Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

Архітектура інформаційної системи автентифікації людини

Студента 2 курсу, 2м групи,
спеціальності 121 «Інженерія
програмного забезпечення»
спеціалізації «Інженерія
програмного забезпечення» _____ Сімаченко Андрія Олександровича
підпис студента

Науковий керівник кандидат
педагогічних наук, доцент
кафедри інженерії програмного
забезпечення та кібербезпеки _____ Жирова Тетяна Олександрівна
підпис керівника

Гарант освітньої програми доктор
економічних наук, професор
кафедри інженерії програмного
забезпечення такібербезпеки _____ Токар Володимир Володимирович
підпис керівника

КИЇВ – 2021

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«10» листопада 2020 р.

Завдання
на випускний кваліфікаційний проєкт студентові
Сімаченку Андрію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проєкту Архітектура інформаційної системи автентифікації людини

Затверджена наказом ректора від «30» листопада 2020 р. № 3224

2. Строк здачі студентом закінченого проєкту 25 листопада 2021 р.

3. Цільова установка та вихідні дані до проєкту

Мета проєкту вирішити проблему побудови гнучкої та універсальної архітектури системи авторизації людини з використанням керування голосом

Об'єкт дослідження основний об'єкт це архітектура програмного забезпечення, також розглядається автентифікація та авторизація, розпізнавання голосу та мовника

Предмет дослідження архітектура серверу автентифікації та авторизації з можливістю використанням голосу в якості паролю

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проекту (перелік питань за кожним розділом):

ВСТУП

РОЗДІЛ 1. ІНФОРМАЦІЙНЕ ДОСЛІДЖЕННЯ ПРОЦЕСУ РОЗРОБКИ АРХІТЕКТУРІ СИСТЕМИ АВТЕНТИФІКАЦІЇ

1.1. Наукова складова, якою забезпечується поняття архітектури розроблюваної системи

1.2. Технічний опис та нормативно-правова база процедур автентифікації ідентифікації.

1.3. Регуляція процесу доступу до даних автентифікованого користувача

1.4. Висновок до розділу 1

РОЗДІЛ 2. СПОСОБИ ВИРІШЕННЯ ЗАВДАНЬ, ТЕХНОЛОГІЇ ВИКОРИСТАНІ У ПРОЦЕСІ РОЗРОБКИ

2.1. Протокол автентифікації та авторизації

2.2. Удосконалення протоколу автентифікації

2.3. Інтеграція модуля голосової автентифікації

2.4. Розроблення процесу автентифікації та визначення необхідних модулів

2.5. Висновок до розділу 2

РОЗДІЛ 3. СПЕЦИФІКАЦІЯ ВИМОГ ДО СИСТЕМИ УПРАВЛІННЯ ЕЛЕКТРОПРИЛАДАМИ ОФІСУ

3.1. Вимоги до архітектури

3.2. Нефункціональні вимоги

3.3. Функціональні вимоги

3.4. Системні вимоги

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проєкту	Строк виконання етапів проєкту	
		за планом	фактично
1.	*Вибір теми випускного кваліфікаційного проєкту*	*21.09.2020*	*21.09.2020*
2.	*Розробка та затвердження завдання на проєкт магістра (стац/заоч)*	*06.11.2020*	*22.12.2020*
3.	*Вступ та перелік літературних джерел*	*27.02.2021*	*27.02.2021*
4.	*Розробка технічного завдання*	*20.03.2021*	*20.03.2021*
5.	*Розділ 1. Інформаційне дослідження процесу розробки архітектурі системи автентифікації*	*16.04.2021*	*16.04.2021*
6.	*Розділ 2. Способи вирішення завдань, технології використані у процесі розробки*	*24.05.2021*	*24.05.2021*
7.	*Розділ 3. Специфікація вимог до системи управління електроприладами офісу*	*21.06.2021*	*21.06.2021*
8.	*Розділ 4. Проектування та розробка енергозберігаючої системи управління електроприладами офісу*	*20.09.2021*	*20.09.2021*
9.	*Розробка програми та методики тестування*	*18.10.2021*	*18.10.2021*
10.	*Написання наукової статті*	*22.05.2021*	*22.05.2021*
11.	*Керівництво користувача*	*21.10.2021*	*21.10.2021*
12.	*Висновки та пропозиції*	*01.11.2021*	*01.11.2021*
13.	*Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)*	*03.11.2021*	*03.11.2021*
14.	*Підготовка автореферату та презентації доповіді*	*03.11.2021*	*03.11.2021*
15.	*Попередній захист випускного кваліфікаційного проєкту*	*22.11.2021 – 25.11.2021*	*22.11.2021*
16.	*Здача зброшурованої випускного кваліфікаційного проєкту*	*25.11.2021*	*25.11.2021*
17.	*Зовнішнє рецензування випускного кваліфікаційного проєкту*	*26.11.2021*	*26.11.2021*
18.	*Підготовка до публічного захисту випускного кваліфікаційного проєкту*		

7. Дата видачі завдання «10» листопада 2020 р.

8. Науковий керівник випускного кваліфікаційного проєкту Жирова Т.О.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Токар В.В.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Сімаченко А.О.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проекту

Науковий керівник випускного кваліфікаційного проєкту
Жирова Т.О.
(підпис, дата)

Відмітка про попередній захист _____
(ПІБ, підпис, дата)

1. Висновок про випускний кваліфікаційний проєкт

Випускний кваліфікаційний проєкт студента Сімаченко А.О.
(прізвище, ініціали)
може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми Токар В.В.
(прізвище, ініціали, підпис)
Завідувач кафедри Криворучко О. В.
(підпис, прізвище, ініціали)
«___»_____20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження робота присвячена проектуванню архітектури інформаційної системи автентифікації людини, з урахуванням сучасних способів взаємодії людини з системою, таких як голос або жести.

В результаті аналізу наукової бази було сформовано підхід по проектування системи, та визначено вимоги до вихідного продукту

Розробка прототипу була виконана на мові C# клієнтська частина на платформі Xamarin.Forms, серверна частина на платформі .NET 5.

Ключові слова: архітектура програмного забезпечення, автентифікація, авторизація, протоколи авторизації, голосова автентифікація, мікросервіси, сервер авторизації

ABSTRACT

According to the purpose, the work is devoted to designing the architecture of the information system of authentication of human authentication, taking into account modern ways of human interaction with the system, such as voice or gestures.

As a result of the analysis of the scientific base, an approach to system design was formed, and the requirements for the source product were determined

Prototype development was performed in C # client part on Xamarin.Forms platform, server part on .NET 5 platform.

Keywords: software architecture, authentication, authorization, authorization protocols, voice authentication, microservices, authorization server

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АПЗ – архітектура програмного забезпечення

БД – база даних

СУБД – система управління базами даних

EF Core – Entity Framework Core

DI – dependency injection

UML – unified modeling language

SOLID – Single responsibility principle, Open-closed principle, Liskov substitution principle, Interface segregation principle, Dependency inversion principle

API – application programming interface

HTTP – hypertext transfer protocol

URI – uniform resource identifier

Html – hypertext markup language

DOM – document object model

2FA – (two factor authorization) дво факторна авторизація

ООП – об’єктно орієнтоване програмування

ЗМІСТ

Зм	Аркуш	№ докум.	Підпис	Дата	КНТЕУ 121-02-19.МР		
					Архітектура інформаційної системи автентифікації людини	Стадія	Аркуш	Аркушів
Зав. каф		Криворучко О.В.		22.12.20		З	3	63
Керівник		Жирова Т.О.		22.12.20				
Гарант		Токар В.В,		22.12.20	Зміст	Факультет інформаційних технологій, 2м курс,2мгрупа		
Розроб		Сімаченко А.О.		22.12.20				

ВСТУП

Аналізуючи поширену цифрову продукцію останніх років, можна помітити, що з часом кількість можливих інтерфейсів взаємодії людини з цифровою системою стає все більше, концепція єдиної точки доступу до всього програє концепції інтернету речей. З'явилися пристрої особистого та широкого використання з нестандартними способами управління: управлінням голосом, поглядом, жестами, механічно, не тільки за допомогою екранів або контролерів вводу, таких як клавіатура чи мишка а і за допомогою гіроскопу шляхом зміни положення пристрою в просторі.

Хоча кількість інструментів та можливостей для розробки цифрових систем зросла необхідність керувати доступом до даних та функцій не зникла а з кожним разом постає все більш гостро. Так наприклад система розумного дому, з функцією голосового асистента, може бути активована малолітньою дитиною, що може не усвідомлювати наслідків своїх команд, таких як управління температурою опалення чи рівнем кондиціонування приміщення, керування небезпечними побутовими приладами чи відключення сигналізації. Щоб уникнути ризиків пов'язаних з небажаним або несанкціонованим використанням того чи іншого програмного продукту, важливо мати інструменти автентифікації та авторизації користувачів навіть на особистих чи побутових програмних системах.

Але широковживані системи та протоколи автентифікації через паролі цифрові ключі, і т.д., хоч і в достатньому степені надійні, не можуть бути практично використані в програмних системах з нетривіальними методами взаємодії з користувачем які були перелічені вище. З таких причин:

					КНТЕУ 121-02-19.МР		
Зм	*Аркуш*	*№ докум.*	*Підпис*	*Дата*	Архітектура інформаційної системи автентифікації людини	*Стадія*	*Аркуш*	*Аркушів*
01.11.21						В	5	63
Зав. каф		Криворучко О.В.		27.02.21				
Керівник		Жирова Т.О.		27.02.21				
Гарант		Токар В.В,		27.02.21				
Розроб		Сімаченко А.О.		27.02.21	Вступ	Факультет інформаційних технологій, 2м курс,2мгрупа		

- фізична архітектура таких пристроїв часто побудована для забезпечення високої енергоефективності жертвуючи продуктивністю, в таких умовах складні механізми автентифікації та авторизації будуть витрачати надто багато ресурсів пристрою;
- неможливо або небезпечно вводити пароль голосом або жестом, те ж саме справедливо і для цифрових ключів;
- небезпечно використовувати авторизовані сесії такі як в класичних інформаційних системах, так як інша персона в зоні дії голосового асистента або датчика жестів зможе керувати системою від імені авторизованого користувача.

Потрібно врахувати, що програмні продукти не просто є способом отримати інформацію або послугу, вони можуть як зберігати та використовувати особисті данні користувачів, так і мають велику цінність та захищені законом. Тому в таких інформаційних системах варто впроваджувати механізми автентифікації та авторизації.

Також впровадження таких механізмів дозволяє забезпечити підтримку таких інформаційних процесів

- Відокремлення трафіку різних користувачів
- Групування запитів за рівнями доступу та можливість аналізувати чутко структуровані дані генеровані операторами системи.
- Забезпечення захисту від зловмисників, що намагаються вкрасти інформацію або нанести шкоду системі чи користувачам.

Тому я вважаю актуальним розглянути проблеми розробки архітектури інформаційних систем авторизації та автентифікації в програмних продуктах де традиційні методи використані бути не можуть, на прикладі системи голосового асистента.

Мета проєкту: вирішити проблему проектування гнучкої та сучасної системи автентифікації людини з використанням нестандартних ознак для ідентифікації, наприклад таких як голос.

Об'єкт дослідження: основний об'єкт це архітектура програмного забезпечення, також розглядається автентифікація та авторизація, розпізнавання голосу та мовника, біометрична автентифікація, нейронні мережі.

Предмет дослідження: архітектура інформаційної системи автентифікації та авторизації з можливістю використанням голосу в якості ознаки верифікації

РОЗДІЛ 1
ІНФОРМАЦІЙНЕ ДОСЛІДЖЕННЯ ПРОЦЕСУ РОЗРОБКИ АРХІТЕКТУРИ СИСТЕМИ АВТЕНТИФІКАЦІЇ

1.1. Наукова складова, якою забезпечується поняття архітектури розроблюваної системи

Перед початком розробки технічної складової потрібно дослідити інформаційні процеси, що забезпечують створення програмного продукту. Почнемо з поняття **архітектури програмного забезпечення** (далі - АПЗ)**.**

За версією Мартіна Фавлера *[6]* - «Архітектура визначається, як концепція системи найвищого рівня у своєму середовищі. Архітектура програмної системи (у певний момент часу) - це її організація або структура значущих компонентів взаємодіючи через інтерфейси, ці компоненти складаються з послідовно менших компонентів та інтерфейсів ». Така трактовка досить добре узгоджується зі стандартом архітектури - ISO/IEC/IEEE 42010. Але в рамках даної роботи не вважаю її вичерпною, такою що повністю та однозначно описує всі аспекти архітектури. Тому пропоную взяти до уваги більш детальне визначення АПЗ запропоноване Філіпом Крюхтеном *[7]* в журналі IEEE Software.

Так за Крюхтеном АПЗ займається розробкою та впровадженням структури програмного забезпечення високого рівня. Це є результатом збірки певної кількості архітектурних елементів у деяких добре підібраних формах для задоволення основних вимог до функціональності та продуктивності системи, а також деякі інші, нефункціональні вимоги такі, як надійність, масштабованість, портативність

та доступність. Короткий варіант цієї концепції звучить так: АПЗ дорівнює елементам та формам у поєднанні з вимогами та обмеженнями.

АПЗ займається абстракцією, декомпозицією та композицією, стилем та естетикою. Для опису АПЗ ми використовуємо модель, що складається з декількох поглядів або перспектив. Розглядаючи великі та складні архітектури, запропонована Крюхтеном коцепція складається з п'яти основних поглядів (див. рис. 1.1):

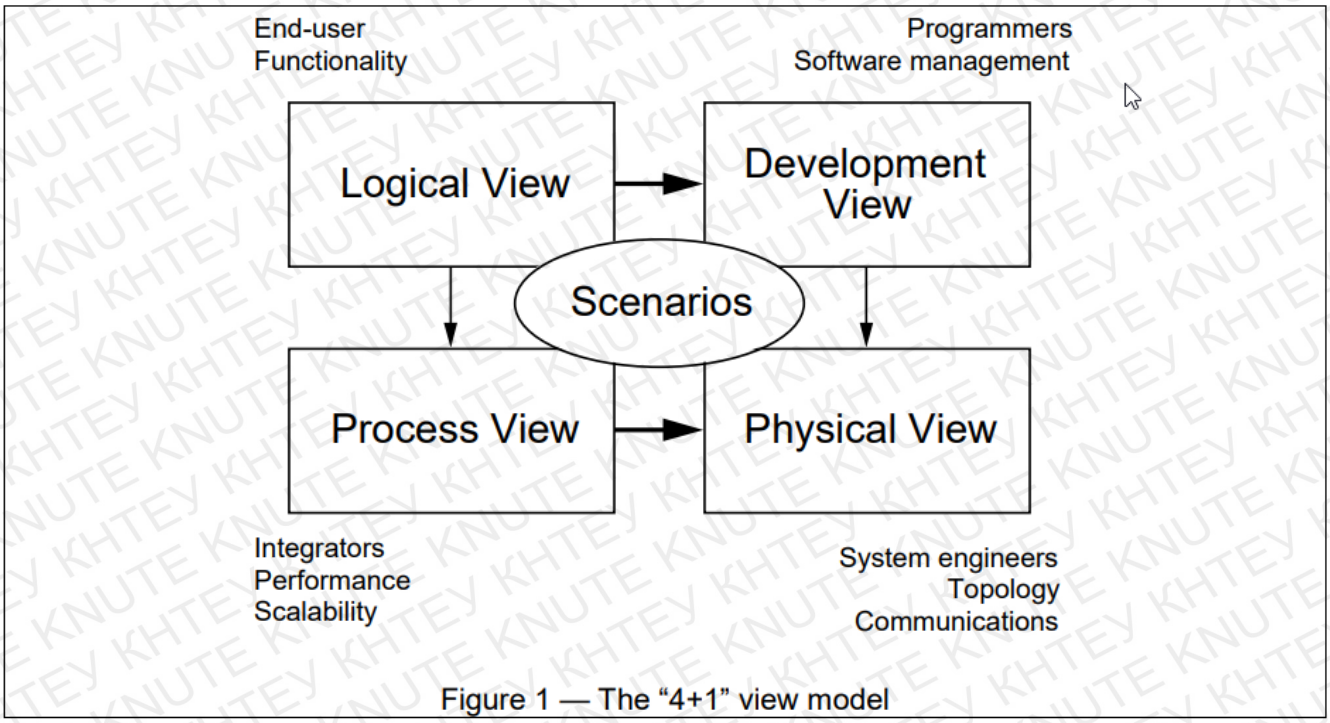


Рис. 1.1 Концепція АПЗ Філіпа Крюхтена

- Логічний погляд, який є об'єктною моделлю проектування (коли використовується об'єктно-орієнтований метод проектування) використовується),
- перегляд процесу, який відображає аспекти паралельності та синхронізації дизайну,
- фізичний вигляд, який описує відображення програмного забезпечення на обладнання та відображає його розподілений аспект,

- погляд на розробку, який описує статичну організацію програмного забезпечення під час його розробки середовище.

Опис архітектури, прийняті рішення, можна організувати навколо цих чотирьох поглядів, і потім ілюструється кількома вибраними варіантами використання або сценаріями, які стають п’ятим поданням.

Таке визначення АПЗ дає розуміння як створювати та описувати програмне забезпечення з усіх сторін та аспектів розробки.

1.2. Технічний опис та нормативно-правова база процедур автентифікації ідентифікації.

Далі важливо детально розібрати процеси автентифікації ідентифікації та авторизації які є основною функцією нашої системи. Такі процедури являються ключовими в системах які використовують приватні данні користувача, тому що вони забезпечують первинних захист від розповсюдження приватних даних, що захищені законом. Тому в законодавстві більшості держав, в тому числі України, закріплено значення цих понять та мінімальні вимоги до їх реалізації. Так згідно до постанови Кабінету Міністрів України від 2006 року *[10]*:

Автентифікація - процедура встановлення належності користувачеві інформації в системі (далі - користувач) пред'явленого ним ідентифікатора;

Ідентифікація - процедура розпізнавання користувача в системі як правило за допомогою наперед визначеного імені (ідентифікатора) або іншої апріорної інформації про нього, яка сприймається системою.

Також закон пропонує такий поділ інформації:

- Відкрита інформація, яка належить до державних інформаційних ресурсів, а також відкрита інформація про діяльність суб'єктів владних повноважень, військових формувань, яка оприлюднюється в Інтернеті, інших глобальних інформаційних мережах і системах або передається телекомунікаційними мережами.

- Конфіденційна інформація, яка перебуває у володінні розпорядників інформації, визначених частиною першою статті 13 Закону України "Про доступ до публічної інформації.
- Службова інформація.
- Інформація, яка становить державну або іншу передбачену законом таємницю (далі - таємна інформація).
- Інформація, вимога щодо захисту якої встановлена законом.

Також закон передбачає визначення до яких видів даних може мати доступ авторизований, ідентифікований та невідомий користувачі. Але не регламентується конкретний фактор за допомогою якого має бути виконано автентифікацію.

Пропоную розглянути методи автентифікації представлені в роботі Річарда Сміта - «Authentication: From Passwords to Public Keys»:

Пароль - це секретна інформацію, яка невідома необізнаним людям. До комп'ютерів це могли бути голосовий пароль або секретний код від фізичного замку. У комп'ютерах це може бути слово-пароль, фраза або персональний ідентифікаційний номер.

Розробники можуть дешево і легко реалізувати паролевий механізм. Секретне слово є ідеальним засобом для автентифікації переміщаються користувачів, тобто, для людей, котрі підключаються до системи з непередбачуваних віддалених місць.

Однак паролі мають слабкі місця. По-перше, їх ефективність залежить від секретності, і їх важко зберегти в таємниці. Існує незліченна кількість способів вивідати або перехопити пароль, і зазвичай немає способу виявити активну розвідку до завдання шкоди. По-друге, розвиток методів нападу зробило для зломщиків визначення паролів, зазвичай обираних людьми, досить простою справою. Навіть якщо вибираються складні паролі, їх де-небудь записують, щоб при необхідності мати під рукою. Але звичайно ж, записаний пароль більш вразливий в плані можливої крадіжки, ніж запам'ятовуємий. Навіть діючі з кращих спонукань люди в

якийсь момент порушують правила використання паролів просто для того, щоб мати можливість скористатися своїм комп'ютером саме тоді, коли в цьому виникне необхідність.

Пристрій автентифікації - відмінною характеристикою є володіння авторизованими людьми якимось конкретним предметом. До появи комп'ютерів це могла бути печатка с особистим підписом або ключ від замка. У комп'ютерах це може бути не більше ніж файл даних, що містить відмінну характеристику. Часто характеристика вбудовується в пристрій, наприклад в картку з магнітною смугою, смарт-карт або в обчислювач пароля. У цій книзі подібні речі називаються пристроями автентифікації. Автентифікацію на основі пристроїв найскладніше обійти, так як використовується унікальний фізичний об'єкт, яким повинен володіти людина, щоб увійти в систему.

На відміну від паролів, власник завжди може відразу сказати, якщо пристрій було вкрадено, і йому складно ділити його з ким-небудь ще і одночасно мати можливість входу в систему.

Основними слабкими місцями є більш висока вартість реалізації і ризик втрати або відмови апаратури. Проблемою може бути також і мобільність.

Біометрика - відмінною характеристикою є якась фізична особливість, унікальна для особи, що перевіряється. До появи комп'ютерів це могли бути особистий підпис, портрет, відбиток пальця або письмовий опис зовнішнього вигляду людини. У разі комп'ютерів відмінна характеристика фізичної особи вимірюється і порівнюється з раніше отриманими даними, знятими з достовірно встановленої особи. У добре відомих методиках для автентифікації використовуються голос людини, відбитки пальців, письмова підпис, форма руки або особливості очей. Надалі подібні речі називаються біометрикою.

Біометрична автентифікація зазвичай є найлегшим підходом для тих людей, які повинні проходити автентифікацію. У більшості випадків добре спроектована

біометрична система просто знімає показання з людини і правильно виконує автентифікацію. В даному випадку очевидно, що відмітна характеристика досить мобільна, тому що є частиною тіла власника.

Однак така система має недоліки. Як правило, в порівнянні з іншими системами обладнання дороге в придбанні, установці і експлуатації. При дистанційному використанні біометричні свідчення схильні до ризику перехоплення: викрадач може відтворити запис свідчень, щоб замаскувати себе під власника, або використовувати їх, щоб того вистежити. Якщо біометричні показники потрапляють. в погані руки, то їх власник не має способу заповнення збитку, так як біометричні особливості неможливо змінити.

Крім того, сам процес автентифікації складний. Складно також зробити систему досить чутливою, щоб вона відкидала сторонніх користувачів і при цьому іноді не відкидала своїх. Біометричні показники також можуть бути визнані непридатними внаслідок фізіологічних змін і тілесних ушкоджень. Був випадок, коли жінка, яка працювала в режимному приміщенні, не була впущена всередину біометричним пристроєм через те, що її вагітність викликала зміна картини кровоносних судин в сітківці очей.

Щоб більш предметно розглянути даний процес було побудовано діаграму послідовності (рис 1.2). На якій зображено процес автентифікації користувача, на прикладі Веб-додатка Facebook.

Основні протоколи та схеми автентифікації: **«Базова»** схема автентифікації базується на моделі, за якою агент користувача повинен автентифікуватися за допомогою ідентифікатора користувача та пароля для кожної області додатку. Сервер дозволить запит, лише якщо він зможе перевірити ідентифікатор користувача та пароль для захищеного URI. Немає додаткових параметрів автентифікації.

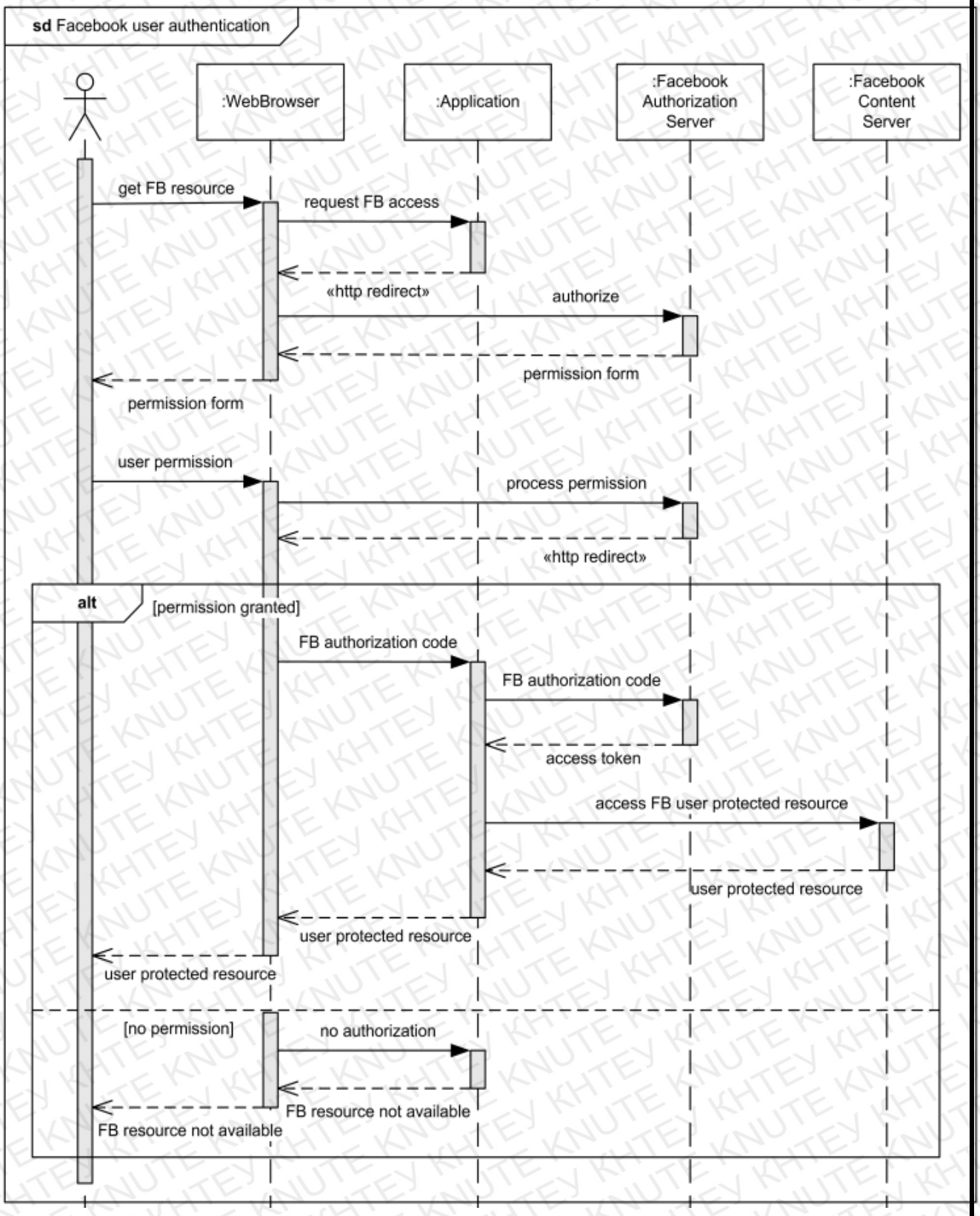


Рис. 1.2 Діаграма послідовностей автентифікації Facebook

Базова схема автентифікації - це незахищений метод фільтрації несанкціонованого доступу до ресурсів на сервері HTTP. Він базується на припущенні, що зв'язок між клієнтом та сервером може розглядатися як перевірений перевізник. Оскільки у відкритій мережі це, як правило, не відповідає дійсності, слід відповідно використовувати базову схему автентифікації. Незважаючи на це, клієнти повинні впроваджувати схему для спілкування із серверами, які її використовують.

Автентифікація за сертифікатами. Сертифікат являє собою набір атрибутів, що ідентифікують власника, підписаний certificate authority (CA). CA виступає в ролі посередника, який гарантує справжність сертифікатів (за аналогією з ФМС, що випускає паспорта). Також сертифікат криптографічно пов'язаний з закритим ключем, який зберігається у власника сертифіката і дозволяє однозначно підтвердити факт володіння сертифікатом.

Автентифікація по токену. Такий спосіб автентифікації найчастіше застосовується при побудові розподілених систем Single Sign-On (SSO), де один додаток (service provider або relying party) делегує функцію автентифікації користувачів іншому додатку (identity provider або authentication service). Типовий приклад цього способу - вхід в додаток через обліковий запис в соціальних мережах. Тут соціальні мережі є сервісами автентифікації, а додаток довіряє функцію автентифікації користувачів соціальних мереж. Реалізація цього способу полягає в тому, що identity provider (IP) надає достовірні відомості про користувача в вигляді токена, а service provider (SP) додаток використовує цей токен для ідентифікації, автентифікації і авторизації користувача. На загальному рівні, весь процес виглядає наступним чином: Клієнт автентифікується в identity provider одним із способів, специфічним для нього (пароль, ключ доступу, сертифікат, Kerberos, ітд.). Клієнт просить identity provider надати йому токен для конкретного SP-додатки. Identity

provider генерує токен і відправляє його клієнту. Клієнт проходить автентифікацію в SP-додатку за допомогою цього токена.

OpenID Connect 1.0 - (рис 1.3) це простий рівень автентифікації поверх протоколу OAuth 2.0. Це дозволяє Клієнтам перевіряти особу Кінцевого Користувача на основі аутентифікації, виконаної Сервером Авторизації, а також отримувати основну інформацію про профіль Кінцевого Користувача взаємосумісним і REST-подібним способом.

OpenID Connect дозволяє клієнтам усіх типів, включаючи веб-клієнти, мобільні та клієнти JavaScript, запитувати та отримувати інформацію про аутентифіковані сеанси та кінцевих користувачів. Набір специфікацій розширюється, дозволяючи учасникам використовувати додаткові функції, такі як шифрування даних про особисті дані, відкриття постачальників OpenID та управління сеансами, коли це має для них сенс.

1.3. Регуляція процесу доступу до даних автентифікованого користувача

Після проходження автентифікації, в складних системах часто додають ще один крок такий як авторизація. Авторизація відноситься до процесу надання привілеїв процесам і зрештою, користувачам. Це відрізняється від автентифікації тим, що автентифікація — це процес, який використовується для ідентифікації користувача. Після ідентифікації (надійно) привілеї, права, майно та допустимі дії користувача визначаються авторизацією. Явно перерахувати дозволені дії кожного користувача (та процесу користувача) щодо всіх ресурсів (об’єктів) неможливо у розумній системі. У реальній системі використовуються певні методики для спрощення процесу надання та перевірки авторизації(ів). Один підхід, популярний в системах UNIX, полягає в тому, щоб призначити кожному об'єкту три класи користувачів: власник, група і світ. Власник-це або автор об’єкта, або користувач, призначений суперкористувачем як власник. Дозволи власника (читання, запис і виконання)

застосовуються лише до власника. Група - це сукупність користувачів, які мають спільні права доступу до об'єкта. Дозволи групи (читання, запис та виконання) застосовуються до всіх користувачів у групі (крім власника). Світ стосується всіх інших, хто має доступ до системи. Світові дозволи (читання, запис та виконання) застосовуються до всіх користувачів (крім власника та учасників групи). Інший підхід - додати до об'єкта список, який явно містить ідентифікацію всіх дозволених користувачів (або груп). Це список контролю доступу.

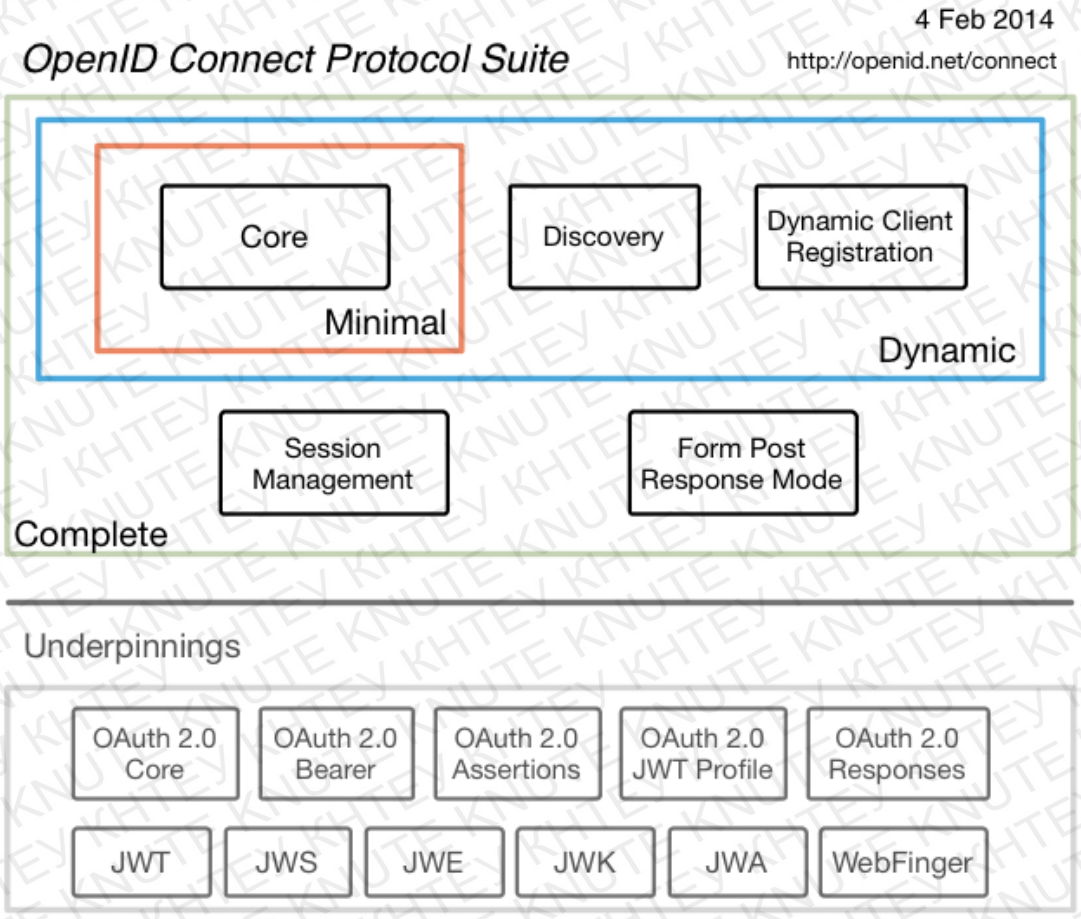


Рис. 1.3 Загальна структура протоколу OpenId Connect

1.4. Висновок до розділу 1

Було опрацьовано технічну документацію таку як технічні журнали та книги, документацію до протоколів та систем-аналогів. Було проаналізовано нормативно-правові акти такі як постанова Кабінету Міністрів, що описує класифікації даних та

послуг, визначає групи користувачів та їх права на доступ до певних класифікацій даних, визначає мінімально вимоги до безпеки, дає вичерпне розуміння процедур автентифікації авторизації, та ідентифікації з правової точки зору. Розглянуто стандарти, що використовуються в процесі розроблення програмного забезпечення. Визначено інформаційну базу на яку слід опиратися в процесі проектування архітектури. Оброблено законодавчу базу якою регламентуються вимоги та специфікації системи автентифікації людини. Проаналізовано різні погляди на процес автентифікації та проектування архітектури.

РОЗДІЛ 2
СПОСОБИ ВИРІШЕННЯ ЗАВДАНЬ ТА ТЕХНОЛОГІЇ ВИКОРИСТАНІ В ПРОЦЕСІ РОЗРОБКИ

2.1. Протокол автентифікації та авторизації

В процесі дослідження інформаційного процесу, що забезпечує розробку програмного продукту, було визначено основні способи автентифікації присутні на ринку:

- Автентифікація по паролю (HTTP, Windows authentication, Kerberos і т.д.)
- З допомогою сертифікату (SSL/TLS).
- За одноразовими паролями (2FA)
- З використанням токенів (OpenIdConnect, Ws-Trust, Ws-Federation)

Щоб зрозуміти чи є такий протокол, який підходить для розробки системи автентифікації з використанням голосової верифікації, потрібно визначити всі переваги та недоліки.

Авторизація за допомогою паролю сама собою являється найбільш простою технологією, але при цьому є досить не гнучкою, та не підходить для нашої системи через те що, не зручно постійно говорити фразу-пароль, перед тим як віддати команду. Тим більше за допомогою активної розвідки дуже просто викрасти пароль та отримати певний контроль над системою.

Розглядаючи систему авторизації за допомогою сертифікатів, можна визначити її як одну з найскладніших в реалізації та впровадженні, але при цьому вона забезпечує високий рівень надійності. Але навіть в цифрових сертифікатів є певні вразливі місця – центри сертифікації, в них працюють люди, що можуть

приймати недобросовісні рішення про видачу чи блокування того чи іншого сертифікату. В рамках розробки системи авторизації з голосовим керуванням неможливо інтегрувати таку схему тому що, ознакою за якою можна буде провести авторизацію має бути голос користувача.

Якщо розглядати варіант з використанням одноразових паролів, він не підходить так само як і статичний пароль, але може служити альтернативним варіантом авторизації, адже кожен пароль генерується заново та має короткий життєвий цикл.

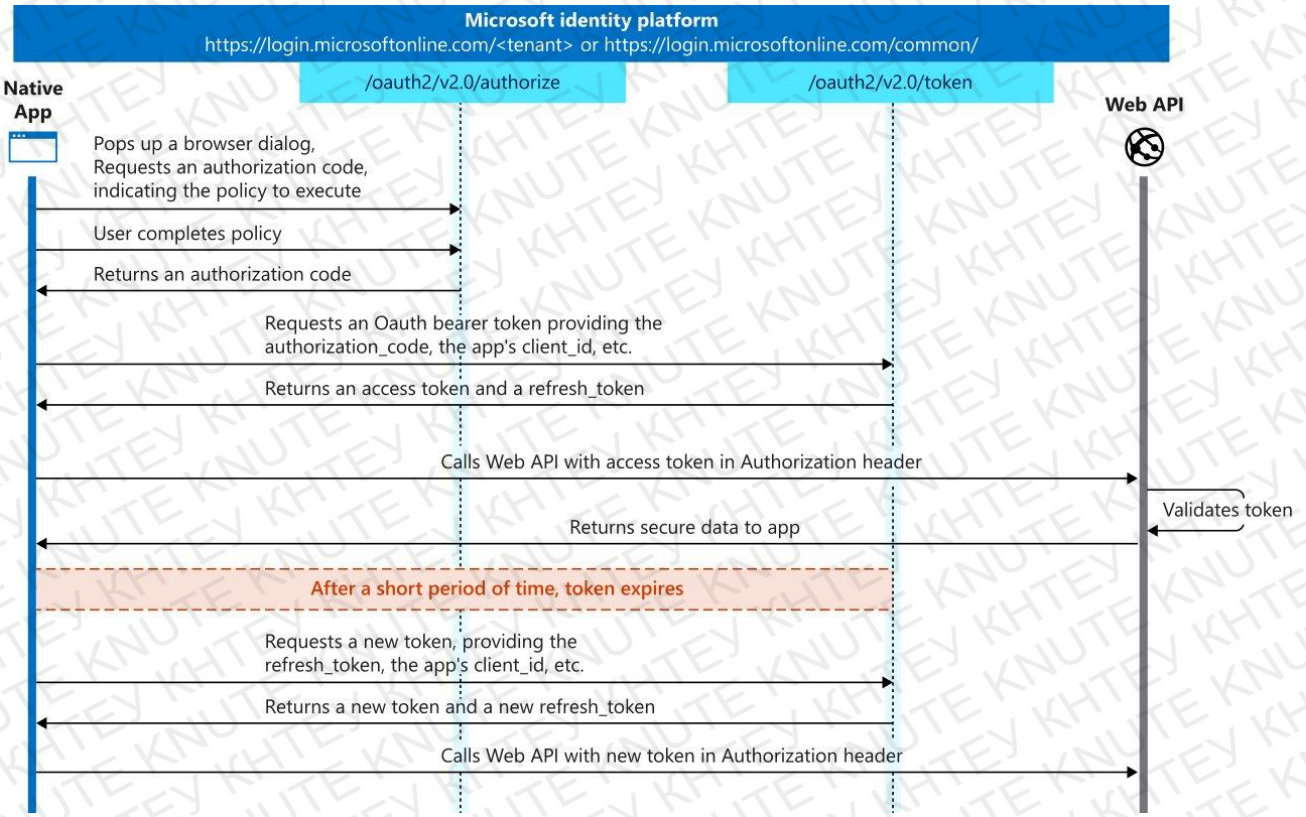


Рис. 2.1 Діаграма послідовності автентифікації по стандарту OpenId Connect

Використання токенів досить легкий в реалізації та гнучкий метод авторизації, має перевагу в тому що токен – шифрований набір даних, має невеликий строк дії, ці фактори ускладнюють процес викрадення персональних даних користувача. При цьому в інтеграції такого підходу є складність в тому, що

ознакою для генерації токену є пароль та логін, які не підходять в рамках даної системи.

Дивлячись на перелічені способи автентифікації було прийнято рішення взяти за основу протокол OpenID Connect *[11]* (рис 2.1) з використанням токенів і вдосконалити його, зробивши сумісним з можливостю керувати голосом. Причини: токен є набором зашифрованих даних, який можна змінити за бажанням. Таким чином можна узгодити перелік очікуваних даних в токені і абстрагуватися від паролю та логіну. Даний протокол є досить гнучким, дає можливість автентифікації та являється обгорткою навколо протоколу авторизації OAuth2.0, що дає нам можливість поєднати авторизацію та автентифікацію в рамках одного процесу.

2.2. Вдосконалення протоколу автентифікації

Перед тим як розпочати розробку схеми інтеграції, потрібно визначитися з ключовими пунктами, яким має відповідати вихідне рішення :

Таблиця 2.1

Вимоги до протоколу автентифікації

Вимога	Опис
Надійність	Вихідний стандарт повинен мати передбачувану поведінку, методи виявлення та усунення помилок та виняткових ситуацій.
Гнучкість	Має бути варіант авторизуватися навіть якщо, за якихось причин голосова авторизація не підходить.
Розширюваність	Необхідно врахувати можливість додавати різні варіанти автентифікації, наприклад за допомогою обличчя або відбитка пальця.
Толерантність до ресурсів	Так як процес роботи з аудіо файлами та визначенням користувача по голосу, потребують досить багато ресурсів процесора та пам’яті.

Для забезпечення таких вимог потрібно передбачити альтернативний спосіб автентифікації, наприклад через логін та пароль. Це буде забезпечувати певний рівень надійності, в тих випадках коли за якихось причин голосова автентифікація не є можливою.

Правильне використання JWT може забезпечити виконання всіх інших вимог. Шифрування певних уніфікованих даних властивих лише для одного користувача, може забезпечити гнучкість та розширюваність, при цьому чим менша кількість таких даних тим простіше працювати з токеном та менше ресурсів витрачається на авторизацію та автентифікацію.

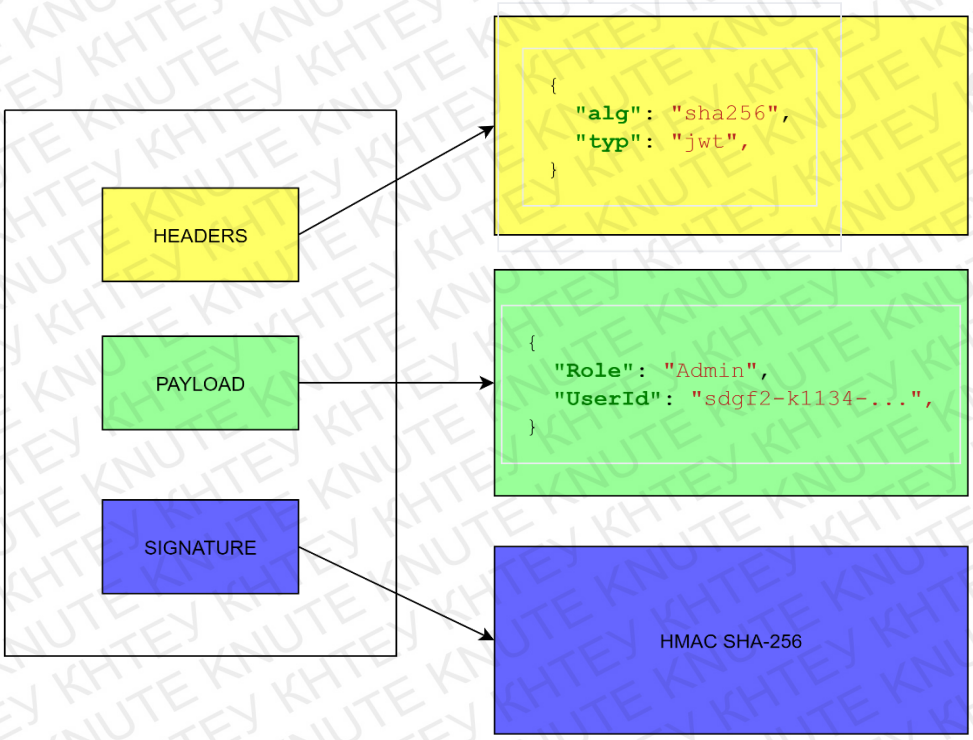


Рис. 2.2 Структура JWT

Враховуючи всі вище наведені критерії було прийнято рішення доповнити стандартну структуру JWT такою інформацією про користувача:

- Роль за допомогою якої можливо видавати право на використання того чи іншого ресурсу.
- Ідентифікатор – унікальна ознака яка дозволяє ідентифікувати користувача.

Отож вихідна структура JWT представлена на рисунку 2.2.

2.3. Інтеграція модуля голосової автентифікації

Також потрібно визначити вимоги до модуля голосової автентифікації:

Таблиця 2.2

Вимоги до модуля голосової автентифікації

Вимога	Опис
Надійність	Вихідний стандарт повинен мати передбачувану поведінку, методи виявлення та усунення помилок та виняткових ситуацій.
Гнучкість	Має бути варіант авторизуватися навіть якщо, за якихось причин голосова авторизація не підходить.
Розширюваність	Необхідно врахувати можливість додавати різні варіанти автентифікації, наприклад за допомогою обличчя або відбитка пальця.
Толерантність до ресурсів	Так як процес роботи з аудіо файлами та визначенням користувача по голосу, потребують досить багато ресурсів процесора та пам'яті.

Щоб обрати найкращий варіант пропоную розглянути основні методи та способи інтеграції такої функції в систему:

- Розробка і підтримка власного модуля.
- Використання і підтримка стороннього open-source рішення.
- Використання спеціалізованого сервісу.

Розглядаючи варіант розробки та підтримки власного алгоритму було визначено основні підходи до реалізації. Перше це визначення вхідних та вихідних даних. Вхідні данні це вибірка голосових профілів з якими порівнюється

контрольний голосовий файл, формат файлів можна обрати стандартний .wav, вихідні данні це ідентифікатор голосового профілю автентифікованого користувача. Алгоритм ідентифікації та автентифікації має бути побудований на рекурентній нейронні мережі.

Рекурентна нейронна мережа (рис. 2.3) (RNN) — це клас штучних нейронних мереж, де зв’язки між вузлами утворюють орієнтований граф уздовж тимчасової послідовності. Це дозволяє йому демонструвати тимчасову динамічну поведінку.

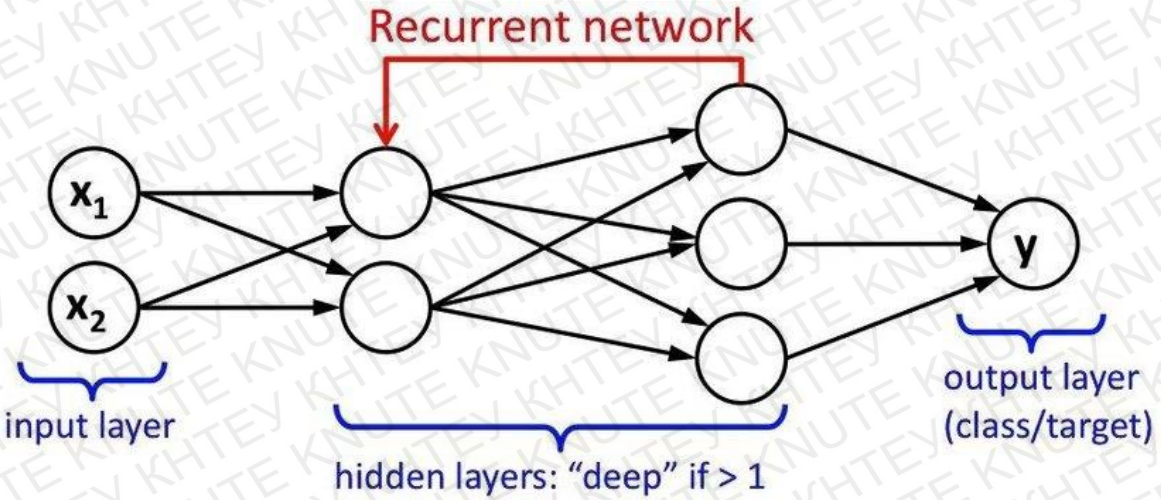


Рис. 2.3 Структура примітивної рекурентної нейронної мережі

Похідні від нейронних мереж із прямим зв'язком, RNN можуть використовувати свій внутрішній стан (пам'ять) для обробки послідовностей змінної довжини вхідних даних. Це робить їх застосовними до таких завдань, як несегментоване, пов’язане розпізнавання рукописного тексту або розпізнавання мовлення. Рекурентні нейронні мережі теоретично завершені за Тюрінгом і можуть запускати довільні програми для обробки довільних послідовностей вхідних даних.

Для роботи самої нейронної мережі потрібно перетворити вхідні данні в виді аудіо файлу в уніфікований набір цифрових даних придатний для математичних маніпуляцій, такий процес називається pre-processing (перед-обробка).

Перед-обробка починається з розділення аудіо файлу на рівні відрізки фіксованої довжини. Кожен відрізок перетворюється на масив з числами висоти

звукової хвилі. Далі така цифрова характеристика звуку має велику кількість різних звукових частот які накладаючись один на одну створюють людську мову. Тому щоб розкласти складний звук на кванти потрібно провести декілька математичних перетворень, наприклад перетворення Фурьє.

Перетворення Фур'є - операція, що співставляє однієї функції речової змінної іншу функцію, взагалі кажучи, комплексної змінної. Ця нова функція визначає коефіцієнти («амплітуди») при розкладанні вихідної функції на елементарні складові — гармонійні коливання з різними частотами (подібно до того, як музичний акорд може бути виражений у вигляді суми музичних звуків, що його становлять).

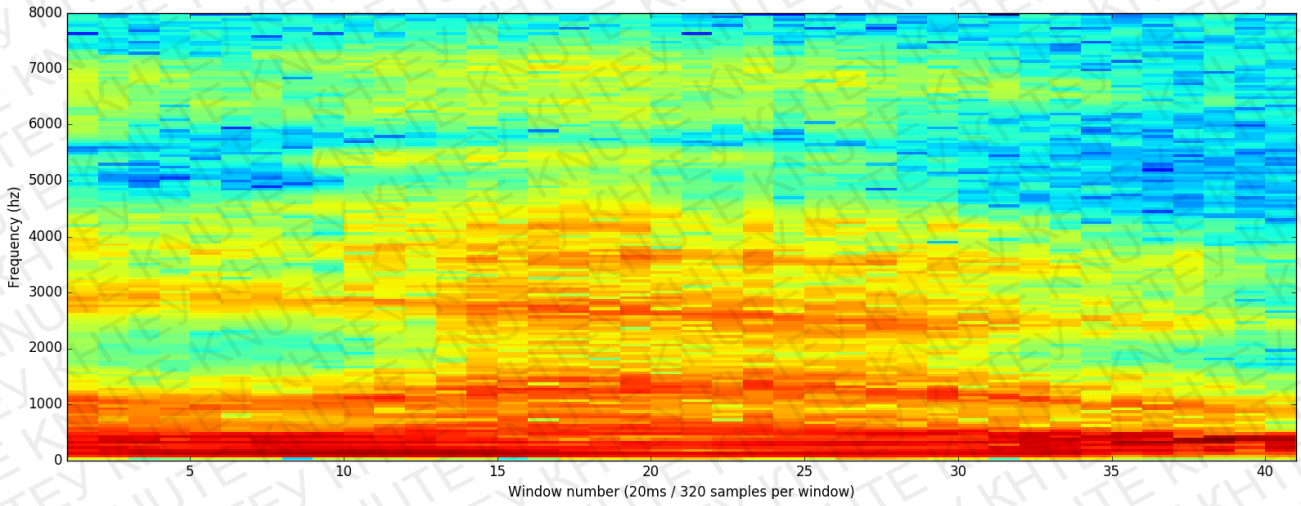


Рис. 2.4 Частотна діаграма звукової хвилі

Таке перетворення розкладає складну звукову хвилю на її найменші складові, що дає нам можливість отримати потужність квантів звуку, та звести сумарну потужність кожної хвилі. Результат такого обчислення є оцінкою важливості кожного частотного діапазону. З таких даних створюється частотна діаграма (рис. 2.4).

Ті самі дії повторюються для кожного з відрізків, результат роботи перед-обробки є вхідними даними нейронної мережі, результатом є коефіцієнт подібності до тієї чи іншої вже аналізованої звукової доріжки.

Переваги розробки власного модуля голосової автентифікації

- Гнучкість можливість розробити вузько-спеціалізований продукт, що ідеально підходить під специфіку задачі, можливість додавати функціонал по мірі необхідності
- Власна підтримка – оперативне вирішення критичних ситуацій
- Передбачувана поведінка власна реалізація має досить передбачувану та логічну поведінку

Недоліки такого підходу

- Ресурси на розробку нейронна мережа це досить складний механізм для побудови якого потрібно багато часу, коштів та спеціалізованих кадрів, звичайний розробник навряд чи зуміє побудувати гарну нейронну мережу в адекватний строк.
- Ресурси на підтримку – так як для підтримки роботи нейронної мережі потрібно багато серверних ресурсів та коштів на утримання спеціалістів.
- Низька ефективність в порівнянні з конкурентами нейронна мережа власного виробництва розроблена силами однієї або декількох команд, не будуть стояти на одному рівні з продуктами компаній у яких нейронні мережі є основним напрямком діяльності.

Також одним із підходів є використання open-source рішення, такий варіант дозволяє не витрачати час та кошти на розробку системи. Основні переваги такого підходу.

- Вимагає мінімальної кількості коштів та часу
- З інтеграцією може справитися штатний спеціаліст

При цьому такий підхід має і низку недоліків

- Погана підтримка якщо такий ресурс взагалі має підтримку, то на швидкість та оперативність можна не розраховувати адже продукт не є комерційним.
- Складність інтеграції, продукт розроблений для виконання певного спектру дій часто має уніфіковані інтерфейси для інтеграції, що потребують доробки або можуть взагалі не підійти по ряду причин.
- Рівень ефективності такої мережі буде більш слабким порівнюючи з його комерційними аналогами.
- Ненадійність – в перспективі в процесі підтримки такого модуля може статися так що дана реалізація вже не забезпечує необхідних вимог, що може вилитися в досить великі проблеми на етапі підтримки продукту.

І останній підхід це використання стороннього сервісу для автентифікації голосом. Такий підхід вимагає найменшу кількість людино/годин, так як все що потрібно це отримати підписку на сервіс та інтегрувати його в свою систему. Такий підхід має ряд переваг

- Гарна служба підтримки - в комерційних сервісах зазвичай досить потужна підтримка так як від цього напряму залежить їх прибуток
- Висока ефективність – так само як і з підтримкою, якщо цей продукт є основною статтею доходу напрямку компанії він є досить конкурентноздатним тому має високий рівень ефективності.
- Проста інтеграція- для легкості використання сервіси зазвичай спрощують та уніфікують методи інтеграції роблячи їх досить простими.
- В процесі використання стороннього сервісу можна вести розробку власного замінника і в процесі розробки

Але при всіх перевагах такий метод має суттєвий недолік: платна підписка – такий підхід постійно потребує коштів, в залежності від об’ємів трафіку кількість коштів може досягати великих розмірів.

Проаналізувавши основні підходи до реалізації модуля голосової автентифікації було обрано оптимальний підхід – використання стороннього сервісу. Адже розробка нейронних мереж занадто складний процес для реалізації його одним спеціалістом, також дана тематика не являється частиною теми даної роботи. Використання open-source проекту не забезпечує бажаний рівень надійності та ефективності роботи модуля.

Серед представлених на ринку сервісів було обрано Microsoft Azure Cognitive Services. Тому що, серед всіх варіантів Microsoft пропонує досить високий рівень послуг, пропонує широкий спектр можливостей, оплату по факту використання та пробний період.

Служба мовлення (Speech) *[12]* — це об'єднання мовлення в текст, синтезу мовлення та мовлення в єдину підписку Azure. Увімкнути свої програми, інструменти та пристрої можна легко за допомогою інтерфейсу Speech CLI, Speech SDK, Speech Devices SDK, Speech Studio або REST API.

Перелічені нижче функції є частиною послуги Speech:

Таблиця 2.3

Можливості та функції сервісу Speech

Сервіс	Послуга	Опис	SDK	REST
Мова в текст	Мова в текст В реальному часі	Перетворення мовлення в текст транскрибує або перекладає аудіопотоки або локальні файли в текст у режимі реального часу, який можуть використовувати або відображати ваші програми, інструменти чи пристрої. Використовуйте мовлення в текст із розумінням мови (LUIS), щоб отримати наміри користувача з транскрибованого мовлення та діяти за голосовими командами.	+	+

	Пакетне перетворення мовлення в текст	Пакетна передача мовлення в текст забезпечує асинхронну транскрипцію мови в текст великих обсягів мовних аудіоданих, що зберігаються в сховищі BLOB-об'єктів Azure. На додаток до перетворення аудіо мовлення в	-	+
		текст, пакетне перетворення мови в текст також дозволяє проводити діаризацію та аналіз настроїв.		
	Розмова на кількох пристроях	Підключіть кілька пристроїв або клієнтів до розмови, щоб надсилати мовні або текстові повідомлення з легкою підтримкою транскрипції та перекладу	+	-
	Оцінка вимови	Оцінка вимови оцінює вимову мовлення та дає доповідачам відгуки про точність і плавність розмовного звуку. Завдяки оцінці вимови ті, хто вивчає мову, можуть практикувати, отримувати миттєвий зворотний зв'язок і покращувати свою вимову, щоб вони могли говорити та презентувати з упевненістю.	+	+
Синтез мовлення	Синтез мовлення	Синтез тексту в мовлення перетворює введений текст у синтезовану мову, схожу на людину, за допомогою мови розмітки мовлення (SSML). Використовуйте нейронні голоси, які є людськими голосами, які живляться від глибоких нейронних мереж.	+	+
	Створюйте власні голоси	Створюйте власні голосові шрифти, унікальні для вашого бренду або продукту.	-	+

Продовження таблиці 2.3

Переклад мовлення	Переклад мовлення	Переклад мовлення дає змогу в реальному часі багатомовний переклад мовлення у ваші програми, інструменти та пристрої. Використовуйте цю службу для перекладу мовлення в мовлення та з мови в текст.	+	-
Голосові помічники	Голосові помічники	Голосові помічники, які використовують службу Speech, дають розробникам можливість створювати природні, схожі на людину інтерфейси розмови для своїх додатків і досвіду. Служба голосового помічника забезпечує швидку та надійну взаємодію між пристроєм та реалізацією помічника, яка використовує прямий мовний канал	+	-
Розпізнавання оратора	Перевірка та ідентифікація мовця	Служба розпізнавання спікера надає алгоритми, які перевіряють та ідентифікують мовців за їх унікальними характеристиками голосу. Розпізнавання спікера використовується для відповіді на запитання «хто говорить?».	+	+

2.4. Розроблення процесу автентифікації та визначення необхідних модулів

За основу процесу виконання автентифікації користувача за допомогою голосу, було обрано стандарт OpenId Connect який було доопрацьовано та вдосконалено, в результаті чого було створено такий процес автентифікації користувача який зображено на рисунку 2.5. На рисунку 2.6 зображено процес неуспішної авторизації.

На діаграмі представлено успішний варіант авторизації користувача, представлено основні потоки даних, та модулі які забезпечують виконання

основних запитів. На наступній діаграмі представлено неуспішні варіанти авторизації та автентифікації користувача.

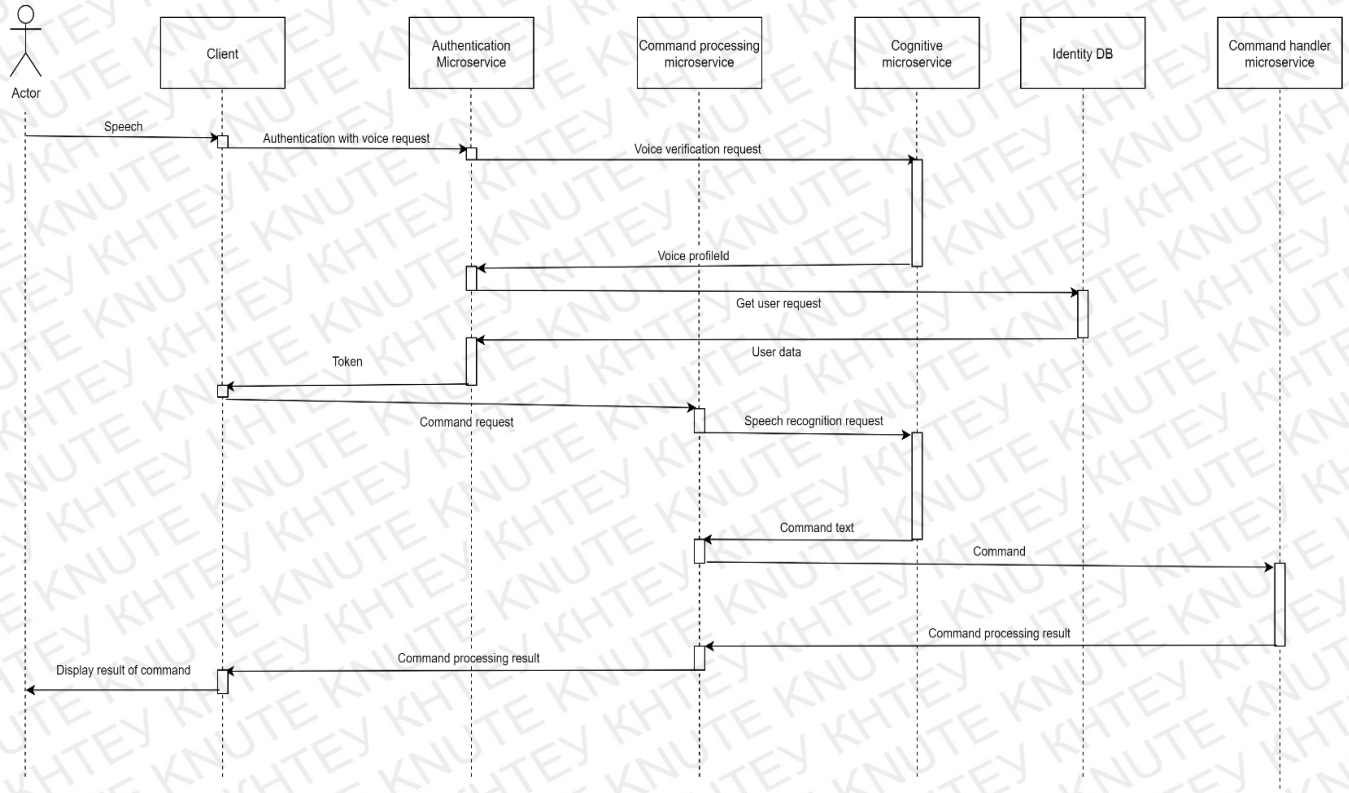


Рис. 2.5 Діаграма послідовності успішної авторизації користувача

Визначившись з можливостями модуля розпізнавання та верифікації мовця слід зазначити, що проста реєстрація та автентифікація аккаунту може бути занадто ресурс затратною через необхідність порівнювати голосову доріжку з голосовими профілями всіх користувачів системи, яких може бути велика кількість.

Тому було прийнято рішення вдосконалити схему автентифікації, шляхом добавляння механізму «сімей» (рис 2.7). Сім'я – група користувачів я рамках якої і проходить ідентифікація користувача. Такий підхід дозволяє впровадити автентифіковані сесії, та зменшити час на процес авторизації, та автентифікації.

Для кращого розуміння процесу автентифікації було розроблені набір діаграм потоків, що дозволяють описати послідовність виконання запитів конкретними модулями системи

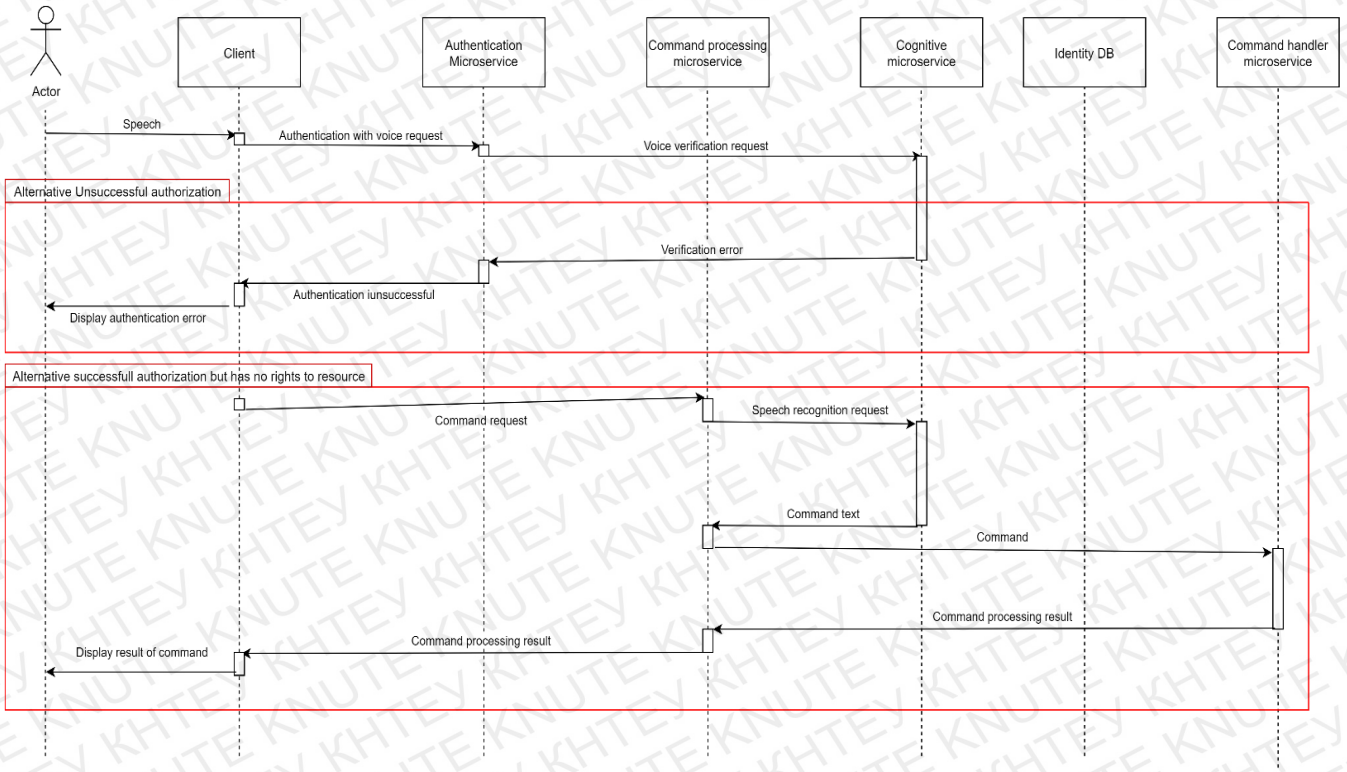


Рис. 2.6 Діаграма послідовності неуспішної авторизації користувача

Діаграма послідовності — це тип діаграми взаємодії, оскільки вона описує, як і в якому порядку група об’єктів працює разом. Ці діаграми використовуються розробниками програмного забезпечення та бізнес-фахівцями, щоб зрозуміти вимоги до нової системи або документувати існуючий процес. Діаграми послідовності іноді називають діаграмами подій або сценаріями подій.

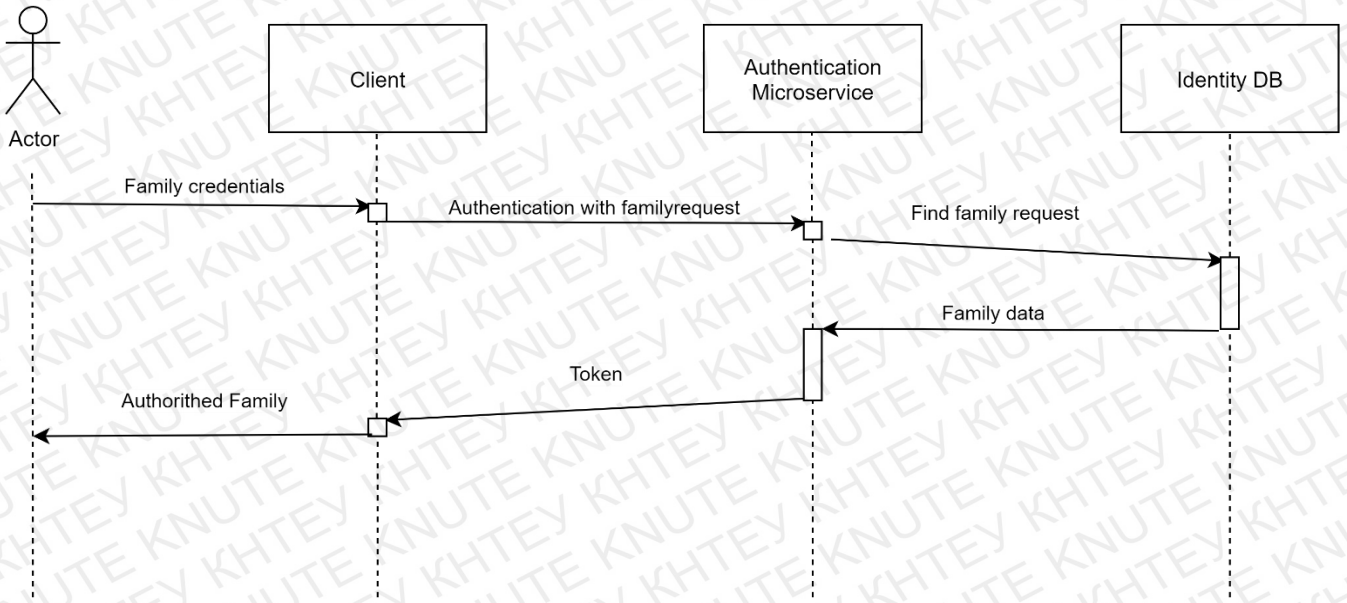


Рис 2.7 Діаграма послідовності успішної авторизації

Наступні сценарії ідеально підходять для використання діаграми послідовності:

Сценарій використання: Сценарій використання – це діаграма того, як ваша система може бути використана. Це чудовий спосіб переконатися, що ви пропрацювали логіку кожного сценарію використання системи.

Логіка методу: так само, як ви можете використовувати діаграму послідовності UML для вивчення логіки варіанту використання, ви можете використовувати її для дослідження логіки будь-якої функції, процедури або складного процесу.

Логіка сервісу: якщо ви вважаєте, що послуга є високорівневим методом, який використовується різними клієнтами, діаграма послідовності — ідеальний спосіб відобразити це.

Діаграма послідовності Visio. Будь-яку діаграму послідовності, створену за допомогою Visio, можна також завантажити в Lucidchart. Lucidchart підтримує імпорт файлів .vsd і .vdx і є чудовою альтернативою Microsoft Visio. Майже всі зображення, які ви бачите в розділі UML цього сайту, були згенеровані за допомогою Lucidchart.

2.4. Висновок до розділу 2

Було розглянуто основні технології для проектування архітектури системи автентифікації та авторизація за допомогою голосу, її розробки. Було вдосконалено протокол автентифікації OpenId Connect, створено діаграми, що відображають процес автентифікації аккаунту або сімейства.

Також було проаналізовано альтернативи вказано їх переваги та недоліки, Розглянуто актуальні методи та способи реалізації автентифікації та авторизації.

РОЗДІЛ 3
СПЕЦИФІКАЦІЯ ВИМОГ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ АВТЕНТИФІКАЦІЇ

3.1. Вимоги до архітектури

Перед початком процесу проектування АПЗ, необхідно визначити вимоги до архітектури. Ось головні з них:

Ефективність системи. Насамперед програма, звичайно ж, повинна вирішувати поставлені завдання та добре виконувати свої функції, причому у різних умовах. Сюди можна віднести такі характеристики, як надійність, безпека, продуктивність, здатність справлятися зі збільшенням навантаження (масштабованість) тощо.

Гнучкість системи. Будь-який додаток доводиться змінювати з часом - змінюються вимоги, додаються нові. Чим швидше і зручніше можна внести зміни до існуючого функціоналу, чим менше проблем і помилок це викличе — тим гнучкіша і конкурентоспроможніша система. Тому в процесі розробки потрібно намагатися оцінювати те, що виходить, щодо того, як вам це потім, можливо, доведеться змінювати. Варто запитати у себе: «А що буде, якщо поточне архітектурне рішення виявиться неправильним?», «Яка кількість коду зазнає при цьому змін?». Зміна одного фрагмента системи має впливати на її інші фрагменти. По можливості, архітектурні рішення не повинні «вирубуватись у камені», і наслідки архітектурних помилок мають бути в розумній мірі обмежені. "Гарна архітектура дозволяє відкладати прийняття ключових рішень" (Боб Мартін) і мінімізує "ціну" помилок.

Розширюваність системи. Можливість додавати до системи нові сутності та функції, не порушуючи її основної структури. На початковому етапі в систему має сенс закладати лише основний та найнеобхідніший функціонал (принцип YAGNI — you ain't gonna need it, «Вам це не знадобиться») Але при цьому архітектура повинна дозволяти легко нарощувати додатковий функціонал у міру потреби. Причому так, щоб внесення найімовірніших змін вимагало найменших зусиль.

Вимога, щоб архітектура системи мала гнучкість і розширюваність (тобто була здатна до змін та еволюції) є настільки важливою, що вона навіть сформульована у вигляді окремого принципу — «Принципу відкритості/закритості» (Open-Closed Principle — другий із п'яти принципів SOLID). Програмні сутності (класи, модулі, функції тощо) повинні бути відкритими для розширення, але закритими для модифікації.

Іншими словами – має бути можливість розширити/змінити поведінку системи без зміни/переписування вже існуючих частин системи.

Це означає, що додаток слід проектувати так, щоб зміна його поведінки і додавання нової функціональності досягалося за рахунок написання нового коду (розширення), і при цьому не доводилося б змінювати вже існуючий код. У такому разі поява нових вимог не спричинить модифікацію існуючої логіки, а зможе бути реалізована насамперед за рахунок її розширення. Саме цей принцип є основою «плагінної архітектури» (Plugin Architecture). Про те, за рахунок яких технік це може бути досягнуто, буде наведено далі.

Масштабованість процесу розробки. Можливість скоротити термін розробки рахунок додавання до проекту нових людей. Архітектура повинна дозволяти розпаралелити процес розробки, так щоб багато людей могли працювати над програмою одночасно.

Тестованість. Код, який легше тестувати, міститиме менше помилок та надійніше працюватиме. Але тести не лише покращують якість коду. Багато

розробників приходять до висновку, що вимога «хорошої тестованості» є також спрямовуючою силою, що автоматично веде до гарного дизайну, і одночасно одним з найважливіших критеріїв, що дозволяють оцінити його якість: "Використовуйте принцип «тестованості» класу як «лакмусовий папірець» хорошого дизайну Навіть якщо ви не напишіть жодного рядка тестового коду, відповідь на це питання в 90% випадків допоможе зрозуміти, наскільки все «добре» або «погано» з його дизайном" (Ідеальна архітектура).

Існує ціла методологія розробки програм на основі тестів, яка так і називається Розробка через тестування (Test-Driven Development, TDD).

Можливість повторного використання. Систему бажано проектувати так, щоб її фрагменти можна було використовувати повторно в інших системах.

Добре структурований, читаний та зрозумілий код. Супроводжуваність. Над програмою, як правило, працює багато людей — одні йдуть, приходять нові. Після написання супроводжувати програму теж, як правило, доводиться людям, які не брали участь у її розробці. Тому хороша архітектура має давати можливість відносно легко та швидко розібратися у системі новим людям. Проект має бути добре структурований, не містити дублювання, мати добре оформлений код та бажано документацію. І по можливості в системі краще застосовувати стандартні, загальноприйняті звичні рішення для програмістів. Чим екзотичніша система, тим складніше її зрозуміти іншим (Принцип найменшого подиву - Principle of least astonishment. Зазвичай, він використовується щодо інтерфейсу користувача, але застосовний і до написання коду).

Отже, враховуючи вище перелічені вимоги до АПЗ, було прийнято рішення будувати власну архітектуру на основі «мікро сервісів». архітектурний стиль мікросервісів — це підхід, у якому єдиний додаток будується як набір невеликих сервісів(рис 3.1), кожен із яких працює у процесі і комунікує з іншими використовуючи легковажні механізми, зазвичай HTTP. Ці сервіси побудовані

навколо бізнес-потреб і розгортаються незалежно з використанням повністю автоматизованого середовища. Існує абсолютний мінімум централізованого керування цими сервісами. Самі собою ці послуги можуть бути написані різними мовами і використовувати різні технології зберігання даних.

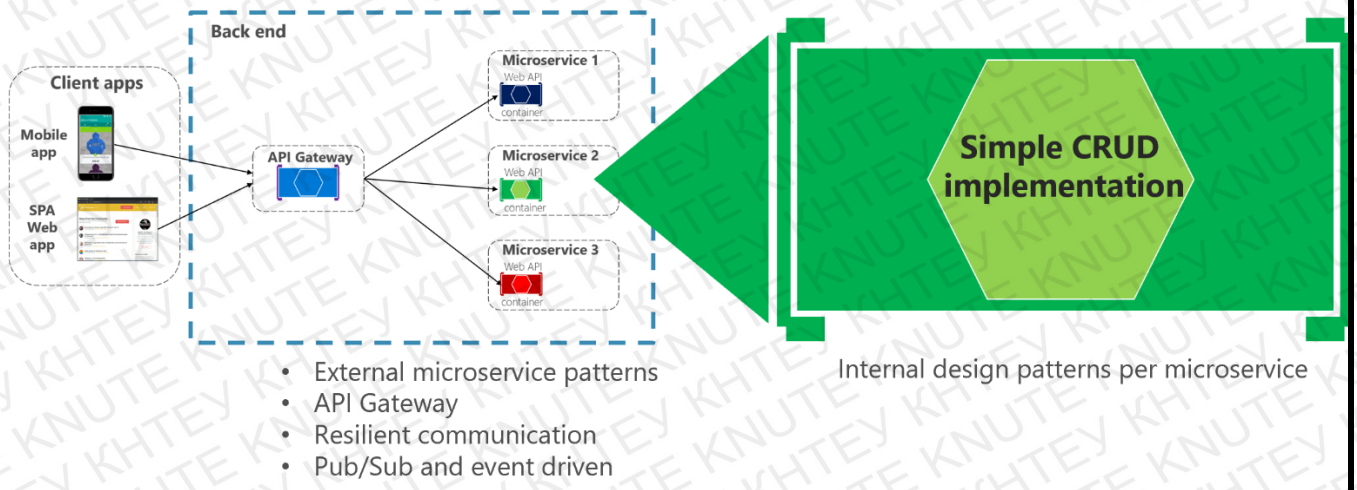


Рис. 3.1 Структура простої системи з використанням мікросервісів

Найкраще порівняти його з монолітом (monolithic style): додатком, побудованим як єдине ціле. Enterprise програми часто включають три основні частини: інтерфейс користувача (що складається як правило з HTML сторінок і javascript-а), база даних (як правило реляційної, з безліччю таблиць) і сервер. Серверна частина обробляє HTTP запити, виконує доменну логіку, запитує та оновлює дані в БД, заповнює HTML сторінки, які потім надсилаються браузеру клієнта. Будь-яка зміна в системі призводить до перескладання та розгортання нової версії серверної частини програми.

Монолітний сервер - досить очевидний спосіб побудови таких систем. Вся логіка обробки запитів виконується в єдиному процесі, при цьому ви можете використовувати можливості вашої мови програмування для поділу програми на класи, функції і namespace-и. Ви можете запускати та тестувати програму на машині розробника та використовувати стандартний процес розгортання для перевірки змін

перед викладанням їх у продуктивне середовище. Ви можете масштабувати монолітну програму горизонтально шляхом запуску декількох фізичних серверів за балансувальником навантаження.

Монолітні програми можуть бути успішними, але все більше людей розчаровуються в них, особливо у світлі того, що все більше програм розгортаються в хмарі. Будь-які зміни, навіть найменші, вимагають перескладання та розгортання всього моноліту. З часом стає важче зберігати хорошу модульну структуру, зміни логіки одного модуля мають тенденцію впливати на код інших модулів. Масштабувати доводиться вся програма цілком, навіть якщо це потрібно тільки для одного модуля цієї програми.

Переваги мікросервісів виявилися досить сильними, щоб такі великі корпоративні гравці як Amazon, Netflix або eBay впровадили їх у свої системи.

- Висока стійкість до відмов: при падінні одного з сервісів, всі інші залишаються в строю. Таким чином, неполадки в окремих сервісах не завадять усьому робочому процесу.
- Гнучкість: можна спробувати впровадити нову технологію малою кров'ю. Це буде значно швидше і при невдачі відкотити зміни просто. Змінюючи локально один із сервісів, ми не ризикуємо всією системою і час, який потрібний для змін, менший.
- Простота: що менше коду (кожний окремий сервіс являє собою цілісну систему (рис 3.2), тому потрібно розбиратися у величезній кількості деталей, які не стосуються цієї конкретної функції), тим простіше програмістам розібратися, що як працює. До того ж, на це піде менше часу.
- Легкість виведення написаного коду на роботу. Невелика кількість коду забезпечує швидку розгортання.

- Масштабованість. Найнеобхідніші та потрібні сервіси можна доповнити та розширити, коли з'явиться така необхідність. Вся система при цьому залишається незмінною.

Незважаючи на популярність і велику кількість плюсів мікросервісної архітектури, вона має і мінуси.

- Зв’язок між самими сервісами складний. Так як кожен функціональний елемент ізольований, потрібна особлива ретельність при побудові між ними грамотної комунікації, адже їм у будь-якому випадку доведеться обмінюватися запитами та відповідями один з одним. Зрозуміло, що зі збільшенням кількості сервісів складність у побудові їхнього повідомлення зростатиме.
- Зростання числа сервісів також тягне у себе зростання кількості баз даних, із якими ці послуги співвідносяться, оскільки, на відміну монолітної архітектури, мікросервіси використовують жодну загальну базу даних.
- Складність тестування полягає в тому, що спочатку потрібно окремо розбиратися з кожним сервісом, а потім тестувати коректну взаємодію його з іншими мікросервісами.
- Мікросервіси гірше підходять для використання всередині окремих організацій, для них вони можуть виявитися невиправдано складними у застосуванні, тоді як для масових інтернет-сервісів вони підходять добре.

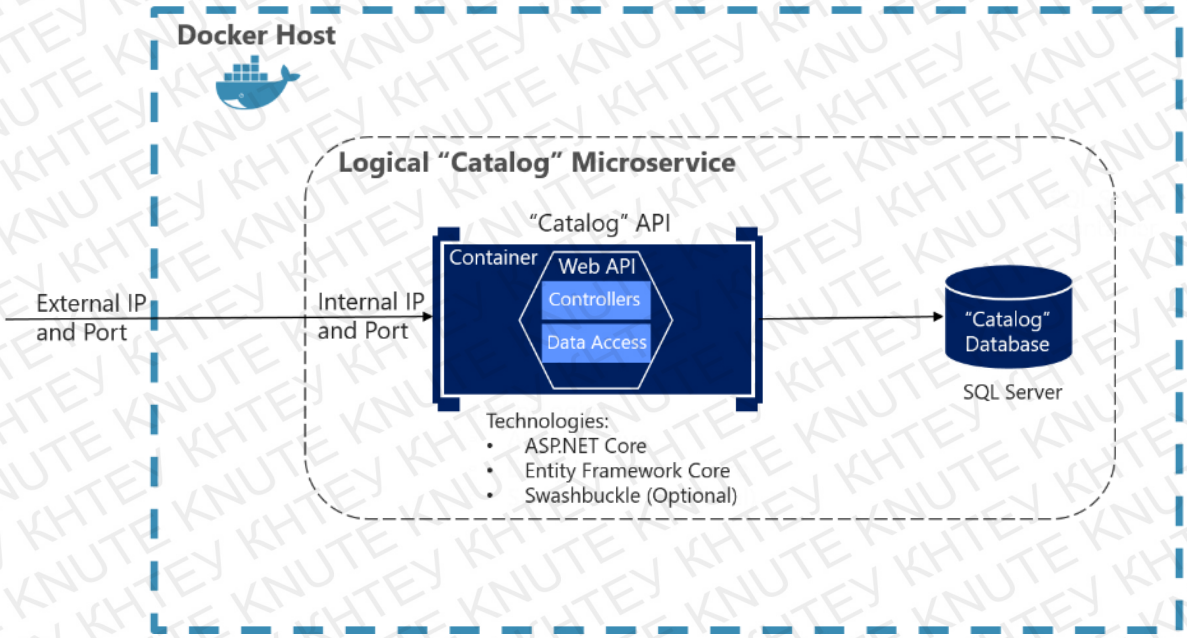


Рис. 3.2 Структура простого CRUD мікросервісу

3.2. Нефункціональні вимоги

Специфікація вимог *[8]*- це процес документування вимог, в якому повність описується поведінка всієї системи. Існує багато видів документації, основні з них це: списки, прототипи, сценарії.

Сховище даних :

- Надійне
- Підтримує великі сесії активності
- Невеликий час на розробку
- Можливість легкого масштабування
- Гнучкість
- Підтримка довгого життєвого циклу ПЗ

На даний момент існує багато СУБД, які в тій чи іншій мірі задовольняють зазначені системні вимоги, найбільш вірогідні претенденти це: MySql, MsSql, SqLite, PostgreSql, MongoDb.

PostgreSql – відповідає більшості вимогам але основні переваги у використанні цієї СУБД, проявляються у великих проектах, а на малих масштабах конкуренти показують себе краще.

MySql – найпопулярніша СУБД, має можливість інтегруватися з .net платформою, в цілому задовольняє всі вимоги.

SqLite – портативна і компактна СУБД, в цілому задовольняє вимогам, але направлена на вбудовані рішення та на мобільну розробку, тому має обмежену здатність до розширення.

MongoDb – документо-орієнтована СУБД, її перевага у простоті та швидкості створення БД, але має проблеми в програмах з довгим життєвим циклом(тому що схема БД визначається самою програмою).

MsSql – нативне рішення для платформи .net, інтегрована в EF Core *[2]*, також задовольняє всі вимоги

Вирішальне рішення у виборі СУБД, в даному випадку, грає досвід роботи з конкретним представником. Найбільше досвіду роботи наявно з MongoDb та MsSql, а так як MsSql це нативне рішення для .net платформи і має перевагу у довжині життєвого циклу то вибір СУБД падає на MsSql.

Рівень доступу до даних:

- Можливість легкої зміни джерела даних
- Масштабованість
- Можливість заміни на mock об’єкт для тестування
- Кросплатформеність

Цей рівень передбачує створення локальної бази даних, також буде правильним передбачити можливість зміни, або добавлення нового джерела даних. Тому

необхідно сильно відокремити рівень від інших для забезпечення слабкої зв'язності. Реалізація патернів «repository» та «unit of work» забезпечить можливість легкого масштабування та зміни джерела даних. Для забезпечення кросплатформеності можна використати платформу asp.net core *[1]* яка може працювати на всіх сучасних операційних системах, та написана на мові c# Рівень
Рівень бізнес логіки:

- Слабка зв'язність
- Можливість розширення
- Кросплатформеність

Цей рівень інкапсулює в собі основні логічні операції та перетворення, що забезпечують виконання основних функцій системи. Даний рівень, називають «високорівневим», так як він не займається вводом-виводом напряму, та складається з інтерфейсів, сервісів, конвертерів, інтеракторів. Всі ці шаблони описують певні патерни роботи з даними. В якості технології було обрано платформу .NET 5 на мові c#.

.NET (раніше відома як .NET Core) - це модульна платформа для розробки програмного забезпечення з відкритим вихідним кодом. Сумісна з операційними системами як Windows, Linux і macOS. Була випущена компанією Microsoft. Платформа має власну спільноту на GitHub. Підтримує такі мови програмування: C#, Visual Basic .NET (частково) та F#.

Рівень представлення:

- Можливість додати різні види інтерфейсу
- Зручний та інформативний дизайн інтерфейсу
- Кросплатформеність

В контексті розробки такої системи рівень представлення має досить другорядний характер, він має забезпечувати зрозумілий інтерфейс можливість записувати аудіо файли в форматі .wav та задовольняти загальні вимоги до

сучасного інтерфейсу. В якості технології я обрав бібліотеку Xamarin Forms для будування кростплатформених мобільних застосунків.

Xamarin.Forms — це фреймворк інтерфейсу користувача з відкритим вихідним кодом. Xamarin.Forms дозволяє розробникам створювати програми Xamarin.Android, Xamarin.iOS і Windows з єдиної спільної кодової бази.

Xamarin.Forms дозволяє розробникам створювати користувацькі інтерфейси в XAML із кодом на C#. Ці інтерфейси відображаються як ефективні власні елементи керування на кожній платформі.

3.3. Функціональні вимоги

Для специфікації функціональних вимог, було обрано метод створення сценаріїв використання.

В першій таблиці описується процес отримання інформації про пристрій.

Таблиця 3.1

Сценарій використання №1

Назва процесу	Реєстрація сімейства
Короткий опис	Створення сімейства
Суб'єкт	Оператор системи
Передумова	Унікальна назва та пароль до сімейства
Основний потік	Оператор відкриває вікно реєстрації сімейства, вводить унікальне ім'я та пароль, натискає кнопку створити сім'ю. Сім'я створена
Альтернативний потік	Оператор відкриває вікно реєстрації сімейства, вводить унікальне ім'я та пароль, натискає кнопку створити сім'ю. Помилка - така сім'я вже існує

Таблиця 3.2

Сценарій використання №2

Назва процесу	Додавання голосового профілю користувача
Короткий опис	Процес зчитки голосових параметрів користувача для можливості подальшої ідентифікації та автентифікації його за голососм
Суб'єкт	Оператор системи
Передумова	Користувач позитивно відповідає на пропозицію зчитати голос
Основний потік	1.) Оператор натискає на кнопку створити VoiceId 2.) Починається запис голосу 3.) Користувач читає текст з екрану. 4.) Натискає кнопку прочитано 5.) На екран виводиться кількість повторів що ще необхідно записати. 6.) Повторюються кроки з 2-го по 5-тий поки кількість повторів не стане нульовою. Голосовий профіль створено
Альтернативний потік	1. Оператор натискає на кнопку створити VoiceId 2. Починається запис голосу 3. Користувач читає текст з екрану. 4. Натискає кнопку прочитано 5. На екран виводиться кількість повторів що ще необхідно записати. 6. Повторюються кроки з 2-го по 5-тий поки кількість повторів не стане нульовою. Помилка – неможливо розпізнати мовлення.

Таблиця 3.3

Сценарій використання №3

Назва процесу	Змінити температуру термостату
Короткий опис	Голосова команда зміни температури термостату в мікросервісу, що абстрагує в собі роботу систему управління опаленням
Суб'єкт	Оператор системи
Передумова	Користувач має голосовий профіль
Основний потік	Користувач натискає кнопку запису, віддає команду голосом, «scale temperature up on 2 points», система отримує голосову команду, автентифікує та авторизує користувача за голосом, відправляє голосову команду на перетворення в текст, з тексту виділяються ключові точки, відправляється запит на підняття температури, користувач має право на таку дію, мікросервіс виконує запит , повертається результат про успішне виконання запиту
Альтернативний потік А	Користувач натискає кнопку запису, віддає команду голосом, «scale temperature up on 2 points», система отримує голосову команду. Помилка – користувача не автентифіковано.
Альтернативний потік Б	Користувач натискає кнопку запису, віддає команду голосом, «scale temperature up on 2 points», система отримує голосову команду, автентифікує та авторизує користувача за голосом, відправляє голосову команду на перетворення в текст, з тексту виділяються ключові точки, відправляється запит на підняття температури, користувач не має права на таку дію. Помилка - користувача не авторизовано.

Таблиця 3.4

Сценарій використання №4

Назва процесу	Запит в пошуковий сервіс
Короткий опис	Голосова команда з запитом на пошук чогось в пошуковому сервісі
Суб'єкт	Оператор системи
Передумова	Користувач має голосовий профіль
Основний потік	Користувач натискає кнопку запису, віддає команду голосом, «find, what is the software», система отримує голосову команду, автентифікує та авторизує користувача за голосом, відправляє голосову команду на перетворення в текст, з тексту виділяються ключові точки, відправляється запит в мікросервіс веб пошуку, користувач має право на таку дію, мікросервіс виконує запит , повертається результат про успішне виконання запиту
Альтернативний потік А	Користувач натискає кнопку запису, віддає команду голосом, «find, what is the software», система отримує голосову команду, автентифікує та авторизує користувача за голосом. Помилка – користувача не автентифіковано
Альтернативний потік Б	Користувач натискає кнопку запису, віддає команду голосом, «find, what is the software», система отримує голосову команду, автентифікує та авторизує користувача за голосом, відправляє голосову команду на перетворення в текст, з тексту виділяються ключові точки, відправляється запит в мікросервіс веб пошуку, користувач має право на таку дію. Помилка – користувача не авторизовано

Таблиця 3.5

Сценарій використання №5

Назва процесу	Додавання нового аккаунту
Короткий опис	В систему додається новий профіль користувача
Суб'єкт	Оператор системи
Передумова	Додавання нового користувача
Основний потік	Відкривається вікно регістрації користувача, користувач вводить пароль та унікальній логін. Натискає кнопку зареєструватися. Аккаунт створено.
Альтернативний потік	Відкривається вікно регістрації користувача, користувач вводить пароль та унікальній логін. Натискає кнопку зареєструватися. Помилка – такого користувача вже зареєстровано.

Таблиця 3.5

Сценарій використання №5

Назва процесу	Видалення аккаунту
Короткий опис	В системі видаляється профіль користувача
Суб'єкт	Оператор системи
Передумова	Видалення користувача
Основний потік	Відкривається вікно підтвердження вибору. Вводиться пароль, та повторюється знову. Система валідує пароль. Валідація успішна. Система архівує профіль користувача. Запускає планувальник видалення запису на один місяць.
Альтернативний потік	Відкривається вікно підтвердження вибору. Вводиться пароль, та повторюється знову. Система валідує пароль. Валідація не успішна. Помилка - пароль введено неправильно

3.4. Системні вимоги

Серверна частина, очікуються вимоги для мікросервісу авторизації та виконання команд.

- Операційна система Linux в контейнерах Docker,
- Версія платформи .Net 5
- Мінімальна оперативна пам’ять RAM 1 gb
- Місце на диску 10gb

Серверна частина мікросервісу пошуку в веб, та управління температурою опалення:

- Операційна система Linux в контейнерах Docker,
- Версія платформи .Net 5
- Мінімальна оперативна пам’ять RAM 512 mb
- Місце на диску 100 mb

Реальний пристрій управління температурою опалення:

- Wi-fi модуль
- Мікроконтроллер
- Пам’ять 10-20 мб

3.5. Висновок до розділу 3

В ході розробки вимог до архітектури та до самої системи автентифікації людини за допомогою голосу, було створено специфікації у вигляді сценаріїв використання, та задокументовано нефункціональні та системні вимоги у вигляді списку. Було проаналізована основні інструменти в розробці програмного забезпечення, що можуть бути використані для досягнення поставлених вимог.

РОЗДІЛ 4
ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ АВТЕНТИФІКАЦІЇ ЛЮДИНИ

4.1. Практичні особливості проектування та розробки складних систем

- Рівень представлення – в контексті розробки такої системи рівень представлення має досить другорядний характер, він має забезпечувати мінімально зрозумілий інтерфейс та швидке оновлення даних. Особливістю можна вважати необхідність миттєвого оновлення інтерфейсу після надходження даних від нижчих рівнів системи(рівня логіки). Так як система не передбачує необхідності в багатьох операторах (адміністраторах), відпадає необхідність відділяти інтерфейс від основної системи, в окрему автономну одиницю, для реалізації кросплатформеного представлення.
- Рівень логіки – цей рівень використовує нижчі рівні, які можуть працювати з даними або з мережевими вузлами. Тому більшість операцій мають бути асинхронними, щоб не блокувати основний потік роботи програми, допускається використання окремих потоків для використання різних модулів системи. Необхідно передбачити можливість розширення списку використовуваних технологій, забезпечити слабку зв'язність між компонентами системи.
- Рівень мережевих операцій – на даному рівні реалізуються основні методи по роботі з мережевими вузлами. Особливістю даного рівня є те, що існує вірогідність розширення списку використовуваних технологій передачі інформації, тобто розширення функціоналу даного рівня не повинне впливати на інші рівні системи. Рівень доступу до даних – передбачує створення локальної бази даних, також буде правильним передбачити можливість зміни, або

добавлення нового джерела даних. Тому необхідно сильно відокремити рівень від інших для забезпечення слабкої зв'язності.

На основі виділених мною особливостей можна скласти попередню діаграму архітектури (рис. 4.1). Така діаграма не остаточна, а потрібна для більш наглядної демонстрації основних модулів системи і зв'язків між ними.

Передача даних крізь модуль може бути реалізована, за допомогою паттерна DTO(Data transfer object), з використанням автомапера властивостей для протягування даних крізь рівні.

Також для зменшення зв'язаності класів та модулів, можна використати паттерн «Інверсія управління» (Inversion of controll), з його реалізацією у вигляді фреймворку «Впровадження залежностей»(Dependency injection) який буде керувати потоком виконання, передавати потрібні залежності модулям.

У процесі проектування необхідно звертати увагу на базові принципи ООП *[4][5]*, та принципи SOLID*[9]*, дотримання цих правил в проектування великої системи, забезпечить виконання основних вимоги до якості ПЗ.

4.2. Проектування сервісу автентифікації

Враховуючи вимоги до бази даних було обрано реляційну БД. Для зберігання даних в таких БД, використовують таблиці та систему первинних та вторинних ключів для ідентифікації записів та відображення зв'язків між ними.

SQL Server *[2]* є однією з найбільш популярних систем управління базами даних (СКБД) в світі. Дана СУБД підходить для самих різних проектів: від невеликих додатків до великих високонавантажених проектів.

SQL Server був створений компанією Microsoft. Перша версія вийшла в 1987 році. А поточною версією є версія 16, яка вийшла в 2016 році і яка буде використовуватися в поточному керівництві.

SQL Server довгий час був винятково системою управління базами даних для Windows, проте починаючи з версії 16 ця система доступна і на Linux.

SQL Server характеризується такими особливостями як:

- Продуктивність. SQL Server працює дуже швидко.
- Надійність і безпека. SQL Server надає шифрування даних.
- Простота. З даної СУБД відносно легко працювати і вести адміністрування.

Таблиця 4.1

Опис таблиці User

Назва	Обмеження
User	Назва таблиці
UserId	Первинний ключ, унікальний ідентифікатор(Guid)
Login	Назва, стрічка довжина 256 символів
PasswordHash	Хеш паролю,
PasswordSalt	Сигнатура шифрування паролю, масив байтів
Role	Роль користувача, цілочислений 16біт
VoiceidAccessType	Тип доступу до голосової верифікації
VoiceId	Ключ голосового профілю, стрічка 50 символів
FamilyId	Зовнішній ключ на таблицю Family, унікальний ідентифікатор(Guid)

Таблиця 4.2

Опис таблиці UsersFamily

Назва	Обмеження
UsersFamily	Назва таблиці
FamilyId	Первинний ключ, унікальний, цілочислений
PasswordHash	Хеш паролю, масив байтів
Name	Назва, стрічка довжина 100 символів
PasswordSalt	Сигнатура шифрування паролю, масив байтів

Визначившись зі структурою та полями в БД, можна приступати до процесу розробки архітектури мікросервісу автентифікації. Потрібно зазначити, що мова c#

та платформа .Net повністю кросплатформені, тому вважаю за розумне винести моделі рівня передачі даних в окрему бібліотеку, звідки їх можливо імпортувати в наш мікросервіс, в проєкт з користувацьким інтерфейсом чи в інший застосунок, так модель досить добре узгоджується з паттернами DDD або предметно орієнтованого проєктування.

Предметно-орієнтоване проектування (рідше проблемно-орієнтоване, англ. domain-driven design, DDD) - це набір принципів та схем, спрямованих на створення оптимальних систем об'єктів. Зводиться створення програмних абстракцій, які називаються моделями предметних областей. У ці моделі входить бізнес-логіка, що встановлює зв'язок між реальними умовами застосування продукту і кодом.

Предметно-орієнтоване проектування не є якоюсь конкретною технологією чи методологією. DDD - це набір правил, які дозволяють приймати правильні проектні рішення. Даний підхід дозволяє значно прискорити процес проектування програмного забезпечення у незнайомій предметній галузі.

Для візуального представлення проектованої архітектури було вирішено створити діаграму класів на UML.

Діаграма класів UML – це графічне позначення, яке використовується для побудови та візуалізації об’єктно-орієнтованих систем.

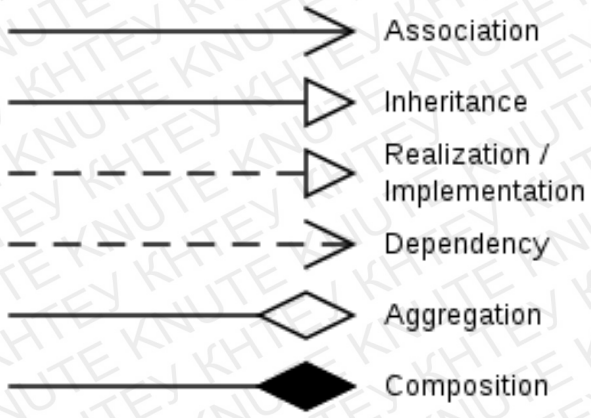


Рис. 4.1 Типи зв’язків в діаграмі UML

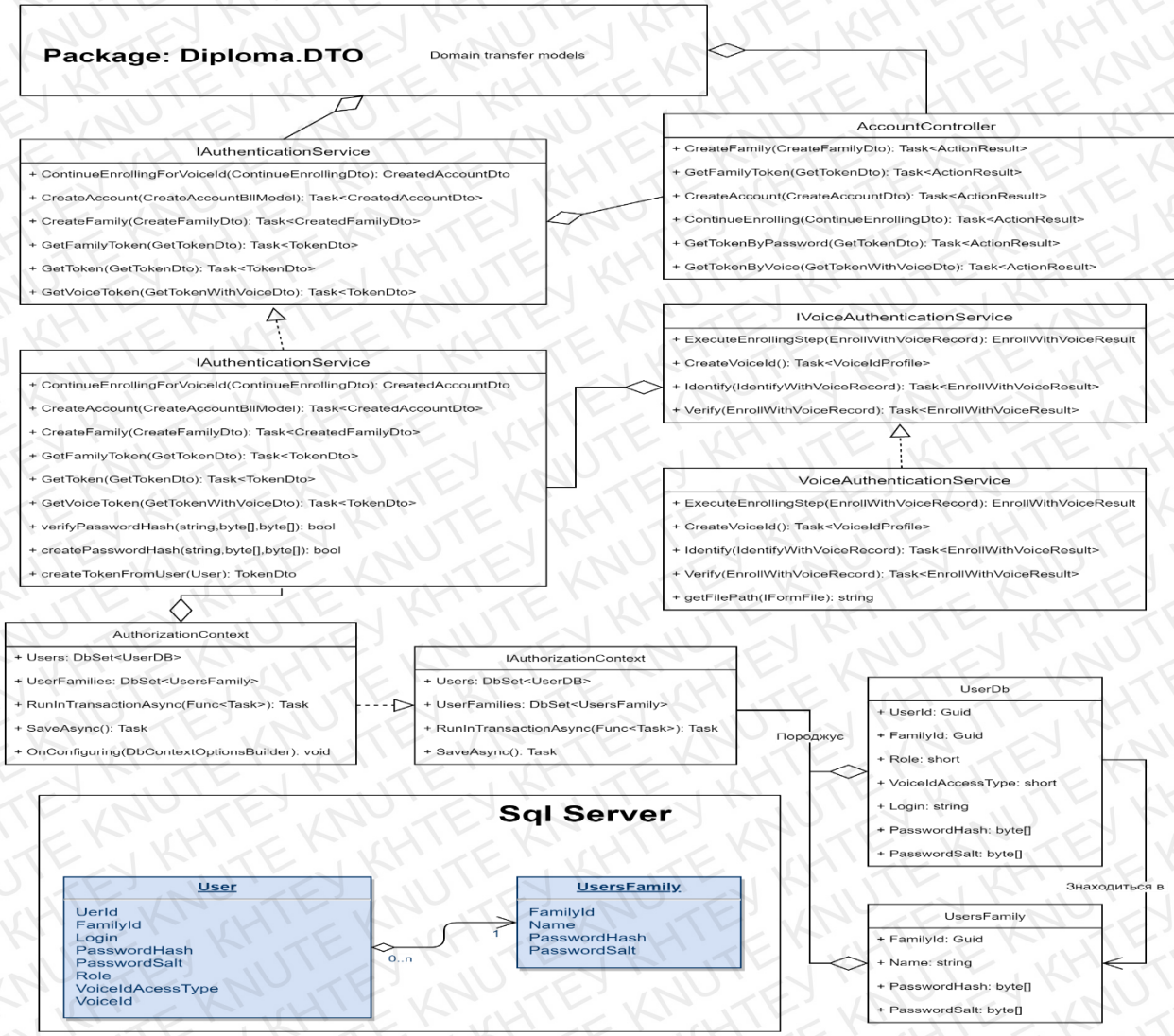


Рис. 4.2 Діаграма класів мікросервісу автентифікації

Діаграма класів на уніфікованій мові моделювання (UML) — це тип діаграми статичної структури, яка описує структуру системи, показуючи: класи, їх атрибути, операції (або методи), і відносини між об'єктами.

В ході проектування було визначено структуру бази даних та створено модель структури мікросервісу (рис. 4.2).

4.3. Розробка мікросервісу автентифікації

Перед початком розробки мікросервісу потрібно розгорнути сервер БД. Після розгортання потрібно створити базу даних та таблиці. Скріпт створення елементів на мові Transact-Sql :

```
create database DiplomaIdentity

use DiplomaIdentity

Create table [User](
        [UserId] uniqueidentifier  primary key,
        [Login] nvarchar(50) not null unique,
        PasswordHash varbinary(max) not null,
        PasswordSalt varbinary(max) not null,
        [Role] smallint not null,
        [VoiceId] nvarchar(50) not null unique,
        [VoiceIdAccessType] smallint not null,
        [FamilyId] uniqueidentifier not null,
);

Create table [UsersFamily] (
        [FamilyId] uniqueidentifier primary key,
        [Name] nvarchar(50) not null unique,
        PasswordHash varbinary(max) not null,
        PasswordSalt varbinary(max) not null,
);
```

Потім створити класи описані в моделі архітектури, також потрібно визначитися з набором сервісів що потрібно зареєструвати в DI-контейнері як сервіс, також потрібно додати ініціалізацію контексту БД з використанням захищеної стрічки доступу до серверу SQL (рис. 4.3).

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddHttpClient();
    services.AddCutomAuthentication(Configuration);
    services.AddControllers();

    services.AddDbContext<AuthenticationContext>(options =>
    {
        options.UseSqlServer(Configuration.GetConnectionString("AuthenticationContext"));
    });
    services.AddScoped<IVoiceAuthenticationService, VoiceAuthenticationService>();
    services.AddScoped<IAuthenticationService, AuthenticationService>();
    services.AddScoped<IAuthenticationContext, AuthenticationContext>();
    services.AddScoped<IAuthorizationSettingsService, AuthorizationSettingsService>();
}
```

Рис. 4.3 Реєстрація сервісів, бази даних,

Наступним кроком має бути налаштування автентифікації та авторизації. Для цього потрібно створити метод розширення до IServiceCollection для винесення

роботи з автентифікацією та авторизацією в окремий клас. Створивши розширення потрібно описати структуру JWT та створити нову політику авторизації з обов'язковим полем FamilyId (ідентифікатор сімейства користувачів, необхідний для обмеження кількості голосових аккаунтів з якими необхідно порівнювати вхідний запис)

```
public static IServiceCollection AddCutomAuthentication(this IServiceCollection serviceCollection, IConfiguration configuration)
{
    var optionsHelper = new AuthorizationSettingsService(configuration);

    var optionsData = optionsHelper.GetAuthOptions();

    serviceCollection.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
            .AddJwtBearer(options =>
            {
                options.RequireHttpsMetadata = false;
                options.TokenValidationParameters = new TokenValidationParameters
                {
                    ValidateIssuer = true,
                    ValidIssuer = optionsData.ISSUER,
                    ValidateAudience = true,
                    ValidAudience = optionsData.AUDIENCE,
                    ValidateLifetime = true,
                    IssuerSigningKey = optionsData.KEY,
                    ValidateIssuerSigningKey = true,
                };
            });
    serviceCollection.AddAuthorization(option =>
    {
        option.AddPolicy("Family", policy =>
        {
            policy.RequireClaim("FamilyId");
        });
    });
    return serviceCollection;
}
```

Рис. 4.4 Створення та реєстрація схеми та політики авторизації

Реалізувавши необхідні операції по налаштування проекту потрібно реалізувати всі необхідні методи в класах та протестувати їх роботу.

4.4. Проектування мікросервісу виконання операцій

Перед проектування мікросервісу виконання операції потрібно визначити основні функції, даного сервісу:

- Перевід голосу в текст
- Розпізнавання текстової команди
- Виконання команди

Для того щоб задовольнити дані вимоги, потрібно створити окремий сервіс для виконання даних алгоритмів також необхідно передбачити провайдер

мережевих операцій з іншими мікросервісами. Для реалізації мережевих операцій можливо використовувати IHttpClient клас-поставник мережевих запитів платформи .Net, або бібліотеку Refit що дозволяє автогенерувати реалізації в процесі розгортання проєкту. Для кращого співставлення тексту команди та можливої операції можна створити карту порівняння команди, в виді дерева або списка, збереженого в базу даних або серіалізованого в файл. Хоча для таких задач варто створювати нейронну мережу, так як процес аналізу сенсу тексту досить складний і простим алгоритмом неможливо покрити всі варіанти промови однієї і тієї ж команди, але в рамках даної роботи це не має практичної необхідності, тому було прийнято рішення створити базу даних з кореляціями ключових слів та можливих команд.

4.5. Розробка мікросервісу виконання операцій

Реалізація даного сервісу включає в себе кроки такі як і в розробці попереднього сервісу, але потрібно звернути увагу, на створення бази даних, якщо проаналізувати створення контексту бази даних та реєстрація політики автентифікації автентифікації, то там зустрічається велика кількість загального коду який повторюється в кожному з мікросервісів, тому я вважаю що слідуючи принципам SOLID, то потрібно винести сервіси отримання налаштувань автентифікації (рис. 4.6), та базовий контекст бази даних в окрему бібліотеку коду який можливо використовувати в інших мікросервісах.

Також потрібно винести в окремі бібліотеки класи що описують моделі рівня трансферу даних. Таке рішення зумовлено необхідністю забезпечення легкого доступу до моделей DTO з усіх інших мікросервісів, також окрему бібліотеку можливо легко інтегрувати в інші системи через пакетний менеджер nugget.

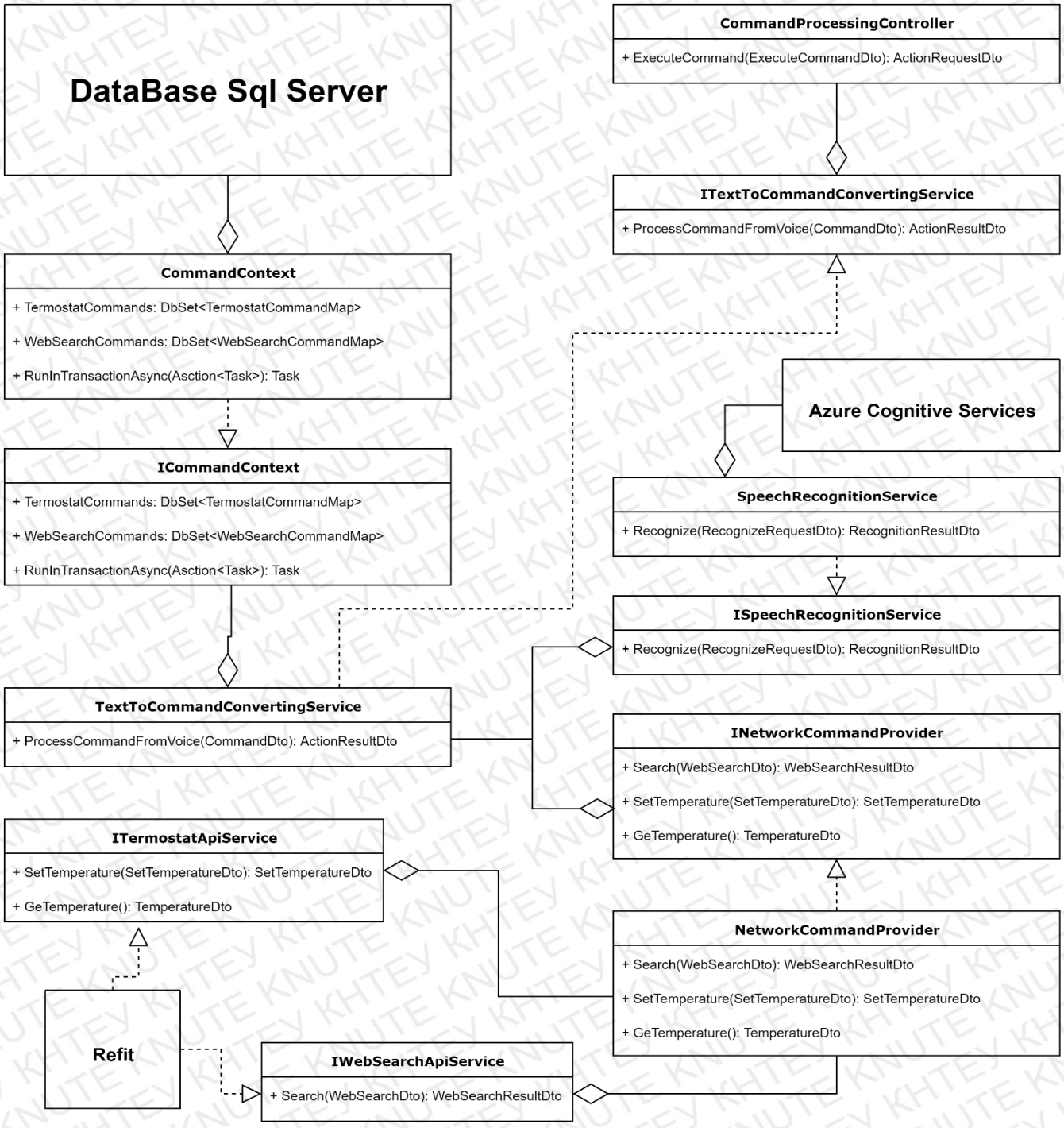
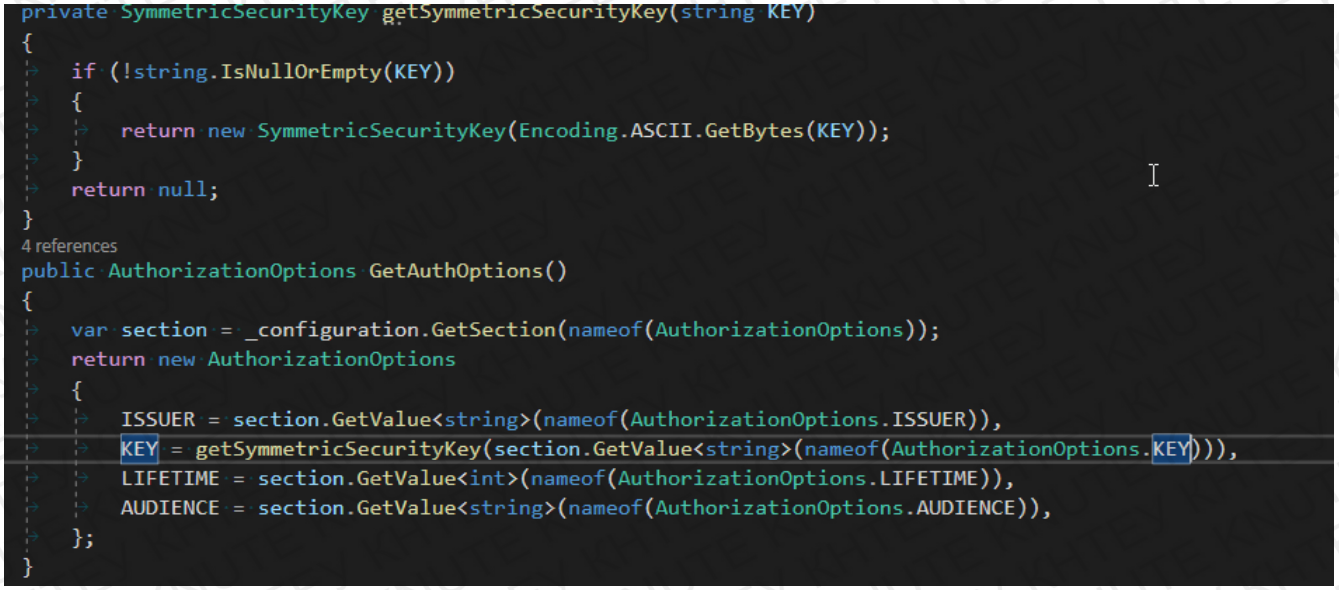


Рис. 4.5 Діаграма класів мікросервісу виконання операцій



```
private SymmetricSecurityKey getSymmetricSecurityKey(string KEY)
{
    if (!string.IsNullOrEmpty(KEY))
    {
        return new SymmetricSecurityKey(Encoding.ASCII.GetBytes(KEY));
    }
    return null;
}
4 references
public AuthorizationOptions GetAuthOptions()
{
    var section = _configuration.GetSection(nameof(AuthorizationOptions));
    return new AuthorizationOptions
    {
        ISSUER = section.GetValue<string>(nameof(AuthorizationOptions.ISSUER)),
        KEY = getSymmetricSecurityKey(section.GetValue<string>(nameof(AuthorizationOptions.KEY))),
        LIFETIME = section.GetValue<int>(nameof(AuthorizationOptions.LIFETIME)),
        AUDIENCE = section.GetValue<string>(nameof(AuthorizationOptions.AUDIENCE)),
    };
}
```

Рис. 4.6 Методи сервісу отримання налаштувань автентифікації

Надалі потрібно розробити класи та методи описані в моделі архітектури мікросервісу, доповнюючи їх необхідними приватними методами для ефективного виконання алгоритму, та забезпечення структури у відповідності до стандартів чистого коду.

4.6. Проектування та розробка мікросервісів виконання команд, та користувацького інтерфейсу

Мікросервіс веб пошуку в рамках розробки проектування архітектури системи даний модуль має демонстративну функцію і носить репрезентативний характер, тому було прийнято рішення створити простий мікросервіс що імітує роботу сервісу пошуку, при цьому сам мікросервісний підхід дозволяє швидко та безболісно замінити модуль на реальний, або доробити існуючий без перекомпіляції всіх інших проєктів. Такий сервіс буде мати один контролер з кінцевою точкою яка віддає умовний результат пошуку, в разі успішної авторизації, та 401 статус код в іншому разі.

Мікросервіс управління термостатом – даний мікросервіс також може бути реалізований як імітація роботи справжнього пристрою, також обрана архітектура

дає можливість замінити такий модуль на реальний без прив'язки до мови або платформи на якій працює сам пристрій. Даний мікросервіс буде містити контролер та методи, що дозволяють отримати показники температури та змінити їх. Такої поведінки досить щоб зімітувати реальний пристрій.

Користувацький інтерфейс – в контексті даної системи інтерфейс потрібен для забезпечення більш зручної демонстрації роботи процесу автентифікації та виконання команд голосом, тому було вирішено створити простий користувацький інтерфейс з використанням технології Xamarin.Forms що дозволяє створювати мобільні додатки на мові c#, дає інструменти легко та ефективно використовувати мікрофон пристрою, та підтримує більшість кросплатформених фреймворків платформи .Net таких як Refit. Інтерфейс містить сторінки: реєстрації сімейства, реєстрації користувача, створення голосового профілю, виконання голосової команди. Даний перелік вікон забезпечує зручне користування системою.

4.7. Висновок до розділу 4

В ході проектування та розробки було розглянуто основні стандарти, принципи та методи створення складних програмних систем засобами об'єктно орієнтованої мови програмування, було спроектовано модель програмної системи та розроблено робочий прототип системи у відповідності до моделі.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Дипломна робота містить детальний аналіз проблем у проектуванні систем з нетривіальними способами взаємодії з користувачем, виявлено недоліки стандартних підходів до проектування систем, вдосконалено існуючого протоколу автентифікації.

Розглянуто наявні методи проектування та розробки систем автентифікації .

Розроблено специфікації вимог до системи, визначено основну архітектуру та основні технології, які задовольняють визначеним вимогам.

Спроектовано та розроблено прототип програмного продукту у відповідності до поставлених вимог.

Розроблена програмна система має гнучку мікросервісну архітектуру, дозволяє не залежати від конкретного середовища виконання та дає можливість легко додавати або змінювати модулі без шкоди для інших частин системи. Такий архітектурний підхід дозволяє виявити сервіси з найбільшим навантаженням та легко масштабувати їх в ширину, для забезпечення стабільної роботи з використанням найменшої кількості ресурсів.

Створений програмний продукт розширює можливості голосового управління, дає можливість автентифікувати та авторизувати користувача, та може бути легко інтегрована в інші системи, як вже готові так і нові.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ASP.NET Documentation. Learn to use ASP.NET Core to create web apps and services [Електронний ресурс], 2020. Режим доступу : https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1
2. Entity Framework core documentation. Modern object-database mapper for .NET [Електронний ресурс], 2020. Режим доступу : https://docs.microsoft.com/en-us/ef/
3. Стандарт системи та системної інженерії – визначення архітектури. [електронний ресурс] Дійсний від 2017 року Режим доступу: https://www.iso.org/standard/50508.html
4. Walter Sobkiw Sustainable Development Possible with Creative System Engineering / Walter Sobkiw New Jersey: CassBeth 2008.
5. McConnell, Steve Rapid Development: Taming Wild Software Schedules - McConnell, Steve / Redmond, WA: Microsoft Press 1996.
6. M.Fowler Who needs and architect? / M.Fowler – Addison-Weslye:IEEE Software 2003.
7. Philippe Kruchten Architectural Blueprints—The “4+1” View Model of Software / Architecture IEEE Software [інтернет ресурс] 1995 Режим доступу: https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf
8. Стандарт системи та системної інженерії ISO/IEC/IEEE 24765:2010 дійсний від 2010 року
9. Ерік Фрімен, Елізабет Робсон Head First. Патерни проєктування Видавництво ФАБУЛА 2020

10. Постанова Кабінету Міністрів України. Про затвердження Правил забезпечення захисту інформації в інформаційних, телекомунікаційних та інформаційно-телекомунікаційних системах. [електронний ресурс] 2006 режим доступу: https://zakon.rada.gov.ua/laws/show/373-2006-%D0%BF#Text
11. Протокол OpenId Connect. [електронний ресурс] 2014 Режим доступу: https://openid.net/connect/
12. Огляд служби Azure – "Speech" [електронний ресурс] 2021 Режим доступу: https://docs.microsoft.com/ru-ru/azure/cognitive-services/speech-service/overview

ДОДАТКИ

Додаток А

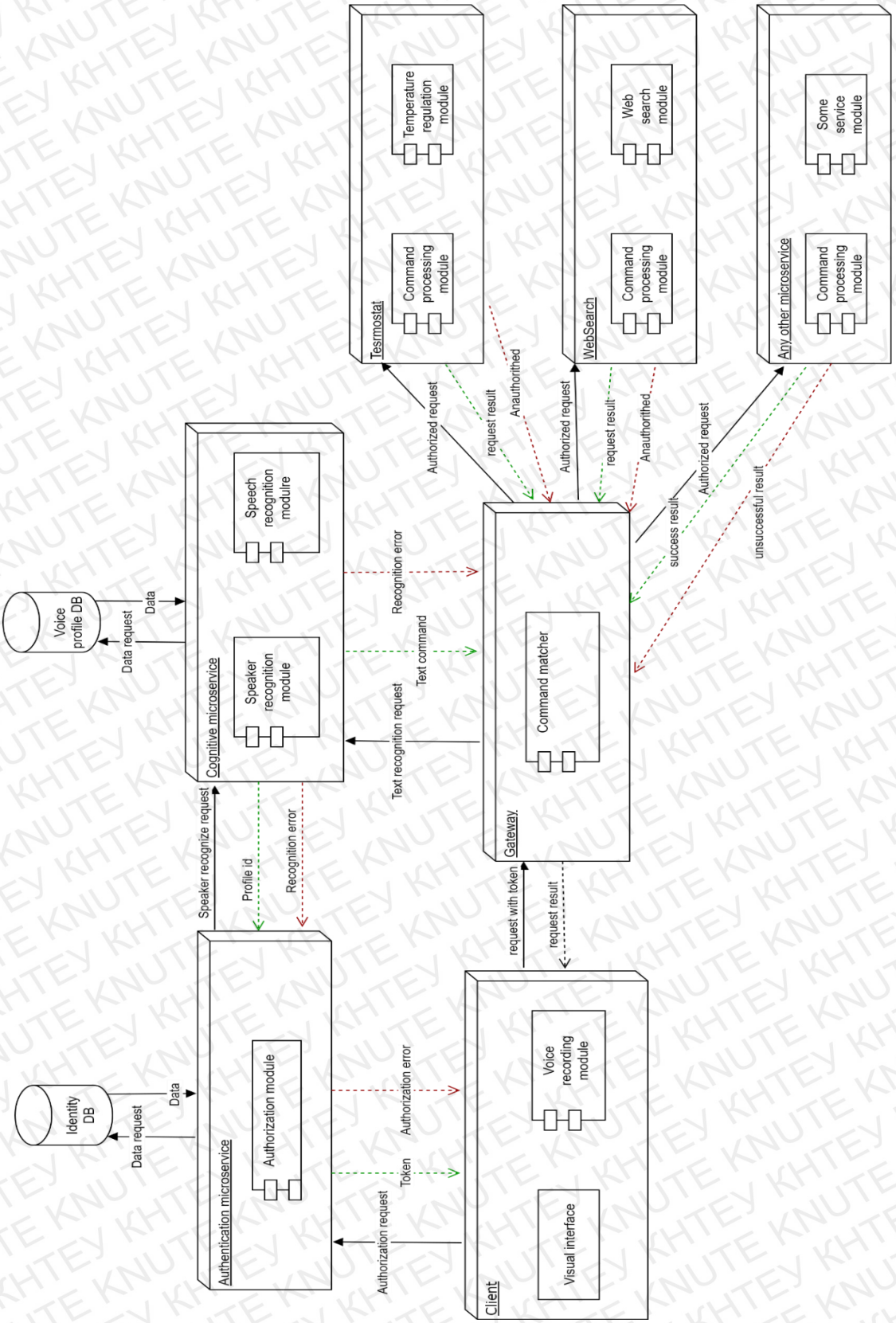


Рис. А. Діаграма пакетів архітектури системи автентифікації