**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

### ЗВІТ

До лабораторної роботи № 1

**З дисципліни:** *“Архітектура та проєктування програмного забезпечення”*

**На тему:** *“Проєктування компонентів архітектури програмного забезпечення та їх взаємодії”*

**Лектор:**

асист. каф. ПЗ

Луцик І.І.

**Виконав:**

ст. гр. ПЗ-45

Хруставчук М.Л.

**Прийняв:**

асист. каф. ПЗ

МалийР.М.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2025 р.

∑= \_\_\_\_\_ .

Львів – 2025

**Тема роботи:** Проєктування компонентів архітектури програмного забезпечення та їх взаємодії.

**Мета роботи:** Ознайомлення з процесом проєктування архітектури програмного забезпечення, формалізацією вимог до системи та використанням UML-діаграм (діаграми компонентів, взаємодій, розгортання) для моделювання архітектури ПЗ.

**TЕОРЕТИЧНІ ВІДОМОСТІ**

Архітектура програмного забезпечення є фундаментом у процесі створення інформаційних систем. Вона визначає загальну структуру системи, її основні компоненти, способи взаємодії між ними, а також забезпечує виконання функціональних і нефункціональних вимог, таких як масштабованість, безпека, надійність і супроводжуваність. Архітектурне проєктування виступає проміжною ланкою між аналізом вимог і реалізацією коду, формуючи цілісне бачення майбутнього застосунку.

Компонент архітектури – це незалежний програмний модуль із чітко визначеною відповідальністю, інтерфейсами та залежностями. Компонентний підхід спрощує супровід системи, повторне використання коду та розподіл завдань між командами. Правильна структуризація компонентів дозволяє легко вносити зміни й масштабувати систему, не порушуючи її цілісність.

Вибір архітектурного стилю залежить від вимог предметної області. Найпоширенішими є:

* Багатошарова архітектура (Layered) – розподіляє систему на рівні представлення, бізнес-логіки та доступу до даних.
* Клієнт–серверна – забезпечує взаємодію між клієнтським застосунком і сервером, який обробляє запити.
* Мікросервісна – система складається з невеликих незалежних сервісів, що взаємодіють через REST або message broker.
* Подієво-орієнтована (Event-driven) – обмін повідомленнями через події, що підвищує асинхронність і продуктивність.

Для візуалізації архітектури використовують UML-діаграми, які дозволяють описати як статичну структуру, так і динамічну поведінку системи. Найважливіші з них:

* Діаграма компонентів – показує взаємозв’язки між основними модулями системи;
* Діаграма взаємодії (sequence) – демонструє порядок обміну повідомленнями між об’єктами;
* Діаграма розгортання (deployment) – відображає фізичне розташування компонентів у середовищі виконання.

Метою архітектурного проєктування є створення логічної моделі системи, що забезпечує цілісність, узгодженість і відповідність технічним та бізнес-вимогам. Використання UML дозволяє стандартизовано документувати рішення, спрощувати комунікацію між розробниками та підтримувати систему протягом усього життєвого циклу.

**ЗАВДАННЯ**

1. Проаналізувати предметну область у відповідності до обраного варіанту, визначити основні групи користувачів, випадки використання та основні компоненти програмного забезпечення.
2. Визначити архітектурний стиль та спроєктувати основні рівні взаємодії.
3. Описати компоненти ПЗ.
4. Побудувати UML-діаграми: компонентів, взаємодії, розгортання.
5. Обґрунтувати вибір архітектури й шаблонів проєктування.

Предметна область – “Мобільний застосунок для логопедичних занять”.

**ХІД ВИКОНАННЯ**

**1. Проаналізувати предметну область, визначити групи користувачів, випадки використання та компоненти ПЗ.**

*Аналіз предметної області.*

Предметна область охоплює логопедичну терапію для дітей із порушеннями мовлення, а саме:

* Постановку, автоматизацію та контроль правильності вимови звуків.
* Формування звичок правильної артикуляції.
* Мотивацію дитини до регулярних занять через ігрові механіки.
* Взаємодію логопеда і дитини в єдиній системі.

Система спрощує роботу логопеда, робить процес тренувань більш приємним для дитини, а також застосовується модуль штучного інтелекту, який оцінює правильність вимови та допомагає відстежувати прогрес учня.

*Основні групи користувачів.*

* Дитина – виконує артикуляційні вправи, повторює звуки, отримує підказки та візуальний зворотний зв’язок.
* Логопед – створює індивідуальні плани занять, перевіряє результати, додає нові вправи та аналізує прогрес.

*Основні функціональні сценарії (випадки використання).*

* Реєстрація та авторизація користувачів.
* Виконання вправ учнем і надсилання результатів.
* Автоматичний аналіз вимови за допомогою AI-модуля.
* Отримання оцінки й підказок для покращення вимови.
* Створення та редагування вправ логопедом.
* Перегляд статистики занять і прогресу учнів.
* Адміністративне керування системою.

*Основні компоненти програмного забезпечення:* Mobile Client (React Native), User Service, Exercise Service, Speech AI Service, Progress Service, API Gateway, Database.

**2. Визначити архітектурний стиль та спроєктувати рівні взаємодії.**

Для системи обрано мікросервісну клієнт–серверну архітектуру. Вона дозволяє розподілити логіку між незалежними сервісами, що спрощує масштабування, підтримку та подальший розвиток проєкту.

Система складається з таких основних рівнів:

* 1. Клієнтський рівень – мобільний застосунок на React Native, який надає інтерфейс користувача та взаємодіє із сервером через REST API.
  2. Рівень API Gateway – приймає всі запити від клієнта, виконує маршрутизацію, автентифікацію та передає дані у відповідний мікросервіс.
  3. Рівень мікросервісів – набір незалежних сервісів (User, Exercise, Speech AI, Progress), кожен із яких реалізує певну бізнес-функцію системи. Комунікація між ними здійснюється через HTTP.
  4. Рівень даних – кожен мікросервіс має власну базу даних (UserDB, ExerciseDB, ProgressDB), що реалізує принцип *Database per Service*.

Такий підхід дозволяє створити масштабовану систему з чітким розділенням відповідальностей між компонентами, що полегшує її розробку та супровід.

**3. Описати компоненти ПЗ.**

* 1. Mobile Client (React Native) – реалізує інтерфейс для учня та логопеда, забезпечує реєстрацію, авторизацію, виконання вправ і перегляд результатів. Клієнт взаємодіє з бекендом через REST API та отримує дані у форматі JSON.
  2. User Service – обробляє запити реєстрації, автентифікації та керування обліковими записами. Забезпечує роботу з JWT-токенами та визначення ролей користувачів.
  3. Exercise Service – надає можливість створювати, редагувати та видаляти вправи. Забезпечує доступ логопеда до методичних матеріалів і планів занять.
  4. Speech AI Service – приймає аудіофайли від клієнта, аналізує їх і повертає оцінку правильності вимови.
  5. Progress Service – відповідає за збереження результатів занять та прогресу учнів.
  6. API Gateway – приймає всі зовнішні запити, виконує перевірку авторизації, маршрутизує їх до відповідних сервісів (User, Exercise, Speech AI або Progress) та повертає узгоджену відповідь клієнту.
  7. Database – єдина база даних, у якій зберігаються всі основні сутності системи.

**4. Побудувати UML-діаграми:**

На діаграмі компонентів (рис. 1) показано основні складові системи та взаємозв’язки між ними. Мобільний клієнт взаємодіє з API Gateway, який маршрутизує запити до окремих мікросервісів: User, Exercise, Speech AI та Progress. Усі сервіси працюють із єдиною базою даних, де зберігається інформація про користувачів, вправи та результати.

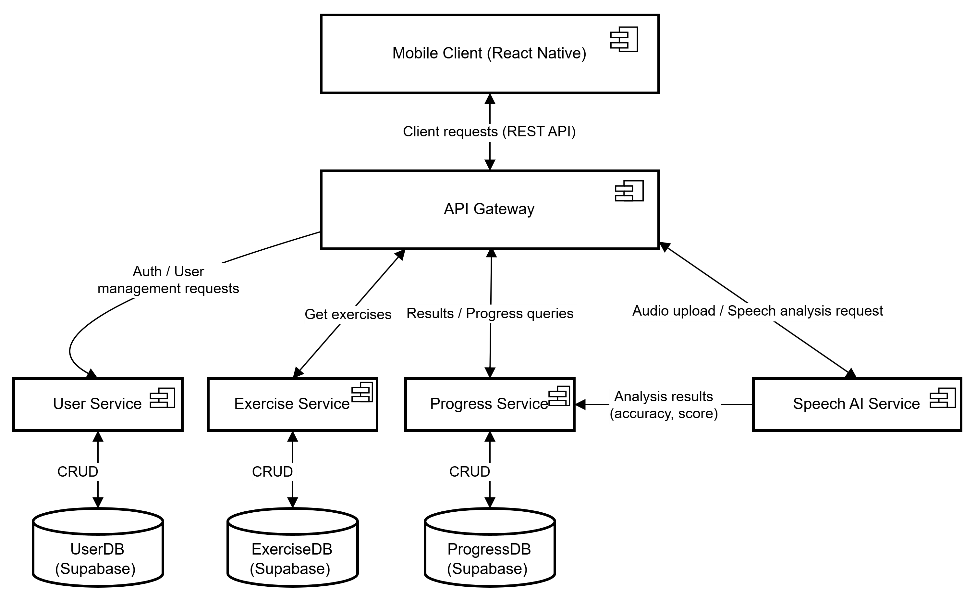


Рис. 1. Діаграма компонентів

Діаграма взаємодії (рис. 2) відображає послідовність обміну даними під час виконання вправи. Учень авторизується в системі, отримує завдання, надсилає аудіо для аналізу, після чого сервіс розпізнавання мовлення оцінює правильність вимови, а Progress Service зберігає результат. Логопед має можливість переглядати статистику та звіти про прогрес учнів.

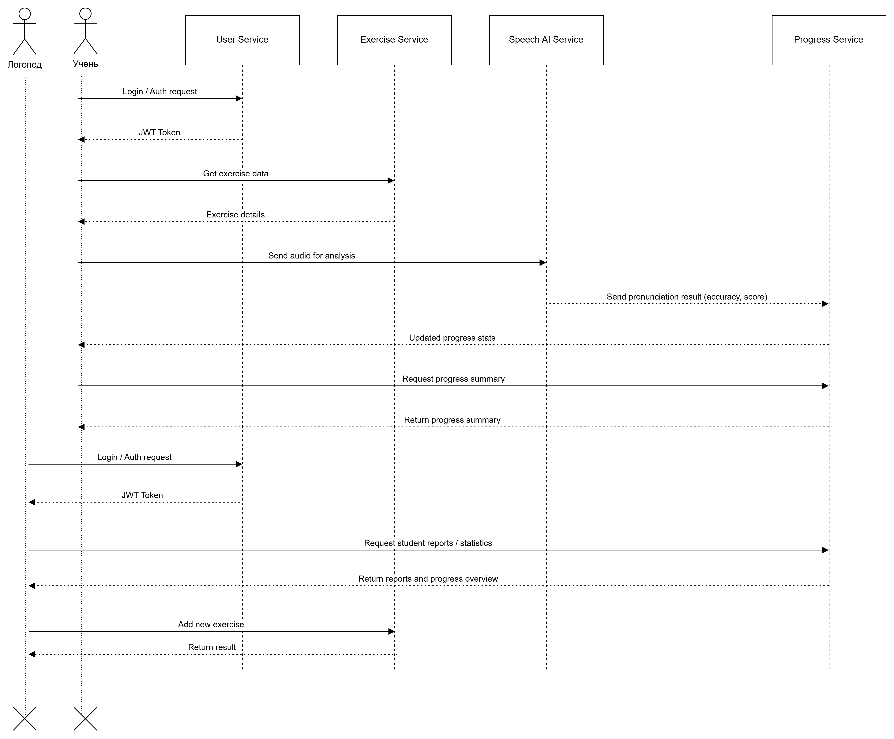


Рис. 2. Діаграма взаємодії

На діаграмі розгортання (рис. 3) показано розміщення компонентів системи у середовищі виконання. Мобільний клієнт на React Native взаємодіє з серверною частиною, що включає API Gateway та набір мікросервісів на ASP.NET Core. Для аналізу вимови використовується окремий AI-сервіс, а кожен сервіс має окрему базу даних, розгорнуту на Supabase (PostgreSQL), що забезпечує ізоляцію та незалежність компонентів.

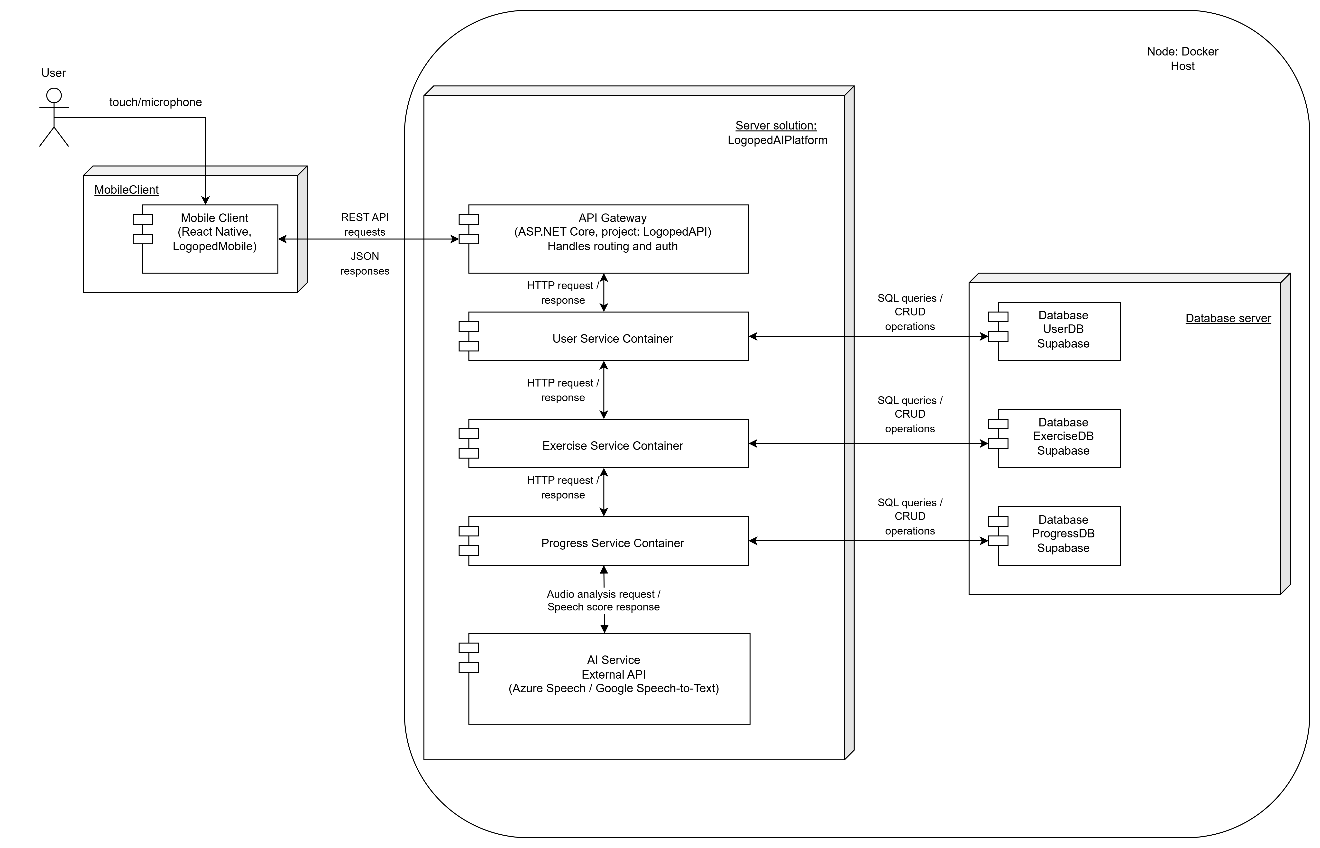


Рис. 3. Діаграма розгортання

**5. Обґрунтувати вибір архітектури й шаблонів проєктування**

Обрана мікросервісна архітектура забезпечує гнучкість, модульність і простоту масштабування. Кожен сервіс виконує одну бізнес-функцію, що полегшує оновлення та подальший розвиток системи.

Основними шаблонами проєктування виступають:

* Repository – для розділення бізнес-логіки від доступу до даних у кожному сервісі.
* Dependency Injection – для зручного керування залежностями у .NET-сервісах.
* API Gateway Pattern – для централізованої маршрутизації запитів і захищеного доступу до внутрішніх сервісів.
* Adapter – для взаємодії з зовнішнім сервісом розпізнавання мовлення.

Обрана структура забезпечує зрозумілу логіку, ізольованість модулів і стабільну роботу системи з можливістю подальшого розширення функціональності.

**ВИСНОВКИ**

У ході виконання лабораторної роботи було проаналізовано предметну область, визначено основні групи користувачів, сценарії використання та розроблено архітектуру програмного забезпечення. Для проєкту обрано мікросервісну архітектуру з клієнт–серверною взаємодією, що забезпечує масштабованість, гнучкість та незалежність окремих компонентів.

Було спроєктовано основні сервіси системи: Mobile Client, User Service, Exercise Service, Speech AI Service, Vision AI Service та Progress Service. Для кожного визначено функціональні обов’язки та взаємодію між ними. Побудовано UML-діаграми компонентів, взаємодії (послідовності) та розгортання, які відображають логічну структуру, порядок обміну повідомленнями та фізичне розміщення елементів системи.

Отримані результати демонструють узгодженість архітектури з вимогами до сучасних розподілених систем. Запропонована структура дозволяє легко додавати нові сервіси, змінювати існуючі модулі без порушення роботи інших частин системи, а також забезпечує основу для подальшої реалізації та контейнеризації застосунку у Docker.