Прикарпатський національний університет імені Василя Стефаника
Фізико-технічний факультет
Кафедра комп'ютерної інженерії та електроніки

Угера Андрій Романович
Andrii Uhera

УДК 004:42

Спеціальність 123 «Комп'ютерна інженерія»

Кваліфікаційна робота
на здобуття освітнього ступеня бакалавра

Розробка та розгортання мікросервісного додатку для отримання інформації про розклад занять студента.
Development and deployment of a microservice application for obtaining information about a student's class schedule.

Науковий керівник:
кандидат к.т.н, Михайло КОТИК
Рецензент:
Кандидат технічних наук, старший викладач кафедри Комп'ютерних наук та інформаційних систем, Михайло ПЕТРИШИН

Івано-Франківськ
2024

Формат	Поз.	Позначення	Найменування	К-ть	Прим.
А4			Структурна схема типового мікросервісного додатку	1	
А4			Діаграма мікросервісів додатку	1	
А4			Пояснювальна записка	64	

Змн.	Арк.	№ докум.	Підпис	Дата	123.КІ-41.23		
Розробив		Угера А.Р.			Специфікація	Літ.	Арк. 2 / Аркушів 1
Перевірив		Котик М.В.					
Н. Контр.							
Затвердив							

АНОТАЦІЯ

Пояснювальна записка дипломного проекту складається з трьох розділів, містить 1 додаток, 18 рисунків та 12 джерел – загалом 71 сторінки.

Об’єкт дослідження: мікросервісний додаток.

Мета дипломного проекту: розробка програмного забезпечення для управління розкладом університетів. Головною метою є створення зручного та ефективного інструменту, який надасть студентам, викладачам та адміністраторам можливість швидко та зручно отримувати доступ до інформації про розклад занять.

У першому розділі описано визначення проблеми, яку вирішує дипломний проект та загальне ознайомлення з мікросервісною архітектурою.

У другому розділі розглянуті існуючі рішення для управління розкладом університетів, їх переваги та недоліки. Також різні методи удосконалення додатку.

У третьому розділі описано ключові моменти процесу проектування програмного забезпечення. Включаючи розробку архітектури системи, проектування графічного інтерфейсу користувача, вибір технологій та інших аспектів.

У додатку наведено: код нашої програми з дотриманням всіх вимог поставленими на початку розробки дипломного проекту.

ABSTRACT

The explanatory note of the diploma project consists of three chapters, contains 1 appendix, 18 figures and 12 sources - a total of 71 pages.

Object of research: microservice application.

The purpose of the thesis project: to develop software for managing university schedules. The main goal is to create a convenient and effective tool that will provide students, teachers and administrators with the ability to quickly and conveniently access information about class schedules.

The first chapter describes the definition of the problem that the thesis project solves and a general introduction to microservice architecture.

The second chapter discusses existing solutions for managing university schedules, their advantages and disadvantages. Also, various methods of improving the application.

The third section describes the key points of the software design process. Including system architecture development, graphical user interface design, technology selection, and other aspects.

The appendix contains: the code of our program in compliance with all the requirements set at the beginning of the development of the diploma project.

ПЕРЕЛІК ОСНОВНИХ СКОРОЧЕНЬ

DevOps - Набір практик, що поєднує розробку програмного забезпечення з його обслуговуванням і експлуатацією (Development і Operations)

Docker - Інструментарій для управління ізольованими Linux-контейнерами

HTTP - Протокол передачі даних, що використовується в комп'ютерних мережах (HyperText Transfer Protocol)

CIAM - Технології, які допомагають організаціям здійснювати цифрову взаємодію зі своїми клієнтами (Customer identity access management)

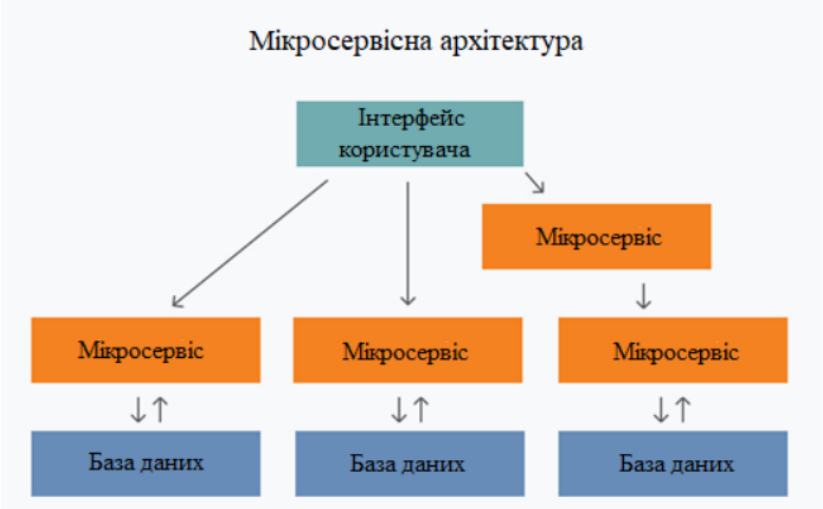
2FA - Метод додаткового захисту (Two-Factor Authentication)

MFA - Розширена автентифікація, метод контролю доступу до інформації (Multi-Factor Authentication)

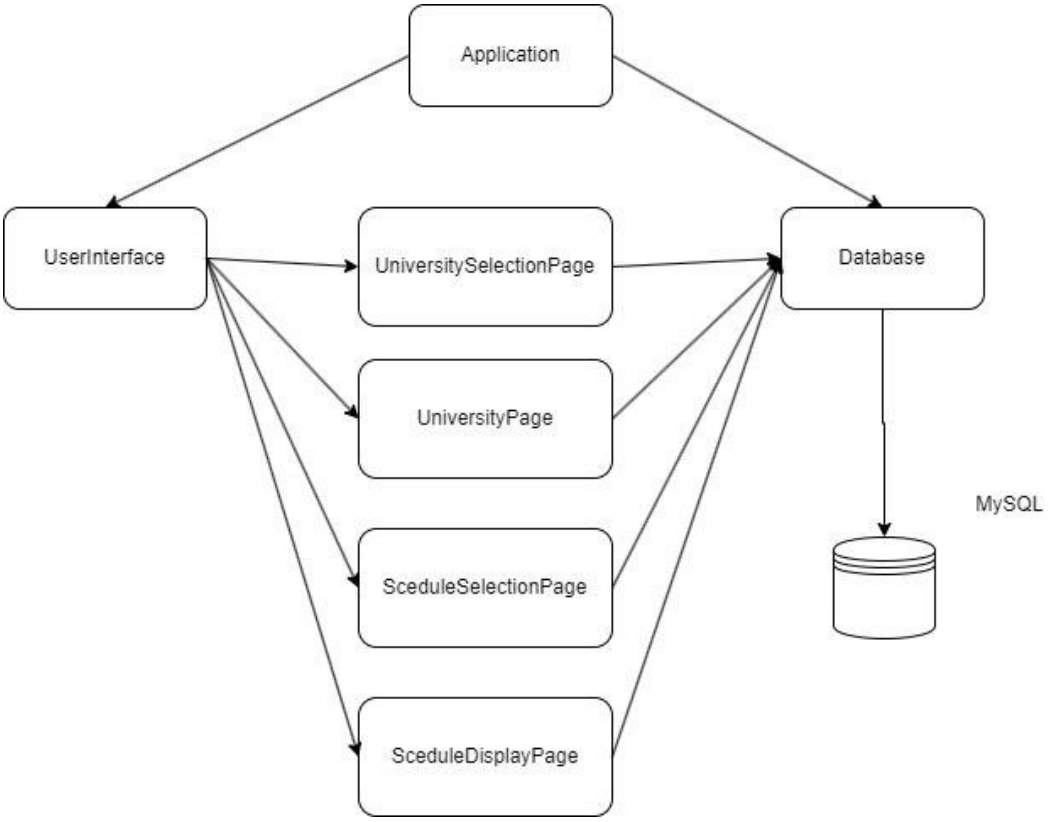
Face ID - Сканер 3D форми особи для захисту входу (Face Identification)

Touch ID - Сканер відбитків пальців для захисту входу (Touch Identification)

SSO - Технологія, що дозволяє користувачеві переходити з одного онлайн-сервісу до іншого, без повторної автентифікації (Single Sign-On)



					123.КІ-41.23		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		Угера А.Р.			Структурна схема типового мікросервісного додатку	*Літ.*	*Арк.* 5 / *Аркушів* 1
Перевірив		Котик М.В.					
Н. Контр.							
Затвердив							



					123.КІ-41.23		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		Угера А.Р.			Діаграма мікросервісів додатку	*Літ.*	*Арк.* 6 / *Аркушів* 1
Перевірив		Котик М.В.					
Н. Контр.							
Затвердив							

Пояснювальна записка

до кваліфікаційної роботи

на тему:

«Розробка та розгортання мікросервісного додатку для отримання інформації про розклад занять студента.»

					123.КІ-41.23			
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розробив		Угера А.Р.			. Пояснювальна записка	*Літ.*	*Арк.*	*Аркушів*
Перевірив		Котик.М.В.					7	64
Н. Контр.								
Затвердив								

ЗМІСТ

ВСТУП

У сучасному світі технології інформаційних систем активно трансформуються, надаючи нові можливості для оптимізації процесів та поліпшення якості життя. У цьому контексті розробка та розгортання мікросервісного додатку стають ключовими етапами впровадження сучасних інформаційних технологій. Особливо актуальною є задача створення мікросервісного додатку для отримання інформації про розклад занять студента, оскільки це не лише сприяє ефективній організації навчального процесу, але й відповідає вимогам сучасного інформаційного суспільства.

Метою даної дипломної роботи є розробка та розгортання мікросервісного додатку, який дозволить студентам швидко та зручно отримувати інформацію про розклад занять. Для досягнення цієї мети передбачається використання сучасних підходів до архітектури мікросервісних систем, а також інструментів для автоматизації процесу розгортання та управління мікросервісами.

В рамках дослідження буде проведений аналіз сучасних підходів до розробки мікросервісних додатків, вибір оптимальної архітектури для запропонованого додатку, а також етапи розробки та розгортання системи. Окрім того, буде розглянуто питання забезпечення безпеки та масштабованості мікросервісного додатку для забезпечення його ефективної роботи в умовах реального виробничого середовища.

У результаті виконання цієї дипломної роботи очікується розробка функціонально повного та ефективного мікросервісного додатку для отримання інформації про розклад занять студента, який буде готовий до впровадження в реальне виробниче середовище та забезпечить високу якість обслуговування користувачів.

Мета дослідження: розробка функціонально повного додатку, використання сучасних підходів до архітектури мікросервісних систем та підготовка до впровадження в реальне виробниче середовище.

Завдання дипломного проєкту: розробити мікросервісний додаток нового зразку на ринку для отримання інформації про розклад занять студента який полегшить користування в пошуку потрібної інформації.

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ

1.1. Мікросервісна архітектура.

Мікросервісна архітектура - це підхід до створення програмного забезпечення, в якому додаток розбивається на невеликі і незалежні компоненти, які називаються мікросервісами. Кожен мікросервіс відповідає за конкретну функціональну частину додатку і може бути розгорнутий та масштабований окремо.

1.1.1. Опис концепції мікросервісної архітектури

Принципи мікросервісної архітектури:

- Розподіленість: мікросервіси є окремими компонентами програмного забезпечення, які функціонують і розвиваються незалежно один від одного.
- Масштабованість: кожен мікросервіс може бути масштабований окремо, що дозволяє забезпечити ефективне використання ресурсів та відповідати потребам навантаження.
- Резистентність до відмов: якщо один мікросервіс виявляє несправність, інші можуть продовжувати працювати без перебоїв.
- Легкість розвитку і впровадження змін: через розподіленість та залежності, зміни в одному мікросервісі не впливають на інші, що спрощує розробку та впровадження змін.

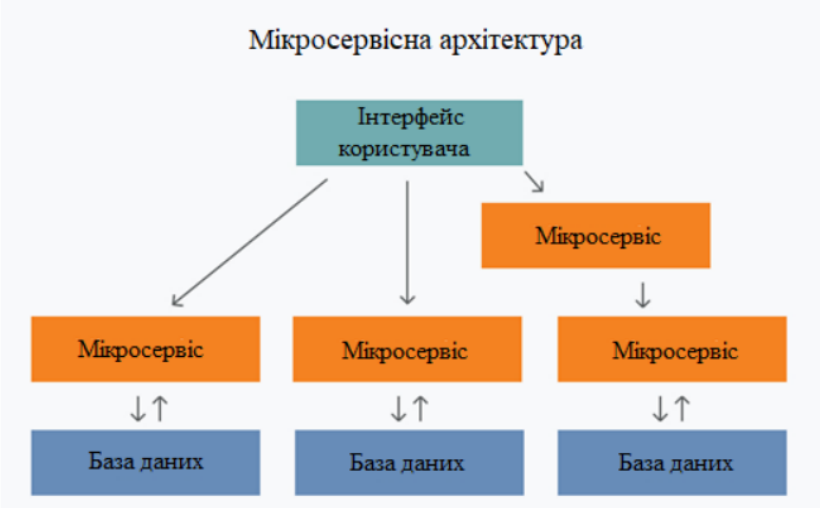


Рис. 1.1. Структурна схема типового мікросервісного додатку

Характеристики мікросервісної архітектури:

- Розгорнутість (Deployability): мікросервіси можуть бути незалежно розгорнуті та оновлені.
- Поміркованість (Modularity): кожен мікросервіс повинен вирішувати конкретну задачу або сервіс.
- Децентралізація (Decentralization): кожен мікросервіс володіє власною базою даних та бізнес-логікою.
- Автономність (Autonomy): мікросервіси можуть функціонувати та оновлюватися незалежно один від одного.

Описуючи концепцію мікросервісної архітектури в деталях, важливо врахувати її основні принципи, характеристики та переваги, а також приклади використання та сценарії розгортання в реальних проектах.

1.1.2. Переваги та недоліки в порівнянні з монолітною архітектурою.

Порівняння мікросервісної архітектури з монолітною - це важливий аспект розуміння переваг та недоліків кожного підходу. Давайте розглянемо їх детальніше:

Переваги мікросервісної архітектури:

- Гнучкість і масштабованість: мікросервіси можуть бути розгорнуті та масштабовані незалежно один від одного. Це дозволяє ефективно реагувати на зміни обсягу трафіку та вимоги до системи.
- Легкість розвитку і впровадження змін: розвиток та оновлення окремих мікросервісів можуть проводитися швидше та безпечніше, оскільки зміни в одному сервісі не впливають на інші.
- Резистентність до відмов: якщо один мікросервіс виявляє несправність, це не призводить до повного збою системи. Інші сервіси можуть продовжувати працювати без перебоїв.
- Легкість у впровадженні нових технологій: мікросервіси можуть використовувати різні технології та мови програмування, що дозволяє використовувати найкращі підходи для кожного конкретного завдання.

Недоліки мікросервісної архітектури:

- Складність управління: з великою кількістю мікросервісів може збільшитися складність управління системою, включаючи моніторинг, логування та відлагодження.
- Складність узгодження: у мікросервісній архітектурі важливо забезпечити узгодженість між сервісами, що може бути складним завданням, особливо в великих системах.
- Затрати на інфраструктуру: розгортання та управління великою кількістю мікросервісів може призвести до збільшення витрат на інфраструктуру та обслуговування.

Переваги монолітної архітектури:

Простота управління: одна програма чи сервіс може бути керована та відлагоджена зі значно меншими зусиллями.

Більш просте узгодження: у монолітної архітектури зазвичай менше проблем з узгодженням між частинами системи.

Знижені витрати на інфраструктуру: розгортання та управління одним монолітним додатком може бути дешевше, ніж з великою кількістю мікросервісів.

Недоліки монолітної архітектури:

- Складність масштабування: монолітні додатки можуть бути складними у масштабуванні, оскільки весь додаток потрібно масштабувати разом.

Більш великі витрати на розвиток і підтримку: розвиток та підтримка великого монолітного додатку може бути витратнішою порівняно з розробкою мікросервісів.

1.1.3. Основні принципи розробки мікросервісних додатків

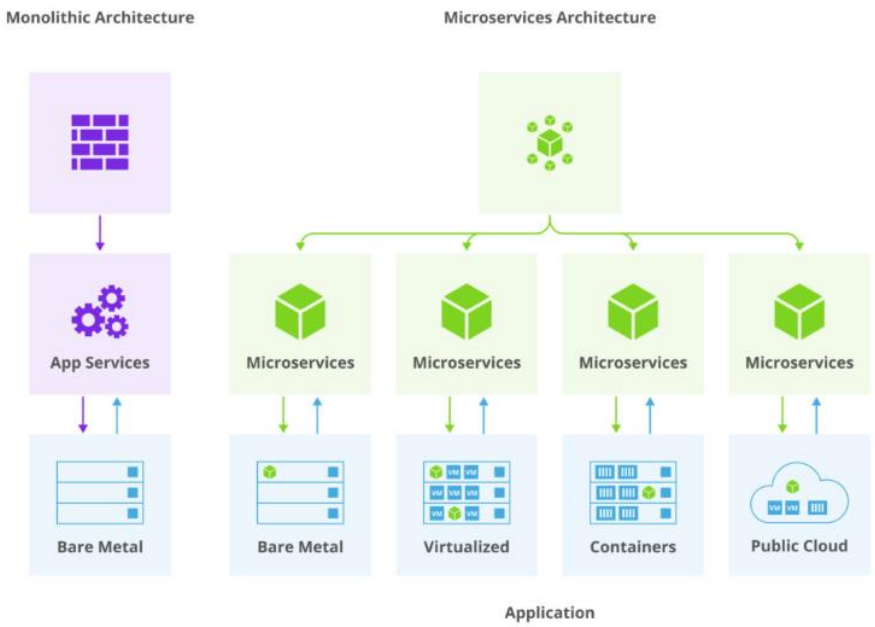


Рис. 1.2. Як працює мікросервісна архітектура

Основні принципи розробки мікросервісних додатків визначають підходи та методології, які допомагають забезпечити ефективність, масштабованість та гнучкість системи. Давайте розглянемо детальніше кожен з цих принципів:

Розподіленість функціональності: функціональність додатку розбивається на окремі сервіси, кожен з яких виконує конкретну функцію або послугу. Це дозволяє збільшити гнучкість та простоту управління, оскільки кожен сервіс може розвиватися та масштабуватися незалежно.

Самостійність сервісів: кожен мікросервіс має бути незалежним і самостійним. Це означає, що він повинен мати свою власну базу даних, бізнес-логіку та інтерфейси. Самостійність дозволяє розвивати та масштабувати кожен сервіс окремо.

Крос-функціональність команд: команди, які розробляють мікросервіси, повинні бути крос-функціональними, тобто включати розробників, тестувальників, аналітиків та інших спеціалістів. Це сприяє швидкому вирішенню проблем та розвитку кожного сервісу.

Автоматизація процесів: важливо автоматизувати процеси розгортання, тестування та моніторингу мікросервісів. Це допомагає забезпечити стабільну та надійну роботу системи.

Культура DevOps: впровадження культури DevOps дозволяє зменшити час впровадження змін та підвищити швидкість реакції на зміни в середовищі. Це створює умови для швидкого впровадження нового функціоналу та вирішення проблем.

Моніторинг та логування: важливо забезпечити ефективний моніторинг та логування кожного сервісу, щоб вчасно виявляти проблеми та вирішувати їх.

Управління конфігураціями: для забезпечення стабільності та надійності системи необхідно ефективно управляти конфігураціями кожного сервісу та його залежностями.

Ці принципи допомагають забезпечити ефективний та успішний розвиток мікросервісних додатків, зменшуючи ризики та підвищуючи швидкість реакції на зміни в середовищі.

1.2. Інструменти розробки мікросервісних додатків.

Інструменти для розробки мікросервісних додатків допомагають забезпечити ефективне управління мікросервісною архітектурою, автоматизувати процеси розгортання, моніторингу та керування сервісами.

1.2.1. Огляд популярних інструментів та технологій, які використовуються для розробки мікросервісних додатків.

Огляд популярних інструментів та технологій, що використовуються для розробки мікросервісних додатків, допомагає розробникам вибрати найбільш підходящі інструменти для їхніх конкретних потреб. Давайте розглянемо деякі з найбільш популярних:

Docker: Docker є одним з найпопулярніших інструментів для контейнеризації додатків. Він дозволяє упаковувати додатки та їхні залежності в контейнери, що забезпечує консистентність середовища розгортання та прискорює розробку, тестування та розгортання мікросервісів.

Kubernetes: Kubernetes є платформою для автоматизації розгортання, масштабування та керування контейнеризованими додатками. Він дозволяє ефективно керувати та оркеструвати мікросервісами у великих і складних системах.

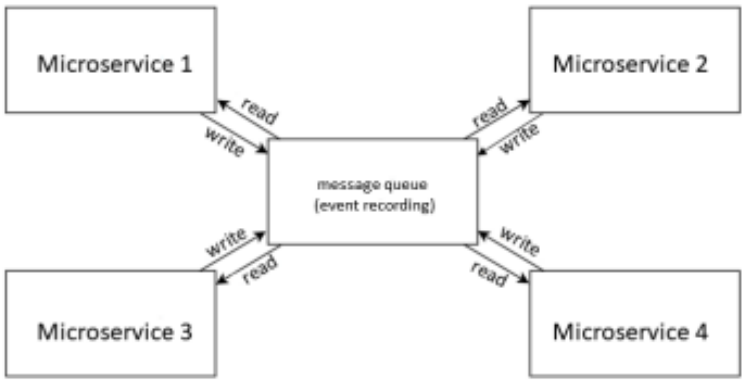


Рис. 1.3. Комунікація подій

Spring Boot: Spring Boot - це фреймворк для розробки Java-додатків, який дозволяє швидко створювати та розгортати мікросервіси. Він надає широкий набір інструментів для розробки, тестування та управління мікросервісами.

Node.js: Node.js - це платформа для розробки серверних додатків на JavaScript. Вона часто використовується для створення легких та ефективних мікросервісів, особливо в області веб-розробки.

GraphQL: GraphQL - це мова запитів та середовище виконання, яке дозволяє клієнтам отримувати саме ті дані, які їм потрібні. Вона стає все популярнішою для розробки мікросервісів через свою гнучкість та здатність працювати з різноманітними джерелами даних.

Рис. 1.4. Комунікація повідомленнями

Consul, etcd, ZooKeeper: Ці інструменти використовуються для керування конфігураціями, розподіленим керуванням станом та виявленням служб для мікросервісних архітектур.

Prometheus, ELK Stack (Elasticsearch, Logstash, Kibana): Ці інструменти використовуються для моніторингу та логування мікросервісних додатків. Вони дозволяють відстежувати різні метрики та аналізувати журнали для виявлення проблем та вирішення їх.

Ці інструменти та технології є лише деякими з численних, що доступні для розробки мікросервісних додатків. Вибір конкретних інструментів залежить від ваших потреб, вимог проекту та експертизи команди розробників.

1.2.2. Аналіз їхніх можливостей та відмінностей.

Функціональність і можливості: різні інструменти та технології можуть мати різні набори функціональності. Наприклад, Docker забезпечує контейнеризацію додатків, Kubernetes - оркестрацію контейнерів, Spring Boot - фреймворк для розробки Java-додатків тощо. Під час аналізу слід оцінити, які саме можливості надають інструменти та технології, і як це відповідає вимогам та потребам вашого проекту.

Продуктивність та швидкодія: деякі інструменти можуть мати кращу продуктивність та швидкодію, ніж інші. Наприклад, Node.js відомий своєю високою швидкодією у веб-розробці, тоді як Spring Boot може забезпечувати велику швидкодію для Java-додатків. Важливо аналізувати продуктивність та швидкодію кожного інструменту та технології та обирати той, який найкраще відповідає потребам вашого проекту.

Масштабованість: деякі інструменти та технології можуть бути більш масштабованими за інші. Наприклад, Kubernetes має потужні інструменти для автоматизації масштабування та керування мікросервісами у великих системах. Важливо враховувати потреби у масштабованості вашого проекту та обирати інструменти, які найкраще відповідають цим потребам.

Вартість і витрати: деякі інструменти можуть бути безкоштовними або мати вільну ліцензію, тоді як інші можуть вимагати платних підписок або витрат

на підтримку та обслуговування. Важливо враховувати вартість та витрати на впровадження та підтримку кожного інструменту та технології та обирати той, який найкраще підходить для вашого бюджету.

Екосистема та підтримка: важливо також враховувати екосистему та рівень підтримки кожного інструменту та технології. Наприклад, Docker та Kubernetes мають широку спільноту та активну підтримку, що робить їх популярними виборами для мікросервісних додатків.

1.3. Методи розгортання мікросервісних додатків.

Розгортання мікросервісних додатків може бути реалізоване за допомогою різних методів, в залежності від ваших вимог до масштабування, доступності та інших факторів.

1.3.1. Огляд різних методів розгортання мікросервісів

Огляд різних методів розгортання мікросервісів включає в себе розгляд різних підходів до розгортання та управління мікросервісами від початкового розроблення до етапу виробництва. Давайте розглянемо кілька основних методів:

Ручне розгортання: це метод, коли розробники ручно розгортають кожен мікросервіс на серверах або в хмарному середовищі. Хоча цей підхід може бути простим для невеликих проектів, він може стати неефективним та складним для керування великим числом сервісів на виробничому етапі.

Інструменти контейнеризації (наприклад, Docker): використання контейнеризації дозволяє упаковувати мікросервіси та їхні залежності у стандартизовані контейнери, які можна легко розгортати та запускати на будь-якому середовищі, що підтримує контейнери. Цей підхід забезпечує консистентність середовища розгортання та полегшує автоматизацію процесу розгортання.

Інструменти оркестрації (наприклад, Kubernetes): оркестрація дозволяє автоматизувати керування та масштабуванням мікросервісами. Вона надає засоби для автоматизованого розгортання, масштабування, керування та моніторингу мікросервісів у великих та складних системах. Kubernetes є одним

з найпопулярніших інструментів оркестрації та надає широкий набір функціональності для управління мікросервісами.

Серверлесс архітектура: серверлесс архітектура дозволяє розробникам написати код для виконання певної функціональності, а потім автоматично масштабувати та керувати цим кодом без необхідності управління інфраструктурою. Цей підхід може бути корисним для певних видів додатків, особливо для обробки подій або функцій, які виконуються рідко.

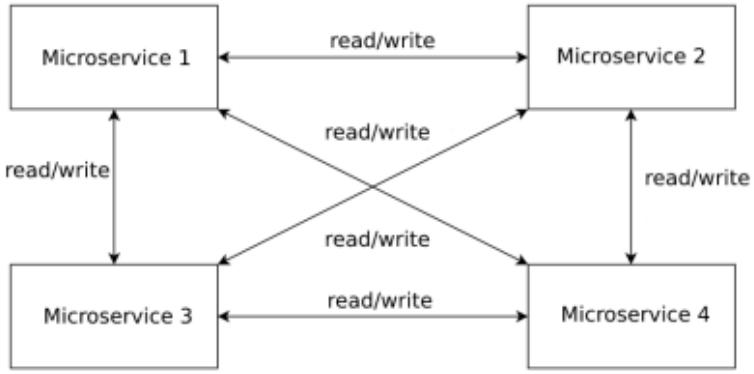


Рис. 1.5. HTTP-зв'язок

Мікросервіси в контейнерах на платформі хмарних сервісів: деякі хмарні сервіси, такі як AWS ECS, Azure Kubernetes Service або Google Kubernetes Engine, надають можливість розгортати мікросервіси у контейнерах на їхніх інфраструктурах. Цей підхід дозволяє розробникам використовувати переваги хмарних сервісів для розгортання та керування мікросервісами.

Кожен з цих методів має свої переваги та недоліки, і вибір конкретного методу залежить від вимог та особливостей вашого проекту.

1.3.2. Порівняння підходів, включаючи контейнеризацію (наприклад Docker), оркестрацію (наприклад, Kubernetes) та серверні платформи.

Порівняння різних підходів до розгортання мікросервісних додатків, таких як контейнеризація (наприклад, Docker), оркестрація (наприклад, Kubernetes) та використання серверних платформ, дозволяє з'ясувати переваги та недоліки кожного підходу. Давайте розглянемо їх докладніше:

Контейнеризація (наприклад, Docker):

Переваги: контейнери забезпечують ізольоване середовище для кожного мікросервісу, що дозволяє легко переносити та розгортати їх на різних середовищах, будь то локальна машина розробника, тестовий сервер або виробниче середовище. Контейнеризація забезпечує консистентність середовища, що полегшує розробку, тестування та розгортання додатків. Docker забезпечує швидке розгортання та масштабування мікросервісів завдяки легкості використання та широкій підтримці спільноти.

Недоліки: для керування більшим числом контейнерів та їхніми залежностями може бути складно використовувати лише Docker. Тут потрібно використовувати інші інструменти, такі як Docker Compose або інші інструменти оркестрації. Контейнеризація не надає вбудованої підтримки для автоматизованого масштабування та керування додатками у виробничому середовищі.

Оркестрація (наприклад, Kubernetes):

Переваги: kubernetes надає потужні засоби для автоматизованого керування та оркестрації контейнеризованими додатками. Він дозволяє автоматизувати розгортання, масштабування, керування та моніторинг мікросервісів у великих та складних системах. Kubernetes надає вбудовану підтримку для автоматизованого масштабування та керування мікросервісами, що дозволяє ефективно використовувати ресурси та забезпечувати високу доступність системи.

Недоліки: Налаштування та управління Kubernetes може бути складним та вимагати значних зусиль для розгортання та підтримки. Використання Kubernetes може вимагати додаткових витрат на ресурси та підтримку порівняно з іншими методами розгортання.

Серверні платформи (наприклад, AWS ECS, Azure Kubernetes Service, Google Kubernetes Engine):

Переваги: використання серверних платформ дозволяє легко розгортати та керувати мікросервісами в хмарних сервісах. Ці платформи надають інструменти для автоматизації розгортання, масштабування та керування мікросервісами, що спрощує процес розробки та підтримки.

Недоліки: використання серверних платформ може бути дорожчим порівняно з розгортанням на власних серверах або хостинг-провайдерах. Залежність від конкретного хмарного сервісу може обмежити гнучкість та портативність вашої системи.

Порівняння цих підходів допоможе вам зрозуміти їхні переваги та недоліки та обрати той, який найкраще відповідає вашим потребам та вимогам вашого проекту

РОЗДІЛ 2. АЛГОРИТМІЧНИЙ

2.1 Методи аутентифікації та авторизації

У міру того, як підприємства просуваються до цифрової зрілості в часи надійних хмарних систем і суворої безпеки в Інтернеті, автентифікація та авторизація використовуються разом (а також часто взаємозамінно). Хоча обидва терміни звучать схоже, вони відносяться до абсолютно різних процесів безпеки. В рамках управління ідентифікацією та доступом клієнтів (CIAM) автентифікація перевіряє особу користувача, тоді як авторизація підтверджує, чи має користувач доступ до виконання певної функції. Іншими словами, автентифікація - це ідентифікація користувачів шляхом підтвердження того, ким вони себе називають, тоді як авторизація - це процес встановлення прав і привілеїв користувача. Обидва процеси відіграють однаково важливу роль у захисті конфіденційних даних від порушень та несанкціонованого доступу.

2.1.1. Опис алгоритмів та методів аутентифікації користувачів, які використовуються для доступу до інформації.

1. Автентифікація на основі пароля

Також відома як автентифікація на основі знань, автентифікація на основі пароля покладається на ім'я користувача та пароль або PIN-код. Це найпоширеніший метод автентифікації; кожен, хто входив до комп'ютера, знає, як використовувати пароль.

Автентифікація на основі пароля є найлегшим типом автентифікації для зловмисників. Люди часто використовують паролі повторно і створюють паролі, які можна вгадати, використовуючи слова зі словника та загальнодоступну особисту інформацію. Крім того, працівникам потрібен пароль для кожної програми та пристрою, які вони використовують, що ускладнює запам'ятовування паролів і змушує їх спрощувати паролі, де це можливо. Це робить акаунти вразливими до фішингу та атак грубої сили.

2. Двофакторна/багатофакторна автентифікація

Двофакторна автентифікація (2FA) вимагає, щоб користувачі надавали принаймні один додатковий фактор автентифікації, окрім пароля. Багатофакторна

автентифікація вимагає двох або більше факторів. Додатковими факторами можуть бути будь-який з типів автентифікації користувача, описаних у цій статті, або одноразовий пароль, надісланий користувачеві в текстовому повідомленні або на електронну пошту. Фактори також можуть включати позасмугову автентифікацію, яка передбачає, що другий фактор знаходиться на іншому каналі, ніж оригінальний пристрій, щоб зменшити ризик атак "зловмисника посередині". Цей тип автентифікації посилює безпеку облікових записів, оскільки зловмисникам потрібно більше, ніж просто облікові дані для доступу.

Надійність 2FA ґрунтується на вторинному факторі. Зловмисники можуть легко зламати текстові та електронні повідомлення. Використання біометричних даних або push-повідомлень, які вимагають наявності або наявності у користувача певних даних, забезпечує більш надійний 2FA. Однак будьте обережні при розгортанні 2FA або MFA, оскільки це може додати тертя в UX.

3. Біометрична автентифікація

Біометрія використовує те, чим є користувач. Вона менше покладається на легко викрадений секрет для підтвердження того, що користувач є власником облікового запису. Біометричні ідентифікатори є унікальними, що ускладнює злом акаунтів з їх допомогою.

До найпоширеніших типів біометрії відносяться наступні:

- Сканування відбитків пальців - перевірка автентичності на основі відбитків пальців користувача.
- Сканування долоні ідентифікує користувачів за унікальним візерунком вен.
- Розпізнавання обличчя використовує характеристики обличчя людини для верифікації.
- Розпізнавання райдужної оболонки ока сканує око користувача за допомогою інфрачервоного випромінювання, щоб порівняти його зі збереженим профілем.
- Поведінкова біометрія використовує те, як людина ходить, друкує або поводиться з пристроєм.

Користувачі можуть бути знайомі з біометрією, що полегшує її розгортання в корпоративному середовищі. Багато споживчих пристроїв мають можливості біометричної автентифікації, включаючи Windows Hello та Face ID і Touch ID від Apple. Біометрична автентифікація часто є простішою і швидшою, оскільки не вимагає від користувача згадування секрету чи паролю. Зловмисникам також складніше її підробити.

Технології залишаються найбільшим недоліком біометрії. Не кожен пристрій обробляє біометричні дані однаково, якщо взагалі обробляє. Старі пристрої можуть використовувати лише збережене статичне зображення, яке можна обдурити за допомогою фотографії. Новіше програмне забезпечення, таке як Windows Hello, може вимагати, щоб пристрій мав камеру з ближньою інфрачервоною зйомкою. Це може вимагати більших початкових витрат, ніж інші типи автентифікації. Користувачі також повинні відчувати себе комфортно, надаючи свої біометричні дані компаніям, які все ще можуть бути зламані.

4. Єдиний вхід

Єдиний вхід (SSO) дозволяє працівнику використовувати єдиний набір облікових даних для доступу до декількох додатків або веб-сайтів. Користувач має обліковий запис у постачальника ідентифікаційних даних (IdP), який є надійним джерелом для програми (постачальника послуг). Постачальник послуг не зберігає пароль. IdP повідомляє сайту або додатку за допомогою файлів cookie або токенів, що користувач пройшов через нього верифікацію.

SSO зменшує кількість облікових даних, які користувачеві потрібно запам'ятовувати, що підвищує безпеку. Користувацький інтерфейс також покращується, оскільки користувачам не потрібно входити в кожен обліковий запис щоразу, коли вони звертаються до нього, за умови, що вони нещодавно пройшли автентифікацію в IdP. SSO також може допомогти скоротити час, який служба підтримки витрачає на вирішення проблем з паролями.

Цей метод автентифікації означає, що якщо IdP зазнає витоку даних, зловмисники можуть отримати доступ до кількох облікових записів за допомогою одного набору облікових даних. SSO також вимагає від IT-спеціалістів значних

початкових витрат часу на налаштування та підключення до різних додатків і веб-сайтів.

5. Автентифікація на основі токенів

Автентифікація на основі токенів дозволяє користувачам входити в акаунти за допомогою фізичного пристрою, наприклад, смартфона, ключа безпеки або смарт-карти. Вона може використовуватися як частина MFA або для забезпечення безпарольної роботи. З автентифікацією на основі токенів користувачі підтверджують свої облікові дані один раз протягом заздалегідь визначеного періоду часу, щоб зменшити кількість постійних входів.

Токени ускладнюють зловмисникам доступ до облікових записів користувачів. Зловмисникам потрібен фізичний доступ до токену і знання облікових даних користувача, щоб проникнути в обліковий запис.

Потрібно довіряти працівникам, щоб вони відстежували свої токени, інакше їм можуть заблокувати доступ до облікових записів. Оскільки користувачі будуть заблоковані, якщо вони забудуть або втратять токен, компанії повинні спланувати процес повторної реєстрації.

6. Автентифікація на основі сертифікатів

Автентифікація на основі сертифікатів використовує цифрові сертифікати, видані центром сертифікації, і криптографію з відкритим ключем для перевірки особи користувача. Сертифікат зберігає ідентифікаційну інформацію та відкритий ключ, тоді як приватний ключ зберігається у користувача віртуально.

Автентифікація на основі сертифікатів використовує SSO. IT-спеціалісти можуть розгортати, керувати та відкликати сертифікати. Цей тип автентифікації добре підходить для компаній, які наймають підрядників, яким тимчасово потрібен доступ до мережі.

Розгортання автентифікації на основі сертифікатів може бути дорогим і трудомістким. IT-спеціалісти також повинні створити процес повторної реєстрації на випадок, якщо користувачі не зможуть отримати доступ до своїх ключів - наприклад, якщо їх викрадуть або зламають пристрій.

2.1.2. Управління правами доступу користувачів до різних частин системи.

Управління правами доступу користувачів до різних частин розкладу занять студента, зокрема з урахуванням можливостей, доступних тільки викладачам, є критично важливим аспектом для забезпечення безпеки та конфіденційності інформації.

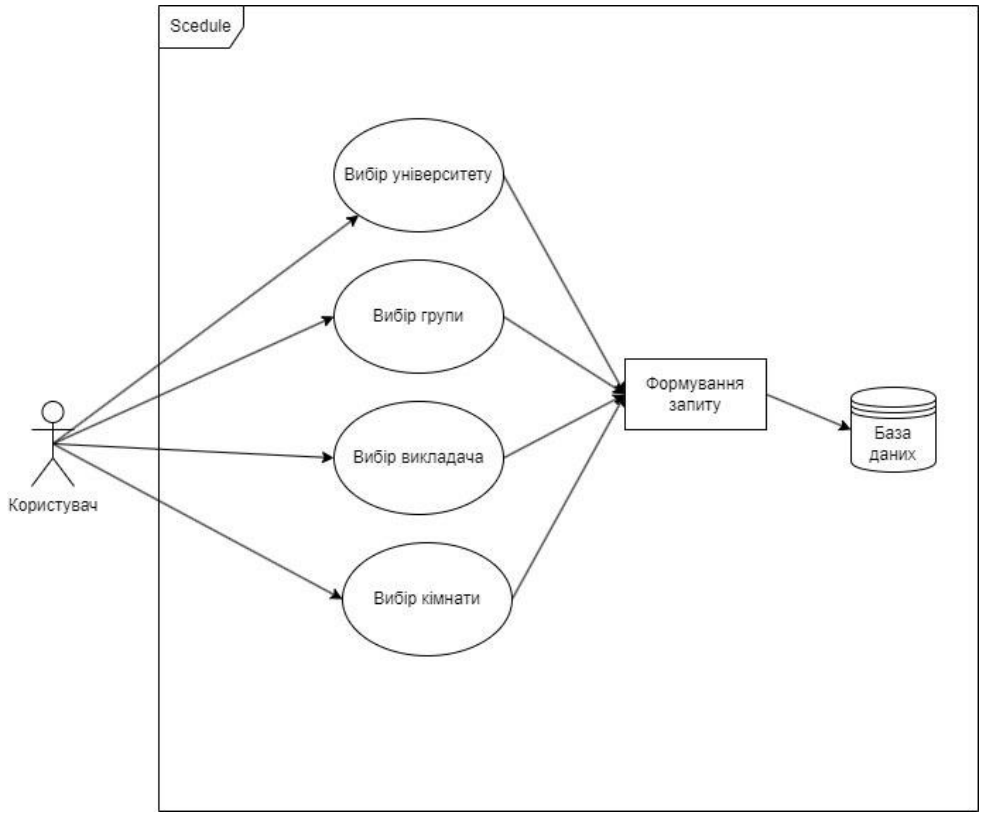


Рис. 2.1. Діаграма варіантів використання

Перш за все, потрібно визначити різні ролі користувачів у системі. У нашому випадку це будуть, наприклад, студенти та викладачі. Ролі визначаються на основі функціональних можливостей, які кожна група користувачів може виконувати. Для студентів доступні лише обмежені функціональні можливості, такі як перегляд розкладу та можливість відмітити свою присутність. Викладачі, з іншого боку, мають розширені можливості, такі як перегляд розкладу, додавання або редагування занять, викладання матеріалів, а також можливість змінювати місце проведення пар.

Реалізація системи управління доступом, яка визначає, які користувачі мають доступ до яких частин функціональності. Для викладачів, які мають

право змінювати місце проведення пар, необхідно встановити відповідні права доступу, які дозволяють їм виконувати цю операцію.

Реалізація механізму авторизації, що перевіряє ідентичність користувача та його роль у системі. Перед наданням доступу до функціональності, такої як зміна місця проведення пар, система повинна перевірити, чи має користувач відповідні права доступу.

Журналювання всіх дій, пов'язаних зі зміною місця проведення пар та іншими діями викладачів. Це дозволяє вести детальний журнал того, хто та коли вносив зміни, що допомагає вирішити можливі суперечки або виявити випадки недопущення.

2.2 Алгоритми оптимізації роботи з базою даних.

У сучасному світі, де все ґрунтується на даних, оптимізація баз даних має вирішальне значення для ефективного управління даними, швидшої обробки запитів і загальної продуктивності системи. Незалежно від того, чи це невеликий додаток, чи велика корпоративна система, бази даних відіграють ключову роль у зберіганні, управлінні та пошуку інформації. Однак, оскільки бази даних зростають у розмірі та складності, їхня продуктивність може постраждати, якщо їх не оптимізувати ефективно. Використання різних методів оптимізації має важливе значення для забезпечення безперебійної роботи баз даних та їхньої оптимальної продуктивності.

2.2.1. Опис алгоритмів та методів оптимізації запитів до бази даних для ефективного отримання розкладу занять студента.

Індексування: індекси мають фундаментальне значення для швидкого пошуку даних. Вони працюють як зміст у книзі, дозволяючи базам даних швидко знаходити інформацію. Однак, надмірна індексація може знизити продуктивність. Вибір правильних стовпців для індексування на основі шаблонів запитів і частоти доступу до даних має вирішальне значення. Регулярний перегляд та оптимізація індексів може значно підвищити продуктивність бази даних.

```
CREATE INDEX idx_name_age ON employees(name, age);
```

Нормалізація: нормалізація бази даних зменшує надмірність і підвищує цілісність даних, організовуючи їх в окремі пов'язані таблиці. Мінімізуючи дублювання даних, вона зменшує вимоги до сховища і запобігає аномаліям при оновленні. Однак надмірна нормалізація може призвести до збільшення кількості операцій об'єднання, що впливає на продуктивність запитів. Баланс між нормалізацією та денормалізацією, коли це необхідно, є ключовим для оптимального проектування бази даних.

```
CREATE TABLE orders (
order_id INT PRIMARY KEY,
customer_id INT,
order_date DATE,
FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

Оптимізація запитів: створення ефективних запитів є життєво важливим для продуктивності бази даних. Такі прийоми, як використання відповідних методів об'єднання, обмеження пошуку даних лише необхідними стовпцями та уникнення непотрібних вкладених або підзапитів, можуть значно підвищити швидкість виконання запитів. Крім того, оптимізація складних запитів за допомогою переписування запитів або використання підказок до запитів може підвищити їхню продуктивність.

```
EXPLAIN SELECT * FROM orders WHERE customer_id = 123;
```

Перегородки: розбиття на розділи передбачає поділ великих таблиць на менші, більш керовані частини. Це підвищує продуктивність, дозволяючи виконувати операції з конкретними розділами, а не з цілими таблицями. Розбиття на розділи може ґрунтуватися на критеріях діапазону, списку або хешу, залежно від характеру даних і шаблонів доступу, тим самим підвищуючи продуктивність запитів і ефективність обслуговування.

```
CREATE TABLE sales (
sale_id INT PRIMARY KEY,
sale_date DATE,
amount DECIMAL(10, 2)
```

```
) PARTITION BY RANGE (YEAR(sale_date)) (
 PARTITION p0 VALUES LESS THAN (1990),
PARTITION p1 VALUES LESS THAN (2000),
PARTITION p2 VALUES LESS THAN (2010),
PARTITION p3 VALUES LESS THAN (2020),
PARTITION p4 VALUES LESS THAN (MAXVALUE)
);
```

Кешування: впровадження механізмів кешування може значно зменшити навантаження на базу даних та покращити час відгуку. Використання кешів у пам'яті або розподілених систем кешування може зберігати дані, до яких часто звертаються, зменшуючи потребу в повторних запитах до бази даних. Однак слід ретельно продумати політику закінчення терміну дії кешу та узгодженість даних, щоб забезпечити їхню точність.

```
-- Pseudocode
DECLARE @cacheKey NVARCHAR(255) = 'query_cache_key';
DECLARE @cachedResult NVARCHAR(MAX);

SET @cachedResult = REDIS.GET(@cacheKey);
IF @cachedResult IS NULL
BEGIN
            -- Execute the query and store the result in the
cache
            SET @cachedResult = EXECUTE_QUERY('SELECT * FROM
large_table');
            REDIS.SET(@cacheKey, @cachedResult, EXPIRY_TIME);
END

 -- Use @cachedResult for further processing
```

Оптимізація апаратного забезпечення: іноді вузькі місця в продуктивності бази даних можна усунути, оптимізувавши апаратні ресурси. Це включає в себе модернізацію апаратних компонентів, таких як процесори, пристрої зберігання

даних і пам'ять, щоб задовольнити зростаючі вимоги робочого навантаження бази даних.

```
-- Example: Increase the size of the query cache
SET GLOBAL query_cache_size = 256M;
```

Регулярне обслуговування: виконання рутинних завдань з обслуговування бази даних, таких як реорганізація індексів, оновлення статистики та очищення даних, допомагає підтримувати оптимальну продуктивність. Регулярне резервне копіювання та перевірка узгодженості бази даних також мають вирішальне значення для цілісності даних і надійності системи.

```
-- Update statistics for a table
UPDATE STATISTICS table_name;
```

Використання стиснення та архівування: стиснення даних та архівування історичної інформації може оптимізувати зберігання та підвищити продуктивність бази даних. Зберігання архівних даних окремо зменшує навантаження на активні бази даних, дозволяючи їм працювати ефективніше.

Стратегії масштабування: зі збільшенням обсягу даних і навантаження на користувачів, використання стратегій масштабування, таких як вертикальне масштабування (збільшення ресурсів сервера) або горизонтальне масштабування (розподіл даних між кількома серверами), може запобігти зниженню продуктивності. Хмарні рішення пропонують можливості масштабування, що дозволяє базам даних адаптуватися до мінливих робочих навантажень.

Оптимізація баз даних - це безперервний процес, який вимагає всебічного розуміння структури бази даних, моделей робочого навантаження та здатності адаптуватися до мінливих вимог. Впроваджуючи комбінацію цих методів оптимізації, а також регулярно переглядаючи і допрацьовуючи конфігурації баз даних, організації можуть гарантувати, що їхні бази даних працюють з максимальною продуктивністю, ефективно підтримуючи їхні додатки та бізнес-процеси.

2.3 Алгоритми синхронізації даних

На даний момент мікросервіс є найпопулярнішим способом розробки програмного забезпечення. Цей підхід передбачає розбиття програми на окремі мікросервіси, кожен з яких виконує свою окрему бізнес-завдання. Для кожного мікросервісу існують наступні характерні вимоги. Наприклад, він повинен бути невеликим і незалежним, повинен виконувати певну бізнес-вимогу, взаємодіяти з іншими мікросервісами, використовуючи патерн розумних кінцевих точок і німих труб, а також дотримуватися децентралізованого управління. Однією з основних частин мікросервісів є взаємодія мікросервісів. Це означає обмін даними між ними. Важливим елементом мікросервісної парадигми парадигми мікросервісів є децентралізація даних. Вона часто реалізується шляхом виділення власної бази даних для для кожного мікросервісу. Це дозволяє ізолювати дані від інших сервісів, тим самим підтримуючи їхню стабільність та безпеку даних.

2.3.1. Опис алгоритмів синхронізації даних між різними компонентами мікросервісного додатку.

Проблему синхронізації даних між сервісами можна вирішити, використовуючи оптимальний метод обміну повідомленнями. Для мінімізації недоліків цього методу можна використати Apache Kafka. Apache Kafka - це потокова платформа, яка публікує потоки даних і підписки на них, а також зберігає на них, а також зберігає та обробляє їх. Kafka має величезні можливості масштабування і надає централізовану платформу для взаємодії мікросервісів. Ще однією перевагою Kafka є можливість налаштування часу зберігання повідомлень, що забезпечує реплікацію, цілісність та зберігання даних протягом будь-який період часу. І, нарешті, потокова обробка підвищує рівень абстракції, що в свою свою чергу, дозволяє Kafka динамічно обчислювати похідні потоки та набори даних на основі потоків даних. Таким чином, розглянувши основні варіанти взаємодії мікросервісів, можна зробити висновок що метод обміну повідомленнями є найбільш придатним для вирішення проблеми синхронізації даних між різними сервісами. При цьому, найкращою реалізацією цього методу є використання message брокера. Наприклад, Apache Kafka, він забезпечує

відправку та обробку повідомлень в реальному часі, а також може гарантувати доставку повідомлень. Для того, щоб ефективно та швидко реалізувати додаток за допомогою мови програмування, спочатку потрібно побудувати математичну модель, що описує систему.

2.3.2. Розгляд методів реалізації асинхронної синхронізації для забезпечення цілісності даних та уникнення конфліктів.

Послідовність синхронізації даних від сервісу виробника до сервісу споживача складається з алгоритму. Спочатку виробник послуг, відповідно до певних вимог, виконує на своєму боці формування та обробку даних на своєму боці. Після цього він конвертує їх у формат JSON і надсилає це повідомлення брокеру Kafka. Повідомлення зберігається в журналі фіксації. Кожен запис містить додаткову інформацію, таку як назва розділу, назва теми, зміщення повідомлення. Це дозволяє подальші маніпуляції зі збором даних від брокера. У цей час споживач послуги слухає теми, на які він підписаний, на наявність нових повідомлень. З цього моменту починається обробка повідомлень починається обробка повідомлень в сервісі споживача.

Як тільки з'являються нові повідомлення, слухач віднімає пакет повідомлень, розмір якого можна налаштувати, і передає його на обробку до служби обробки повідомлень Message Service сервіс обробки повідомлень. Повідомлення розбиваються на Linked Hash Map <Ціле число, Особа>, де ключем є глобальний ідентифікатор особи в системі, а значенням карти є особа з усіма її даними за адресами 5 та контактами. Кожне повідомлення конвертується з JSON в об'єкти особи, адреси та контактів. Після цього, за допомогою класу Validator, особа перевіряється на коректність даних, якщо дані невірні, то видається відповідне повідомлення. некоректні, то виводиться відповідне повідомлення в лог, і це повідомлення не обробляється. Пізніше дані особи поміщаються в картку. Такий алгоритм дає можливість не обробляти проміжні дані особи в межах даного пакету повідомлень. Це дозволяє зменшити кількість обчислень, адже якщо в пакеті повідомлень є 10 записів, що стосуються однієї і тієї ж людини, то він буде оброблений лише 1 раз.

Як тільки сформована фінальна карта всіх оброблених повідомлень, вона передається в пул потоків (Thread Pool) Task Executor - клас, що реалізує управління паралельною обробкою даних. Кожне повідомлення з карти передається в Обробник повідомлень, де обробляється паралельно з обробкою інших повідомлень. інших повідомлень. Розмір Виконавця завдань пулу потоків задається через файл налаштувань. У Message Processor відбувається вся подальша обробка повідомлення, зіставлення глобальних глобальних атрибутів людини з локальними, обчислення деяких параметрів тощо.

В кінці фінальні дані зберігаються в базі даних нашої системи. Як тільки обробка всіх повідомлень пакета закінчена, слухач готовий до отримання нового пакета повідомлень.

2.4 Алгоритми кешування даних.

У сучасну цифрову епоху швидкість і продуктивність є вирішальними факторами для забезпечення успіху додатку. Повільне завантаження та низька продуктивність можуть призвести до розчарування користувачів, збільшення показника відмов та втрати прибутку. Саме тут на допомогу приходить кешування - потужна технологія, яка може значно підвищити продуктивність додатків, тимчасово зберігаючи дані, до яких часто звертаються. Кешування - це процес зберігання даних, до яких часто звертаються, в локальній кеш-пам'яті, яка може бути як на пристрої користувача, так і на сервері. Коли користувач знову запитує ті самі дані, програма може отримати їх з локального кешу замість того, щоб завантажувати з бази даних. Це значно скорочує час, необхідний для завантаження програми, що, в свою чергу, покращує користувацький досвід.

2.4.1. Опис алгоритмів кешування даних для покращення продуктивності та зменшення навантаження на сервери.

Кешування - це метод тимчасового зберігання даних, до яких часто звертаються, для прискорення роботи додатків. Однак важливо забезпечити своєчасне оновлення кешованих даних, щоб користувачі завжди бачили найсвіжішу інформацію. Цей процес відомий як анулювання кешу. Існує кілька методів, які можна використовувати для анулювання кешованих даних,

залежно від типу даних, що кешуються, і стратегії кешування, яка використовується.

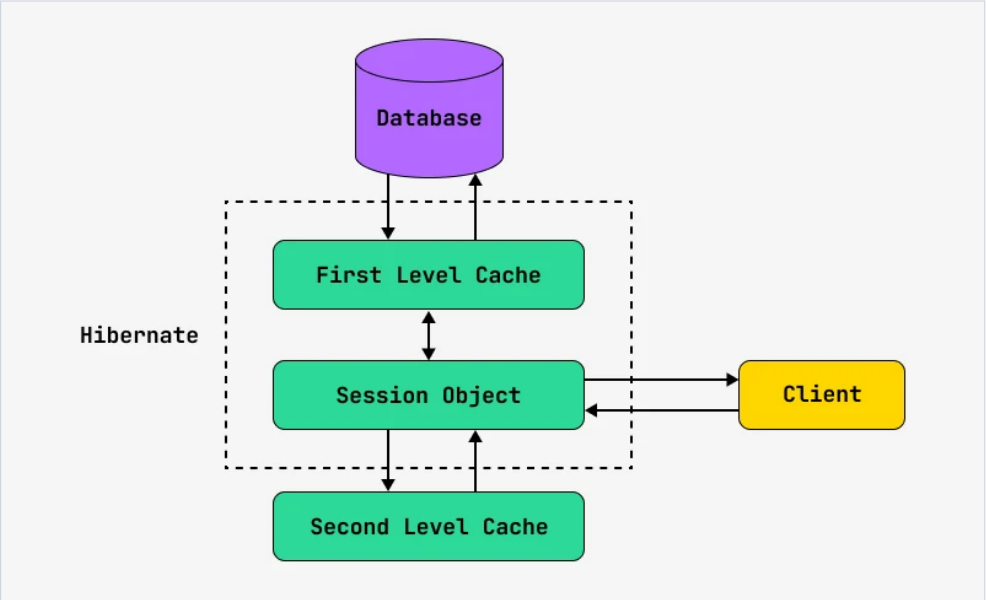


Рис. 2.2. Види кешування в Hibernate

Анулювання за часом: передбачає встановлення терміну придатності для кешованих даних. Після закінчення терміну дії кешовані дані стають недійсними, і їх потрібно завантажувати з сервера повторно. Цей метод зазвичай використовується для вмісту, який не змінюється часто, наприклад, статичних зображень або файлів CSS.

Цей метод корисний, коли є контент, який залишається статичним протягом тривалого періоду часу. Можна встановити час закінчення терміну дії залежно від того, як часто очікувати, що контент буде змінюватися.

Анулювання вручну: передбачає ручне анулювання кешованих даних, якщо відомо, що вони застаріли. Це може зробити адміністратор програми, або скрипт, який відстежує зміни в джерелі даних. Ручне анулювання зазвичай використовується для вмісту, який змінюється нечасто, але потребує негайного оновлення, коли він змінюється.

Анулювання на основі подій: передбачає анулювання кешованих даних, коли відбувається певна подія, наприклад, новий елемент додається до бази

даних. Цей метод зазвичай використовується для контенту, який часто змінюється і потребує оновлення в режимі реального часу.

Цей метод корисний, якщо є контент, який часто змінюється і потребує негайного оновлення.

Анулювання на основі версії: пов'язує номер версії з кешованими даними. Коли дані оновлюються, номер версії збільшується. Це дозволяє додатку визначати, чи є кешовані дані актуальними, порівнюючи номер версії з поточною версією даних. Цей метод зазвичай використовується для контенту, який часто змінюється, але не потребує оновлення в режимі реального часу.

Цей метод корисний, якщо є контент, який часто змінюється, але не потребує негайного оновлення.

Кожна з цих методик скасування кешування має свої сильні та слабкі сторони. Найкращий метод для додатку буде залежати від багатьох факторів, таких як тип даних, що кешуються, і стратегія кешування, що використовується. Важливо регулярно відстежувати та налаштовувати систему перевірки кешування на недійсність, щоб гарантувати, що кешовані дані оновлюються вчасно і точно.

2.4.2. Розгляд стратегій кешування та вибір оптимальних алгоритмів для конкретних сценаріїв використання.

Впровадити кешування в додатку можна кількома способами, залежно від типу кешування, який обирати. Основні з них, які можна виконати, щоб реалізувати кешування:

Визначити дані, до яких часто звертаються в додатку.

Визначити, який тип кешування найкраще підходить для потреб.

Налаштувати систему кешування (Redis або Memcache), щоб зберігати дані та отримувати їх за потреби.

Протестувати свій додаток, щоб переконатися, що кешування працює належним чином.

Слідкувати за системою кешування, щоб переконатися, що вона працює оптимально.

У висновку ми можем підсумувати що кешування - це потужний метод підвищення продуктивності додатків. Скорочуючи час, необхідний для завантаження даних, кешування може значно покращити користувацький досвід і зменшити показник відмов. Впровадивши кешування в додатку, можна підвищити продуктивність, знизити витрати і забезпечити кращий загальний досвід для своїх користувачів. Завдяки правильній стратегії кешування можна забезпечити швидку, надійну та чуйну роботу додатку незалежно від обсягу трафіку, який він отримує.

РОЗДІЛ 3. ПРОГРАМНИЙ

3.1. Упорядкування мікросервісів додатка.

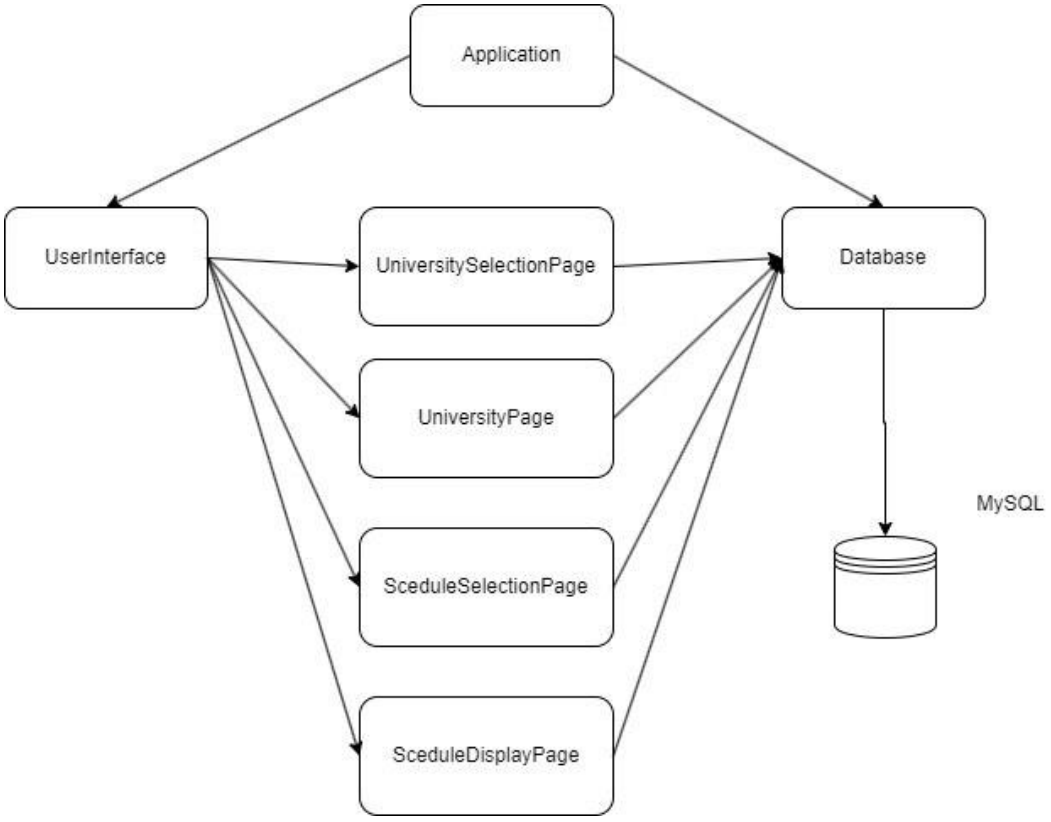


Рис. 3.1. Діаграма мікросервісів додатку

Наш додаток є системою управління розкладом для університетів. Основні компоненти додатку включають класи Application, UserInterface та DataBase, які працюють разом для забезпечення функціональності додатку. Загалом, користувач може шукати та вибирати університет, переглядати розклад за різними критеріями та отримувати докладну інформацію про конкретний університет та його розклад. Всі ці можливості побудовані на основі взаємодії між компонентами додатку та базою даних.

Також використання мікросервісів в даній програмі позволяє нам не переживати якщо будь який із класів перестане функціонувати оскільки всі остальні в цей момент будуть працювати далі і програма не закінчить свою роботу. Це дає нам змогу виправляти помилки не затронувши ті частини коду які до цього не відносяться.

3.2 Мікросервіс Application.

```python
from DataBase import DataBase
from Interface.UserInterface import RunUI

class Application():
    def __init__(self) -> None:
        # Ініціалізуємо об'єкт бази даних з параметрами підключення
        DataBase.setDataBase("localhost", "root", "2501", "test_scedule")

    def Run(self):
        # Запускаємо користувацький інтерфейс
        RunUI()

    @staticmethod
    def ShutDown():
        # Поточно цей метод не має функціональності, але може бути розширений для виконання ді
        pass
```

Рис. 3.2. Клас Application та функції які в ньому знаходяться

Клас Application служить як основний контейнер для нашої програми. У методі __init__ відбувається ініціалізація бази даних з параметрами підключення. Метод Run запускає користувацький інтерфейс. Метод ShutDown поточно не виконує жодних дій, але може бути використаний для виконання певних завершальних дій при завершенні програми.

3.3 Мікросервіс DataBase

```python
class DataBase:
    # Метод для встановлення з'єднання з базою даних
    @staticmethod
    def setDataBase(dbhost, username, password, databaseName):
        global DataBaseHost
        global UserName
        global Password
        global DataBaseName
        global DataBaseConnector

        # Збереження параметрів підключення до бази даних
        DataBaseHost = dbhost
        UserName = username
        Password = password
        DataBaseName = databaseName

        try:
            # Підключення до бази даних
            DataBaseConnector = mysql.connector.connect(
                host=DataBaseHost,
                user=UserName,
                passwd=Password,
                database=DataBaseName
            )
            print("Database connection successfully established.")
        except MySQLConnectorError as e:
            logging.error(f"Failed to connect to the database: {e}")
            DataBaseConnector = None
```

Рис. 3.3. Функція в класі Database яка відповідає за з'єднання з базою даних.

setDataBase(dbhost, username, password, databaseName): ця функція встановлює з'єднання з базою даних. Вона ініціалізує підключення до бази даних з вказаними параметрами хоста, користувача, пароля та назви бази даних. Після встановлення з'єднання вона виводить повідомлення про успішне встановлення з'єднання. Ця функція є ключовою для початку роботи з базою даних, оскільки без неї неможливо взаємодіяти з даними.

```python
# Метод для отримання переліку університетів
@staticmethod
def GetUniversities():
    cursor = DataBase.getCursor()
    if cursor is None:
        return []
    try:
        # Запит до бази даних для отримання назв уніве
        query = "SELECT name FROM universities"
        cursor.execute(query)
        return cursor.fetchall()
    except Exception as e:
        print(f"Database error: {e}")
        return []
    finally:
        cursor.close()
```

Рис. 3.4. Функція в класі Database яка відповідає за повернення назви всіх університетів.

GetUniversities(): ця функція повертає назви всіх університетів. Вона дозволяє отримати загальний перелік університетів, які присутні в базі даних. Це важлива функція для реалізації функціональності, пов'язаної з відображенням інформації про університети.

```python
# Метод для отримання розкладу для заданої групи та університету
@staticmethod
def GetScheduleByGroup(university, groupName):
    cursor = DataBase.getCursor()
    if cursor is None:
        return []
    try:
        # Запит до бази даних для отримання розкладу для заданої групи та університету
        query = """
        SELECT
            u.name AS university_name,
            g.name AS group_name,
            c.name AS course_name,
            t.name AS teacher_name,
            cl.room_number AS classroom,
            s.day_of_week,
            s.start_time,
            s.end_time
        FROM Schedules s
        JOIN _Group g ON s.group_id = g.group_id
        JOIN Courses c ON s.course_id = c.course_id
        JOIN Teachers t ON s.teacher_id = t.teacher_id
        JOIN Classrooms cl ON s.classroom_id = cl.classroom_id
        JOIN Universities u ON g.university_id = u.university_id
        WHERE u.name = %s
        AND g.name LIKE %s;
        """
        cursor.execute(query, (university, groupName))
        return cursor.fetchall()
    except Exception as e:
        print(f"Database error: {e}")
        return []
    finally:
        cursor.close()
```

Рис. 3.5. Функція в класі Database яка відповідає за повернення розкладу для заданої групи та університету.

GetScheduleByGroup(university, groupName): ця функція повертає розклад для заданої групи та університету. Вона є однією з ключових функцій для отримання розкладу з бази даних. Завдяки цій функції можна отримувати детальну інформацію про розклад занять для конкретної групи в конкретному університеті.

3.4 Мікросервіс UserInterface

```
# Перехід на сторінку вибору розкладу залежно від типу
def gotoScheduleSelectionPage(self, type):
    if type == "group":
        self.stacked_widget.setCurrentWidget(self.schedule_group_page)
        self.schedule_group_page.showGroups(self.current_university)
    elif type == "teacher":
        self.stacked_widget.setCurrentWidget(self.schedule_teacher_page)
        self.schedule_teacher_page.showTeachers(self.current_university)
    elif type == "classroom":
        self.stacked_widget.setCurrentWidget(self.schedule_classroom_page)
        self.schedule_classroom_page.showClassrooms(self.current_university)
    else:
        print("Error")
```

Рис. 3.6. Функція в класі UserInterface яка відповідає за перехід на сторінку вибору розкладу залежно від типу.

gotoScheduleSelectionPage(type): ця функція відповідає за перехід на сторінку вибору розкладу залежно від типу. Вона приймає аргумент type, який вказує на тип розкладу (група, викладач або аудиторія). В залежності від значення type, функція встановлює відповідну сторінку в QStackedWidget і викликає метод відповідної сторінки для відображення інформації.

```
# Перехід на сторінку відображення розкладу
def gotoScheduleDisplayPage(self, type, selection):
    self.stacked_widget.setCurrentWidget(self.search_results_page)
    self.search_results_page.load_schedule(self.current_university, type, selec
```

Рис. 3.7. Функція в класі UserInterface яка відповідає за перехід на сторінку відображення розкладу.

gotoScheduleDisplayPage(type, selection): ця функція відповідає за перехід на сторінку відображення розкладу. Вона приймає аргументи type (тип розкладу) і selection (вибірка розкладу) і встановлює сторінку ScheduleDisplayPage в QStackedWidget. Після цього викликається метод load_schedule для завантаження розкладу на сторінку.

```python
# Закриття додатку
def shutDown(self):
    sys.exit(self.application.exec_())
```

Рис. 3.8. Функція в класі UserInterface яка відповідає за закриття додатку.

shutDown(): ця функція відповідає за закриття додатку. Вона викликає функцію exit з модуля sys, щоб закрити додаток.

3.5 Мікросервіс UniversitySelectionPage

```python
# Відображення кнопок університетів
def display_buttons(self, university_list):
    while self.buttons_layout.count():
        button = self.buttons_layout.takeAt(0).widget()
        if button:
            button.deleteLater()

    start_index = self.current_page * self.items_per_page
    end_index = min(start_index + self.items_per_page, len(university_list))

    # Створення кнопок для відображення університетів
    for university in university_list[start_index:end_index]:
        btn = QPushButton(university[0])
        btn.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
        btn.clicked.connect(lambda _, uni=university[0]: self.select_universit
        self.buttons_layout.addWidget(btn)
```

Рис. 3.9. Функція в класі UniversitySelectionPage яка відповідає за відображення кнопок для університетів на сторінці вибору університету.

display_buttons(self, university_list): ця функція відповідає за відображення кнопок для університетів на сторінці вибору університету. Вона очищує поточний вміст макету, а потім створює кнопки для кожного університету у списку university_list, додає їх до макету і налаштовує підключення для обробки подій кліку на кожну кнопку.

```python
# Обробка вибору університету
def select_university(self, university_name):
    print(f"Selected: {university_name}")
    self.university_selected.emit(university_name)
```

Рис. 3.10. Функція в класі UniversitySelectionPage яка викликається при виборі користувачем певного університету.

select_university(self, university_name): ця функція викликається при виборі користувачем певного університету. Вона виводить у консоль повідомлення про обраний університет і емітує сигнал university_selected, щоб повідомити інші частини програми про вибір користувача.

3.6 Демонстрація інтерфейсу програми

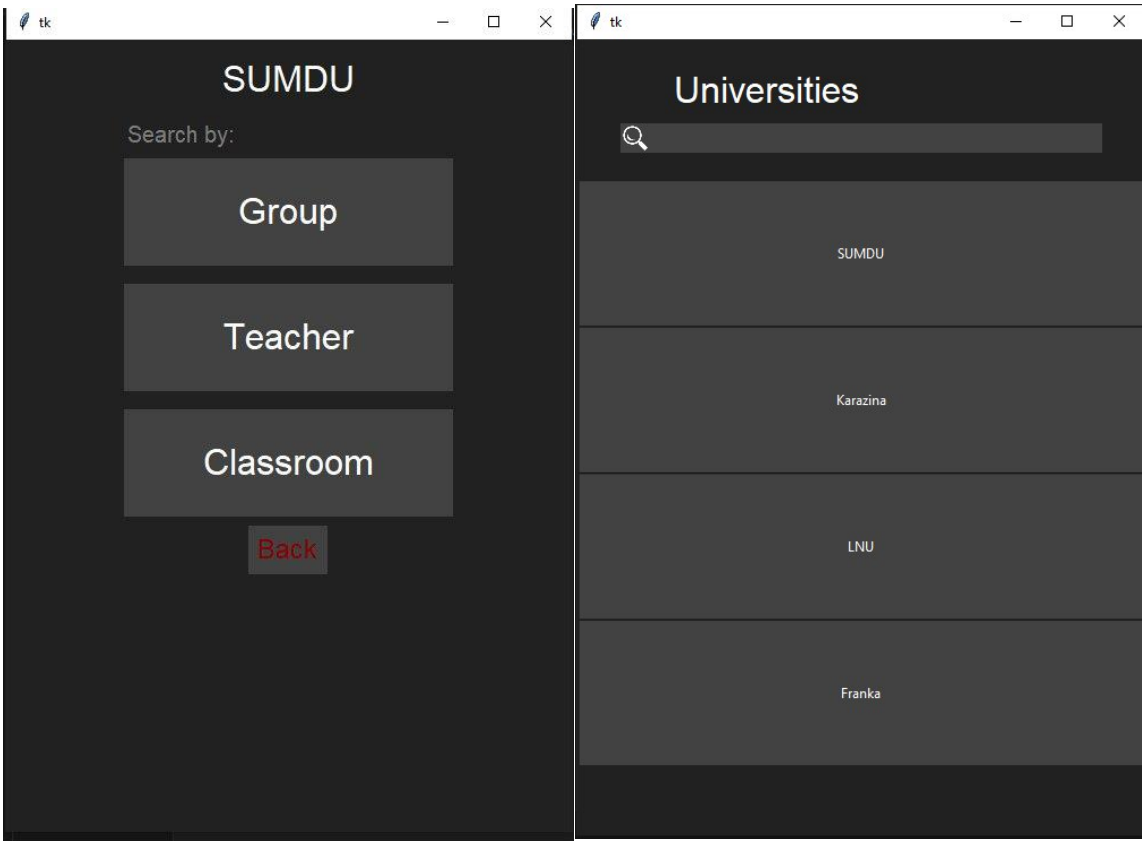


Рис. 3.11. Загальний вигляд інтерфейсу програми.

3.7 Висновки

Наш додаток для управління розкладом університетів є добре структурованою системою, що складається з трьох основних компонентів: класів Application, UserInterface та DataBase. Application є вхідною точкою програми, яка ініціалізує з'єднання з базою даних та запускає інтерфейс користувача. UserInterface відповідає за відображення інтерфейсу та управління ним, в той час як DataBase забезпечує взаємодію з базою даних.

Додаток має ряд сторінок, які дозволяють користувачу вибирати університет, переглядати розклад за різними критеріями та отримувати докладну інформацію про обраний університет та його розклад. Кожна сторінка взаємодіє з базою даних

через відповідний клас, що забезпечує доступ до необхідної інформації. Всі ці компоненти працюють разом, щоб надати користувачеві зручний та ефективний інструмент для управління розкладом навчання

.

ВИСНОВКИ

1 У цій дипломній роботі було проведено дослідження та розроблено мікросервісний додаток для отримання інформації про розклад занять студента. Для досягнення цієї мети було проведено аналіз концепції мікросервісної архітектури та виявлено переваги її використання, такі як гнучкість, масштабованість та відновлюваність системи. Окрім того, були розглянуті основні принципи розробки мікросервісних додатків, включаючи розподілену архітектуру та використання API для комунікації між сервісами.

2 Управління правами доступу користувачів до різних частин розкладу занять студента виявилося ключовим аспектом розробки. З урахуванням потреб користувачів, таких як студенти та викладачі, було реалізовано механізм авторизації та управління правами доступу. Для викладачів були встановлені спеціальні права доступу, зокрема можливість змінювати місце проведення пар, що відображає їхні функціональні потреби.

3 Під час розробки було розглянуто різні інструменти та технології для побудови мікросервісного додатку, а також проведено порівняння їхніх можливостей та відмінностей. Було виявлено, що використання контейнеризації, оркестрації та серверних платформ сприяє покращенню швидкодії та масштабованості додатку.

Загалом, розроблений мікросервісний додаток для отримання інформації про розклад занять студента є ефективним та гнучким рішенням, яке враховує потреби користувачів та використовує передові підходи в області розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Designing and Building Solid Microservice Ecosystems, 2023p., 318 ст.
2. Big Data Optimization: Recent Developments and Challenges, 2016p., 210 ст.
3. Database Performance Tuning and Optimization: Using Oracle, 2006 p., 456 ст.
4. Optimization Algorithms for Query Processing in Distributed, 2001p., 97ст.
5. Distributed Machine Learning and Gradient Optimization, 2022p., 141ст.
6. Soft Computing for Data Analytics, Classification Model, and Control, 2022p., 160ст.
7. IT Logging and Monitoring Methods a Complete Guide, 2019., 291ст.
8. Runtime Verification, 2012., 154ст.
9. DevOps and Containers Security, 2020., 311ст.
10. Docker: Up and Running, 2023., 446ст.
11. Serverless Design Patterns and Best Practices, 2018., 121ст.
12. Serverless Architectures on AWS: With Examples Using AWS Lambda, 2017., 217ст.

ДОДАТОК

Код нашої програми поділений на частини:

Main.py

```
from Application import Application

def main():
    app = Application()
    app.Run()

if __name__ == "__main__":
    main()
```

Application.py

```
from DataBase import DataBase

from Interface.UserInterface import RunUI

class Application():
    def __init__(self) -> None:
       DataBase.setDataBase("localhost", "root", "2501", "test_scedule")

    def Run(self):
        RunUI()

    @staticmethod
    def ShutDown():
        pass
```

DataBase.py

```
import mysql.connector
from mysql.connector import Error as MySQLConnectorError
import logging

class DataBase:

    @staticmethod
    def setDataBase(dbhost, username, password, databaseName):
        global DataBaseHost
        global UserName
        global Password
        global DataBaseName
        global DataBaseConnector

        DataBaseHost = dbhost
        UserName = username
        Password = password
        DataBaseName = databaseName

        try:
            DataBaseConnector = mysql.connector.connect(
                host=DataBaseHost,
                user=UserName,
                passwd=Password,
                database=DataBaseName
            )
            print("Database connection successfully established.")
        except MySQLConnectorError as e:
            logging.error(f"Failed to connect to the database: {e}")
            DataBaseConnector = None
```

```
    @staticmethod
    def getCursor():
        if DataBaseConnector is None or not DataBaseConnector.is_connected():
            logging.error("Database is not connected. Cannot fetch cursor.")
            return None
        return DataBaseConnector.cursor()

    @staticmethod
    def GetUniversity(name):
        cursor = DataBase.getCursor()
        if cursor is None:
            return []
        try:
            query = "SELECT * FROM universities WHERE name LIKE %s"
            cursor.execute(query, ('%' + name + '%',))
            return cursor.fetchall()
        except Exception as e:
            print(f"Database error: {e}")
            return []
        finally:
            cursor.close()

    @staticmethod
    def GetUniversities():
        cursor = DataBase.getCursor()
        if cursor is None:
            return []
        try:
            query = "SELECT name FROM universities"
            cursor.execute(query)
            return cursor.fetchall()
        except Exception as e:
            print(f"Database error: {e}")
            return []
        finally:
            cursor.close()


    @staticmethod
    def GetGroup( university, name):

        cursor = DataBase.getCursor()
        if cursor is None:
            return []
        try:
            query = """
            SELECT * FROM _group g
            JOIN Universities u ON g.university_id = u.university_id
            WHERE u.name = %s AND g.name LIKE %s
            """
            cursor.execute(query, (university, name))
            return cursor.fetchall()
        except Exception as e:
            print(f"Database error: {e}")
            return []
        finally:
            cursor.close()

    @staticmethod
    def GetGrops(university):
        cursor = DataBase.getCursor()
        if cursor is None:
            return []
```

```
        try:
            query = """
            SELECT g.name FROM _group g
            JOIN Universities u ON g.university_id = u.university_id
            WHERE u.name = %s
            """
            cursor.execute(query, (university, ))
            return cursor.fetchall()
        except Exception as e:
            print(f"Database error: {e}")
            return []
        finally:
            cursor.close()

    @staticmethod
    def GetTeachers(university):
        cursor = DataBase.getCursor()
        if cursor is None:
            return []
        try:
            query = """
            SELECT t.name FROM teachers t
            JOIN Universities u ON t.university_id = u.university_id
            WHERE u.name = %s
            """
            cursor.execute(query, (university, ))
            return cursor.fetchall()
        except Exception as e:
            print(f"Database error: {e}")
            return []
        finally:
            cursor.close()

    @staticmethod
    def GetClassrooms(university):
        cursor = DataBase.getCursor()
        if cursor is None:
            return []
        try:
            query = """
            SELECT c.room_number FROM classrooms c
            JOIN Universities u ON c.university_id = u.university_id
            WHERE u.name = %s
            """
            cursor.execute(query, (university, ))
            return cursor.fetchall()
        except Exception as e:
            print(f"Database error: {e}")
            return []
        finally:
            cursor.close()



    @staticmethod
    def GetScedualeByGroup(university, groupName):
        cursor = DataBase.getCursor()
        if cursor is None:
            return []
        try:
            query = """
            SELECT
            u.name AS university_name,
```

```
            g.name AS group_name,
            c.name AS course_name,
            t.name AS teacher_name,
            cl.room_number AS classroom,
            s.day_of_week,
            s.start_time,
            s.end_time
            FROM Schedules s
            JOIN _Group g ON s.group_id = g.group_id
            JOIN Courses c ON s.course_id = c.course_id
            JOIN Teachers t ON s.teacher_id = t.teacher_id
            JOIN Classrooms cl ON s.classroom_id = cl.classroom_id
            JOIN Universities u ON g.university_id = u.university_id
            WHERE u.name = %s
            AND g.name LIKE %s;
            """
            cursor.execute(query, (university, groupName))
            return cursor.fetchall()
        except Exception as e:
            print(f"Database error: {e}")
            return []
        finally:
            cursor.close()

    @staticmethod
    def GetScedualeByGroupAndDate(university, groupName, scheduleDate):
        cursor = DataBase.getCursor()
        if cursor is None:
            return []
        try:
            query = """
        SELECT
        u.name AS university_name,
        g.name AS group_name,
        c.name AS course_name,
        t.name AS teacher_name,
        cl.room_number AS classroom,
        s.day_of_week,
        s.start_time,
        s.end_time
        FROM Schedules s
        JOIN _Group g ON s.group_id = g.group_id
        JOIN Courses c ON s.course_id = c.course_id
        JOIN Teachers t ON s.teacher_id = t.teacher_id
        JOIN Classrooms cl ON s.classroom_id = cl.classroom_id
        JOIN Universities u ON g.university_id = u.university_id
        WHERE u.name = %s
        AND g.name LIKE %s
        AND s.schedule_date = %s;
        """
        # Assuming scheduleDate is passed as 'YYYY-MM-DD'
            cursor.execute(query, (university, f"%{groupName}%", scheduleDate))
            return cursor.fetchall()
        except Exception as e:
            print(f"Database error: {e}")
            return []
        finally:
            cursor.close()


    @staticmethod
    def GetScheduleByTeacher(university, teacherName):
        cursor = DataBase.getCursor()
```

```
        if cursor is None:
            return []
        try:
            query = """
            SELECT
            u.name AS university_name,
            g.name AS group_name,
            c.name AS course_name,
            t.name AS teacher_name,
            cl.room_number AS classroom,
            s.day_of_week,
            s.start_time,
            s.end_time
            FROM Schedules s
            JOIN _Group g ON s.group_id = g.group_id
            JOIN Courses c ON s.course_id = c.course_id
            JOIN Teachers t ON s.teacher_id = t.teacher_id
            JOIN Classrooms cl ON s.classroom_id = cl.classroom_id
            JOIN Universities u ON g.university_id = u.university_id
            WHERE u.name = %s
            AND t.name LIKE %s;
            """
            cursor.execute(query, (university, teacherName))
            return cursor.fetchall()
        except Exception as e:
            print(f"Database error: {e}")
            return []
        finally:
            cursor.close()

    @staticmethod
    def GetScheduleByTeacherAndDate(university, teacherName, scheduleDate):
        cursor = DataBase.getCursor()
        if cursor is None:
            return []
        try:
            query = """
            SELECT
        u.name AS university_name,
        g.name AS group_name,
        c.name AS course_name,
        t.name AS teacher_name,
        cl.room_number AS classroom,
        s.day_of_week,
        s.start_time,
        s.end_time
        FROM Schedules s
        JOIN _Group g ON s.group_id = g.group_id
        JOIN Courses c ON s.course_id = c.course_id
        JOIN Teachers t ON s.teacher_id = t.teacher_id
        JOIN Classrooms cl ON s.classroom_id = cl.classroom_id
        JOIN Universities u ON g.university_id = u.university_id
        WHERE u.name = %s
        AND t.name LIKE %s
        AND s.schedule_date = %s;
            """
            cursor.execute(query, (university, f"%{teacherName}%", scheduleDate))
            return cursor.fetchall()
        except Exception as e:
            print(f"Database error: {e}")
            return []
        finally:
            cursor.close()
```

```
    @staticmethod
    def GetScheduleByClassroom(university, calssroomNumber):
        cursor = DataBase.getCursor()
        if cursor is None:
            return []
        try:
            query = """
            SELECT
            u.name AS university_name,
            g.name AS group_name,
            c.name AS course_name,
            t.name AS teacher_name,
            cl.room_number AS classroom,
            s.day_of_week,
            s.start_time,
            s.end_time
            FROM Schedules s
            JOIN _Group g ON s.group_id = g.group_id
            JOIN Courses c ON s.course_id = c.course_id
            JOIN Teachers t ON s.teacher_id = t.teacher_id
            JOIN Classrooms cl ON s.classroom_id = cl.classroom_id
            JOIN Universities u ON g.university_id = u.university_id
            WHERE u.name = %s
            AND cl.room_number LIKE %s;
            """
            cursor.execute(query, (university, calssroomNumber))
            return cursor.fetchall()
        except Exception as e:
            print(f"Database error: {e}")
            return []
        finally:
            cursor.close()

    @staticmethod
    def GetScheduleByClassroomAndDate(university, calssroomNumber, scheduleDate):
        cursor = DataBase.getCursor()
        if cursor is None:
            return []
        try:
            query = """
             SELECT
        u.name AS university_name,
        g.name AS group_name,
        c.name AS course_name,
        t.name AS teacher_name,
        cl.room_number AS classroom,
        s.day_of_week,
        s.start_time,
        s.end_time
        FROM Schedules s
        JOIN _Group g ON s.group_id = g.group_id
        JOIN Courses c ON s.course_id = c.course_id
        JOIN Teachers t ON s.teacher_id = t.teacher_id
        JOIN Classrooms cl ON s.classroom_id = cl.classroom_id
        JOIN Universities u ON g.university_id = u.university_id
        WHERE u.name = %s
        AND cl.room_number LIKE %s
        AND s.schedule_date = %s;
            """
            cursor.execute(query, (university, f"%{calssroomNumber}%",
scheduleDate))
```

```
            return cursor.fetchall()
        except Exception as e:
            print(f"Database error: {e}")
            return []
        finally:
            cursor.close()


#Utilities
def format_timedelta(td):
    # Convert timedelta object into hours and minutes
    minutes, seconds = divmod(td.seconds + td.days * 86400, 60)
    hours, minutes = divmod(minutes, 60)
    # Format it as HH:MM
    return f"{hours:02d}:{minutes:02d}"


ScheduleDisplayPage.py
from PyQt5 import QtWidgets, QtGui, QtCore
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QPushButton, QLabel, QHBoxLayout,
QScrollArea
from PyQt5.QtCore import pyqtSignal, Qt
from PyQt5.QtGui import QIcon, QPixmap
from datetime import timedelta, datetime
from PyQt5.QtWidgets import QCalendarWidget, QDialog



from DataBase import DataBase


class ScheduleDisplayPage(QWidget):
    goBack = pyqtSignal(str)

    def __init__(self, parent=None):
        super().__init__(parent)
        self.layout = QVBoxLayout(self)

        self.current_date = datetime.now().date()
        self.type = None
        self.current_University = None
        self.current_selection = None

        # Navigation buttons
        self.nav_layout = QHBoxLayout()
        self.left_button = QPushButton("<")
        self.select_date_button = QPushButton("Select Date")
        self.right_button = QPushButton(">")

        self.label_header_style = """
        QLabel {
            font-size: 20px;
            color: #dddddd;
            padding: 5;
            margin: 2px;
        }
        """
        self.setStyleSheet("""
        QScrollBar:vertical, QScrollBar:horizontal {
            border: 2px solid grey;
            background: white;
            border-radius: 10px;
            width: 15px;
```

```
    border-radius: 15px;
 }

QScrollBar::handle:vertical, QScrollBar::handle:horizontal {
    background: #555555;  /* Lime Green color */
    min-height: 20px;  /* Minimum height for vertical scrollbar */
    min-width: 20px;  /* Minimum width for horizontal scrollbar */
    border-radius: 10px;
    }

QScrollBar::add-line:vertical, QScrollBar::add-line:horizontal {
    border: 2px solid grey;
    background: #555555;
    height: 20px;
    subcontrol-position: bottom;
    subcontrol-origin: margin;
    border-radius: 10px;
    }

QScrollBar::sub-line:vertical, QScrollBar::sub-line:horizontal {
    border: 2px solid grey;
    background: #555555;
    height: 20px;
    subcontrol-position: top;
    subcontrol-origin: margin;
    border-radius: 10px;
    }


""")
# Display header and information

formatted_date = self.current_date.strftime('%A, %B %d, %Y')
self.header_label = QLabel(f"Scedule for {formatted_date}.")
self.header_label.setStyleSheet(self.label_header_style)
self.layout.addWidget(self.header_label)



# Area to display schedule buttons
self.buttons_area = QVBoxLayout()
self.scroll_area = QScrollArea()
self.scroll_area.setWidgetResizable(True)
self.scroll_widget = QWidget()
self.scroll_widget.setLayout(self.buttons_area)
self.scroll_area.setWidget(self.scroll_widget)
self.layout.addWidget(self.scroll_area)

self.nav_layout.addWidget(self.left_button)
self.nav_layout.addWidget(self.select_date_button)
self.nav_layout.addWidget(self.right_button)
self.layout.addLayout(self.nav_layout)

# Back button
self.back_button = QPushButton("Back")
self.layout.addWidget(self.back_button)
self.back_button.clicked.connect(self.go_back)

self.left_button.clicked.connect(self.previous_day)
self.right_button.clicked.connect(self.next_day)
self.select_date_button.clicked.connect(self.show_calendar)

# Date control
```

```
    def load_schedule(self, university = "none", type = "none", selection = "none",
date = datetime.now().date()):

        if university != "none":
            self.type = type
            self.current_University = university
            self.current_selection = selection

        print(university, " | ", type, " | ", selection, " | ", date," | ",
self.current_University, " | ", self.type, " | ", self.current_selection, " | ",
date)
        if self.type ==  "group":
            schedule = DataBase.GetScedualeByGroupAndDate(self.current_University,
self.current_selection, date)
        elif self.type == "teacher":
            schedule =
DataBase.GetScheduleByTeacherAndDate(self.current_University,
self.current_selection, date)
        elif self.type == "classroom":
            schedule =
DataBase.GetScheduleByClassroomAndDate(self.current_University,
self.current_selection, date)

        for x in schedule:
            print(x)

        self.display_schedule(schedule)

    def display_schedule(self, schedule):
        formatted_date = self.current_date.strftime('%A, %B %d, %Y')
        self.header_label.setText(f"Scedule for {formatted_date}.")

        for i in reversed(range(self.buttons_area.count())):
            widget_to_remove = self.buttons_area.itemAt(i).widget()
            self.buttons_area.removeWidget(widget_to_remove)
            widget_to_remove.deleteLater()

        if not schedule:
            no_lessons_label = QLabel(f"There are no lessons for
{formatted_date}.")
            no_lessons_label.setStyleSheet(self.label_header_style)
            self.buttons_area.addWidget(no_lessons_label)

        for entry in schedule:
            _, _, subject, teacher, room, _, time_start, time_end = entry
            start_hours, start_minutes = divmod(time_start.seconds // 60, 60)
            end_hours, end_minutes = divmod(time_end.seconds // 60, 60)
            formatted_start = f"{start_hours:02}:{start_minutes:02}"
            formatted_end = f"{end_hours:02}:{end_minutes:02}"

            # Create the button
            schedule_button = QPushButton()
            schedule_button.setMinimumHeight(150)  # Ensure minimum height for
visibility

            button_layout = QVBoxLayout(schedule_button)

            # Create layouts and labels for time, subject, teacher, room
            for text, icon_path in [(f"{formatted_start} - {formatted_end}",
'icons/time_icon.png'),
                                    (subject, 'icons/book_icon.png'),
```

```
                                    (teacher, 'icons/teacher_icon.png'),
                                    (room, 'icons/classroom_icon.png')]:
                info_layout = QHBoxLayout()
                info_layout.setSpacing(2)  # Reduced spacing between icon and text

                icon_label = QLabel()
                icon = QPixmap(icon_path).scaled(20, 20, Qt.KeepAspectRatio,
Qt.SmoothTransformation)
                icon_label.setStyleSheet("background-color: #444444;")
                icon_label.setPixmap(icon)
                icon_label.setFixedSize(24, 24)  # Ensure icon size is fixed

                text_label = QLabel(text)
                text_label.setStyleSheet("font-size: 16px; background-color:
#444444;")  # Lighter grey background for text

                info_layout.addWidget(icon_label)
                info_layout.addWidget(text_label)
                info_layout.setAlignment(Qt.AlignLeft)  # Align the layout content
to the left

                button_layout.addLayout(info_layout)

            # Style the button for better visual effect
            schedule_button.setStyleSheet("""
                QPushButton {
                    border: none;
                    text-align: left;
                    padding: 10px;
                    border: 1px solid gray;
                    font-size: 16px;
                }
                QPushButton:hover {
                    background-color: #444444;
                }
                QPushButton:pressed {
                    background-color: #444444;
                }
            """)

            # Add button to the main layout
            self.buttons_area.addWidget(schedule_button)


    def show_calendar(self):

        self.calendar_stylesheet = """
        QCalendarWidget {
            color: #cccccc; /* Light grey text */
            background-color: #333333; /* Dark background */
            alternate-background-color: #444444;
            selection-background-color: #555555; /* Color for the selected date */
            selection-color: #444444; /* Text color for the selected date */
        }

        QCalendarWidget QAbstractItemView:enabled {
            font-size: 16px;
            color: #cccccc; /* Light grey text */
            background-color: #333333; /* Dark background */
            selection-background-color: #2a82da; /* Blue selection color */
            selection-color: #444444; /* White text for selected item */
        }
```

```
            QCalendarWidget QWidget#qt_calendar_navigationbar {
                background-color: #444444;
            }

            QCalendarWidget QToolButton {
                color: #cccccc; /* Light grey */
                background-color: #444444;
                border: none;
            }

            QCalendarWidget QToolButton:pressed {
                background-color: #555555;
            }

            QCalendarWidget QToolButton::menu-indicator {
                image: none;
            }
            """
            # Create a dialog
            self.dialog = QDialog(self)
            self.dialog.setWindowTitle("Select a Date")
            layout = QVBoxLayout(self.dialog)

            # Create and add a calendar widget
            self.calendar = QCalendarWidget()
            self.calendar.setGridVisible(True)
            self.calendar.setStyleSheet(self.calendar_stylesheet)
            layout.addWidget(self.calendar)

            # Add a button to confirm the date selection
            select_button = QPushButton("Select Date", self.dialog)
            select_button.clicked.connect(self.date_selected)
            layout.addWidget(select_button)

            # Show the dialog
            self.dialog.setLayout(layout)
            self.dialog.exec_()



    def date_selected(self):
            # Retrieve the selected date from the calendar
            selected_date = self.calendar.selectedDate()
            self.current_date = selected_date.toPyDate()
            self.load_schedule(date=self.current_date)
            self.dialog.close()  # Close the dialog after the date is selected



    def previous_day(self):
            self.current_date -= timedelta(days=1)
            self.load_schedule(date = self.current_date)

    def next_day(self):
            self.current_date += timedelta(days=1)
            self.load_schedule(date = self.current_date)

    def go_back(self):
            self.goBack.emit(self.type)
```

ScheduleSelectionPage.py

```
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QMainWindow, QSizePolicy,  QWidget,
QVBoxLayout, QPushButton, QComboBox, QLabel, QStackedWidget, QLineEdit,
QScrollArea, QGridLayout, QTableWidget, QTableWidgetItem
from PyQt5.QtCore import pyqtSignal
from PyQt5.QtGui import QIcon, QFont

from DataBase import DataBase


class ScheduleSelectionByGroupPage(QWidget):
    group_selected = pyqtSignal(str, str)
    goBack = pyqtSignal(str)
    def __init__(self, parent=None):
        super().__init__(parent)
        layout = QVBoxLayout(self)

        self.back_button = QPushButton("Back")

        label_search_style = """
        QLabel {
            font-size: 14px;
            color: #666666;
            padding: 5 0 5 0;
            margin: 2px;
        }
        """

        self.label = QLabel("Search for a Group:")
        self.label.setStyleSheet(label_search_style)
        layout.addWidget(self.label)

        # Search bar setup
        self.search_bar = QLineEdit()
        self.search_bar.setPlaceholderText("Type to search...")
        self.search_bar.setMinimumHeight(40)
        self.search_bar.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Fixed)  #
Ensure it expands horizontally
        self.search_bar.addAction(QIcon("icons/Search_Icon.png"),
QLineEdit.TrailingPosition)
        layout.addWidget(self.search_bar)

        self.scroll_area = QScrollArea(self)
        self.scroll_area.setWidgetResizable(True)
        self.scroll_widget = QWidget()
        self.scroll_area.setWidget(self.scroll_widget)
        self.buttons_layout = QVBoxLayout(self.scroll_widget)
        layout.addWidget(self.scroll_area)
        layout.addWidget(self.back_button)


    def showGroups(self, university_name):

        self.current_university = university_name

        self.all_groups = DataBase.GetGrops(self.current_university)
        self.display_buttons(self.all_groups)
        self.search_bar.textChanged.connect(self.filter_groups)
        self.back_button.clicked.connect(self.go_back)


    def display_buttons(self, group_list):
        self.clear_layout(self.buttons_layout)  # Clear existing content
```

```
        if not group_list:
            no_group_label = QLabel(f"No groups found.")
            no_group_label.setStyleSheet("font-size: 16px; color: #666;")
            self.buttons_layout.addWidget(no_group_label)
        else:
            for group in group_list:
                button = QPushButton(group[0])
                button.clicked.connect(lambda _, g=group[0]:
self.group_button_clicked(g))
                self.buttons_layout.addWidget(button)

    def clear_layout(self, layout):
        while layout.count():
            child = layout.takeAt(0)
            if child.widget():
                child.widget().deleteLater()

    def filter_groups(self, text):
        # Filter groups based on text input
        filtered_groups = [group for group in self.all_groups if text.lower() in
group[0].lower()]
        self.display_buttons(filtered_groups)

    def group_button_clicked(self, group_name):
        print(f"Selected: {group_name}")
        self.group_selected.emit("group", group_name)

    def go_back(self):
        self.goBack.emit(self.current_university)


class ScheduleSelectionByTeacherPage(QWidget):
    teacher_selected = pyqtSignal(str, str)
    goBack = pyqtSignal(str)
    def __init__(self, parent=None):
        super().__init__(parent)
        layout = QVBoxLayout(self)

        self.back_button = QPushButton("Back")

        label_search_style = """
        QLabel {
            font-size: 14px;
            color: #666666;
            padding: 5 0 5 0;
            margin: 2px;
        }
        """

        self.label = QLabel("Search for a Teacher:")
        self.label.setStyleSheet(label_search_style)
        layout.addWidget(self.label)
        # Search bar setup
        self.search_bar = QLineEdit()
        self.search_bar.setPlaceholderText("Type to search...")
        self.search_bar.setMinimumHeight(40)
        self.search_bar.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Fixed)  #
Ensure it expands horizontally
        self.search_bar.addAction(QIcon("icons/Search_Icon.png"),
QLineEdit.TrailingPosition)
        layout.addWidget(self.search_bar)
```

```
        # Scrollable area for buttons
        self.scroll_area = QScrollArea(self)
        self.scroll_area.setWidgetResizable(True)
        self.scroll_widget = QWidget()
        self.scroll_area.setWidget(self.scroll_widget)
        self.buttons_layout = QVBoxLayout(self.scroll_widget)
        layout.addWidget(self.scroll_area)
        layout.addWidget(self.back_button)


    def showTeachers(self, university_name):

        self.current_university = university_name
        self.all_teachers = DataBase.GetTeachers(university_name)

        self.display_buttons(self.all_teachers)
        self.search_bar.textChanged.connect(self.filter_teachers)
        self.back_button.clicked.connect(self.go_back)


    def display_buttons(self, teacher_list):
        self.clear_layout(self.buttons_layout)  # Clear existing content

        if not teacher_list:
            no_teacher_label = QLabel(f"No teachers found.")
            no_teacher_label.setStyleSheet("font-size: 16px; color: #666;")
            self.buttons_layout.addWidget(no_teacher_label)
        else:
            for teacher in teacher_list:
                button = QPushButton(teacher[0])
                button.clicked.connect(lambda _, t=teacher[0]:
self.teacher_button_clicked(t))
                self.buttons_layout.addWidget(button)

    def clear_layout(self, layout):
        while layout.count():
            child = layout.takeAt(0)
            if child.widget():
                child.widget().deleteLater()

    def filter_teachers(self, text):
        # Filter teachers based on text input
        filtered_teachers = [teacher for teacher in self.all_teachers if
text.lower() in teacher[0].lower()]
        self.display_buttons(filtered_teachers)

    def teacher_button_clicked(self, teacher_name):
        print(f"Selected Teacher: {teacher_name}")
        self.teacher_selected.emit("teacher", teacher_name)

    def go_back(self):
        self.goBack.emit(self.current_university)

class ScheduleSelectionByClassroomPage(QWidget):
    classroom_selected = pyqtSignal(str, str)
    goBack = pyqtSignal(str)
    def __init__(self, parent=None):
        super().__init__(parent)
        layout = QVBoxLayout(self)

        self.back_button = QPushButton("Back")

        label_search_style = """
```

```
        QLabel {
            font-size: 14px;
            color: #666666;
            padding: 5 0 5 0;
            margin: 2px;
        }
        """

        self.label = QLabel("Search for a Classroom:")
        self.label.setStyleSheet(label_search_style)
        layout.addWidget(self.label)

        self.search_bar = QLineEdit()
        self.search_bar.setPlaceholderText("Type to search...")
        self.search_bar.setMinimumHeight(40)
        self.search_bar.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Fixed)  #
Ensure it expands horizontally
        self.search_bar.addAction(QIcon("icons/Search_Icon.png"),
QLineEdit.TrailingPosition)
        layout.addWidget(self.search_bar)


        # Scrollable area for buttons
        self.scroll_area = QScrollArea(self)
        self.scroll_area.setWidgetResizable(True)
        self.scroll_widget = QWidget()
        self.scroll_area.setWidget(self.scroll_widget)
        self.buttons_layout = QVBoxLayout(self.scroll_widget)
        layout.addWidget(self.scroll_area)
        layout.addWidget(self.back_button)


    def showClassrooms(self, university_name):

        self.current_university = university_name
        self.all_classrooms = DataBase.GetClassrooms(university_name)
        self.display_buttons(self.all_classrooms)
        self.search_bar.textChanged.connect(self.filter_classrooms)
        self.back_button.clicked.connect(self.go_back)

    def display_buttons(self, classroom_list):
        self.clear_layout(self.buttons_layout)  # Clear existing content

        if not classroom_list:
            no_classroom_label = QLabel(f"No classrooms found.")
            no_classroom_label.setStyleSheet("font-size: 16px; color: #666;")
            self.buttons_layout.addWidget(no_classroom_label)
        else:
            for classroom in classroom_list:
                button = QPushButton(classroom[0])
                button.clicked.connect(lambda _, c=classroom[0]:
self.classroom_button_clicked(c))
                self.buttons_layout.addWidget(button)

    def clear_layout(self, layout):
        while layout.count():
            child = layout.takeAt(0)
            if child.widget():
                child.widget().deleteLater()


    def filter_classrooms(self, text):
        filtered_classrooms = [classroom for classroom in self.all_classrooms if
```

```
text.lower() in classroom[0].lower()]
        self.display_buttons(filtered_classrooms)

    def classroom_button_clicked(self, classroom_number):
        print(f"Selected: {classroom_number}")
        self.classroom_selected.emit("classroom", classroom_number)

    def go_back(self):
        self.goBack.emit(self.current_university)
```

UniversityPage.py

```
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QMainWindow, QSizePolicy,
QWidget,QHBoxLayout, QVBoxLayout, QPushButton, QComboBox, QLabel, QStackedWidget,
QLineEdit, QScrollArea, QGridLayout, QTableWidget, QTableWidgetItem
from PyQt5.QtCore import pyqtSignal, Qt
from DataBase import DataBase


class UniversityPage(QWidget):
    goBack = pyqtSignal()
    gotoScheduleSelectionPage = pyqtSignal(str)

    def __init__(self, parent=None):
        super().__init__(parent)

        self.mainlayout = QVBoxLayout(self)

        btn_layout = QVBoxLayout(self)
        label_layout = QVBoxLayout(self)

        label_style = """
        QLabel {
            font-size: 14px;
            color: #dddddd;
            padding: 2px;
            margin: 2px;
        }
        """
        label_header_style = """
        QLabel {
            font-size: 20px;
            color: #dddddd;
            padding: 15 5 15 5;
            margin: 2px;
        }
        """
        label_btn_style = """
        QLabel {
            font-size: 14px;
            color: #666666;
            padding: 5 px;
            margin: 2px;
        }
        """
        btn_style = """
        QPushButton {
            padding: 50px;
            margin: 2px;
        }
        """
        self.university_name_label = QLabel("")
```

```
        self.info = QLabel("Information: ")
        self.university_address_label = QLabel("Address: ")
        self.university_email_label = QLabel("Email: ")

        self.btn_type_label = QLabel("Search by: ")
        self.btn_type_label.setStyleSheet(label_btn_style)

        self.university_name_label.setStyleSheet(label_header_style)
        self.info.setStyleSheet(label_btn_style)
        self.university_address_label.setStyleSheet(label_style)
        self.university_email_label.setStyleSheet(label_style)



        label_layout.addWidget(self.university_name_label)
        label_layout.addWidget(self.info)
        label_layout.addWidget(self.university_address_label)
        label_layout.addWidget(self.university_email_label)




        self.group_button = QPushButton("Group")
        self.teacher_button = QPushButton("Teacher")
        self.classroom_button = QPushButton("Classroom")

        self.group_button.setStyleSheet(btn_style)
        self.teacher_button.setStyleSheet(btn_style)
        self.classroom_button.setStyleSheet(btn_style)


        btn_layout.addWidget(self.btn_type_label)
        btn_layout.addWidget(self.group_button)
        btn_layout.addWidget(self.teacher_button)
        btn_layout.addWidget(self.classroom_button)
        btn_layout.addStretch()


        back_layout = QVBoxLayout()
        self.back_button = QPushButton("Back")
        back_layout.addWidget(self.back_button)


        self.mainlayout.addLayout(label_layout)
        self.mainlayout.addStretch(1)
        self.mainlayout.addLayout(btn_layout)
        self.mainlayout.addStretch(1)
        self.mainlayout.addLayout(back_layout)


        self.group_button.clicked.connect(lambda:
self.goto_ScheduleSelectionPage("group"))
        self.teacher_button.clicked.connect(lambda:
self.goto_ScheduleSelectionPage("teacher"))
        self.classroom_button.clicked.connect(lambda:
self.goto_ScheduleSelectionPage("classroom"))
        self.back_button.clicked.connect(self.go_back)

    def showUniversityInfo(self, university_name):
        results = DataBase.GetUniversity(university_name)
        if results:
            university_info = results[0]
            if len(university_info) >= 4:
```

```
                _, name, address, email = university_info

                self.university_name_label.setText(f"{name}")
                self.university_address_label.setText(f"Address: {address}")
                self.university_email_label.setText(f"Email: {email}")
            else:
                self.university_name_label.setText("University Name: Format error")
                self.university_address_label.setText("Address: Format error")
                self.university_email_label.setText("Email: Format error")
        else:
            self.university_name_label.setText("University Name: Not found")
            self.university_address_label.setText("Address: Not available")
            self.university_email_label.setText("Email: Not available")

    def go_back(self):
        self.goBack.emit()

    def goto_ScheduleSelectionPage(self, schedule_type):
        self.gotoScheduleSelectionPage.emit(schedule_type)


UniversitySelectionPage.py
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QLineEdit, QPushButton,
QScrollArea, QLabel, QHBoxLayout, QSizePolicy
from PyQt5.QtCore import pyqtSignal, Qt
from PyQt5.QtGui import QIcon, QFont

from DataBase import DataBase

class UniversitySelectionPage(QWidget):
    university_selected = pyqtSignal(str)
    exitApplication = pyqtSignal()

    def __init__(self, parent=None):
        super().__init__(parent)

        layout = QVBoxLayout(self)

        self.title_label = QLabel("Universities")
        self.title_label.setAlignment(Qt.AlignLeft)
        self.title_label.setFont(QFont("Arial", 20, QFont.Bold))
        self.title_label.setStyleSheet("QLabel { padding-bottom: 10px; padding-top:
15px; }")
        layout.addWidget(self.title_label)

        self.search_bar = QLineEdit()
        self.search_bar.setPlaceholderText("Search Universities...")
        self.search_bar.setMinimumHeight(40)
        self.search_bar.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Fixed)  #
Ensure it expands horizontally
        self.search_bar.addAction(QIcon("icons/Search_Icon.png"),
QLineEdit.TrailingPosition)
        layout.addWidget(self.search_bar)

        self.scroll_area = QScrollArea()
        self.scroll_area.setWidgetResizable(True)
        self.scroll_area_widget_contents = QWidget()
        self.scroll_area.setWidget(self.scroll_area_widget_contents)

        self.buttons_layout = QVBoxLayout(self.scroll_area_widget_contents)
        self.scroll_area.setSizePolicy(QSizePolicy.Expanding,
QSizePolicy.Expanding)  # Ensuring scroll area expands
```

```
        layout.addWidget(self.scroll_area)

        temp = DataBase.GetUniversities()
        self.universities = temp

        self.current_page = 0
        self.items_per_page = 5
        self.display_buttons(self.universities)

        self.search_bar.textChanged.connect(self.filter_universities)

        nav_layout = QHBoxLayout()
        self.exit_btn = QPushButton("Exit")
        nav_layout.addWidget(self.exit_btn)
        self.exit_btn.setStyleSheet("""
            QPushButton {
                background-color: #333333;
                color: #D55;  /* Dark red, not too bright */
                border:none;
                padding: 5px;
                border-radius: 10px;
                font-size: 16px;
                font-weight: bold;
            }
            QPushButton:hover {
                background-color: #555555;
                border-color: #777777;
            }
            QPushButton:pressed {
                background-color: #222222;
                border-color: #aaaaaa;
            }
        """)
        self.exit_btn.pressed.connect(self.exit_application)
        layout.addLayout(nav_layout)

    def display_buttons(self, university_list):
        while self.buttons_layout.count():
            button = self.buttons_layout.takeAt(0).widget()
            if button:
                button.deleteLater()



        start_index = self.current_page * self.items_per_page
        end_index = min(start_index + self.items_per_page, len(university_list))

        for university in university_list[start_index:end_index]:
            btn = QPushButton(university[0])
            btn.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)  #
Ensure buttons expand
            btn.clicked.connect(lambda _, uni=university[0]:
self.select_university(uni))
            self.buttons_layout.addWidget(btn)

    def filter_universities(self, text):
        filtered_universities = [university for university in self.universities if
text.lower() in university[0].lower()]

        # Clear the current contents of the layout before displaying new results or
message
        self.clear_layout(self.buttons_layout)
```

```
        if not filtered_universities:
            # If no universities are found, display a message
            no_universities_label = QLabel(f"There is no university named
'{text}'.")
            no_universities_label.setAlignment(Qt.AlignCenter)
            no_universities_label.setStyleSheet("font-size: 16px; color: #555;")
            self.buttons_layout.addWidget(no_universities_label)
        else:
            # Otherwise, display the buttons for the filtered universities
            self.display_buttons(filtered_universities)

    def clear_layout(self, layout):
        while layout.count():
            item = layout.takeAt(0)
            widget = item.widget()
            if widget:
                widget.deleteLater()


    def select_university(self, university_name):
        print(f"Selected: {university_name}")
        self.university_selected.emit(university_name)

    def exit_application(self):
        self.exitApplication.emit()



UserInterface.py
import sys

from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QMainWindow, QStackedWidget, QLabel

from Interface.UniversitySelectionPage import UniversitySelectionPage
from Interface.UniversityPage import UniversityPage
from Interface.SceduleSelectionPage import ScheduleSelectionByClassroomPage,
ScheduleSelectionByGroupPage, ScheduleSelectionByTeacherPage
from Interface.SceduleDisplayPage import ScheduleDisplayPage

from DataBase import DataBase

class UserInterface(QMainWindow):
    def __init__(self, app):
        super().__init__()
        self.application = app

        self.setWindowTitle('University Schedule App')
        self.setGeometry(50, 50, 500, 900)

        self.stacked_widget = QStackedWidget(self)
        self.setCentralWidget(self.stacked_widget)

        self.university_selection_page = UniversitySelectionPage()
        self.University_page = UniversityPage()

        self.schedule_group_page = ScheduleSelectionByGroupPage()
        self.schedule_teacher_page = ScheduleSelectionByTeacherPage()
        self.schedule_classroom_page = ScheduleSelectionByClassroomPage()
        self.search_results_page = ScheduleDisplayPage()

        self.stacked_widget.addWidget(self.university_selection_page)
        self.stacked_widget.addWidget(self.University_page)
```

```
        self.stacked_widget.addWidget(self.schedule_group_page)
        self.stacked_widget.addWidget(self.schedule_teacher_page)
        self.stacked_widget.addWidget(self.schedule_classroom_page)

        self.stacked_widget.addWidget(self.search_results_page)


self.university_selection_page.university_selected.connect(self.gotoUnivesityPage)
        self.university_selection_page.exitApplication.connect(self.shutDown)


self.University_page.gotoScheduleSelectionPage.connect(self.gotoScheduleSelectionPa
ge)

        self.University_page.goBack.connect(self.gotoUniversitySelectionPage)


self.schedule_group_page.group_selected.connect(self.gotoScheduleDisplayPage)

self.schedule_teacher_page.teacher_selected.connect(self.gotoScheduleDisplayPage)

self.schedule_classroom_page.classroom_selected.connect(self.gotoScheduleDisplayPag
e)

        self.schedule_group_page.goBack.connect(self.gotoUnivesityPage)
        self.schedule_teacher_page.goBack.connect(self.gotoUnivesityPage)
        self.schedule_classroom_page.goBack.connect(self.gotoUnivesityPage)

        self.search_results_page.goBack.connect(self.gotoScheduleSelectionPage)

    def gotoUniversitySelectionPage(self):
        self.stacked_widget.setCurrentWidget(self.university_selection_page)

    def gotoUnivesityPage(self, university_name):
        self.current_university = university_name
        self.stacked_widget.setCurrentWidget(self.University_page)
        self.University_page.showUniversityInfo(self.current_university)

    def gotoScheduleSelectionPage(self, type):
        if type == "group":
            self.stacked_widget.setCurrentWidget(self.schedule_group_page)
            self.schedule_group_page.showGroups(self.current_university)
        elif type == "teacher":
            self.stacked_widget.setCurrentWidget(self.schedule_teacher_page)
            self.schedule_teacher_page.showTeachers(self.current_university)
        elif type == "classroom":
            self.stacked_widget.setCurrentWidget(self.schedule_classroom_page)
            self.schedule_classroom_page.showClassrooms(self.current_university)
        else:
            print("Error")

    def gotoScheduleDisplayPage(self, type, selection):
        self.stacked_widget.setCurrentWidget(self.search_results_page)
        self.search_results_page.load_schedule(self.current_university, type,
selection)

    def shutDown(self):
        sys.exit(self.application.exec_())
```

```
def RunUI():
    app = QApplication(sys.argv)
    app.setStyleSheet("""
    QWidget {
        background-color: #222222;
        color: #dddddd;
    }
    QScrollArea{
        border: none;
    }
    QPushButton {
        border: none;
        border-radius: 10px;
        background-color: #444444;
        padding: 10px;
        font-size: 14px;
    }
    QPushButton:hover {
         background-color: #555555;
    }
    QPushButton:pressed {
        background-color: #666666;
    }
    QLineEdit {
        border: none;
        border-radius: 10px;
        background-color: #444444;
        padding: 5px 15px;
        font-size: 16px;
        }
    QLineEdit:hover {
        background-color: #555555;
    }
    """)
    window = UserInterface(app)
    window.show()
    sys.exit(app.exec_())
```