Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

# КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Програмно-апаратна система зберігання даних з віддаленим доступом на основі Nextcloud
Назва теми

КвРКІ 200113.20.01.13 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент IV курсу, група КІ2-20-1 _____ А.С. Крупецький
Підпис, Ініціали, прізвище

Керівник _____ В.В. Яцків
Підпис, дата, Ініціали, прізвище

Нормоконтролер _____ С.М. Лисенко
Підпис, дата, Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис, Ініціали, прізвище

«21» червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень БАКАЛАВР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

"10" 01 2024 р.

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА**

Крупецькому Анатолію Сергійовичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Програмно-апаратна система зберігання даних з віддаленим доступом на основі Nextcloud

Керівник проекту (роботи) Яцків В.В., д.т.н., проф.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.02.2024 р. № 8

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2024 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз програмно-апаратної системи зберігання даних з віддаленим доступом на основі Nextcloud

Вибір компонентів розробки програмно-апаратної системи зберігання даних з віддаленим доступом на основі Nextcloud

Програмно-апаратна реалізація системи зберігання даних з віддаленим доступом на основі Nextcloud

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Схема взаємодії компонентів системи

Схема бази даних

Схема функціонування програми з користувачем та сервером

6. Консультанти розділів дипломного проекту (роботи)

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Нормоконтроль | Лисенко С.М., професор кафедри КІІС | | |
| Антиплагіат | Нічепорук А.О., доцент кафедри КІІС | | |

7. Дата видачі завдання « 17 » 01 2024 р.

## КАЛЕНДАРНИЙ ПЛАН

| №з/п | Назва етапів (розділів) дипломного проекту (роботи) | Термін виконання етапів проекту (роботи) | Примітка |
|---|---|---|---|
| 1 | Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником | 17.01.2024 | виконано |
| 2 | Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження | 15.02.2024 | виконано |
| 3 | Робота над розділом 1 – аналіз програмно-апаратної системи зберігання даних з віддаленим доступом на основі Nextcloud | 17.02.2024 | виконано |
| 4 | Робота над розділом 2 – вибір компонентів розробки програмно-апаратної системи зберігання даних з віддаленим доступом | 15.03.2024 | виконано |
| 5 | Робота над розділом 3 – програмно-апаратна реалізація системи зберігання даних з віддаленим доступом на основі Nextcloud | 08.04.2024 | виконано |
| 6 | Оформлення пояснювальної записки згідно вимог | 25.05.2024 | виконано |
| 7 | Попередній захист ВКР | 30.05.2024 | виконано |
| 8 | Захист ВКР на засіданні ЕК | 24.06.2024 | |

Студент _____ А.С. Крупецький
Підпис Ініціали, прізвище

Керівник роботи _____ В.В. Яцків
Підпис Ініціали, прізвище

| № рядка | формат | Позначення | Найменування | Кіл. листів | № екз | Примітка |
|---|---|---|---|---|---|---|
| | | | Текстові документи | | | |
| 1 | | КвРКІ 200113.20.01.13 ПЗ | Пояснювальна записка | 57 | | |
| | | | | | | |
| | | | Графічні матеріали | | | |
| 2 | | КвРКІ 200113.20.01.13 Е8 | Схема взаємодії компонентів системи | 1 | | |
| 3 | | КвРКІ 200113.20.01.13 Е8 | Схема бази даних | 1 | | |
| 4 | | КвРКІ 200113.20.01.13 Е8 | Схема функіонування програми з користувачем та сервером | 1 | | |

| Зм | Арк | № докум | Підпис | Дата | КвРКІ 200113.20.01.13 ВП | | | |
|---|---|---|---|---|---|---|---|---|
| Розробив | | Крупецький | | | Відомість проекту | Літера | Аркуш | Аркушів |
| Перевір. | | Яцків | | | | У | 1 | 1 |
| Н. контр. | | Лисенко | | | | ХНУ, КІ2-20-1 | | |
| Затв. | | Говорущенко | | 21.06 | | | | |

# АНОТАЦІЯ

Тема кваліфікаційної роботи: «Програмно-апаратна система зберігання даних з віддаленим доступом на основі Nextcloud».

Автор роботи: Крупецький Анатолій Сергійович.

Керівник роботи: Яцків Василь Васильович.

Пояснювальна записка: 57 с., 13 рис., 4 дод., 54 джерела.

Графічна частина: 3 креслення.

Зберігання даних, програмно-апаратна система, архітектура, віддалений доступ, база даних.

Метою дипломної роботи є розробка та впровадження програмно-апаратної системи зберігання даних з віддаленим доступом на основі платформи Nextcloud. Робота спрямована на створення ефективного інструменту для організації роботи з даними, забезпечення їх безпеки та доступності для користувачів з будь-якого місця та пристрою.

Об'єктом дослідження є процес зберігання даних з можливістю віддаленого доступу.

Предметом дослідження є методи та алгоритми зберігання даних на основі Nextcloud, включаючи аналіз вимог, вибір технологій, розробку програмного забезпечення та налаштування апаратної інфраструктури.

Підпис студента

30.05.2024

Дата

## ЗМІСТ

# ВСТУП

Хмарне середовище представляє собою підхід до обчислень та зберігання даних, де користувачі можуть отримувати доступ до різних обчислювальних ресурсів через Інтернет. Замість використання власних обчислювальних потужностей, люди можуть орендовувати або використовувати послуги від постачальників хмарних послуг.

У хмарному середовищі користувачі можуть легко масштабувати свої обчислювальні ресурси в залежності від потреб, забезпечуючи ефективне використання ресурсів та оплату лише за фактично використані об'єми. Також характерною особливістю є доступність з будь-якого місця, що робить цю технологію зручною для роботи на відстані та глобальної співпраці. Хмарні технології також сприяють спільному використанню ресурсів, спрощують самообслуговування та автоматизацію, а також пропонують гнучку систему оплати за використані ресурси.

Різні постачальники хмарних послуг, такі як Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP) та Nextcloud, надають користувачам можливість використовувати ці технології для зберігання даних, обчислень, вирішення задач та інших завдань.

Метою дипломної роботи є розробка та впровадження програмно-апаратної системи зберігання даних з віддаленим доступом на основі платформи Nextcloud. Робота спрямована на створення ефективного інструменту для організації роботи з даними, забезпечення їх безпеки та доступності для користувачів з будь-якого місця та пристрою.

Об’єктом дослідження є процес зберігання даних з можливістю віддаленого доступу.

Предметом дослідження є методи та алгоритми зберігання даних на основі Nextcloud, включаючи аналіз вимог, вибір технологій, розробку програмного забезпечення та налаштування апаратної інфраструктури.

# 1 АНАЛІЗ ПРОГРАМНО-АПАРАТНОЇ СИСТЕМИ ЗБЕРІГАННЯ ДАНИХ З ВІДДАЛЕНИМ ДОСТУПОМ НА ОСНОВІ NEXTCLOUD

## 1.1 Аналіз предметної області і виявлення наявних проблем і завдань

Розпочати слід з того, що таке хмарне середовище – це модель збереження даних у комп'ютері, в якій цифрові дані накопичуються в логічні пули, а фізичне зберігання охоплює кілька серверів (зазвичай у кількох місцях). Фізичне середовище, як правило, належить хостинговим компаніям, які ним керують. Ці постачальники хмарних систем зберігання даних відповідають за утримання наявної інформації та доступ до неї, а також за роботу фізичного середовища. Користувачі можуть купувати у постачальників послуг хмарного сховища можливість накопичувати там дані [1].

Хмарні технології зародилися в 1950-х роках, коли вчені вперше заговорили про концепцію розділення часу. Полягала вона в наступному: комп’ютери вартували надто дорого, тому придбати їх усім співробітникам було неможливо, але замість цього декілька людей могли б разом працювати на одному загальному процесору. Така ідея з’явилася в 1954 році та її реалізація розпочалась в 1959 році, а перше комерційне рішення було випущено в 1964 році [2].

Відношення до обчислювальної потужності як до ресурсу, подібному до електрики та води, привело до появи комп’ютерного бюро, де клієнти могли придбати необхідний об’єм потужності для виконання розрахунків. Ця модель функціонувала до 1980-х років – тоді з’явилися персональні комп’ютери, після чого вона втратила свою актуальність.

Другим важливим фактором, що вплинув на сучасні хмарні сервіси, стала можливість підключення до глобальної мережі. Це основний принцип технології: користувачі повинні мати доступ до сервісу з будь-якого куточку світу.

Перші процесори та їх користувачі, як правило, знаходилися в одній будівлі. Локальні мережі працювали в США вже з кінця 1950-х років, а вже через 10 років вчений Джозеф Карл Робнет Ліклайдер (Joseph Carl Robnett Licklider) запропонував

створити з обчислювальних центрів глобальну мережу. В 1962 році науковець очолив проект по створенню мережі Міністерства оборони США, гірського комплексу Шайенн (бункер в Колорадо) та Стратегічного командування ВВС США.

У 1966 році почався розвиток ARPANET, початковий вигляд зображено на рисунку 1.1, найбільшого проекту, ядро якого на початку 1990-х еволюціонувало в сучасний інтернет. Нова мережа розвивалась, сервіси, які в ній були доступні залучали все більше людей, а відповідно, вимагали все більше обчислювальної потужності. Історія вийшла на друге коло[3].

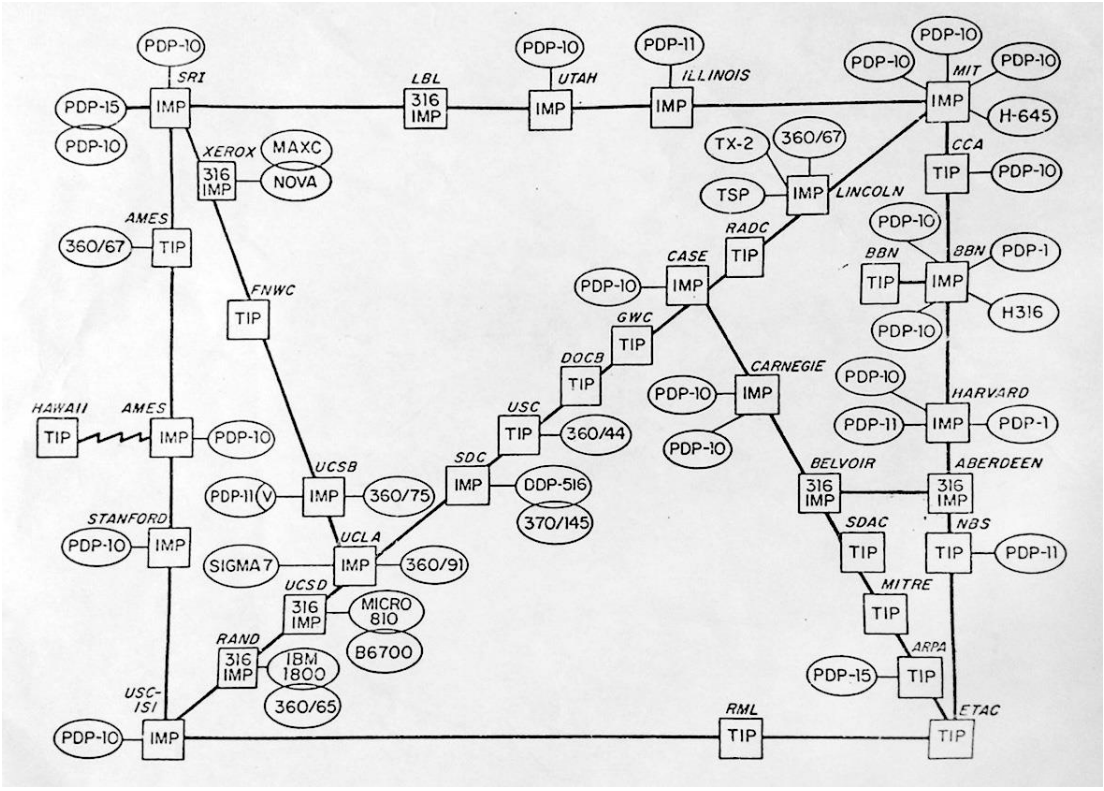



Рисунок 1.1 – Логістична мапа «ARPANET» [4]

Третій значний фактор в історії хмарних сервісів стала віртуалізація. Користувачам необхідні були цифрові системи, котрі не залежать від конкретного інструментарію та дозволяють починати та закінчувати роботу у будь-який момент.

Вперше таку концепцію експериментально запровадили ще в 1966 році, а комерційний варіант з’явився в 1972 році, зображено на рисунку 1.2, його

представила компанія IBM [5]. Сучасні функції віртуалізації x86 були додані до лінійки процесорів Intel у 2005 році (VT-x) та до процесорів AMD у 2006 році (AMD-V).

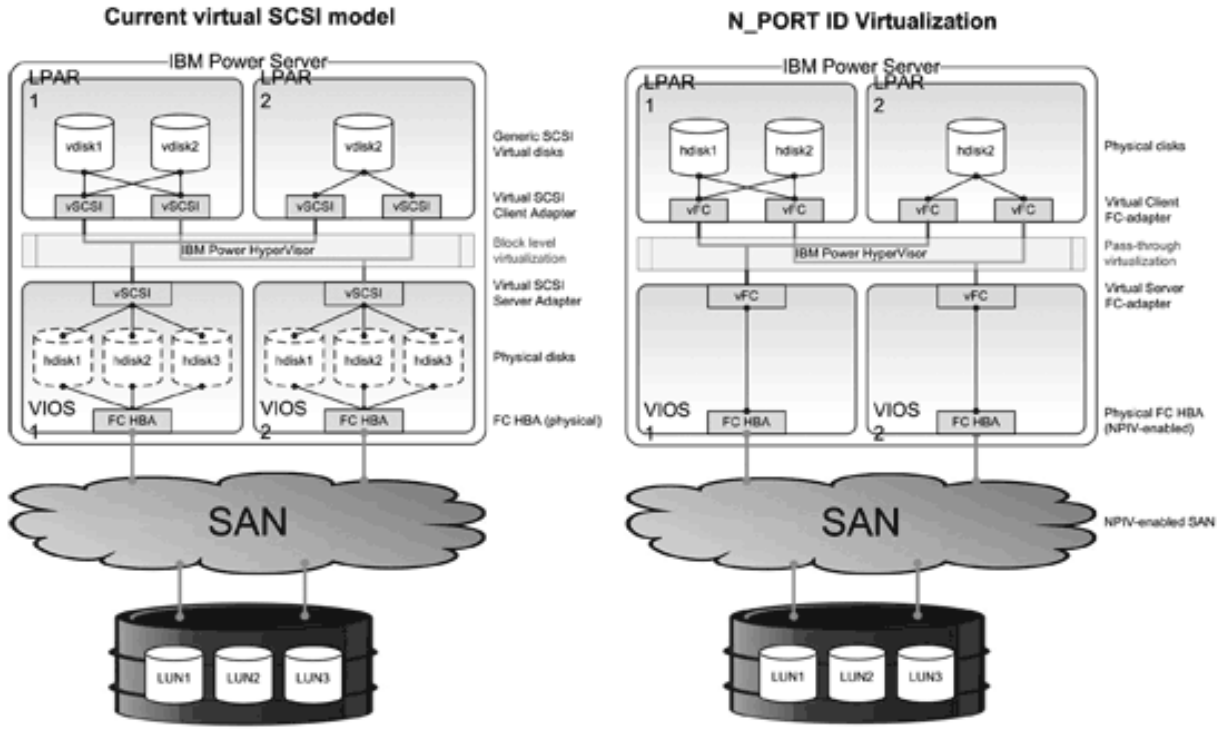



Рисунок 1.2 – Система віртуалізації IBM [6]

Складно сказати, хто і коли увів термін «хмара». З розвитком інтернету стали популярні онлайн-сервіси. Їх стали називати SaaS (Software as a Service — «програмне забезпечення як послуга»), аби відрізнити від десктопних версій, які потрібно встановлювати на комп’ютер.

У інтернет ажіотажу було 2 важливих наслідки. По-перше, швидко зросла кількість розробників, тому потрібно було б спростити процес розміщення нових програм. Так народилась ідея PaaS (Platform as a Service — «платформа як послуга»). Першим таким сервісом став Zimki , запущений у 2006 році [7]. У 2008 році Google представила на розсуд App Engine, який вже пізніше став хмарною платформою Google [8].

По-друге, декотрі інтернет-компанії стали дуже великими та потребували велику кількість обчислювальних потужностей. Вони були потрібні їм лише в певні

моменти, наприклад, інтернет-магазинам під час розпродажів у «чорну п'ятницю». Але більшу частину часу весь об'єм потужностей був не потрібний, та бізнес став передавати їх третім особам – це привело до створення IaaS (Infrastructure as a Service — «інфраструктура як послуга»).

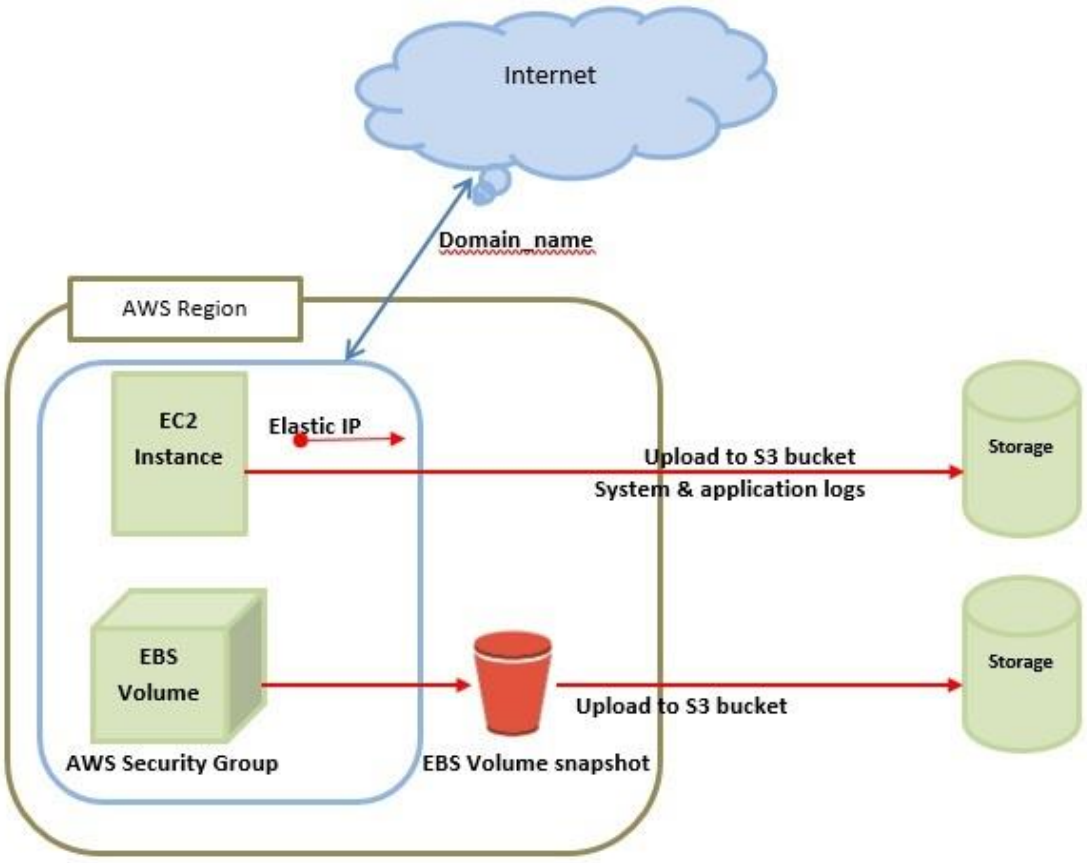



Рисунок 1.3 – Базова архітектура Amazon Web Services [9]

Amazon Web Services став першим IaaS-сервісом, або хмарою у сьогоденному розумінні, зображено на рисунку 1.3. Microsoft запустила подібний сервіс Azure у 2010 році, а Google — Google Compute Engine у 2012 році. Інші компанії незабаром зрозуміли потенціал хмарних технологій та приєдналися до перегонів, але Amazon, Microsoft та Google все ще значно випереджають їх [10].

Багато з сьогоднішніх хмарних технологій стали результатом багатолітньої роботи у певній сфері та перед публічним запуском тестувались всередині компаній.

Цікавим прикладом являється Google Spanner — перша та поки що єдина розподільна реляційна база даних з гарантійною узгодженістю. Вона використовується для підтримки роботи усієї рекламної системи Google — класичним реляційним базам це не під силу [11].

У цілому, хмарні технології рухаються до автоматизації як можна більшого кількості аспектів розробки. Для цього мінімізується об’єм необхідного коду, налаштування середовища та ймовірність помилок.

Сучасні проблеми хмарних середовищ:

1. Питання безпеки та збереження конфіденційності даних в хмарному середовищі залишаються актуальними. Великі обсяги даних можуть стати об'єктом кібератак.

2. З підвищенням попиту на хмарні ресурси виникають питання щодо масштабованості та продуктивності платформ.

3. Хоча хмарні середовища зазвичай добре забезпечені, проблеми з доступністю час від часу можуть виникати, призводячи до відмов в обслуговуванні.

4. Вартість використання хмарних послуг та нестабільність цін можуть становити виклики для бюджетного планування.

5. Управління даними, їх зберігання та вилучення з хмарного середовища.

## 1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

Використання хмарних технологій має численні переваги та може бути вигідним для різних видів бізнесів та організацій.

Основні причини використання хмарних технологій включають:

1. Замість інвестування в власні обчислювальні ресурси, організації можуть використовувати хмарні послуги та оплачувати лише ті ресурси, які вони реально використовують. Це може значно зменшити витрати на обладнання та підтримку.

2. Хмарні рішення надають можливість швидко масштабувати ресурси в залежності від потреб. Це дозволяє еластично адаптуватися до змін обсягів роботи та підтримувати оптимальну продуктивність.

3. Завдяки хмарним технологіям користувачі можуть отримувати доступ до своїх даних та програм в будь-якому місці та в будь-який час, що полегшує роботу на відстані та дозволяє глобальним командам ефективно співпрацювати.

4. Хмарні середовища надають інструменти для автоматизації процесів розгортання, масштабування та керування ресурсами. Це спрощує управління інфраструктурою та підтримку.

5. Постачальники хмарних послуг постійно впроваджують нові технології та оновлення, що дозволяє організаціям користуватися останніми розробками без необхідності великих інвестицій та трудомістких оновлень.

6. Багато хмарних провайдерів надають високий рівень безпеки та автоматичне резервне копіювання даних. Це дозволяє забезпечити надійність та захист інформації від втрати чи несанкціонованого доступу.

7. Використання хмарних технологій дозволяє бізнесу швидше реагувати на зміни у внутрішньому та зовнішньому середовищі, сприяє швидкому впровадженню нових ідей та вирішенню завдань.

Загально кажучи, хмарні технології можуть стати стратегічним інструментом для підтримки бізнес-потреб, полегшуючи роботу, зменшуючи витрати та забезпечуючи високий рівень доступності та ефективності.

Хоча хмарні середовища мають численні переваги, вони також мають свої недоліки, які можуть впливати на певні аспекти використання цих технологій.

Декілька основних недоліків хмарних середовищ включають:

- Для доступу до хмарних послуг та даних користувачам необхідне стійке та високошвидкісне Інтернет-з'єднання. Відсутність зв'язку може призвести до обмеження доступу та працездатності.

- Передача та зберігання даних в хмарних середовищах може викликати питання щодо приватності та безпеки. Обробка конфіденційної інформації може бути суттєвою проблемою, особливо у випадку чутливих даних.

- Захист від кіберзлочинності та загроз безпеки залишається важливою проблемою. Хмарні системи можуть стати об'єктом кібератак, які можуть призвести до втрати даних чи порушення доступності сервісів.

- Користувачі можуть стикатися з обмеженим контролем та прозорістю стосовно інфраструктури та управління хмарними послугами, що може впливати на їхню спроможність відстежувати та перевіряти дії.

- Хоча використання хмарних послуг може зменшити капітальні витрати, можуть виникнути непередбачені витрати через додаткові послуги, платні опції або неочевидні вартості.

- Організації, які повністю переходять до хмарних рішень, можуть втратити частину контролю над своєю інфраструктурою та даними, оскільки вони знаходяться під управлінням постачальників хмарних послуг.

- Відмови або збої в роботі хмарних сервісів можуть призвести до тимчасової недоступності або втрати даних, особливо у випадку низької відмовостійкості.

- З погляду юридичної сторони, виникають питання про відповідальність за захист та використання даних в хмарних середовищах, а також можливі правові обмеження.

Враховуючи ці недоліки, важливо ретельно розглядати та приймати рішення у виборі хмарного середовища.

Види хмарних послуг, зображено на рисунку 1.4 [12]:

- IaaS — Infrastructure as a Service. Постачальник IaaS надає свої сервери та обчислювальні потужності, тобто доступ до інфраструктури. В такому випадку можна встановлювати на сервери будь-які ОС та програми. Це дозволяє компаніям розгортати та обслуговувати власні застосунки, зберігати всі дані, пов’язані з роботою цієї програми чи сайту на серверах постачальника.

Тоді не потрібно мати свій фізичний сервер та обслуговувати його — всі ці турботи лежать на плечах постачальника послуг. Проте компанія не матиме можливості керування базовою інфраструктурою клауду, адже це — «територія» провайдера. Приклад такого хмарного сервісу — хостинг від NETFORCE Ukraine [13].

- SaaS — Software as a Service. Це програми на основі передплати. Користувач просто логіниться в системі через браузер чи застосунок в телефоні та отримує доступ до її функцій. Програму не потрібно встановлювати на свій комп'ютер та обслуговувати її.

Компанії часто використовують такі інструменти у роботі, наприклад, Google Workspace, Slack, Nextcloud або Jira. ПЗ та інфраструктура в таких випадках розміщуються та управляється Google, Slack, Nextcloud та Atlassian (якщо мова про Jira).

- PaaS — Platform as a Service. PaaS надає розробникам платформу для створення та розгортання ПЗ. Сюди входять інфраструктура, ОС, середовище для розробки програм та інструменти для керування БД. Це полегшує та пришвидшує розробку. Приклади таких рішень: Heroku та Microsoft Azure [14].

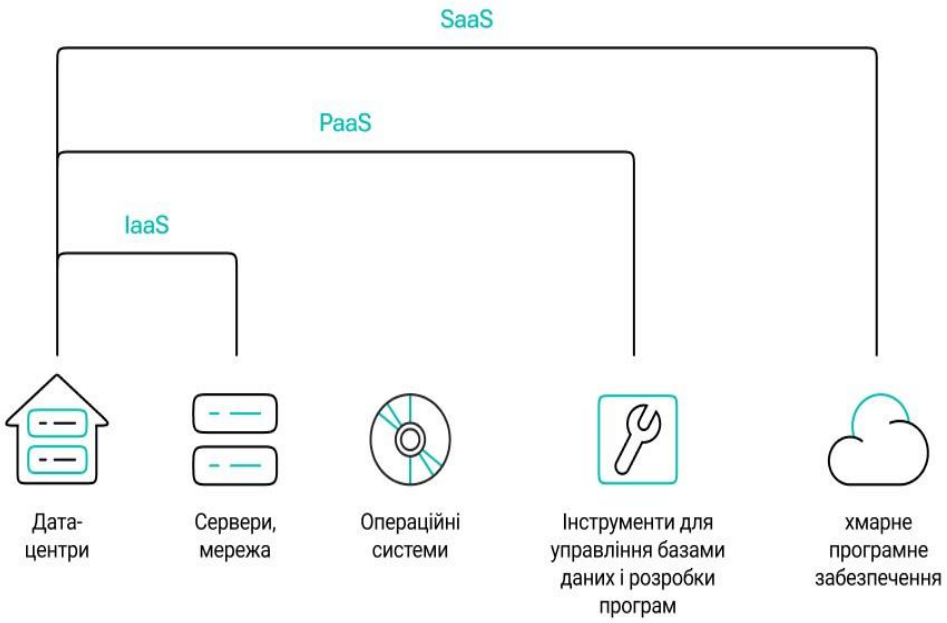



Рисунок 1.4 – Види хмарних послуг [15]

Amazon Web Services (AWS) є однією з найбільших та найвпливовіших хмарних платформ у світі, надаючи широкий спектр послуг для підтримки різних бізнес-потреб [16].

Розглянемо детальніше характеристики AWS:

– AWS володіє центрами обробки даних та регіональними зонами доступності по всьому світу, що дає змогу користувачам розміщувати свої дані та додатки ближче до кінцевих споживачів.

– AWS надає можливість масштабування ресурсів в залежності від потреб бізнесу, що дозволяє еластично адаптуватися до змін обсягів роботи та забезпечувати високу доступність.

Основні сервіси для розробки:

– AWS Elastic Compute Cloud (EC2) дозволяє створювати та масштабувати віртуальні сервери для обчислень.

– Amazon Simple Storage Service (S3) - надійне та масштабоване сховище для зберігання та отримання будь-яких обсягів даних.

– AWS пропонує різноманітні бази даних, такі як Amazon RDS (Relational Database Service), Amazon DynamoDB (NoSQL) та інші.

– Amazon SageMaker - платформа для розробки та впровадження моделей машинного навчання.

– AWS Analytics пропонує послуги для обробки, аналізу та візуалізації даних, включаючи Amazon Redshift та Amazon Quicksight.

Зручність та недоліки:

– AWS надає широкий спектр послуг, що дозволяє користувачам знаходити рішення для різних завдань без необхідності використання кількох платформ.

– Зручно, але в той же час вимагає технічної експертизи для ефективного використання всіх функцій та послуг, що пропонуються.

– Широкий асортимент може ускладнювати процес розуміння та вибору відповідних послуг для конкретних завдань.

Загалом, AWS відомий своєю глобальною інфраструктурою, масштабованістю та широким спектром послуг для розробки, але користувачам може знадобитися технічна експертиза для максимальної ефективності використання.

Microsoft Azure - це хмарна платформа, яка, хоч і має меншу кількість послуг порівняно з AWS, вибірково надає глибоку інтеграцію з екосистемою Microsoft [17].

Переваги Microsoft Azure:

– Azure пропонує широкий спектр хмарних послуг, включаючи обчислення, сховище даних, бази даних, машинне навчання та багато іншого.

– Його перевага полягає в тісній інтеграції з продуктами Microsoft, такими як Windows Server, Active Directory, та Office 365, що робить його привабливим для організацій, які вже використовують ці технології.

– Azure надає гнучкість для розгортання та управління різними типами додатків, включаючи ті, що базуються на мовах програмування та технологіях, які підтримуються Microsoft.

– Для підприємств, що використовують технології Microsoft, Azure стає зручним вибором, оскільки забезпечує спрощену інтеграцію із серверами, сервісами Active Directory, Exchange, SharePoint та іншими продуктами.

Недоліки інтеграції Microsoft Azure та прив'язка до постачальника:

– Інтеграція з екосистемою Microsoft може бути перевагою для тих, хто вже використовує ці продукти, але водночас це може створювати залежність та прив'язку до постачальника послуг, особливо у випадку розгортання широкомасштабних інфраструктур.

– Прив'язка до екосистеми Microsoft може зробити міграцію до інших хмарних провайдерів або локальної інфраструктури менш зручною та більш специфічною.

– Інтеграція та робота з Azure вимагає технічної експертизи, особливо при використанні специфічних продуктів Microsoft.

Azure надає можливості для гнучкого розгортання та інтеграції з екосистемою Microsoft, але разом з цим вносить специфічні виклики, такі як можливість прив'язки та ускладнення міграції до інших рішень. Обираючи між AWS та Azure, бізнесам важливо ретельно враховувати свої потреби, технічні можливості та стратегічні цілі.

Google Cloud Platform (GCP) представляє собою конкурентне хмарне рішення, схоже на AWS та Azure, але з основним акцентом на роботі з великими даними, машинним навчанням та аналітикою [18].

Характеристика Google Cloud Platform:

– GCP відомий своїм потужним інструментарієм для роботи з великими обсягами даних, зокрема, BigQuery для швидкого та масштабованого аналізу.

– Платформа активно розвивається у сфері машинного навчання, пропонуючи інструменти, такі як TensorFlow, та послуги для створення та впровадження моделей.

– GCP намагається привернути користувачів вигідними умовами та привабливим ціноутворенням, щоб завоювати більший ринок.

Однак, варто відзначити, що політика тарифів може змінюватися, і компанія іноді переводить користувачів на інші тарифи, що може вплинути на вартість використання платформи у довгостроковій перспективі.

Nextcloud хоч і не займає перших позицій, як AWS і Microsoft Azure, також має свої особливості та переваги, зокрема щодо глобальної інфраструктури та здатності до розширення.

Nextcloud — це відкрите програмне забезпечення для зберігання та обміну даними, яке може бути розгорнуте в різних середовищах [19]:

– Nextcloud може бути розгорнуто на серверах, що розташовані в різних частинах світу, дозволяючи користувачам отримувати доступ до своїх даних з будь-якого місця [20].

– Залежно від потреб організації, Nextcloud може бути масштабоване для підтримки росту користувачів та обсягів даних.

– Nextcloud пропонує різноманітні функціональні можливості, такі як спільна робота, календарі, завдання, обмін файлами, відеоконференції тощо [21].

– Відкрита архітектура Nextcloud дозволяє інтегрувати його з різними сервісами та додатками, такими як LDAP для управління користувачами чи мережеві сховища для зберігання даних [22].

– Користувачі можуть вибрати між самостійним розгортанням Nextcloud на своєму сервері або використанням готових хмарних служб, що пропонують Nextcloud як готове рішення.

– Схоже на AWS, для оптимального використання Nextcloud може знадобитися технічна експертиза для налаштування та управління, зокрема, визначення найбільш підходящих функцій для бізнес-потреб.

Nextcloud надає користувачам більший контроль над своїми даними та конфігурацією порівняно із застосуванням готових хмарних рішень.

Отже, Nextcloud дозволяє гнучко вибирати між самостійним розгортанням та використанням готових рішень, а також має можливість інтеграції та масштабованості для відповіді на потреби різних бізнес-сценаріїв.

Кожен постачальник, включаючи AWS, Azure та GCP, має свої переваги та обмеження. AWS славиться широким спектром послуг, Azure - інтеграцією з продуктами Microsoft, GCP - фокусом на великих даних та машинному навчанні, а Nextcloud - своєю орієнтацією на безпеку та контроль над даними, забезпечуючи локальне зберігання.

Перед вибором хмарного постачальника важливо детально ознайомитися з послугами та розглянути їхні характеристики, щоб забезпечити вибір, який найкраще відповідає потребам бізнесу.

Прийняття стратегічного вибору між AWS, Azure, GCP та Nextcloud може базуватися на гнучкості платформи, партнерствах, вартості та стратегічних цілях бізнесу.

Вибір хмарного постачальника повинен враховувати потреби конкретного бізнесу та його технологічні стратегії на довгострокову перспективу.

## 1.3 Підходи до вирішення задачі за темою дослідження та постановка задачі

Так як потрібно створити систему зберігання даних з віддаленим доступом буде доцільним використання хмарних технологій Nextcloud. Для програмування може бути використана мова PHP, Python, Java або інші.

Завданнями роботи є:

– дослідити процедури функціонування хмарного середовища;

– провести теоретичний аналіз сфери;

– охарактеризувати структуру предметної області та базову модель;

– описати уже існуючі механізми реалізації, виділити наявні проблеми в галузі та шляхи їх вирішення;

– на основі проведених досліджень визначити основні функції системи, сформувати низку функціональних та нефункціональних вимог, розробити модель функцій;

– сформувати об’єкт та мету для наступних досліджень;

– оцінити ступінь виконання поставлених завдань.

На основі цього розробити програмно-апаратну систему зберігання даних з віддаленим доступом за допомогою хмарного сервісу Nextcloud.

## 1.4 Висновки

У межах розділу 1 проведено аналіз предметної області, спрямований на виявлення наявних проблем та завдань. Визначені основні проблеми, що виникають у сфері зберігання даних з віддаленим доступом, такі як недостатня ефективність, обмежені можливості управління та обробки даних, а також проблеми забезпечення безпеки. Далі був здійснений порівняльний аналіз переваг та недоліків існуючих рішень. Оцінено різні підходи до зберігання даних з

віддаленим доступом, включаючи хмарні сервіси, локальні сервери та віртуальні приватні мережі. З'ясовано, що існуючі рішення часто мають обмеження або не відповідають усім потребам користувачів.

На основі проведеного аналізу було сформульовано постановку задачі дипломної роботи. Головною метою є розробка програмно-апаратної системи зберігання даних з віддаленим доступом, яка буде ефективно вирішувати існуючі проблеми та відповідати потребам користувачів. Для досягнення цієї мети необхідно детально розглянути архітектуру системи, вибрати оптимальні технології та інструменти розробки, а також провести ефективне тестування та впровадження системи. Такий підхід дозволить створити продукт, який буде відповідати сучасним вимогам та вирішувати актуальні проблеми у галузі зберігання даних з віддаленим доступом.

# 2 ВИБІР КОМПОНЕНТІВ РОЗРОБКИ ПРОГРАМНО-АПАРАТНОЇ СИСТЕМИ ЗБЕРІГАННЯ ДАНИХ З ВІДДАЛЕНИМ ДОСТУПОМ НА ОСНОВІ NEXTCLOUD

## 2.1 Визначення мови програмування

Зробимо порівняння сучасних мов програмування за декількома ключовими критеріями: простота використання, продуктивність, сфера застосування, та спільнота та екосистема.

Python – це дуже проста у використанні мова, підходить для початківців. Має середню продуктивність, інтерпретована мова. Основні сфери застосування: наука про дані, машинне навчання, веб-розробка, автоматизація. Має дуже велику спільноту, багато бібліотек та фреймворків [23].

JavaScript – це мова, яка є середньо-важкою у використання, має простий синтаксис, але є нюанси з асинхронністю. Продуктивність середня, але дуже швидка для веб-розробки завдяки оптимізованим рішенням браузерів. Основні сфери застосування: веб-розробка (клієнтська і серверна), мобільні додатки (з використанням фреймворків типу React Native). Має велику спільноту, багата екосистема (React, Angular, Vue.js) [24].

Java – мова, яка неважка у використанні, має строгий синтаксис та потреба в детальному описі структури. Має високу продуктивність, завдяки JIT-компіляції. Основні сфери застосування: корпоративні додатки, мобільні додатки (Android), великі системи. Має дуже велику спільноту, багато бібліотек та інструментів [25].

C# – це мова подібна до Java з деякими спрощеннями. Має високу продуктивність, завдяки JIT-компіляції та оптимізаціям. Основні сфери застосування: десктопні додатки, ігри (Unity), веб-додатки (ASP.NET). Має велику спільноту, особливо в сфері ігор та бізнес-додатків [26].

C++ – мова, яка важка у використанні, має складний синтаксис та концепції. Забезпечує високу продуктивність, низькорівневий контроль над пам'яттю. Основні сфери застосування: системне програмування, ігри, вбудовані системи,

продуктивні застосунки. Має велику спільноту, багато бібліотек, але менше готових рішень [27].

Go – важка у використанні, має простий синтаксис та парадигми. Має високу продуктивність, компільована мова з гарною підтримкою конкурентності. Основні сфери застосування: мережеві сервіси, мікросервіси. Спільнота поступово збільшується, особливо серед DevOps та мережевих розробників [28].

Rust – мова, яка є важкою у використанні, має складний синтаксис та концепції, але забезпечує безпеку пам'яті. Забезпечує дуже високу продуктивність, у порівнянні з C/C++. Основні сфери застосування: системне програмування, високопродуктивні програми. Спільнота стрімко збільшується, особливо серед розробників систем та високопродуктивних додатків [29].

Swift – це важка у використанні мова, яка має сучасний та зручний синтаксис. Забезпечує високу продуктивність, є компільованою мовою. Основні сфери застосування: додатки для iOS та macOS. Має велику спільноту, підтримується Apple та багато інструментів для розробки під iOS/macOS [30].

Java має кілька значущих переваг, які роблять її популярною мовою програмування. Вона є простою у використанні, оскільки мова має строгий синтаксис і потребує детального опису структури, що сприяє підтримуваності та читабельності коду. Ця особливість робить Java надійною для великих команд розробників і проектів, де важлива чітка організація коду.

Продуктивність мови Java висока завдяки використанню JIT-компіляції (Just-In-Time), яка дозволяє виконувати код швидко та ефективно. Віртуальна машина Java (JVM) оптимізує виконання програми, забезпечуючи високу продуктивність навіть для великих і складних додатків. JVM також забезпечує платформну незалежність, дозволяючи запускати Java-додатки на будь-якій операційній системі з підтримкою JVM.

Java широко застосовується в різних сферах, зокрема в корпоративних додатках, де потрібна надійність і масштабованість. Вона є основною мовою для розробки мобільних додатків на платформі Android, що робить її важливою для

розробників мобільних додатків. Крім того, Java використовується для створення великих розподілених систем, серверних додатків і веб-сервісів, завдяки своїй здатності обробляти великі обсяги даних і підтримувати високі навантаження.

Спільнота та екосистема мови програмування Java дуже великі і активно розвиваються. Існує безліч бібліотек, фреймворків та інструментів, що полегшують розробку і дозволяють швидко вирішувати різноманітні завдання. Наприклад, Spring та Hibernate є популярними фреймворками для розробки веб-додатків та роботи з базами даних. Велика спільнота розробників означає, що для Java існує багато ресурсів для навчання, документації та підтримки, що сприяє швидкому вирішенню проблем і розвитку нових ідей [31].

Інші переваги Java включають вбудовану підтримку багатозадачності, завдяки якій можна створювати ефективні багатопотокові програми, та високу безпеку, яка забезпечується через строгі правила управління пам'яттю та вбудовані засоби безпеки. Усі ці фактори роблять Java однією з найпопулярніших і найнадійніших мов програмування у світі.

Основна перевага Java полягає в підтримці об’єктно-орієнтованого програмування (ООП). Це дозволяє розробникам організовувати код як об’єкти, які містять і дані, і поведінку, уможливлюючи модульне проектування програмного забезпечення. Ось деякі функції ООП, які надає Java [32]:

– Об’єднання даних із пов’язаними методами в межах класів забезпечує інкапсуляцію, гарантуючи цілісність даних і спрощуючи обслуговування коду.

– Використовування успадкування для створення нових класів на основі існуючих, дозволяючи ієрархічні зв’язки між класами.

– Завдяки поліморфізму різні об’єкти можуть по-різному реагувати на той самий виклик методу під час виконання завдяки динамічному зв’язуванню.

Однією з суттєвих переваг використання Java є його функція незалежності від платформи, яка надає можливість запустити будь-де програму, через інтерпретацію байт-коду, а не пряму компіляцію в інструкції для конкретної машини.

Спрощений досвід кодування, де вихідний код компілюється в байт-код, що виконується віртуальною машиною під назвою JVM, а саме:

– JVM діє як проміжний рівень між апаратним забезпеченням і програмами Java, що забезпечує незалежність від платформи.

– Якщо на пристрої встановлено JVM, програми Java можуть безперебійно працювати на різних платформах.

Java піклується про керування пам’яттю за допомогою вбудованого збирача сміття. Він автоматично відновлює пам’ять, зайняту об’єктами, які більше не використовуються, усуваючи ручне звільнення пам’яті та зменшуючи помилки програмування, пов’язані з керуванням пам’яттю, докладніше про це:

– За допомогою сучасних методів збирання сміття, як-от позначення та очищення або генераційний GC, Java підвищує ефективність під час керування життєвими циклами об’єктів.

– Збирач сміття визначає та звільняє об’єкти, які більше не потрібні або недоступні через втрату посилань.

– Завдяки автоматичному управлінню пам'яттю розробники можуть більше зосередитися на логіці додатків, а не турбуватися про звільнення вручну.

Під час виконання програмного забезпечення неминуче виникають помилки. Java надає надійні механізми обробки винятків для виявлення помилок і забезпечення плавного виконання програми:

– Блок try-catch допомагає розробникам витончено обробляти як відомі, так і непередбачені винятки.

– Java розрізняє перевірені (під час компіляції) винятки, які вимагають явної обробки, і неперевірені (під час виконання) винятки, де явна обробка необов’язкова.

– Використання ієрархії класів типів винятків дозволяє розробникам з точністю відловлювати конкретні винятки або фіксувати широкі категорії для загальних протоколів обробки помилок.

Java пропонує чудову підтримку можливостей багатопоточності для ефективної одночасної обробки. Ця функція дозволяє програмістам виконувати кілька потоків одночасно в одній програмі, а саме:

– Систематичне використання ресурсів ЦП шляхом одночасного виконання потоків забезпечує оптимальну продуктивність системи.

– Java надає синхронізовані блоки та методи для забезпечення синхронізації потоків і уникнення проблем із пошкодженням даних.

– Можливість використання паралелізму, використовуючи функції багатопоточності Java для завдань, які можна обробляти одночасно.

Java може похвалитися комплексною стандартною бібліотекою (також відомою як «Бібліотека класів Java» або «API»), яка надає розробникам великий набір функцій, класів та інтерфейсів. Ця бібліотека спрощує типові завдання програмування, дозволяючи розробникам пришвидшити процес розробки:

– Стандартна бібліотека включає в себе безліч корисних класів, що охоплюють такі основні функції, як файлові операції, мережа, обробка рядків тощо.

– Структура бібліотеки пропонує різноманітні структури даних і реалізацію алгоритмів простим у використанні способом, наприклад, ArrayLists, LinkedLists, HashMaps, що дозволяє ефективно маніпулювати даними.

Безпека має вирішальне значення під час розробки додатків, що включають конфіденційну інформацію користувача або корпоративні активи. Java містить кілька вбудованих функцій безпеки, які забезпечують надійний захист:

– Перед виконанням байт-коду в середовищі пісочниці платформ JVM воно проходить сувору перевірку, щоб запобігти виконанню зловмисного коду.

– Контроль рівнів доступу для членів класу за допомогою модифікаторів доступу, таких як приватний, захищений публічний у поєднанні з механізмом контролю видимості пакетів

– Java надає криптографічні API через пакет javax.crypto, що забезпечує безпечне шифрування/дешифрування та механізми хешування.

Потужність і гнучкість мови Java роблять її ідеальним вибором для програмістів у різних областях. Від підтримки об’єктно-орієнтованого програмування до можливостей незалежності від платформи через JVM до функції автоматичного керування пам’яттю – Java надає розробникам потужні інструменти, надаючи їм можливість ефективного формування складних програм.

Java 17 – це реліз формату Long-Term Support із підтримкою до 2029 року. Минулою LTS-версією була Java 11, відома з 2018 року. Можна й надалі користуватися нею, адже знайдені вразливості у безпеці будуть гарантовано виправлятися до вересня 2026 року [33].

У більшості проєктів використовуються LTS-релізи. Проте кожні півроку виходять нові версії формату non-LTS. Це, наприклад, від 12-ї до 16-ї версії включно, а також Java 18 і 19. Проте підтримка таких варіантів завершується із дебютом нового виконання. Саме так розвивається мова програмування Java, переваги LTS-релізів в такому разі очевидні.

Були додані Sealed-класи, розширені екземпляри (це Enhanced instanceof), Switch-вирази, а також текстові блоки й рекорди (або Records). У цьому LTS-релізі Java переваги та недоліки інколи йдуть поруч. Наприклад, з’явився Vector API. Але він є тільки в інкубаторі, і модуль підключається окремою командою. Ще додано Memory API і Foreign Function, а Security Manager помічений Deprecate. Плюс зміни в MacOS, в якій тепер є рендеринг із використанням Metal API. До того ж, з’явилися Null Pointer Exception, які показують багато інформації щодо помилок. І не просто рядок, де зафіксована помилка, а конкретний об’єкт із Null.

Сучасна криптографія та контекстно-залежні фільтри десеріалізації були модернізовані. А псевдовипадкове число розширили, тобто воно тепер ще більш псевдовипадкове. Це безсумнівні переваги мови програмування Java 17, які важливі для проекту.

Розглянемо головні нововведення jdk, в яких було видалено Finalize з деяких методів, раніше після «загибелі» об’єкту він ставав у чергу, аби Finalize реалізував своє призначення. Але його виконання за специфікацією навіть до завершення

роботи вже JVM не було гарантованим. В результаті від цього функціоналу в JDK поступово відмовляються.

Зміни інтерфейсів Packer та Unpacker, реалізації класу Pack200 використовується здебільшого для внутрішніх потреб. А конструктор класу URLDecoder у цьому релізі повністю видалений.

Чимало класів AWT та Swing змінили видимість, велика частина з них були публічними, тобто були доступними звідусіль. Але в цій версій багато класів приховано. Втім, це не стосується основних інтерфейсів, що можна занести до переваг мови програмування Java 17.

Основні зміни jvm відбулися в:

1. Garbage-колекторах.
2. Компіляторі Just-in-Time.

CMS видалено, через що потрібно обирати збирач сміття. За замовчуванням це G1. Також існує ZGC, який представлено ще в 11-й версії як експериментальний. В G1 з'явився Abortable Mixed Collection. Частина сміття буде збиратися без додаткових умов, тому це дозволяє повністю зібрати їх за цикл. А ще G1 тепер вміє аллокувати пам'ять та бути NUMA-Aware. Тобто для багатонодних архітектур буде під нові об'єкти виокремлюватися власне місце в пам'яті, яке розташовано у виконавцю ближче до Thread.

Затримки, які викликає робота Garbage Collector, виправлено через зміни у роботі JVM. Завдяки ним більше процесів можуть тривати одночасно із виконанням застосунку. Тепер GraalVM знаходиться в окремому проєкті. Ключовим нововведенням Java Flight Recorder є можливість стрімингу подій. Вони будуть записані у JFR. Це дозволяє бачити зміни в форматі real time.

G1 — це скорочена назва Garbage-First collector. Він є алгоритмом складання сміття, який представлений у віртуальній машині Oracle HotSpot Java (JVM) 6. Цей збирач виступає довгостроковою заміною CMS та створений для багатопроцесорних комп'ютерів із великим обсягом пам'яті. G1 досягає потрібних цілей з високою ймовірністю за час припинення збору сміття. При цьому він

повинен показувати високу пропускну здатність на фоні незначних потреб в налаштуваннях.

ParallelGC інколи подається як збирач пропускної спроможності. Під час роботи він ставить на паузу потоки та виконує в декілька потоків свою задачу. Завдяки цьому мета досягається без перерв, максимально продуктивно. В більшості випадків це ефективний спосіб зменшити час, потрібний для роботи збирача (у порівнянні із програмою). Хоча інколи паузи застосунку через роботу GC можуть бути й довгими.

Сумісність є ключовим принципом побудови усіх релізів Java. Проте в новій версії пропала чимала кількість методів, які раніше мали відмітку Deprecated.

## 2.2 Вибір середовища програмування

Середовища розробки для Java, відомі як інтегровані середовища розробки (IDE), забезпечують розробникам зручні інструменти для написання, тестування та відлагодження коду. Ось деякі з найпопулярніших IDE для Java [34]:

1. IntelliJ IDEA вважається одним з найкращих IDE для Java завдяки своїм потужним функціям і зручному інтерфейсу. Вона пропонує розумний автозавершення коду, інтеграцію з системами контролю версій, підтримку для розробки веб-додатків та багаті можливості для рефакторингу коду. Існує дві версії: безкоштовна Community Edition та платна Ultimate Edition [35].

2. Eclipse є одним з найбільш використовуваних IDE для Java. Воно пропонує широкий спектр плагінів, які розширюють його функціональність, включаючи підтримку для інших мов програмування та технологій. Eclipse підтримує налагодження, рефакторинг, контроль версій та багато інших функцій, що роблять його потужним інструментом для розробників [36].

3. NetBeans є офіційним IDE для Java від Oracle і має зручний інтерфейс користувача, вбудовані інструменти для створення GUI, підтримку для веб-

розробки та можливість інтеграції з багатьма популярними фреймворками. Воно також підтримує налагодження, автозавершення коду та аналіз коду [37].

4. Android Studio є офіційним IDE для розробки додатків на платформі Android. Воно побудоване на основі IntelliJ IDEA і включає спеціалізовані інструменти для розробки мобільних додатків, такі як візуальний редактор інтерфейсу, Android Emulator та інструменти для аналізу продуктивності додатків [38].

5. BlueJ є простим у використанні IDE, орієнтованим на навчання програмуванню на Java. Воно має зручний інтерфейс, який дозволяє легко створювати та маніпулювати об'єктами, що робить його ідеальним для новачків і студентів [39].

6. JDeveloper є інтегрованим середовищем розробки від Oracle, яке підтримує не лише Java, але й інші мови та технології. Воно пропонує багатий набір інструментів для розробки додатків, включаючи візуальні розробники, інструменти для тестування та налагодження, а також підтримку для розробки веб-додатків та сервісів [40].

7. MyEclipse є комерційним розширенням для Eclipse, яке додає численні додаткові інструменти та функції для розробки Java-додатків. Воно включає підтримку для роботи з базами даних, веб-сервісами, інструменти для розробки мобільних додатків та багато інших корисних розширень [41].

8. DrJava – це легкий IDE, орієнтований на студентів та початківців. Він має простий інтерфейс, інтерактивну консоль та підтримує основні функції для написання, компіляції та налагодження Java-коду [42].

IntelliJ IDEA – це одна з найпопулярніших та найпотужніших інтегрованих середовищ розробки (IDE) для Java. Розроблена компанією JetBrains, IntelliJ IDEA відома своїм інтелектуальним автозавершенням коду, багатими можливостями для рефакторингу та інтеграцією з великою кількістю інструментів та технологій [43].

Інтелектуальне автозавершення IntelliJ IDEA пропонує розширене автозавершення коду, яке розуміє контекст і допомагає швидко знаходити потрібні

класи, методи та змінні. Це значно прискорює процес написання коду та зменшує кількість помилок.

Аналіз коду в реальному часі IDE постійно аналізує код під час написання, пропонуючи рекомендації щодо покращення та виявляючи потенційні помилки. Це дозволяє розробникам оперативно виправляти проблеми і підтримувати високу якість коду.

Потужні засоби рефакторингу IntelliJ IDEA надає широкий набір інструментів для рефакторингу коду, включаючи перейменування, виділення методів, зміни ієрархії класів і багато іншого. Це дозволяє легко і безпечно змінювати структуру коду.

Інтеграція з системами контролю версій IDE підтримує інтеграцію з популярними системами контролю версій, такими як Git, SVN, Mercurial і Perforce. Це дозволяє зручно керувати версіями коду, виконувати коміти, злиття та інші операції безпосередньо з середовища розробки.

Підтримка розробки веб-додатків IntelliJ IDEA включає інструменти для розробки веб-додатків, такі як підтримка для HTML, CSS, JavaScript, TypeScript та інших веб-технологій. Вона також інтегрується з популярними веб-фреймворками, такими як Spring, JavaServer Faces (JSF), Struts і багато інших.

Інструменти для налагодження і тестування IDE має вбудований відладчик, який дозволяє легко відстежувати виконання коду, ставити точки зупинки, переглядати значення змінних та виконувати інші налагоджувальні дії. Вона також підтримує інтеграцію з популярними фреймворками для тестування, такими як JUnit та TestNG.

Інтеграція з базами даних IntelliJ IDEA має вбудовані інструменти для роботи з базами даних, які дозволяють підключатися до баз даних, виконувати SQL-запити, переглядати схеми баз даних та виконувати інші операції з даними.

Підтримка плагінів IDE має велику екосистему плагінів, які дозволяють розширювати її функціональність. Можна знайти плагіни для підтримки різних мов

програмування, фреймворків, інструментів для підвищення продуктивності та багато іншого.

IntelliJ IDEA доступна у двох версіях. Community Edition – це безкоштовна версія, яка включає основні функції для розробки на Java, Groovy, Kotlin та інших мовах програмування. Вона підходить для невеликих проектів та навчання. Ultimate Edition – це платна версія, яка включає всі можливості Community Edition та додаткові інструменти для розробки корпоративних додатків, веб-додатків, роботи з базами даних та інші розширені функції. Вона призначена для професійного використання та великих проектів.

IntelliJ IDEA є потужним інструментом для розробки на Java, який надає розробникам широкий набір функцій для підвищення продуктивності, покращення якості коду та зручного керування проектами. Завдяки своїм інтелектуальним можливостям, інтеграції з численними інструментами та підтримці сучасних технологій, IntelliJ IDEA є відмінним вибором як для початківців, так і для досвідчених розробників.

IntelliJ IDEA з погляду можливостей постачається у двох варіантах: безкоштовного Community edition, і платного Ultimate edition з розширеною функціональністю.

Community edition призначене для JVM- і Android-розробки. Безкоштовна версія підтримує Java, Kotlin, Groovy і Scala; Android; Maven, Gradle і SBT; працює з системами контролю версій Git, SVN, Mercurial і CVS.

Ultimate edition пристосоване для веб- і enterprise-розробки. Ця версія IDE працює не тільки з Git, SVN, Mercurial і CVS, а й із Perforce, ClearCase і TFS. У ньому ви зможете писати JavaScript і TypeScript, звісно ж, є підтримка Java EE, Spring, GWT, Vaadin, Play, Grails і низки інших фреймворків. І, звичайно, не обійшлося без SQL та інструментів для роботи з базами даних.

IntelliJ IDEA підкуповує своїм глибоким розумінням коду, розумною ергономікою, вбудованими функціями для розробки і підтримкою багатьох мов [44].

Підсвічування синтаксису і просте автодоповнення коду – звичайна річ для будь-яких сучасних Java-редакторів. IDEA пішла далі, запропонувавши "розумне автодоповнення". Цей термін означає, що середовище розроблення показує список найбільш релевантних символів, які можна застосувати в наданому контексті. Список символів залежить не тільки від контексту як такого, «загальноприйнятого», а й від стилю програмування розробника, від того, наскільки часто він використовує ті чи ті оператори. «Завершення ланцюжка» і взагалі показує список символів, що застосовуються, що допустимі через методи або геттери в поточному контексті. Крім того, у випадку зі статичними членами або константами IDEA автоматично додає будь-які необхідні оператори імпорту (import). У всіх випадках автодоповнення, IDEA намагається вгадати тип символу під час виконання, уточнити свій вибір і навіть застосувати приведення типів, якщо необхідно.

Код Java часто містить фрагменти з інших мов у вигляді рядків. IDEA може вводити код SQL, XPath, HTML, CSS або JavaScript у рядкові літерали Java. У цьому сенсі IDE може здійснювати рефакторинг коду кількома мовами. Наприклад, якщо ви перейменуєте клас у JPA-відображенні, IDEA оновить відповідний клас сутностей і виразів JPA. IDEA Ultimate знаходить дублікати та схожі фрагменти і також застосовує до них рефакторинг.

IntelliJ IDEA аналізує код під час завантаження і безпосередньо під час введення. Воно вказує на передбачувані проблеми і, за бажанням, пропонує список ймовірних швидких правок до виявлених проблем, зображено на рисунку 2.1.

IntelliJ IDEA спроєктована таким чином, щоб не вибивати зі стану потокової продуктивності. Для всіх дій, які можуть знадобитися під час написання коду, існують комбінації клавіш для їхнього швидкого виклику, зокрема – визначення символів у спливаючих віконцях, зображено на рисунку 2.2.

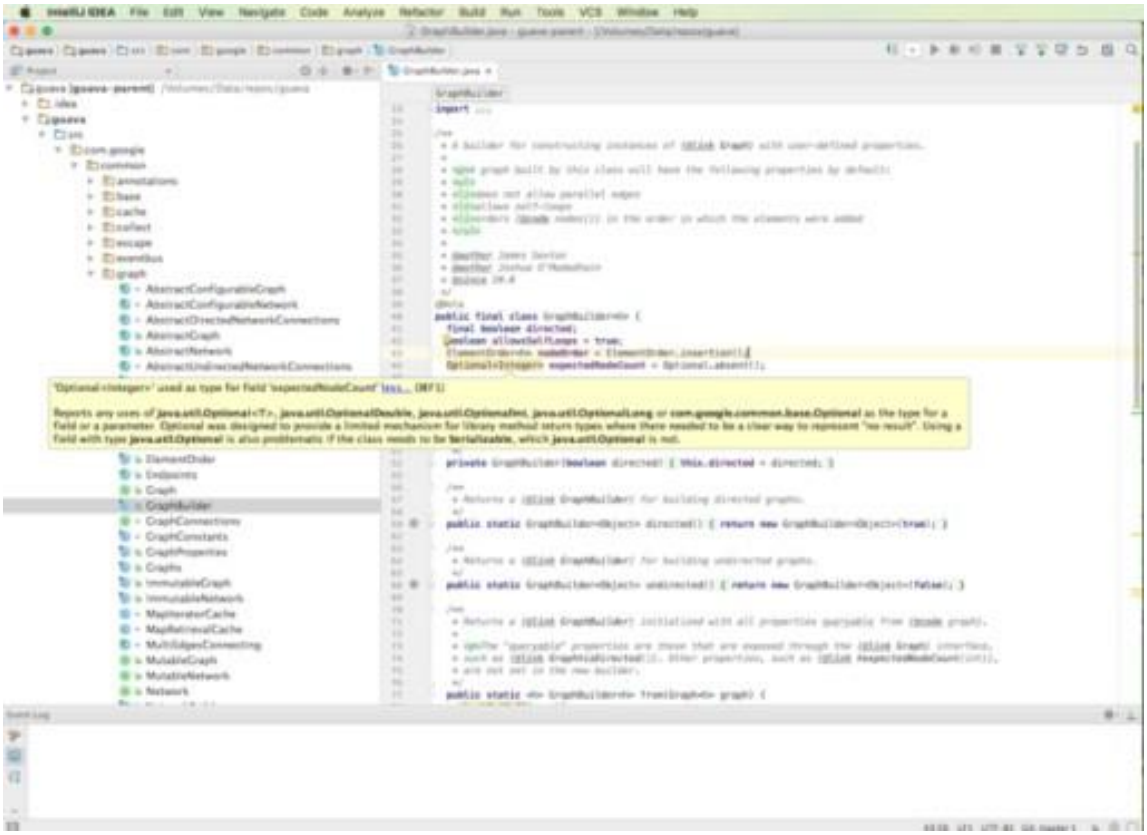


Рисунок 2.1 – IntelliJ IDEA показує кількість попереджень (warnings) і припущення, що ґрунтуються на статистичному аналізі Java-коду [45]

Виділяється відлагоджувач IDEA, в якому значення змінних відображаються безпосередньо у вікні редактора, поруч із відповідним вихідним кодом. У разі зміні стану змінної, колір підсвічування також змінюється.

IntelliJ IDEA забезпечує єдиний інтерфейс взаємодії з більшістю систем контролю версій, зокрема з Git, SVN, Mercurial, CVS, Perforce і TFS. Ви можете керувати змінами безпосередньо в IDE, що дуже зручно. Коли я тестував IDEA, у мене виникало бажання, щоб остання зміна у вихідному коді показувалася у вікні редактора у вигляді анотації (як це відбувається, наприклад, у Visual Studio). Як виявилося, у IDEA є для цього спеціальний плагін.

Також IDEA оснащена інструментами для збірки, середовищем виконання тестів, інструментами покриття (coverage tools) і вбудованим термінальним вікном. У IntelliJ немає власного профайлера, але за допомогою плагінів до неї можна під'єднати сторонні. Наприклад, YourKit, створений колишнім провідним

розробником JetBrains або VisualVM (це перепакована версія профайлера NetBeans).

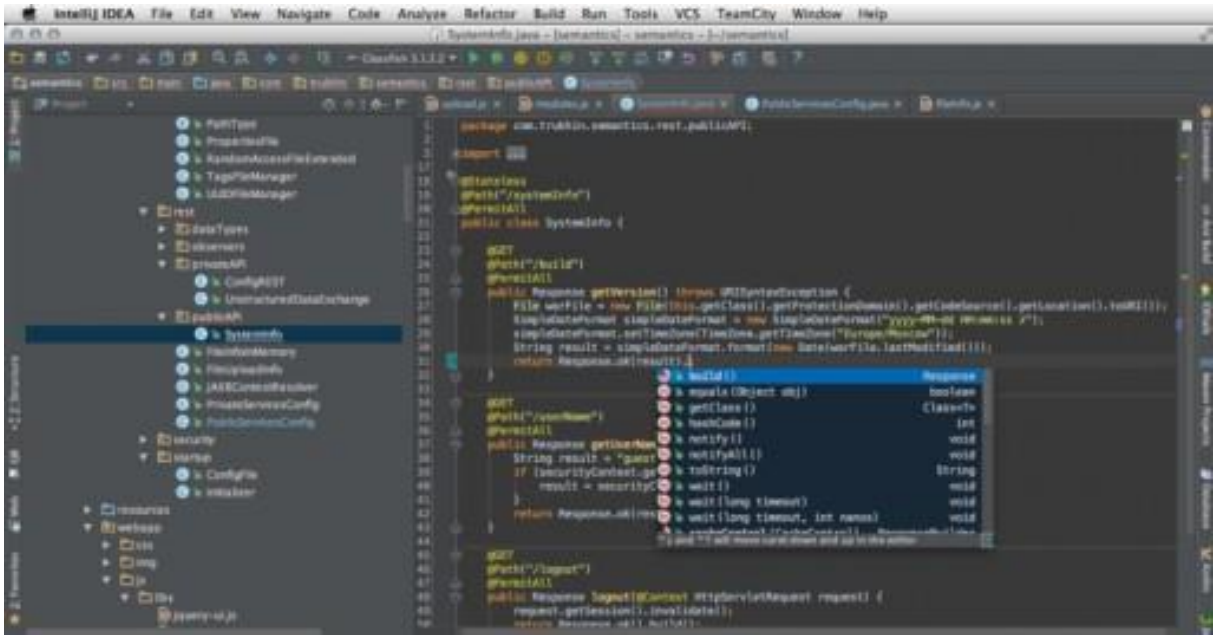


Рисунок 2.2 – Спливаюче віконце визначення символів в IntelliJ IDEA [46]

Налагодження в Java може бути важким, коли відбуваються проблеми з класами, вихідних кодів яких немає. Комплект IDEA містить декомпілятор для таких випадків.

Серверне програмування на Java передбачає часту взаємодію з базами даних, тож програмісти IDEA версії Ultimate оцінять зручність інструментів для роботи з SQL і БД. Але якщо комусь їхніх можливостей буде замало, можна придбати версію IDEA Ultimate із вбудованою SQL IDE (DataGrip). Щоправда, це буде дещо дорожче, ніж звичайна підписка IDEA Ultimate.

IntelliJ IDEA підтримує всі основні сервери застосунків JVM, і дає змогу розгортати і проводити налагодження на цих серверах, що нівелює добре знайомі всім програмістам Java Enterprise труднощі. IDEA також підтримує Docker (за допомогою плагіна, який додає до середовища розробки спеціальне вікно інструментів Docker. До речі, плагінів у IDEA – просто море.

IDEA розширило підтримку коду Spring, Java EE, Grails, Play, Android, GWT, Vaadin, Thymeleaf, Android, React, AngularJS та інших фреймворків. Ви, мабуть, помітили, що не всі з них пов'язані з Java. IDEA безпосередньо з коробки "розуміє" й інші мови – Groovy, Kotlin, Scala, JavaScript, TypeScript і SQL. Якщо ви не знайшли в цьому переліку потрібної вам мови, наразі існує 19 мовних плагінів IntelliJ, зокрема, для забезпечення підтримки R, Elm і D.

## 2.3 Вибір контейнера для проекту

Контейнеризація – це технологія, яка дозволяє запускати додатки і їхні залежності в ізольованих середовищах, званих контейнерами. Контейнери забезпечують стабільну роботу додатків на різних платформах, оскільки вони включають в себе всі необхідні бібліотеки, конфігураційні файли та інші залежності.

Найпоширеніші контейнери:

1. Docker – це платформа для створення, доставки та запуску контейнерів. Вона використовує модель клієнт-сервер, де Docker Client спілкується з Docker Daemon для управління контейнерами. Docker підтримує створення образів за допомогою Dockerfile, які містять всі необхідні залежності для додатків. Контейнери запускаються швидко і використовують менше ресурсів, ніж віртуальні машини. Docker Hub – це хмарний репозиторій для зберігання та поширення образів. Docker інтегрується з Kubernetes для оркестрації контейнерів, автоматизуючи розгортання та масштабування додатків. Він вирішує безліч завдань, пов'язаних зі створенням контейнерів, розміщенням в них додатків, управлінням процесами, а також тестуванням ПЗ і його окремих компонентів [47].
2. Podman – це інструмент для управління контейнерами, який не потребує демонів. Він підтримує більшість команд Docker CLI, що робить перехід від Docker до Podman відносно простим. Podman дозволяє запускати контейнери без привілеїв

адміністратора, підвищуючи безпеку, та підтримує rootless контейнери. Він інтегрується з Kubernetes через CRI-O [48].

3. CRI-O – це контейнерний runtime, створений для використання з Kubernetes. Він забезпечує мінімалістичний та легковагий runtime, сумісний зі специфікацією Container Runtime Interface (CRI), що дозволяє Kubernetes управляти контейнерами. CRI-O підтримує OCI-образи та інтегрується з Kubernetes для ефективного управління контейнерами [49].

4. Containerd – це інструмент для управління життєвим циклом контейнерів, розроблений як легкий і ефективний runtime. Він підтримує запуск, зупинку, перезавантаження контейнерів та управління їхніми образами. containerd інтегрується з Kubernetes та підтримує OCI-образи, надаючи API для управління контейнерами та образами [50].

5. LXC (Linux Containers) – це контейнерна технологія, яка надає контейнерну віртуалізацію на рівні операційної системи. LXD – це менеджер для LXC, який надає зручний інтерфейс для управління контейнерами та підтримує створення контейнерів, схожих на віртуальні машини. Ця технологія забезпечує високу продуктивність та ізоляцію додатків [51].

6. RKT – це контейнерний runtime, розроблений CoreOS (зараз частина Red Hat). Він фокусується на безпеці та сумісності з різними контейнерними форматами, включаючи AppC та OCI. Хоча проект більше не активно розвивається, rkt все ще використовується в деяких середовищах для підвищеної безпеки контейнеризації [52].

7. OpenVZ – це контейнерна технологія для віртуалізації на рівні операційної системи, яка дозволяє запускати декілька ізольованих примірників Linux на одному фізичному сервері. Вона є частиною комерційної пропозиції Virtuozzo і забезпечує високу продуктивність, підтримку ізольованих примірників ОС та розширені можливості управління ресурсами [53].

Основні можливості Docker [54]:

1. Можливість розміщення в ізольованому оточенні різнорідної начинки, що включає різні комбінації виконуваних файлів, бібліотек, файлів конфігурації, скриптів, файлів jar, gem, tar тощо.

2. Підтримка роботи на будь-якому комп'ютері на базі архітектури x86_64 з системою на базі ядра Linux, починаючи від ноутбуків, закінчуючи серверами та віртуальними машинами. Можливість роботи поверх немодифікованих сучасних ядер Linux (без накладення патчів) і в штатних оточеннях всіх великих дистрибутивів Linux, включаючи Fedora, RHEL, Ubuntu, Debian, SUSE, Gentoo і Arch.

3. Використання легковагих контейнерів для ізоляції процесів від інших процесів і основної системи.

4. Оскільки контейнери використовують свою власну самодостатню файлову систему, не важливо де, коли і в якому оточенні вони запускаються.

5. Ізоляція на рівні файлової системи: кожен процес виконується у повністю окремій кореневій ФС.

6. Ізоляція ресурсів, наприклад, споживання системних ресурсів, таких як витрата пам'яті і навантаження на CPU, можуть обмежуватися окремо для кожного контейнера з використанням cgroups.

7. Ізоляція на рівні мережі, де кожен ізольований процес має доступ тільки до пов'язаного з контейнером мережевого простору імен, включаючи віртуальний мережевий інтерфейс і прив'язані до нього IP-адреси.

8. Коренева файлова система для контейнерів створюється з використанням механізму copy-on-write (окремо зберігаються тільки змінені і нові дані), що дозволяє прискорити розгортання, знижує витрату пам'яті і економить дисковий простір.

9. Всі стандартні потоки (stdout/stderr) кожного виконуваного в контейнері процесу накопичуються і зберігаються у вигляді логу.

10. Змінена файлова система одного контейнера може використовуватися як основа для формування нових базових образів і створення інших контейнерів, без необхідності оформлення шаблонів або ручного налаштування складу образів.

11. Можливість використання інтерактивної командної оболонки, наприклад, до стандартного вводу будь-якого контейнера може бути прив'язаний псевдо-tty для запуску shell.

12. Підтримка використання різних систем зберігання, які можуть підключатися як плагіни. Серед підтримуваних драйверів зберігання заявлені aufs, device mapper (використовуються снапшоти LVM), vfs (на основі копіювання директорій) і Btrfs. Очікується поява драйверів для ZFS, Gluster і Ceph.

13. Можливість створення контейнерів, що містять складні програмні стеки, через зв'язування між собою вже існуючих контейнерів, що містять складові частини формованого стека. Зв'язування здійснюється не через злиття вмісту, а через забезпечення взаємодії між контейнерами (створюється мережевий тунель).

## 2.4 Висновки

У межах розділу 2 проведено вибір компонентів розробки програмно-апаратної системи зберігання даних з віддаленим доступом на основі Nextcloud. В ході порівняльного аналізу були враховані різні аспекти, такі як продуктивність, масштабованість, зручність використання та екосистема інструментів.

Щодо мов програмування, було розглянуто Java, Python, та Go. Java була обрана завдяки своїй великій спільноті розробників, широкому спектру бібліотек та фреймворків, а також стабільності та надійності. У якості середовища розробки було обрано IntelliJ IDEA, оскільки воно забезпечує потужні інструменти для розробки на Java, має дружній інтерфейс та велику кількість розширень для роботи з різними технологіями. Щодо контейнерізації основних складових програм, було вирішено використовувати Docker, оскільки він надає стандартизоване середовище

для розгортання та управління додатками, що спрощує процес розробки та розгортання програмного забезпечення.

Загалом, вибір цих компонентів дозволить забезпечити ефективний розвиток та експлуатацію програмно-апаратної системи зберігання даних з віддаленим доступом на основі Nextcloud.

# 3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗБЕРІГАННЯ ДАНИХ З ВІДДАЛЕНИМ ДОСТУПОМ НА ОСНОВІ NEXTCLOUD

## 3.1 Опис реалізації системи апаратного та програмного забезпечення і її модулів

Розроблена програма використовує мікросервісну архітектуру для забезпечення масштабованості, гнучкості та розширюваності. Кожен мікросервіс виконує окрему функцію і взаємодіє з іншими мікросервісами через REST API або за допомогою брокера повідомлень RabbitMQ. Програма побудована з використанням Spring Boot, що забезпечує швидку розробку та впровадження.

Мікросервісна архітектура дозволяє масштабувати окремі компоненти системи незалежно один від одного. Це означає, що можна збільшити ресурси тільки для тих мікросервісів, які відчувають високе навантаження, не впливаючи на інші частини системи. У монолітній архітектурі масштабування означає масштабування всього додатку, що може бути менш ефективним і більш витратним.

Розробка на основі мікросервісів дозволяє командам працювати над різними частинами системи паралельно без значного впливу на інші частини. Кожен мікросервіс може бути розроблений, оновлений і розгорнутий незалежно від інших, що прискорює розробку і впровадження нових функцій. У монолітній архітектурі будь-яка зміна потребує ретельного тестування всього додатку, що уповільнює процес розробки.

Мікросервіси дозволяють уникнути централізованих залежностей і знижують ризик повного збою системи. Якщо один мікросервіс виходить з ладу, інші можуть продовжувати працювати, забезпечуючи певний рівень функціональності. У монолітній архітектурі збій в одному компоненті може призвести до зупинки всього додатку.

Мікросервісна архітектура дозволяє використовувати різні технології і інструменти для кожного мікросервісу, вибираючи найкраще рішення для

конкретного завдання. Це забезпечує більшу гнучкість у виборі технологій. У монолітній архітектурі вся система зазвичай написана на одній технології, що може обмежувати можливості оптимізації.

Оновлення і розгортання мікросервісів відбувається незалежно, що знижує ризик впровадження помилок і скорочує час на розгортання нових версій. Кожен мікросервіс може бути оновлений без зупинки всього додатку. У монолітній архітектурі будь-яке оновлення потребує зупинки додатку та його повного перезапуску, що може призвести до простоїв.

Для великих проектів, де над додатком працює кілька команд, мікросервісна архітектура дозволяє розділити проект на менші частини, якими можуть займатися окремі команди. Це полегшує управління проектом і підвищує продуктивність. У монолітній архітектурі всі команди повинні працювати над одним великим кодовим базом, що може ускладнювати координацію і управління.

У мікросервісній архітектурі можна використовувати різні рівні безпеки для різних сервісів, обмежуючи доступ до чутливих даних і функцій. Це дозволяє підвищити загальний рівень безпеки системи. У монолітній архітектурі всі компоненти мають однаковий рівень доступу, що може збільшувати ризик безпеки.

Мікросервіси можуть бути оптимізовані окремо для досягнення кращої продуктивності і швидшого часу відгуку. Це дозволяє зосередитися на оптимізації критичних компонентів системи. У монолітній архітектурі оптимізація одного компоненту може вимагати змін в інших частинах системи, що може ускладнювати процес.

Мікросервісна архітектура добре поєднується з DevOps підходом, що дозволяє автоматизувати процеси розгортання, моніторингу і управління системою. Це сприяє швидшому випуску нових версій і більш стабільній роботі системи. Монолітна архітектура може бути менш гнучкою для впровадження DevOps практик через складність і розмір коду.

В кожному мікросервісі реалізовний структурний патерн MVC. MVC (Model-View-Controller) - це архітектурний шаблон, який розділяє додаток на три основні компоненти: «Модель» (Model), «Представлення» (View) і «Контролер» (Controller). Це розділення допомагає організувати код, підвищити його підтримуваність і масштабованість, а також забезпечити розділення обов'язків.

Model відповідає за управління даними додатку, його логікою і правилами бізнесу. Вона представляє стан додатку і може оновлюватися відповідно до дій користувача або інших подій.

Model отримує дані з бази даних, обробляє їх та передає далі. Вона виконує всі CRUD-операції (створення, читання, оновлення, видалення). Model реалізує бізнес-логіку додатку. Всі правила і процеси, які керують обробкою даних, зосереджені в цьому компоненті. Model зберігає стан додатку та може повідомляти про його зміни відповідним компонентам.

View відповідає за відображення даних, які надає Model. Воно займається рендерингом інтерфейсу користувача і взаємодією з користувачем, але не містить бізнес-логіки.

View відповідає за візуалізацію даних, отриманих від Model. Це може бути HTML-сторінка, GUI-додаток або інше відображення даних. View обробляє введення користувача (наприклад, натискання кнопок, введення тексту) і передає ці дані Controller. View оновлюється у відповідь на зміни в Model, забезпечуючи актуальність відображуваної інформації.

Controller виступає посередником між Model і View. Він приймає вхідні дані від користувача через View, обробляє їх (можливо, змінюючи стан Model), і визначає, яке View використовувати для відображення.

Controller отримує дані від користувача через View і обробляє їх (наприклад, перевіряє правильність введення). Controller взаємодіє з Model для зміни її стану, зберігання або отримання даних. Після обробки даних і оновлення Model, Controller визначає, яке View використовувати для відображення результатів.

Переваги використання MVC:

1. Розділення додатку на Model, View і Controller дозволяє ізолювати різні аспекти додатку. Це сприяє більш зрозумілій і організованій структурі коду, що полегшує підтримку і розширення додатку.

2. Кожен компонент MVC можна тестувати окремо, що полегшує процес виявлення і виправлення помилок. Наприклад, Model можна тестувати незалежно від інтерфейсу користувача, що спрощує перевірку бізнес-логіки.

3. Компоненти MVC можуть бути повторно використані в різних частинах додатку або навіть в інших проектах. Це знижує дублювання коду і сприяє більш ефективному використанню ресурсів.

4. Завдяки розділенню обов'язків різні команди розробників можуть працювати над різними аспектами додатку паралельно. Наприклад, команда, що займається інтерфейсом користувача, може працювати над View, тоді як інша команда працює над бізнес-логікою і Model.

5. MVC полегшує внесення змін до додатку, оскільки зміни в одному компоненті (наприклад, View) не вимагають змін в інших компонентах (Model або Controller).

В контексті мікросервісної архітектури, кожен мікросервіс реалізований з використанням патерну MVC, що забезпечує чітке розділення обов'язків і підвищує підтримуваність системи. Це дозволяє розробникам зосередитися на конкретних аспектах функціональності кожного мікросервісу, знижуючи складність коду і покращуючи його організацію.

Наприклад, в мікросервісі "dispatcher", MVC дозволяє чітко розділити обробку вхідних повідомлень від Telegram API (Controller), перенаправлення цих повідомлень (Model) і відображення результатів (View). У мікросервісі "node", MVC забезпечує чітке розділення бізнес-логіки (Model), обробки запитів (Controller) і взаємодії з базою даних (View).

Синхронна та асинхронна комунікації між мікросервісами, їх переваги та недоліки:

1. Синхронна комунікація (REST):

1.1. REST API легко зрозуміти та використовувати, що робить його популярним вибором для інтеграції між сервісами. Він широко підтримується різними мовами програмування та фреймворками.

1.2. При синхронній комунікації кожен запит отримує негайну відповідь, що дозволяє легко відстежувати і налагоджувати взаємодію між сервісами.

1.3. Розробники можуть легко тестувати і налагоджувати сервіси за допомогою стандартних HTTP-клієнтів, таких як Postman.

1.4. REST використовує стандартизовані HTTP-методи (GET, POST, PUT, DELETE), що сприяє кращій інтеграції та взаємодії з іншими веб-сервісами.

1.5. Якщо один з сервісів недоступний або працює повільно, це може призвести до затримок або відмов у обслуговуванні запитів.

1.6. Синхронна природа REST означає, що кожен запит очікує на відповідь перед продовженням роботи, що може стати вузьким місцем при великій кількості одночасних запитів.

1.7. Синхронна комунікація створює сильнішу залежність між сервісами, що може ускладнити їхнє незалежне розгортання та масштабування.

1.8. REST не забезпечує автоматичну обробку збоїв та повторні спроби, що може вимагати додаткової реалізації механізмів для підвищення стійкості.

2. Асинхронна комунікація (RabbitMQ):

2.1. Мікросервіси можуть надсилати повідомлення до RabbitMQ незалежно від стану отримувача. Це знижує залежність від доступності інших сервісів.

2.2. Асинхронна комунікація дозволяє обробляти запити паралельно, що підвищує продуктивність системи та дозволяє краще масштабувати окремі компоненти.

2.3. RabbitMQ дозволяє рівномірно розподіляти навантаження між кількома інстанціями сервісів, що підвищує загальну ефективність.

2.4. RabbitMQ підтримує механізми повторної доставки повідомлень у разі збоїв, що підвищує надійність системи.

2.5. Відстеження та налагодження асинхронних потоків даних може бути складнішим порівняно з синхронними запитами через затримки в обробці повідомлень.

2.6. Асинхронна природа RabbitMQ може призводити до затримок між надсиланням повідомлення та його обробкою, що може бути критичним для деяких додатків.

2.7. Використання RabbitMQ вимагає налаштування і підтримки додаткової інфраструктури, що може збільшити складність і витрати на управління системою.

2.8. Взаємодія з чергами повідомлень вимагає ретельного проектування логіки обробки повідомлень для уникнення дублювання і забезпечення коректної обробки даних.

Синхронна комунікація (REST) добре підходить для випадків, де важлива негайна відповідь і прозорість взаємодії. Вона простіша в реалізації та налагодженні, але менш стійка до відмов і може стати вузьким місцем при великому навантаженні.

Асинхронна комунікація (RabbitMQ) забезпечує вищу продуктивність, незалежність від доступності сервісів і кращу стійкість до збоїв. Вона дозволяє більш ефективно розподіляти навантаження, але вимагає додаткових витрат на інфраструктуру і може бути складнішою в налагодженні.

## 3.2 Опис основних компонентів програми (мікросервісів) та їх роботи в програмі

Dispatcher – це мікросервіс, який приймає повідомлення від Telegram API та перенаправляє їх на інший мікросервіс під назвою Node. Для комунікації він використовує REST для синхронної взаємодії та RabbitMQ для асинхронної передачі даних. Dispatcher функціонує як контролер, отримуючи дані з Telegram API та викликаючи відповідні сервіси для подальшої обробки повідомлень, зображено на рисунку 3.1.

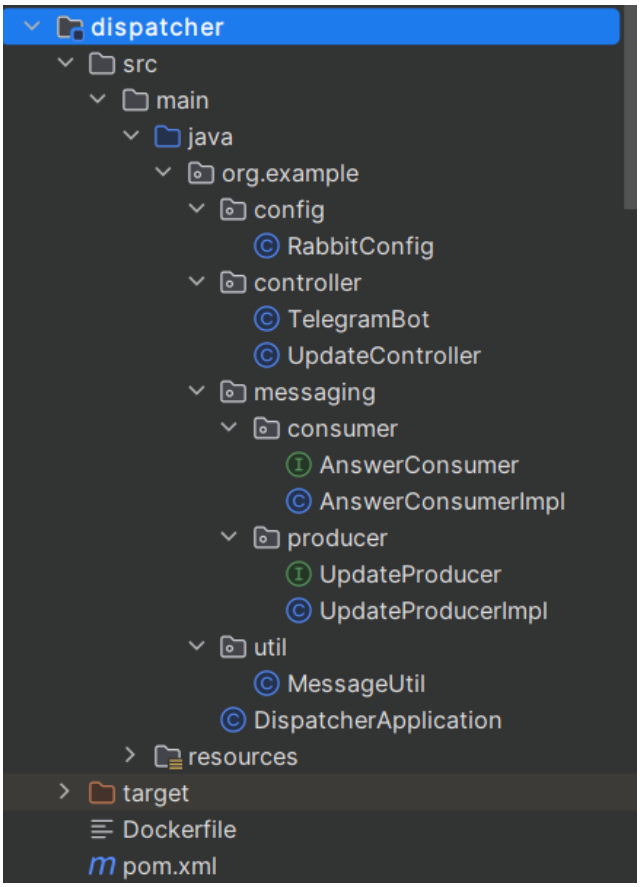



Рисунок 3.1 – Структура мікросервіса Dispatcher

Node - це мікросервіс у програмній архітектурі, який відповідає за виконання основної бізнес-логіки програми та має безпосередній доступ до бази даних PostgreSQL, де зберігаються всі необхідні дані. Його роль полягає в забезпеченні інтеграції з іншими мікросервісами, використовуючи REST для синхронного обміну даними та RabbitMQ для асинхронного. Node взаємодіє з мікросервісом Dispatcher, отримуючи від нього повідомлення від користувачів Telegram та перенаправляючи їх у відповідні сервіси. У своїй реалізації Node включає компоненти, які відповідають за обробку бізнес-логіки, а також репозиторії для ефективної роботи з базою даних, зображено на рисунках 3.2 та 3.6.

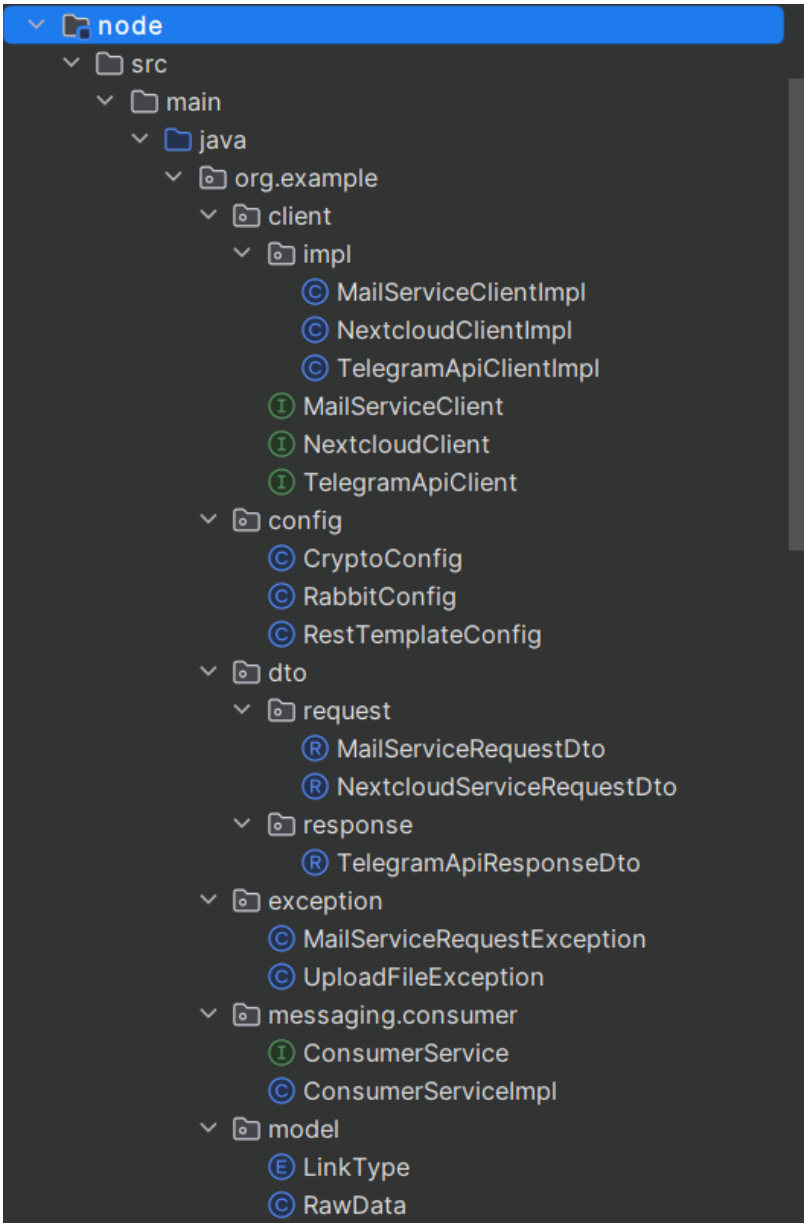



Рисунок 3.2. – С труктура мікросервіса Node

Mail-service – це мікросервіс, який відповідає за авторизацію користувачів у програмі. Він використовує REST для взаємодії з іншими мікросервісами. У його реалізації містяться контролери для обробки запитів на авторизацію, сервіси для перевірки облікових даних та інтеграції з поштовими сервісами, зображено на рисунку 3.3.

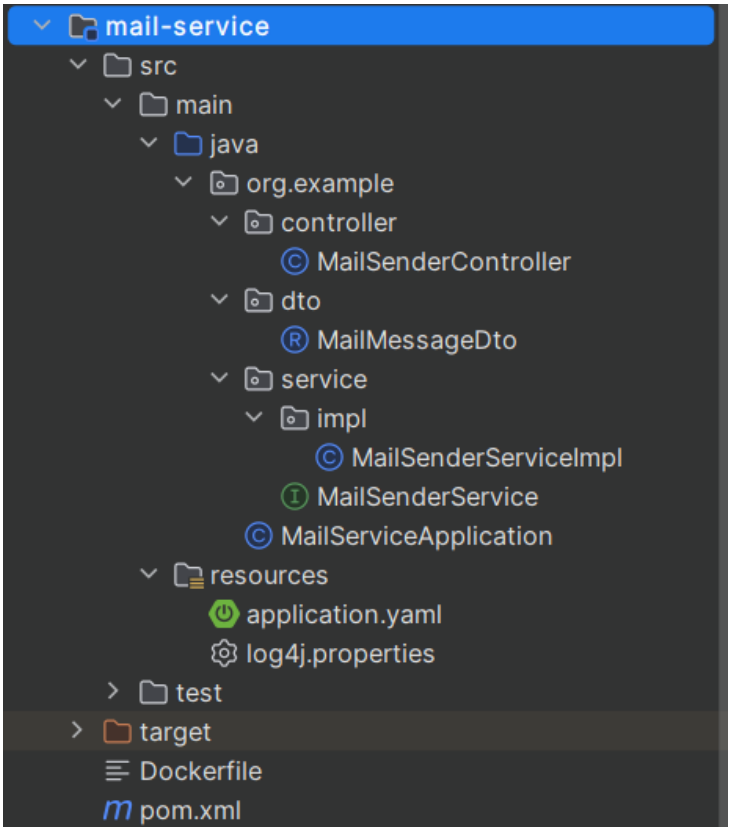



Рисунок 3.3 - Структура мікросервіса Mail-service

Rest-service – це мікросервіс, який приймає HTTP-запити від користувачів і обробляє їх. Для синхронної взаємодії з іншими мікросервісами він використовує REST (Representational State Transfer) – архітектурний стиль, який використовується для створення розподілених систем, основаних на веб-сервісах. У цьому стилі комунікація між клієнтом і сервером відбувається за допомогою стандартних HTTP-запитів, таких як GET, POST, PUT, DELETE. REST використовує ресурси (наприклад, URL адреси) для ідентифікації та взаємодії з ресурсами на сервері. Основні принципи REST включають уніфікований інтерфейс, безстановість, кешування. У його реалізації є контролери для обробки різних запитів користувачів, а саме взаємодія з Mail мікросервісом для авторизації користувачів шляхом підтвердження їхніх електронних пошт, сервіси для виконання бізнес-логіки та репозиторії для доступу до даних, зображено на рисунку 3.4.

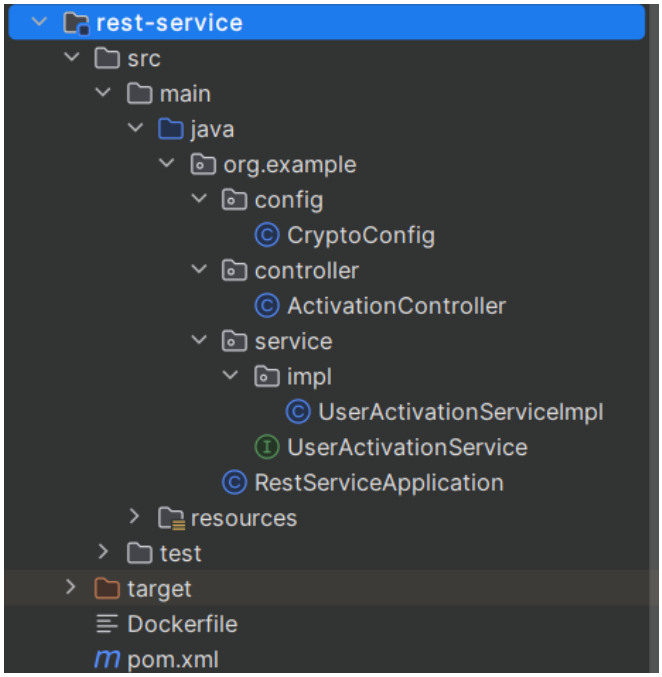



Рисунок 3.4 - Структура мікросервіса Rest-service

Nextcloud-service – це мікросервіс, який взаємодіє з сервером Nextcloud для роботи з файлами та даними. Він використовує REST для комунікації з іншими мікросервісами та Nextcloud API. У його реалізації є сервіси для обробки запитів до Nextcloud та обробки відповідей від нього, зображено на рисунку 3.5.

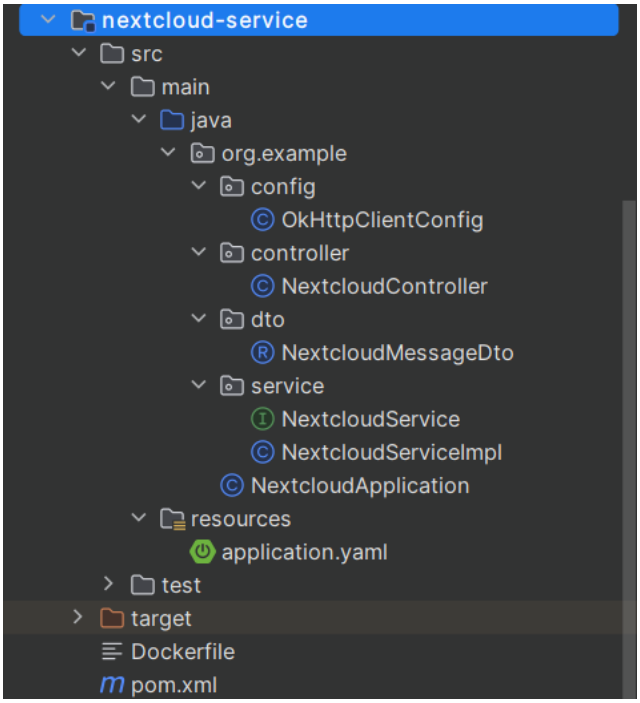



Рисунок 3.5 - структура мікросервіса Nextcloud-service

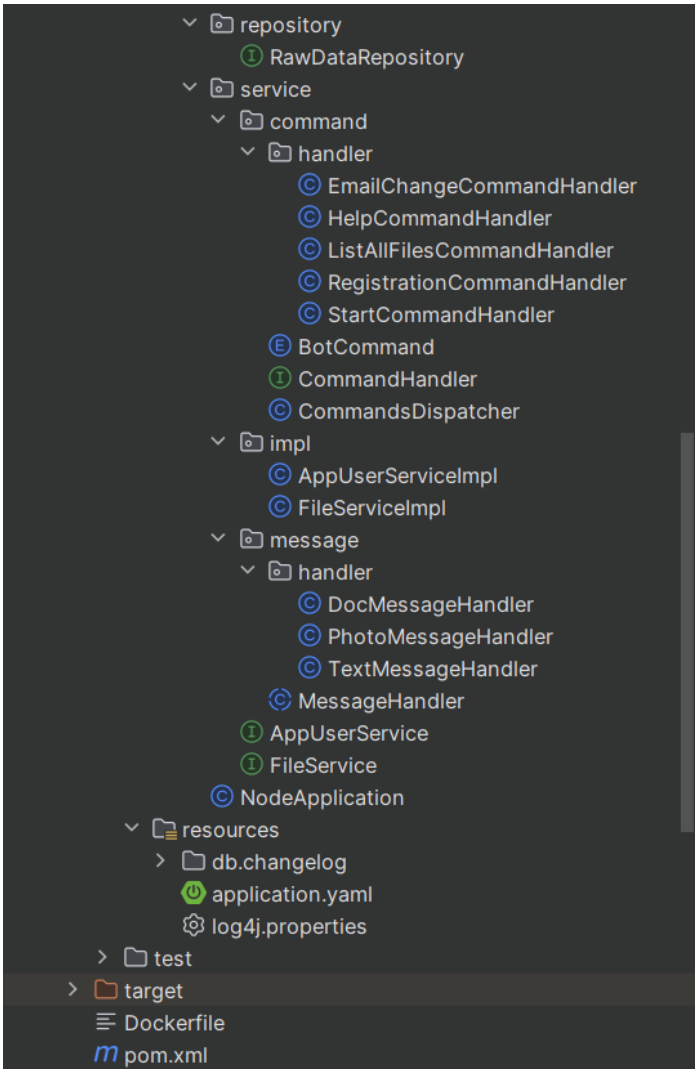



Рисунок 3.6 - Структура мікросервіса Node

## 3.3 Опис процесу створення баз даних

У рамках цього проекту було створено мікросервісну архітектуру для управління базою даних, використовуючи Spring Data JPA, PostgreSQL, Docker та Liquibase. Основною метою було забезпечити консистентний стан бази даних, а також полегшити її налаштування та розгортання.

Першим кроком було створення контейнера з базою даних PostgreSQL у Docker. Це дозволяє легко створювати та керувати базою даних у ізольованому середовищі. Нижче наведено docker-compose.yml для налаштування контейнера:

```
services:
  db-app:
```

```
    image: postgres:14.5
    container_name: postgresql
    environment:
      - POSTGRES_DB=telegram-db
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=pass
    ports:
      - 5400:5432
    volumes:
      - db-data:/var/lib/postgresql/data
    restart: always
volumes:
  db-data:v
```

Для запуску контейнера необхідно виконати команду:

docker-compose up -d

Це створить і запустить контейнер з базою даних PostgreSQL, який буде доступний на порту 5432.

В мікросервісі Rest-service та Node-service було налаштовано підключення до бази даних через файл конфігурації application.yml. Це дозволяє мікросервісам взаємодіяти з базою даних під час запуску:

application.yml (Node-service):

```
server:
  port: 8085
spring:
  rabbitmq:
    host: localhost
    port: 5672
    username: user
```

```
    password: pass
  datasource:
    url: jdbc:postgresql://localhost:5400/telegram-db
    username: user
    password: pass
  jpa:
    hibernate:
      ddl-auto: validate
    open-in-view: false
    show-sql: true
```

Проект використовує кілька сутностей (entity), які були визначені в окремих файлах змін Liquibase. Нижче наведено детальний опис кожної сутності:

1. raw_data

Сутність "raw_data" використовується для зберігання необроблених даних, які можуть бути отримані з різних джерел або систем. Це дані, що потребують подальшої обробки або аналізу.

Скрипт SQL для створення таблиці raw_data:

```
CREATE TABLE IF NOT EXISTS raw_data (
  id BIGINT GENERATED BY DEFAULT AS IDENTITY,
  update JSONB NOT NULL,
  CONSTRAINT raw_data_pk PRIMARY KEY (id)
);
```

2. binary_content

Сутність "binary_content" використовується для зберігання бінарного контенту, такого як файли, зображення, документи тощо в зручному для зберігання представленні.

Скрипт SQL для створення таблиці binary_content:

```
CREATE TABLE IF NOT EXISTS binary_content (
  id BIGINT GENERATED BY DEFAULT AS IDENTITY,
  bytes bytea,
  CONSTRAINT binary_content_pk PRIMARY KEY (id)
);
```

3. app_doc

Сутність "app_doc" призначена для зберігання документів, які можуть бути створені або завантажені користувачами. Це може включати текстові документи, звіти тощо.

Скрипт SQL для створення таблиці binary_content:

```
CREATE TABLE IF NOT EXISTS app_doc (
  id BIGINT GENERATED BY DEFAULT AS IDENTITY,
  telegram_file_id VARCHAR(255) NOT NULL,
  doc_name VARCHAR(255),
  mime_type VARCHAR(255),
  file_size INTEGER,
  binary_content_id BIGINT NOT NULL,
  CONSTRAINT app_doc_pk PRIMARY KEY (id),
  CONSTRAINT binary_content_fk FOREIGN KEY (binary_content_id) REFERENCES binary_content(id)
);
```

4. app_photo

Сутність "app_photo" використовується для зберігання фотографій, які можуть бути завантажені користувачами або створені системою.

Скрипт SQL для створення таблиці app_photo:

```
CREATE TABLE IF NOT EXISTS app_photo (
  id BIGINT GENERATED BY DEFAULT AS IDENTITY,
  telegram_file_id VARCHAR(255) NOT NULL,
  file_size INTEGER,
  binary_content_id BIGINT NOT NULL,
  CONSTRAINT app_photo_pk PRIMARY KEY (id),
  CONSTRAINT binary_content_fk FOREIGN KEY (binary_content_id) REFERENCES binary_content(id)
);
```

5. app_user

Сутність "app_user" використовується для зберігання інформації про користувачів додатку, включає в себе базові дані про користувача, такі як ім'я, електронна пошта, чи активований він в системі тощо.

Скрипт SQL для створення таблиці app_user:

```
CREATE TABLE IF NOT EXISTS app_user (
  id BIGINT GENERATED BY DEFAULT AS IDENTITY,
  telegram_user_id BIGINT NOT NULL,
  telegram_chat_id BIGINT,
  username VARCHAR(255),
  first_name VARCHAR(255),
  last_name VARCHAR(255),
  email VARCHAR(255),
  first_login_date TIMESTAMP,
  state VARCHAR(255),
  is_active BOOLEAN DEFAULT 'false',
```

```
  CONSTRAINT app_user_pk PRIMARY KEY (id)
);
```

Кожна з цих сутностей грає важливу роль у функціональності додатку, забезпечуючи зберігання та обробку різних типів даних. Завдяки використанню Liquibase, структура бази даних є керованою та легко змінюваною, що забезпечує гнучкість та консистентність даних у мікросервісній архітектурі.

Цей проект використовує модуль фреймворка Spring - Spring Data JPA для полегшення роботи з сутностями бази даних за допомогою зручних інтерфейсів, що спрощують велику частину роботи з базою даних та реалізовують весь основний функціонал. JPA(Java Persistence API) – це стандарт Java EE для роботи з об'єктно-реляційним відображенням даних в базі даних.

Spring Data JPA спрощує розробку застосунків, що використовують базу даних, шляхом автоматичної генерації реалізації репозиторіїв на основі інтерфейсів. Це означає, що вам не потрібно писати власні запити SQL - просто створюйте методи в інтерфейсі репозиторія, і Spring Data JPA забезпечить генерацію необхідних SQL-запитів.

Однією з ключових переваг Spring Data JPA є підтримка високого рівня абстракції, яка дозволяє розробникам працювати з базою даних, не думаючи про конкретні деталі її реалізації. Крім того, він надає можливості автоматичного розпізнавання запитів на основі назв методів, управління транзакціями та обробку винятків.

Spring Data JPA також підтримує використання анотацій для визначення взаємозв'язків між об'єктами та таблицями бази даних, а також для налаштування різних параметрів роботи з базою даних, таких як кешування, транзакції та інші. Це дозволяє підтримувати стандартизований та легкозмінний код, що спрощує розробку та підтримку програм з використанням бази даних в середовищі Spring.

Liquibase використовується для контролю структури бази даних та забезпечення її консистентного стану. Було створено головний файл змін

(db.changelog-master.xml), який містить посилання на окремі файли змін, підключено контроль бази даних було безпосередньо до Node-service, зображено на рисунку 3.7.

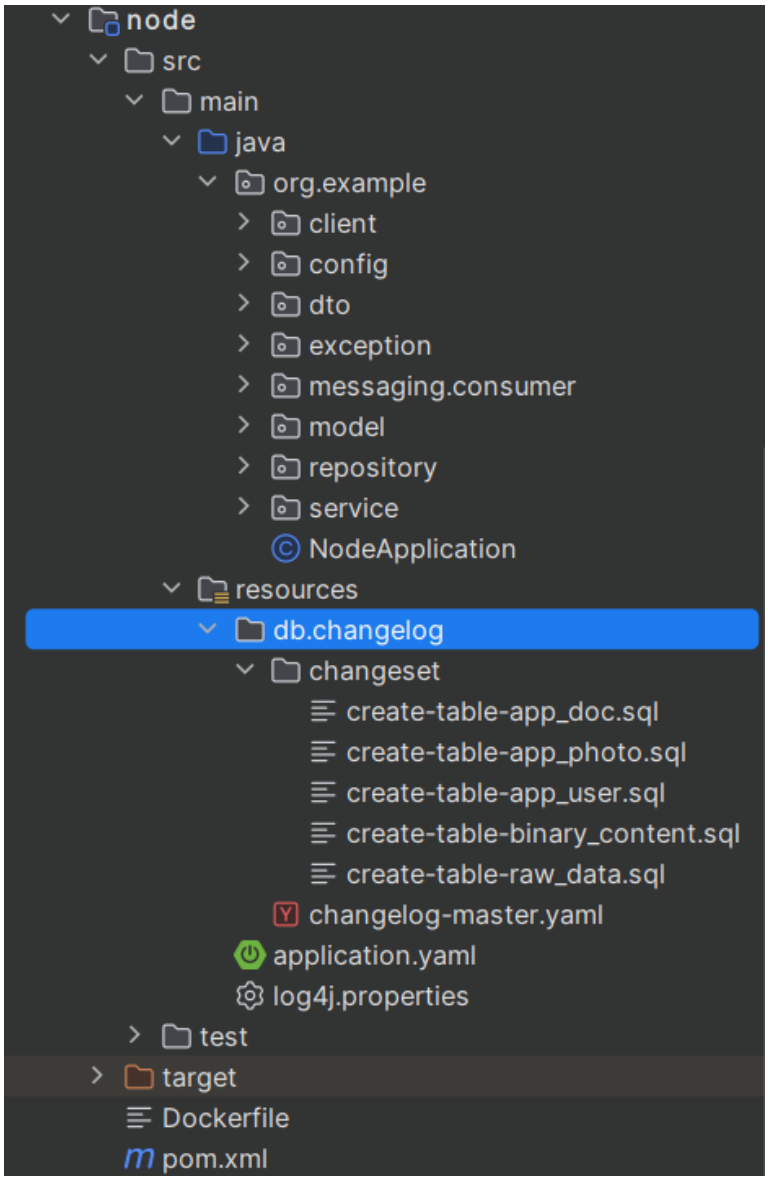



Рисунок 3.7 – Структура налаштування Liquibase у мікросервісі Node

Основний файл конфігурації (changelog-master.yaml):

```
databaseChangeLog:
  - include:
      file: db/changelog/changeset/create-table-raw_data.sql
```

```
- include:
    file: db/changelog/changeset/create-table-app_user.sql
- include:
    file: db/changelog/changeset/create-table-binary_content.sql
- include:
    file: db/changelog/changeset/create-table-app_doc.sql
- include:
    file: db/changelog/changeset/create-table-app_photo.sql
```

В даному файлі було описано всі файли-скрипти, що ініціалізують базу даних і в подальшому контролюються Liquibase.

Цей проект демонструє, як можна ефективно використовувати Docker, PostgreSQL, Liquibase та Spring Data JPA для створення та управління базою даних у мікросервісній архітектурі. Використання контейнерів дозволяє легко розгортати та масштабувати базу даних, тоді як Liquibase забезпечує контроль за її структурою та консистентністю. Spring Data JPA спрощує роботу з сутностями та їх збереженням у базі даних.

## 3.4. Висновки

У межах розділу 3 було використано мікросервісну архітектуру, що сприяло зменшенню складності системи шляхом розділення функціональності на невеликі та самостійні компоненти. Це полегшує розробку та підтримку системи, оскільки кожен мікросервіс може бути розроблений, протестований та масштабований окремо від інших. Крім того, такий підхід дозволяє гнучко змінювати архітектуру системи та впроваджувати нові функції без необхідності переписування всього додатку.

У проекті було використано багато сучасних інструментів для роботи з базами даних та їх сутностями. Це включало в себе такі інструменти, як Spring Data

JPA, Liquibase та PostgreSQL для зручної роботи з об'єктно-реляційним відображенням даних, системи моніторингу та адміністрування баз даних для контролю та оптимізації їхньої продуктивності, а також інструменти для розробки та відлагодження SQL-запитів для ефективної роботи з даними.

Nextcloud є ключовим елементом програмно-апаратної реалізації, надаючи масштабовану та надійну платформу для зберігання та управління даними. Він забезпечує віддалений доступ до файлів та інформації, синхронізацію даних між різними пристроями та резервне копіювання даних, що забезпечує безпеку та надійність у роботі з інформацією.

В цілому, комбінація мікросервісної архітектури та використання Nextcloud як основної платформи для зберігання даних створює потужний та гнучкий інструмент для роботи з інформацією в розподілених середовищах. Це дозволяє компаніям та організаціям ефективно керувати своїми даними та забезпечує їх доступність та безпеку.

# ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень було розроблено комплексну систему зберігання даних з віддаленим доступом на основі Nextcloud, що включає мікросервісну архітектуру, інтеграцію з базами даних, а також використання сучасних інструментів для забезпечення надійності, безпеки та ефективності роботи з даними.

У першому розділі проведено аналіз предметної області, спрямований на виявлення наявних проблем та завдань. Визначені основні проблеми, що виникають у сфері зберігання даних з віддаленим доступом, був здійснений порівняльний аналіз переваг та недоліків існуючих рішень. Оцінено різні підходи до зберігання даних з віддаленим доступом, включаючи хмарні сервіси, локальні сервери та віртуальні приватні мережі. З'ясовано, що існуючі рішення часто мають обмеження або не відповідають усім потребам користувачів.

У другому розділі проведено вибір компонентів розробки програмно-апаратної системи зберігання даних з віддаленим доступом на основі Nextcloud. В ході порівняльного аналізу були враховані різні аспекти, такі як продуктивність, масштабованість, зручність використання та екосистема інструментів. Щодо мов програмування, було розглянуто такі альтернативи як Java, Python, та Go. Java була обрана завдяки своїй великій спільноті розробників, широкому спектру бібліотек та фреймворків, а також стабільності та надійності. У якості середовища розробки було обрано IntelliJ IDEA, оскільки воно забезпечує потужні інструменти для розробки на Java.

У третьому розділі було використано мікросервісну архітектуру, що сприяло зменшенню складності системи шляхом розділення функціональності на невеликі та самостійні компоненти. Це полегшує розробку та підтримку системи, оскільки кожен мікросервіс може бути розроблений, протестований та масштабований окремо від інших. Крім того, такий підхід дозволяє гнучко змінювати архітектуру системи та впроваджувати нові функції без необхідності переписування всього

додатку. У проекті було використано багато сучасних інструментів для роботи з базами даних та їх сутностями. Це включало в себе такі інструменти, як Spring Data JPA, Liquibase та PostgreSQL для зручної роботи з об'єктно-реляційним відображенням даних, системи моніторингу та адміністрування баз даних для контролю та оптимізації їхньої продуктивності, а також інструменти для розробки та відлагодження SQL-запитів для ефективної роботи з даними. Nextcloud є ключовим елементом програмно-апаратної реалізації, надаючи масштабовану та надійну платформу для зберігання та управління даними. Він забезпечує віддалений доступ до файлів та інформації, синхронізацію даних між різними пристроями та резервне копіювання даних, що забезпечує безпеку та надійність у роботі з інформацією.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ


1. Хмарні сховища. URL: https://uk.wikipedia.org/wiki/%D0%A5%D0%BC%D0%B0%D1%80%D0%BD%D1%96_%D1%81%D1%85%D0%BE%D0%B2%D0%B8%D1%89%D0%B0 (дата звернення: 17.02.2024).
2. Звідки взялися хмарні технології? Перспективи та історія розвитку. URL: https://www.imena.ua/blog/cloud-computing-history/ (дата звернення: 17.02.2024).
3. Історія інтернету від ARPANET до сьогодні. URL: https://ucloud.ua/istoriya-internetu-vid-arpanet-do-sogodni/ (дата звернення: 17.02.2024).
4. Логістична мапа «ARPANET». URL: https://m.joyreactor.cc/post/4550170 (дата звернення: 17.02.2024).
5. Історія IBM (International Business Machines). URL: https://busines.in.ua/istoriya-ibm-international-business-machines/ (дата звернення: 17.02.2024).
6. Система віртуалізації серверів IBM. URL: https://www.jetinfo.ru/sistema-virtualizatsii-serverov-ibm-power/ (дата звернення: 17.02.2024).
7. Zimki, the world's first Platform as a Service that belonged to Canon. URL: https://news.ycombinator.com/item?id=28236217 (дата звернення: 17.02.2024).
8. Що таке Google App Engine? URL: https://blog.back4app.com/ru/%D1%87%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-google-app-engine/ (дата звернення: 17.02.2024).
9. AWS Architecture. URL: https://mindmajix.com/aws-architecture (дата звернення: 17.02.2024).
10. Amazon Web Services – базова архітектура. URL: https://coderlessons.com/tutorials/veb-razrabotka/izuchite-amazon-web-services/amazon-web-services-bazovaia-arkhitektura (дата звернення: 17.02.2024).

11. Google Cloud Spanner: огляд можливостей та перші враження від бази даних. URL: https://dou.ua/lenta/articles/google-cloud-spanner-review/ (дата звернення: 17.02.2024).

12. Хмарні обчислення: історія, можливості, перспективи. URL: https://aw.club/global/uk/blog/cloud-computing-history-opportunities-benefits (дата звернення: 17.02.2024).

13. NetForce Ukraine:у лідери галузі за два роки. URL: https://nbr.com.ua/news/netforce-ukraine-u-lideri-galuzi-za-dva-roki (дата звернення: 17.02.2024).

14. Основи хмарних технологій. URL: https://itedu.center/ua/blog/ru/review/6-facts-you-should-know-about-cloud-technologies/ (дата звернення: 17.02.2024).

15. Види хмарних послуг. URL: https://itedu.center/ua/blog/wp-content/uploads/2023/05/image-18.png (дата звернення: 17.02.2024).

16. Що таке AWS? URL: https://aws.amazon.com/ru/what-is-aws/ (дата звернення: 18.02.2024).

17. Платформа Microsoft Azure. URL: https://techexpert.ua/it-products/platforma-microsoft-azure/ (дата звернення: 18.02.2024).

18. Що таке GCP та як ви можете використовувати його для свого бізнесу. URL: https://cloudfresh.com/ua/cloud-blog/shho-take-gcp-ta-yak-vy-mozhete-vykorystovuvaty-jogo-dlya-svogo-biznesu/ (дата звернення: 18.02.2024).

19. Nextcloud ознайомлення з посібником користувача. URL: https://docs.nextcloud.com/server/latest/user_manual/uk/index.html (дата звернення: 18.02.2024).

20. Сервер з власною хмарою. URL: https://habr.com/ru/companies/hostkey/articles/738136/ (дата звернення: 18.02.2024).

21. Nextcloud – ваше хмарне сховище. URL: https://blog.host4.biz/uk/next-cloud-hmarne-shovysche (дата звернення: 18.02.2024).

22. Nextcloud. URL: https://uk.wikipedia.org/wiki/Nextcloud (дата звернення: 18.02.2024).

23. Що таке мова програмування Python? URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-jazik-programmirovanija-python/ (дата звернення: 15.03.2024).

24. Що таке JavaScript. URL: https://cases.media/en/article/sho-take-javascript (дата звернення: 15.03.2024).

25. Java. URL: https://newbieprogramming.fandom.com/uk/wiki/Java (дата звернення: 15.03.2024).

26. Що таке C#? URL: https://beetroot.academy/blog/shcho-take-c-chi-pidhodit-meni-cya-mova-programuvannya-chomu-vona-kruta (дата звернення: 15.03.2024).

27. Що таке C++? Базовий Concepts мови програмування C++. URL: https://www.guru99.com/uk/cpp-tutorial.html (дата звернення: 15.03.2024).

28. Мова програмування Golang – що це? URL: https://lemon.school/blog/mova-programuvannya-golang-shho-tse (дата звернення: 15.03.2024).

29. Rust — все про мову програмування, її особливості та синтаксис. URL: https://nofluffjobs.com/uk/log/robota-v-it/rust-a-programming-language/ (дата звернення: 15.03.2024).

30. Основи, мова програмування Swift. URL: https://book.swift.org.ua/book/1_language_guide/0_the_basics/ (дата звернення: 15.03.2024).

31. Популярні фреймворки Java: Spring та Hibernate. URL: https://mate.academy/blog/java-development/java-spring-hibernate-frameworks/ (дата звернення: 15.03.2024).

32. Особливості мови Java. URL: https://it-rating.ua/osoblivosti-movi-java (дата звернення: 15.03.2024).

33. Огляд Java 17: переваги та недоліки. URL: https://www.nixsolutions.com/ua/blog/it-novosti/oglyad-java-17-perevagi-ta-nedoliki/ (дата звернення: 15.03.2024).

34. Освоюємо Java/Встановлення і налаштування середовища розробки. URL: https://uk.wikibooks.org/wiki/%D0%9E%D1%81%D0%B2%D0%BE%D1%8E%D1%94%D0%BC%D0%BE_Java/%D0%92%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F_%D1%96_%D0%BD%D0%B0%D0%BB%D0%B0%D1%88%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%BE%D0%B2%D0%B8%D1%89%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8 (дата звернення: 19.03.2024).

35. Features overview. URL: https://www.jetbrains.com/idea/features/ (дата звернення: 19.03.2024).

36. Як використовувати Eclipse IDE. URL: https://foxminded.ua/eclipse-ide/#:~:text=Eclipse%20IDE%20%2D%20%D1%86%D0%B5%20%D0%BF%D0%BE%D0%BF%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%B5%20%D1%96%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5,%D1%82%D0%B0%D0%BA%D0%BE%D0%B6%20%D0%BF%D1%96%D0%B4%D1%82%D1%80%D0%B8%D0%BC%D1%83%D1%94%20%D1%96%D0%BD%D1%88%D1%96%20%D0%BC%D0%BE%D0%B2%D0%B8%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F. (дата звернення: 19.03.2024).

37. Netbeans - ide для Java, Python, PHP, Javascript, C/C++, Ada. URL: https://ualinux.com/uk/ubuntu-apps-programming/netbeans (дата звернення: 19.03.2024).

38. Android Studio: переваги та особливості. URL: https://qagroup.com.ua/publications/android-studio-perevagy-ta-osoblyvosti/ (дата звернення: 19.03.2024).

39. BlueJ IDE для вивчення Java в інтерактивному та візуальному плані. URL: https://blog.desdelinux.net/uk/bluej-%D1%96%D0%B4%D0%B5%D1%8F-%D0%B2%D0%B8%D0%B2%D1%87%D0%B8%D1%82%D0%B8-Java-%D0%B2-

%D1%96%D0%BD%D1%82%D0%B5%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%BC%D1%83-%D1%82%D0%B0-%D0%B2%D1%96%D0%B7%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC%D1%83-%D0%BF%D0%BB%D0%B0%D0%BD%D1%96/ (дата звернення: 19.03.2024).

40. Introduction to Oracle JDeveloper. URL: https://docs.oracle.com/cd/E16162_01/user.1112/e17455/gs_jdev.htm (дата звернення: 19.03.2024).

41. What is MyEclipse? URL: https://www.solvusoft.com/en/file-extensions/software/genuitec/myeclipse/ (дата звернення: 19.03.2024).

42. What is DrJava? URL: https://www.cs.cornell.edu/courses/cs211/2005sp/Software/drjava.html (дата звернення: 19.03.2024).

43. IntelliJ IDEA. URL: https://uk.wikipedia.org/wiki/IntelliJ_IDEA (дата звернення: 19.03.2024).

44. Eclipse, NetBeans чи IntelliJ IDEA? Обираємо IDE для Java-розробки. URL: https://javarush.com/ua/groups/posts/4027-eclipse-netbeans-ili-intellij-idea-vihbiraem-ide-dlja-java-razrabotki (дата звернення: 19.03.2024).

45. IntelliJ IDEA показує кількість попереджень (warnings) і припущення, що ґрунтуються на статистичному аналізі Java-коду. URL: https://cdn.javarush.com/images/article/6cd2b31b-7d7e-4e9e-bac2-0aece34ebe33/512.webp (дата звернення: 19.03.2024).

46. Спливаюче віконце визначення символів в IntelliJ IDEA. URL: https://cdn.javarush.com/images/article/e15e0f42-3347-4643-9a87-a8b2513a1108/512.webp (дата звернення: 19.03.2024).

47. Що таке Docker і навіщо він? URL: https://qagroup.com.ua/publications/shcho-take-docker-i-navishcho-vin/ (дата звернення: 22.03.2024).

48. Podman. Documentation. URL: https://docs.rockylinux.org/uk/gemstones/containers/podman/ (дата звернення: 22.03.2024).

49. What is CRI-O? URL: https://cri-o.io/ (дата звернення: 22.03.2024).

50. What Is Containerd? URL: https://www.geeksforgeeks.org/what-is-containerd/ (дата звернення: 22.03.2024).

51. LinuxContainers. URL: https://linuxcontainers.org/ (дата звернення: 22.03.2024).

52. What is RKT? URL: https://www.redhat.com/en/topics/containers/what-is-rkt (дата звернення: 22.03.2024).

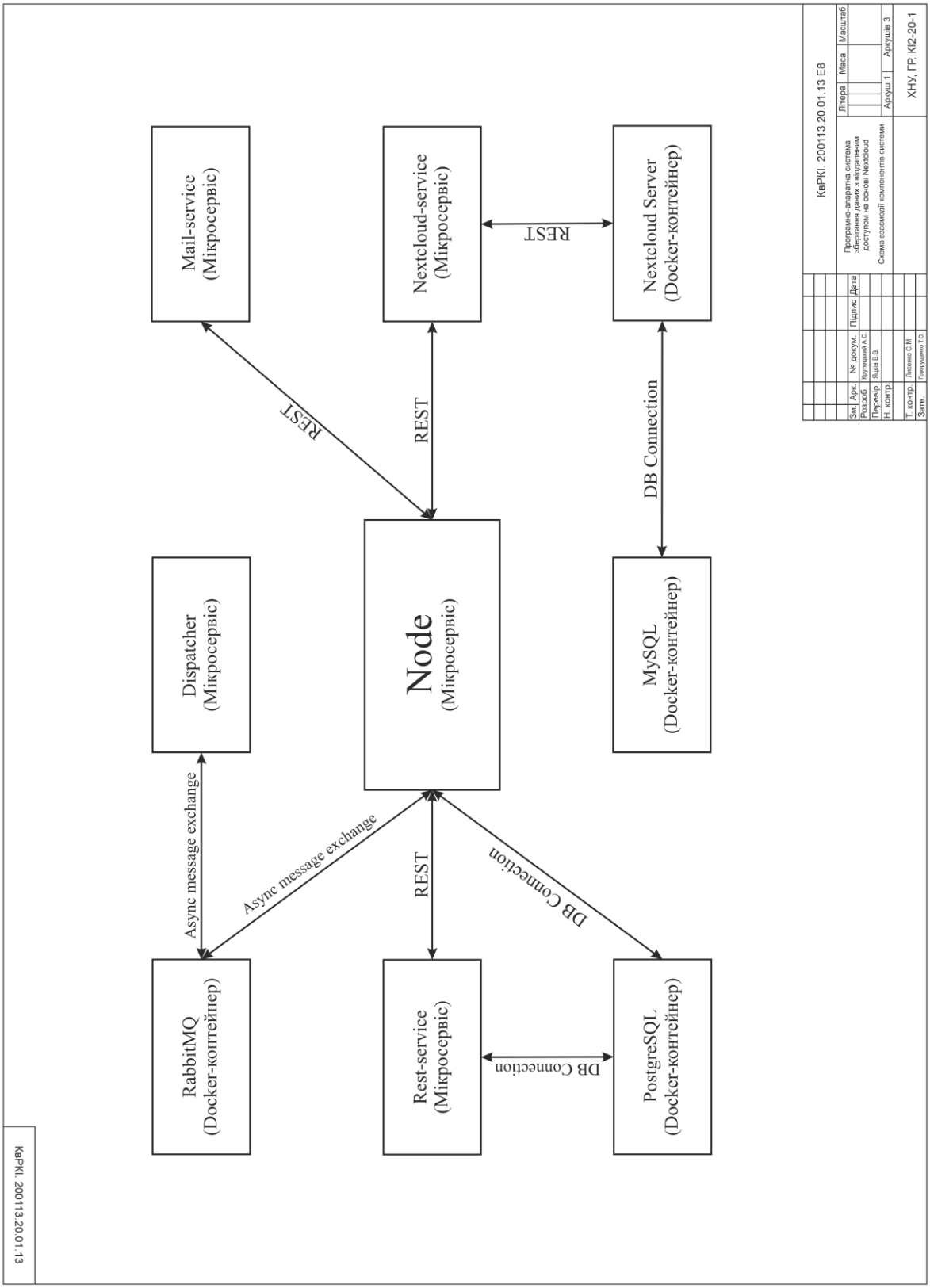
53. OpenVZ або Xen. URL: https://uh.ua/ua/kb/otvety/vps-vds/openvz-vs-xen.html (дата звернення: 22.03.2024).

54. Docker. URL: https://uk.wikipedia.org/wiki/Docker (дата звернення: 22.03.2024).

**Додаток А**

(обов’язковий)

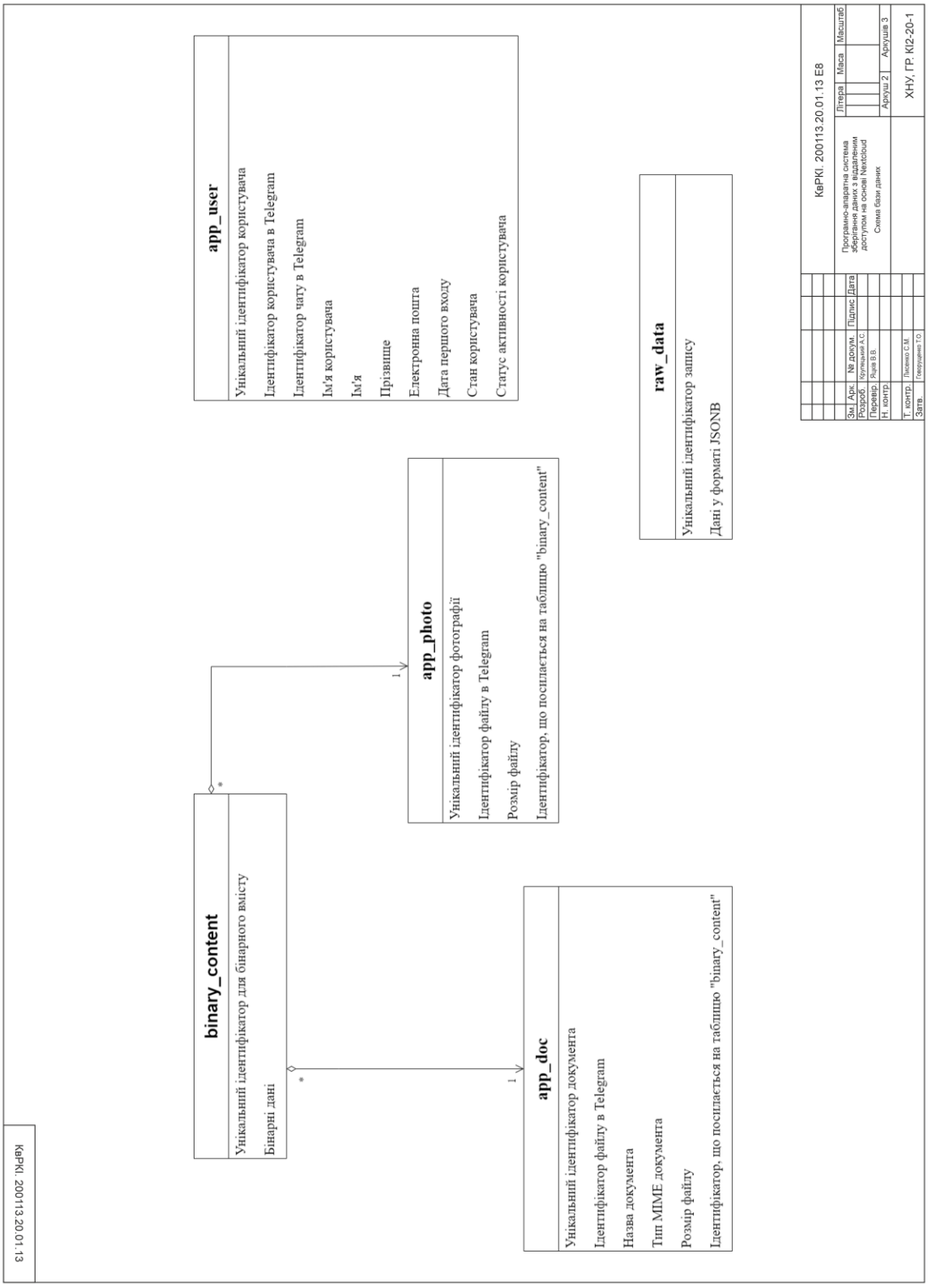
Копія креслення «Схема взаємодії компонентів системи»



**Додаток Б**

(обов’язковий)

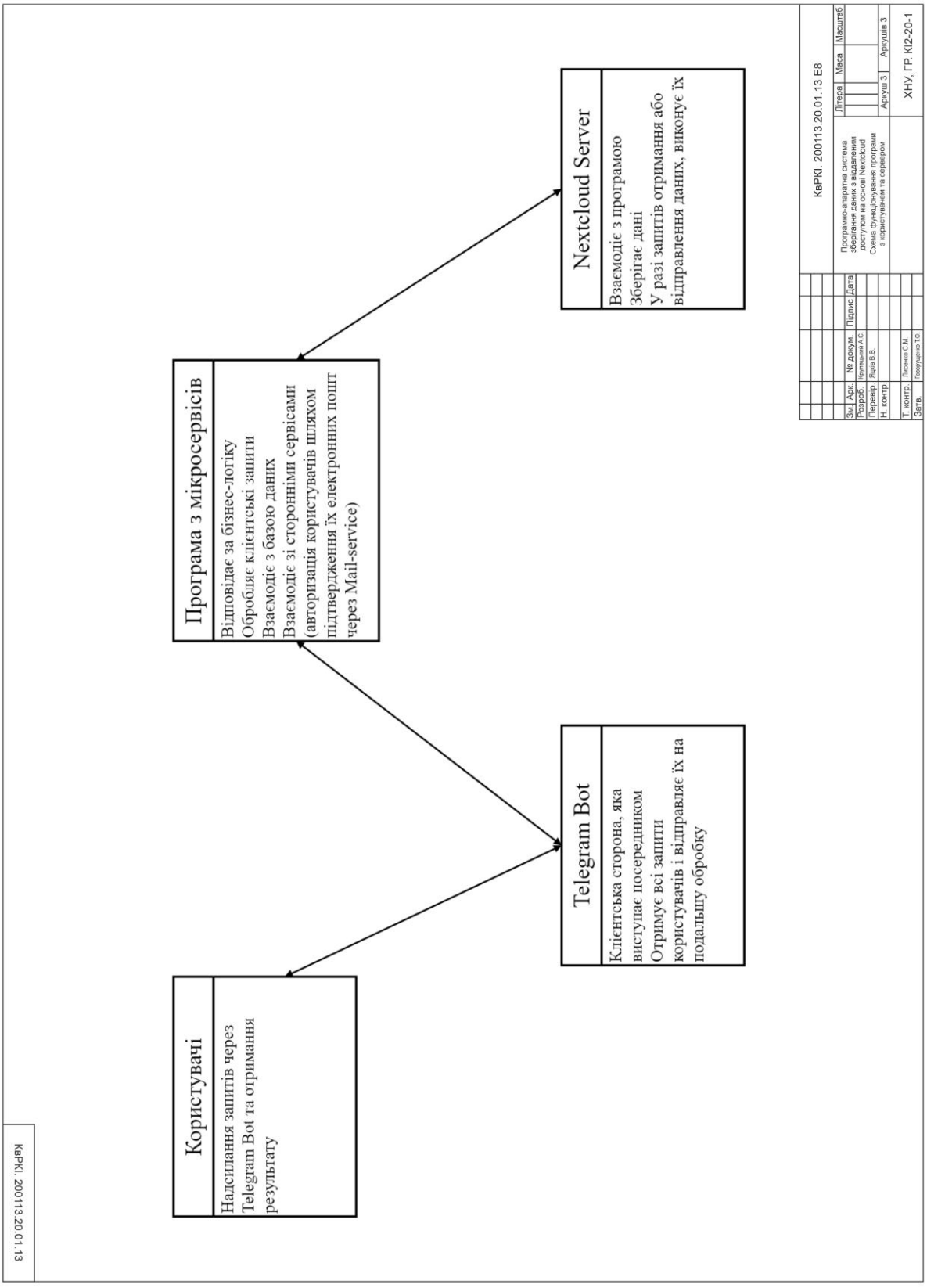
Копія креслення «Схема бази даних»



## Додаток В

(обов’язковий)

Копія креслення «Схема функціонування програми з користувачем та сервером»



**Додаток Г**

Код програми

Мікросервіс Dispatcher

```
package org.example.config;

import static org.example.model.RabbitQueue.ANSWER_MESSAGE;
import static org.example.model.RabbitQueue.DOC_MESSAGE_UPDATE;
import static org.example.model.RabbitQueue.PHOTO_MESSAGE_UPDATE;
import static org.example.model.RabbitQueue.TEXT_MESSAGE_UPDATE;

import org.springframework.amqp.core.Queue;
import
org.springframework.amqp.support.converter.Jackson2JsonMessageConverter;
import org.springframework.amqp.support.converter.MessageConverter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RabbitConfig {
   @Bean
   public MessageConverter jsonMessageConverter() {
      return new Jackson2JsonMessageConverter();
   }

   @Bean
   public Queue textMessageQueue() {
      return new Queue(TEXT_MESSAGE_UPDATE);
   }
```

```
    @Bean
    public Queue docMessageQueue() {
        return new Queue(DOC_MESSAGE_UPDATE);
    }

    @Bean
    public Queue photoMessageQueue() {
        return new Queue(PHOTO_MESSAGE_UPDATE);
    }

    @Bean
    public Queue answerMessageQueue() {
        return new Queue(ANSWER_MESSAGE);
    }
}
```

```
package org.example.controller;

import javax.annotation.PostConstruct;
import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.bots.TelegramLongPollingBot;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;

@Log4j
@RequiredArgsConstructor
```

```
@Component
public class TelegramBot extends TelegramLongPollingBot {
    @Value("${bot.name}")
    private String botName;
    @Value("${bot.token}")
    private String botToken;

    private final UpdateController updateController;

    @PostConstruct
    public void init() {
        updateController.registerBot(this);
    }

    @Override
    public String getBotUsername() {
        return botName;
    }

    @Override
    public String getBotToken() {
        return botToken;
    }

    @Override
    public void onUpdateReceived(Update update) {
        updateController.processUpdate(update);
    }
```

```
    public void sendAnswerMessage(SendMessage answer) {
        if (answer != null) {
            try {
                execute(answer);
            } catch (TelegramApiException e) {
                log.error(e.getMessage());
            }
        } else {
            log.error("Can't send message as answer is null");
        }
    }
}
```

```
package org.example.controller;

import static org.example.model.RabbitQueue.DOC_MESSAGE_UPDATE;
import static org.example.model.RabbitQueue.PHOTO_MESSAGE_UPDATE;
import static org.example.model.RabbitQueue.TEXT_MESSAGE_UPDATE;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j;
import org.example.messaging.producer.UpdateProducer;
import org.example.util.MessageUtil;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
import org.telegram.telegrambots.meta.api.objects.Update;

@Log4j
@RequiredArgsConstructor
```

```
@Component
public class UpdateController {
    private TelegramBot telegramBot;
    private final MessageUtil messageUtil;
    private final UpdateProducer updateProducer;

    public void registerBot(TelegramBot telegramBot) {
        this.telegramBot = telegramBot;
    }

    public void sendView(SendMessage sendMessage) {
        telegramBot.sendAnswerMessage(sendMessage);
    }

    public void processUpdate(Update update) {
        if (update == null) {
            log.error("Received update is null");
            return;
        }

        if (update.hasMessage()) {
            distributeMessageByType(update);
        } else {
            log.error("Unsupported update is received " + update);
        }
    }

    private void distributeMessageByType(Update update) {
        Message message = update.getMessage();
```

```
            if (message.hasText()) {
                processTextMessage(update);
            } else if (message.hasDocument()) {
                processDocumentMessage(update);
            } else if (message.hasPhoto()) {
                processPhotoMessage(update);
            } else {
                setUnsupportedMessageTypeView(update);
            }
        }

        private void setUnsupportedMessageTypeView(Update update) {
            sendView(messageUtil.generateAnswerMessage(update,
                    "Unsupported message type, required only text, document or photo"));
        }

        private void processTextMessage(Update update) {
            updateProducer.produce(TEXT_MESSAGE_UPDATE, update);
        }

        private void processDocumentMessage(Update update) {
            updateProducer.produce(DOC_MESSAGE_UPDATE, update);
        }

        private void processPhotoMessage(Update update) {
            updateProducer.produce(PHOTO_MESSAGE_UPDATE, update);
        }
}
```

```
package org.example.messaging.consumer;
```

```
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;

public interface AnswerConsumer {
    void consume(SendMessage sendMessage);
}
```

```
package org.example.messaging.consumer;

import static org.example.model.RabbitQueue.ANSWER_MESSAGE;

import lombok.RequiredArgsConstructor;
import org.example.controller.UpdateController;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;

@Service
@RequiredArgsConstructor
public class AnswerConsumerImpl implements AnswerConsumer {
    private final UpdateController updateController;

    @Override
    @RabbitListener(queues = ANSWER_MESSAGE)
    public void consume(SendMessage sendMessage) {
        updateController.sendView(sendMessage);
    }
}
```

```
package org.example.messaging.producer;
```

```
import org.telegram.telegrambots.meta.api.objects.Update;

public interface UpdateProducer {
    void produce(String rabbitQueue, Update update);
}
```

```
package org.example.messaging.producer;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.objects.Update;

@Log4j
@Service
@RequiredArgsConstructor
public class UpdateProducerImpl implements UpdateProducer {
    private final RabbitTemplate rabbitTemplate;

    @Override
    public void produce(String rabbitQueue, Update update) {
        log.debug(update.getMessage().getText());
        rabbitTemplate.convertAndSend(rabbitQueue, update);
    }
}
```

```
package org.example.util;

import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
```

```
import org.telegram.telegrambots.meta.api.objects.Update;

@Component
public class MessageUtil {
    public SendMessage generateAnswerMessage(Update update, String text) {
        SendMessage answer = new SendMessage();
        answer.setChatId(update.getMessage().getChatId());
        answer.setText(text);
        return answer;
    }
}
```

```
package org.example;

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

@SpringBootApplication
public class DispatcherApplication {
    public static void main(String[] args) {
        SpringApplication.run(DispatcherApplication.class);
    }
}
```

Мікросервіс Mail-service

```
package org.example.controller;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j;
import org.example.dto.MailMessageDto;
import org.example.service.MailSenderService;
```

```
import org.springframework.http.ResponseEntity;
import org.springframework.mail.MailException;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@Log4j
@RestController
@RequestMapping("/mail")
@RequiredArgsConstructor
public class MailSenderController {
    private final MailSenderService mailSenderService;

    @PostMapping("/send/activation")
    public ResponseEntity<String> sendMailActivationMessage(
            @RequestBody MailMessageDto mailMessageDto
    ) {
        try {
            mailSenderService.sendActivationMessage(mailMessageDto);
            return ResponseEntity
                    .ok()
                    .body("The activation link has been sent to your email");
        } catch (MailException ex) {
            log.error("MailSenderService failure");
            return ResponseEntity
                    .internalServerError()
                    .body("Can't send mail activation message. Please try later!");
        }
```

```
        }
}
```

```
package org.example.dto;

public record MailMessageDto(
        String userId,
        String mailTo
) {}
```

```
package org.example.service.impl;

import lombok.RequiredArgsConstructor;
import org.example.dto.MailMessageDto;
import org.example.service.MailSenderService;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class MailSenderServiceImpl implements MailSenderService {
    private static final String MAIL_ACTIVATION_SUBJECT =
            "Activation of account registration";

    @Value("${spring.mail.username}")
    private String sendFromEmail;
    @Value("${service.activation.url}")
    private String activationServiceUrl;
```

```
    private final JavaMailSender mailSender;

    @Override
    public void sendActivationMessage(MailMessageDto mailMessageDto) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom(sendFromEmail);
        message.setSubject(MAIL_ACTIVATION_SUBJECT);
        message.setText(generateMailActivationText(mailMessageDto.userId()));
        message.setTo(mailMessageDto.mailTo());
        mailSender.send(message);
    }

    private String generateMailActivationText(String id) {
        return String.format(
                "To finish account registration follow the link:\n%s",
                activationServiceUrl.replace("{id}", id));
    }
}
```

```
package org.example.service;

import org.example.dto.MailMessageDto;

public interface MailSenderService {
    void sendActivationMessage(MailMessageDto mailMessageDto);
}
```

```
package org.example;

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

```
@SpringBootApplication
public class MailServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(MailServiceApplication.class);
    }
```

UNICHECK
by Turnitin

Ім'я користувача:
**Кафедра КІ**

ID перевірки:
**1016370684**

Дата перевірки:
**18.06.2024 08:21:49 EEST**

Тип перевірки:
**Doc vs Internet + Library**

Дата звіту:
**18.06.2024 08:30:47 EEST**

ID користувача:
**100005591**

Назва документа: **Крупецький_Програмно-апаратна система зберігання даних з віддаленим доступом на осн...**

Кількість сторінок: **67** Кількість слів: **11839** Кількість символів: **91223** Розмір файлу: **1.77 MB** ID файлу: **1016177806**

## 27.4%
## Схожість

Найбільша схожість: **7.22%** з Інтернет-джерелом (https://javarush.com/ua/groups/posts/4027-eclipse-netbeans-ili-intellij..

26.7% Джерела з Інтернету | 458 ..... Сторінка 69

2.82% Джерела з Бібліотеки | 225 ..... Сторінка 77

## 0.03% Цитат

Цитати | 1 ..... Сторінка 78

Посилання | 1 ..... Сторінка 78

## 0%
## Вилучень

**Немає вилучених джерел**

Tue Jun 18 07:25:25 EEST 2024, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

# Anti-Plagiarism v-15.257

**Максимальне співпадіння з одним документом 3.0%**

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 10%**

| | Документ | | Сумарний збіг по Базі Даних | |
|---|---|---|---|---|
| ID: 131207<br>Назва: БКР Програмно-апаратна система зберігання даних з віддаленим доступом на основі Nextcloud<br>Додано в БД: 2024-06-18<br>Автора: А.С. Крупецький<br>Керівники: В.В. Яцків<br>Консультанти:<br>Опоненти: | Символи | Лексеми | Символи | Лексеми |
| | 81069 | 669 | 3955 (5%) | 32 (5%) |

Джерело плагіату

| ID | Опис | Наявність плагіату в документі | |
|---|---|---|---|
| | | Символи | Лексеми |

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

# РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Крупецький Анатолій Сергійович

Тема: Програмно-апаратна система зберігання даних з віддаленим доступом на основі Nextcloud

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___57___

1. Короткий зміст роботи та прийнятих рішень: Метою дипломної роботи є розробка та впровадження програмно-апаратної системи зберігання даних з віддаленим доступом на основі платформи Nextcloud.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі проведено аналіз предметної області, спрямований на виявлення наявних проблем та завдань. Визначені основні проблеми, що виникають у сфері зберігання даних з віддаленим доступом, був здійснений порівняльний аналіз переваг та недоліків існуючих рішень. Оцінено різні підходи до зберігання даних з віддаленим доступом, включаючи хмарні сервіси, локальні сервери та віртуальні приватні мережі. З'ясовано, що існуючі рішення часто мають обмеження або не відповідають усім потребам користувачів. У другому розділі проведено вибір компонентів розробки програмно-апаратної системи зберігання даних з віддаленим доступом на основі Nextcloud. В ході порівняльного аналізу були враховані різні аспекти, такі як продуктивність, масштабованість, зручність використання та екосистема інструментів. Щодо мов програмування, було розглянуто такі альтернативи як Java, Python, та Go. Java була обрана завдяки своїй великій спільноті розробників, широкому спектру бібліотек та фреймворків, а також стабільності та надійності. У якості середовища розробки було обрано IntelliJ IDEA, оскільки воно забезпечує потужні інструменти для розробки на Java. У третьому

розділі було використано мікросервісну архітектуру, що сприяло зменшенню складності системи шляхом розділення функціональності на невеликі та самостійні компоненти. Це полегшує розробку та підтримку системи, оскільки кожен мікросервіс може бути розроблений, протестований та масштабований окремо від інших. Крім того, такий підхід дозволяє гнучко змінювати архітектуру системи та впроваджувати нові функції без необхідності переписування всього додатку. У проекті було використано багато сучасних інструментів для роботи з базами даних та їх сутностями. Це включало в себе такі інструменти, як Spring Data JPA, Liquibase та PostgreSQL для зручної роботи з об'єктно-реляційним відображенням даних, системи моніторингу та адміністрування баз даних для контролю та оптимізації їхньої продуктивності, а також інструменти для розробки та відлагодження SQL-запитів для ефективної роботи з даними. Nextcloud є ключовим елементом програмно-апаратної реалізації, надаючи масштабовану та надійну платформу для зберігання та управління даними. Він забезпечує віддалений доступ до файлів та інформації, синхронізацію даних між різними пристроями та резервне копіювання даних, що забезпечує безпеку та надійність у роботі з інформацією.

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: мікросервіси потребують більше ресурсів для розгортання та підтримки, ніж монолітні додатки.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: робота виконана на належному науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: добре.

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Радельчук Галина Іванівна, канд. техн. наук, доцент кафедри ТПЗ ХНУ

"21" червня 2024 р. _____(підпис)

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Крупецького Анатолія Сергійовича
ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ2-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

17 червня 2024 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Програмно-апаратна система зберігання даних з віддаленим доступом на основі Nextcloud

Автор: Крупецький Анатолій Сергійович

Спеціальність: 123– Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Яцків Василь Васильович, д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

| № | Висновок | Позначка про відповідність |
|---|---|---|
| 1 | Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту. | відповідає |
| 2 | Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи | |
| 3 | Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат. | |
| 4 | Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту. | |

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення, розміщені в розділах аналізу існуючих аналогів та прототипів, не описують безпосередньо авторське дослідження та не стосуються результатів роботи.
2) усі запозичення є фрагментарними або мають належним чином оформлені посилання.
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчать посилання системи на збіги з 10-40 джерелами на один фрагмент речення.
4) в окремих випадках система фіксує як запозичення послідовності чотирирозрядних двійкових кодів, які є вхідними даними для багатьох задач і не можуть розглядатися як об'єкт авторських прав.
5) всі зафіксовані системою ознаки модифікації тексту стосуються комбінування латинських символів з україномовними скороченнями індексів у формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ідентичності/схожості, складає 27.4% і адресується до 683 першоджерела, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ В.В. Яцків

Гарант ОП _____ С.М. Лисенко

Завідувач кафедри КІІС _____ Т. О. Говорущенко