Лабораторна робота 6

Розробка архітектури інформаційної системи

Мета роботи: Ознайомитись з процесом та здобути практичні навички проєктування архітектури інформаційної системи.

Опис змісту лабораторної роботи:

Дана робота передбачає подовження роботи студентів в команді над розробкою інформаційної системи, яка є продовженням перших трьох лабораторних робіт

Дана робота розрахована на 3 лабораторних заняття:

- Першіе заняття передбачає консультативну форму проведення. На основі матеріалів лекцій та наведених прикладів студенти в команді працюють над завданнями, поставленими в роботі.
- Другі два заняття відведені для усного захисту розробленого проєкту інформаційної системи.

Оцінювання:

- За представлений звіт та захист роботи можна отримати відповідно по 5 балів, разом – 10 балів. Звіт може бути представлений у вигляді презентації або у вигляді текстового документу за вибором команди.
- Якщо команда додатково розробила та представила MVP за проєктом, то вона може додатково отримати до 5 бонусних балів (кількість бонусних балів залежить від кількості розроблених основних функцій системи та ступеня їх пропрацьованості)

Завдання та їх розподіл між студентами в команді:

1. Спроєктувати архітектуру інформаційної системи (за вихідними даними з лабораторних робіт 1-3). Вона повинна містити:
   - ✓ схему основних компонентів системи (вибір оптимальних архітектурних стилей/патернів для реалізації, поділ на компоненти, логічна схема) – **Студент 1**;
   - ✓ запропоновані технології для реалізації системи (технічна схема) – **Студент 2**;
   - ✓ механізм обміну повідомленнями між компонентами або опис, як буде відбуватись потік даних (наприклад, при багатошаровій архітектурі) – **Студент 3**.
2. Навести аргументацію обраних рішень.
3. Зробити висновки за роботою.

ПРИКЛАД 1

Опис застосунку

Мобільний додаток «O-Notes» вирішує проблему втрачання “актуальності” інформації з плином часу та зниження “залученості” людини після завершення роботи. Головна ідея застосунку полягає в оптимізованій системі нагадувань, що слугує закріпленню засвоєної інформації. Застосунок використовує здавалося б не кращу звички постійно відволікатись на телефон, або просто використовує нові реалії, в яких переважна більшість людей має телефони у найближчій доступності.

Вихідні дані

Для коректного виконання роботи швидко наведемо головні характеристики застосунку, що були отримані при виконанні лабораторних 1-3.

Функціональні вимоги:

- Користувач проводить CRUD операції з нотатками в не залежності від режиму доступу чи інтернет-підключення (прямо на пристрої)
- В застосунку реалізовано зв’язок з технічною підтримкою
- ЗМІНА: Авторизація в застосунку може здійснюватись тільки користувачами (застосунок має складову «соціальна мережа»)
- ЗМІНА: Робітники клієнтоорієнтовних відділів працюють із застосунком через додаткове програмне забезпечення, яке не входить до основного застосунку (не розглядається)
- Застосунок має систему публікації нотаток
    - Перевірений користувач може подати створені матеріали в систему
    - Модератор може прийняти матеріали перевірених користувачів до публікації, або видалити з розгляду (не розглядається)
- Застосунок використовує зовнішню базу даних для зберігання нотаток спільноти

Визначимо **нефункціональні вимоги.**

Кількість користувачів можна оцінити через відому для інших записників – отримуємо число, що вимірюється мільйонами

Кількість «онлайн» користувачів застосунку можна оцінити через відому для інших соціальних мереж – отримуємо дуже велике число.

Але можна оптимізувати роботу застосунку так, що функціональна частина «соціальна мережа» буде активізуватися лише в окремих сценаріях. А більша частина роботи буде проходити в режимі офлайн, або в парі «користувач – сховище нотаток».

Зовнішня база даних – нехай ідеальна в налаштуванні та використанні. Вона має бути розрахована на об'єм даних, що рахується приблизно як (обмеження об'єму файлу по режиму доступу користувача)х(максимальна кількість файлів)х(кількість користувачів). Нехай одна нотатка буде не більше ніж 5Мб. Нехай користувач публікує по нотатці в день кожен день. нехай БД архівується кожні п'ять років. Тобто від архіву до архіву буде близько двох тисяч нотаток на одного користувача. Нехай користувачів тисяча. Отримуємо 5х2000х1000 Мб, або 10 Терабайт даних на кожні п'ять років, ще й на додаткову інформацію по користувачах місце залишиться.

Збій в роботі застосунку можна очікувати з боку сховища або від перенавантаження на соціальну складову. Приймемо критичність збою за середню, адже застосунком все ще можна користуватись в режимі офлайн, і всі нотатки користувача зберігаються локально. Навіть залишається можливість переглянути «улюблене» с нотатків спільноти.

Дизайн застосунку повинен бути інтуітивно зрозумілим.

Компоненти

У визначенні компонент будемо спиратись на сформовані вимоги та на приклади існуючих застосунків (нотатки та соц мережі).

Очевидно, що головними сутностями будуть Користувач та Нотатка. Введемо сервіс Користувач, що буде відповідати за управління обліковим записом та даними користувача. Введемо сервіс Нотатки, через який будуть здійснюватись CRUD операції з нотатками. Зазначимо, що цей сервіс не стосується алгоритмів, що стоять за «фізичною» взаємодією з редактором нотаток, адже це реалізовано в коді застосунку. Сервіс Нотатки стосується роботи з нотатками в контексті соціальної частини застосунку.

Також введемо сервіси Авторизація та Відображення. Авторизація визначає, чи співпадають дані регістрації з тими, що вже закріплені за існуючими клієнтами. Відображення відповідає за розгортку бінарних даних нотатки у візуальне представлення нотатки, та за зворотній процес.

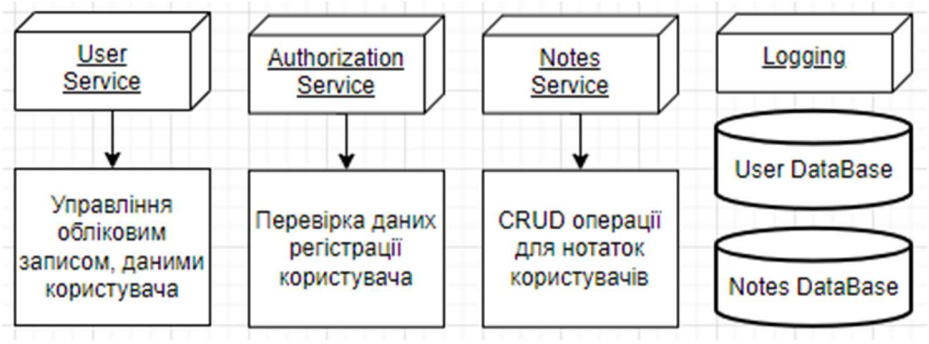
Але відображення реалізоване як конкретний наперед відомий алгоритм в коді застосунку. Тому його сприйняття як окремий сервіс веде до конструкції «застосунок звертається до застосунку для відображення графічної інформації», що не є правильним. Тобто Відображення не є сервісом, але є частиною функціональних можливостей застосунку.

Після авторизації та в залежності від її результату користувач може користуватись методами сервісу Користувач. Сервіс Користувач в свою чергу використовує сервіс

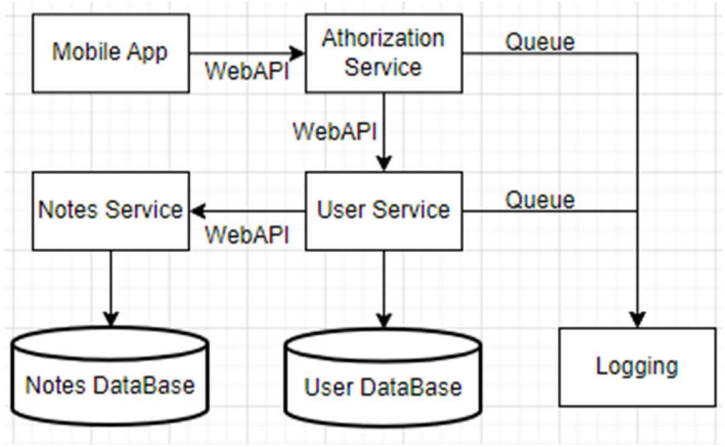
Нотатки, щоб створювати, переглядати, змінювати та видаляти нотатки в соціальній частині застосунку.

Не забудемо про сервіс логування, який буде записувати дії користувача та пов'язані з ним події в системі. На даному етапі будуть логуватися спроби увійти в застосунок та конкретні команди, що викликаються сервісом Користувач.

Помітимо, що об'єкти нотаток цілком автономні, а функціонування об'єктів користувача ніяк не змінюється від того, які саме нотатки до нього прив'язані. Таким чином використовуєм два сховища, для нотаток та для користувачів. Остаточно маємо наступні компоненти.



Логічна схема додатку



Обґрунтуємо вибір транспортного механізму між сервісами.

Як було визначено раніше, перетворення «бінарний файл – нотатка на екрані» відбувається безпосередньо в застосунку. Переходимо до інших систем.

Перед тим, як отримати доступ до соціальної частини застосунку, користувач проходить через авторизацію. Система Авторизація використовує систему Користувач, щоб пересвідчитись у правильності введених даних. Якщо дані вірні, застосунок отримує доступ до методів системи Користувач. Якщо ні, користувачу залишається офлайн-версія застосунку. В кожній ланці цього процесу взаємодія між сервісами відбувається один-на-

один. Спрацювання від сервісів очікується миттєво. Це прямо вказує на синхронний режим доступу та пояснює використання WebAPI.

Надсилання логфайлу до сервісу логування є однобічним, тобто реакція сервісу логування на отриманий файл ніяк не змінює поведінку сервіса-відправника. Взаємодія відбувається один-на-один. Відповідь від сервіса логування не очікується. Це дозволяє скористатися асинхронним режимом доступу та прямо вказує на використання Черги.

Нефункціональні вимоги в подальшому допоможуть визначитись, якими саме базами даних потрібно буде скористатися.

ПРИКЛАД 2

Опис проєкту Проєкт "динамічна черга" вирішує таку проблему, як втрати часу в чергах через формування "вікон" шляхом створення додатку, що дозволяє користувачам повідомляти про неможливість використати їх місце в черзі та вивільнити це місце, а також дозволяє записатись на вивільнене місце або перейти з свого поточного місця на звільнене місце, якщо воно знаходиться в черзі попереду поточного.

Функціональні вимоги

- Мобільний додаток користувача.
- Десктопний додаток працівників закладу для контролю черги.
- Можливість користувача записуватись до черг, виходити з них та змінювати в них місце.
- Можливість працівників контролювати черги (створювати, знищувати, встановлювати кількість місць, видаляти користувачів з черг).

Нефункціональні вимоги

- Система розміщується на сервері компанії-розробника.
- Кількість організацій-замовників < 100, в майбутньому може збільшитись.
- Кількість одночасно активних черг однієї організації ~10.
- Кількість користувачів на одну чергу < 100, існування черги ~7 днів.
- Дані про організацію структуровані і не перевищують 5 мб+1мб на кожного працівника компанії, що має доступ до даних про черги (~10).
- Кількість даних для створення та заповнення черги не перевищує 1 мб.
- Дані користувачів структуровані і не перевищують 10 кб.
- Кількість користувачів <10000, можливе зростання в майбутньому.
- За грубою оцінкою, маємо верхнє обмеження кількості даних - 1500 мб даних організацій, 10000 мб даних користувачів. Загальний обсяг ~11.5 гб, крім того

додається ~0.1 гб даних створених черг щотижня. Але треба враховувати можливість зростання даних в майбутньому.

- Система отримує запити відносно рідко і швидкодія не є критичною. Надійність системи є критичною, втрати повідомлень необхідно уникнути.

Логічна схема

Крім додатків користувача та працівника виділяються компоненти користувацької роботи з чергами та керування чергами, а також логування. Крім того, необхідно забезпечити базу даних

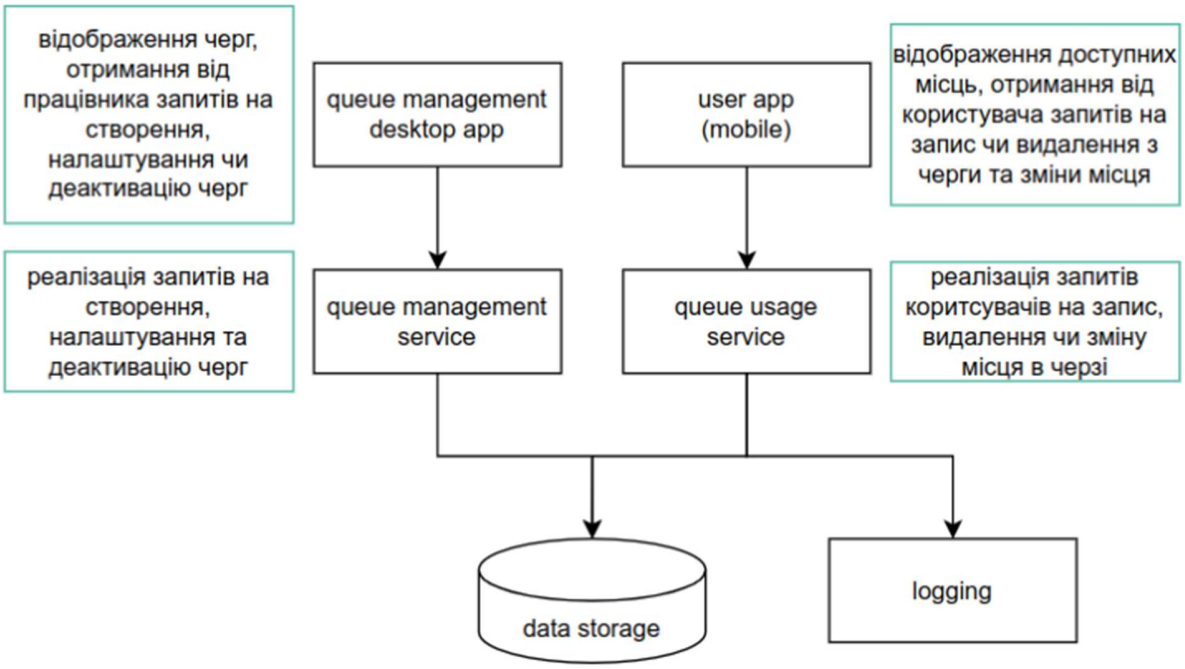


Рисунок 1 – Логічна схема

Розберемо компоненти детальніше.

- Queue management service - сервіс керування з чергою. Містить рівні бізнес-логіки та доступу до даних. Тип: web-api, бо він надає доступ до себе додаткам на пристроях працівників певних організацій, які не підключені безпосередньо до сервісу, а натомість використовують мережу. Технології: Java - мова програмування загального призначення, має помірну швидкодію і використання пам’яті, дозволяє реалізовувати багатопотокову обробку.
- Queue management desktop app - програма на пристрої користувача (у даному випадку - працівника організації-замовника), що дозволяє керувати чергами. Містить рівень інтерфейсу користувача. Тип: настільний додаток. Технології: Java -

мова програмування загального призначення, дозволяє створювати десктопні додатки з допомогою платформи JavaFX. Має помірну швидкодію.

- Queue usage service - сервіс використання черги. Містить рівні бізнес-логіки та доступу до даних. Тип: web-api, бо він надає доступ до себе додаткам на пристроях користувачів, які не підключені безпосередньо до сервісу, а натомість використовують мережу. Технології: Java - мова програмування загального призначення, має помірну швидкодію і використання пам'яті, дозволяє реалізовувати багатопотокову обробку.
- User app - додаток користувача (у даному випадку - клієнта організації-замовника), що дає можливість використовувати черги. Тип: мобільний додаток. Технології: Java - мова програмування загального призначення, дозволяє створювати мобільні додатки з допомогою платформи Android Studio. Має помірну швидкодію.
- Logging - логування подій, що стаються в системі. Тип: сервіс, бо працює автономно. Має рівень отримання даних від інших компонентів, бізнес-логіки та доступу до даних (збереження). Технології: Java.
- База даних - реляційна, бо майже всі дані в системі структуровані. Швидкодія не є критичною. PostgreSQL - безкоштовна система керування реляційними базами даних з відкритим кодом, отже підходить найкраще для даної задачі.

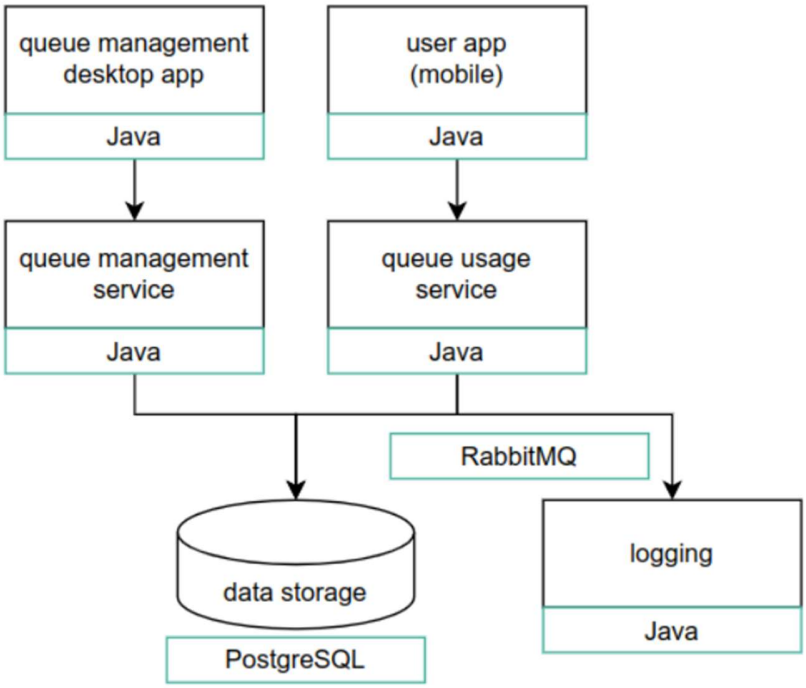


Рисунок 2 – Технічна схема

Розглянемо зв’язки між компонентами. Додаток працівника зв’язаний з сервісом керування чергами, додаток користувача - з сервісом використання черги. В обох випадках зв’язок мережевий і синхронний (бо втрати даних необхідно уникнути, користувачі повинні отримувати підтвердження успішного виконання запитів), отже використовується REST API. Логування повільніше і отримує повідомлення про певні дії в системі через певні проміжки часу, отже для зв’язку логування і сервісів контролю і керування чергами використовується черга, а саме RabbitMQ, бо в системі невелике навантаження і RabbitMQ проста в налаштуванні.

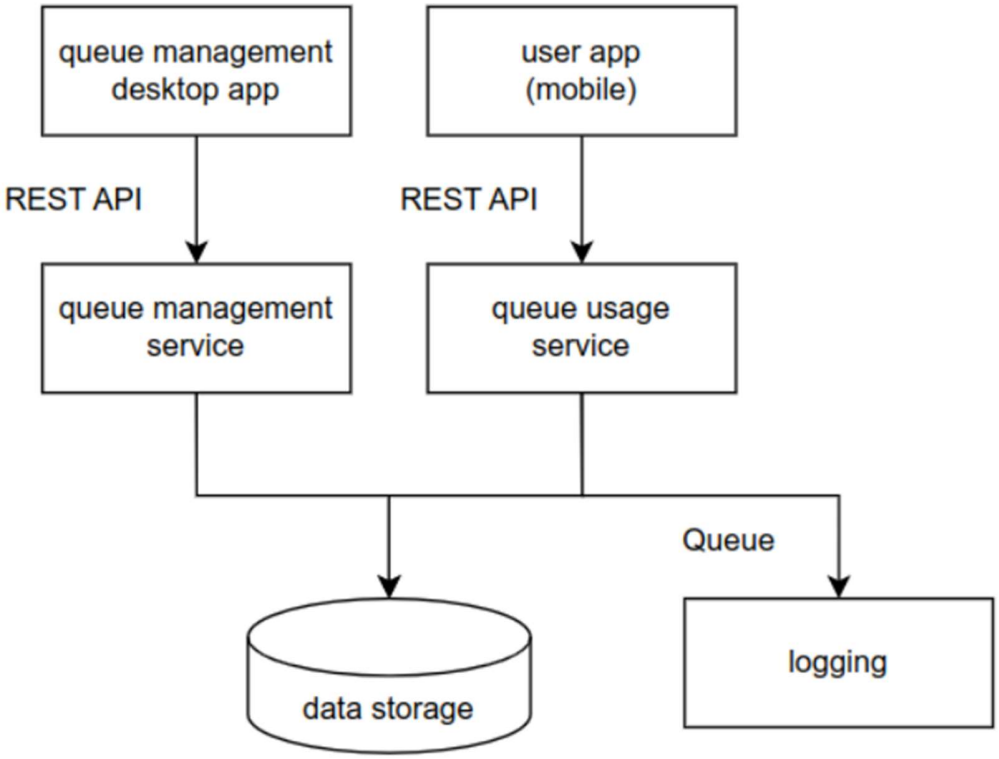


Рисунок 3 – Зв’язки між компонентами

Так як надійність системи є важливою, логування треба зарезервувати. Так як логування використовує небагато ресурсів і єдина причина резервування це ймовірність виходу з ладу резервування, може бути задіяний алгоритм is alive. Резервування сервсів керування та використання черги необхідне для забезпечення роботи багатьох користувачів одночасно, отже використовуються алгоритми load balance. Резервування додатків не здійснюється.

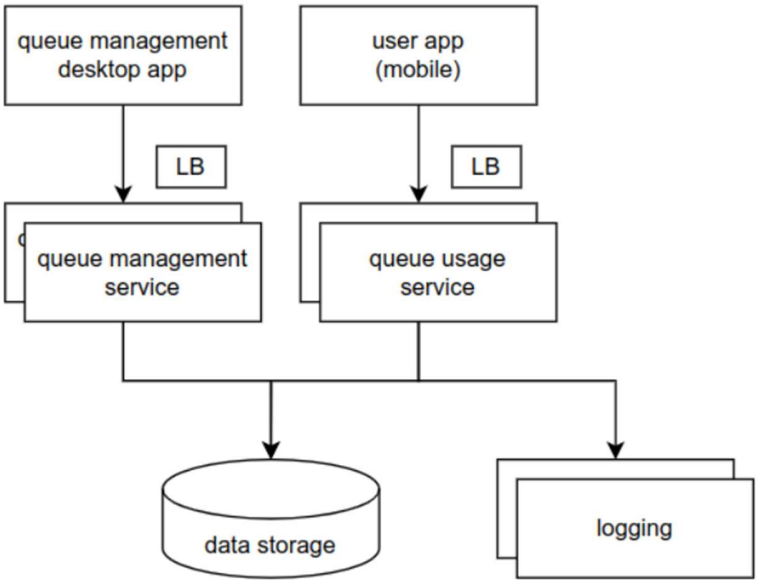


Рисунок 4 – Резервування компонентів