НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Кафедра автоматики та управління в технічних системах

Курсова робота

3 дисципліни «Компоненти програмної інженерії - 3. Якість програмного забезпечення та тестування»

Тема: Розробка інтелектуальної системи розпізнавання людини по голосу

Керівник
ас. Хмелюк М.С.

Виконавець
ст. Яцук О.Д.
зал. книжка № ІТ-7129
гр. ІТ-71

«Допущений до захисту»

_____
(особистий підпис керівника)
«__» _____2020р.

_____
(особистий підпис виконавця)
«__» _____2020р.

Захищений з оцінкою

_____
(оцінка)

Члени комісії:

_____ _____
(особистий підпис) (розшифровка підпису)

_____ _____
(особистий підпис) (розшифровка підпису)

Київ – 2020

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра АВТОМАТИКИ ТА УПРАВЛІННЯ В ТЕХНІЧНІХ СИСТЕМАХ

Дисципліна «Основи програмування»

Курс 4 Група ІТ-71 Семестр VII

ЗАВДАННЯ

на курсову роботу студента

Яцука Олександра Дмитровича

1.Тема роботи Розробка інтелектуальної системи розпізнавання людини по голосу

2. Строк здачі студентом закінченої роботи 15.12.2020

3. Вихідні дані до роботи:

мова програмування C#, .net framework, база даних MSSql, веб-сервер IIS, середній час відгуку не повинен перевищувати 0.2 сек, gitlab у якості CI tool, Angular

4. Зміст розрахунково – пояснювальної записки (перелік питань, що підлягають розробці)

розробити основний веб-інтерфейс користувача з використанням Angular framework, розробити основну серверну частину з використанням .net framework, розробити мікросервіс для розпізнавання людини по голосу, розробити мікросервіс аутентифікації

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

Use-Case, діаграми компонент, станів, послідовностей для 2 мікросервісів та data-flow діаграма

6. Дата видачі завдання 07.10.2020

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів виконання курсової роботи	Термін виконання етапів роботи	Підписи або примітки
1.	Видача завдання	7.10.2020	
2.	Отримання та узгодження теми курсової роботи	20.10.2020	
3.	Архітектура проекту	25.10.2020	
4.	BDD сценарії	30.10.2020	
5.	Розбивка по спринтам (agile)	1.11.2020	
6.	Розробка необхідних діаграм	5.11.2020	
7.	Написання BDD тестів	8.11.2020	
8.	Юніт-тестування мікросервісів	15.11.2020	
9.	Кодування програми	22.11.2020	
10.	Написання інтеграційних тестів	24.11.2020	
11.	Стрес тестування системи	30.11.2020	
12.	Автотести (CI/CD)	2.12.2020	
13.	Розробка діалового інтерфейсу програми	2.12.2020	
14.	Selenium – тестування веб-застосунку	6.12.2020	
15.	Налагодження та перевірка програми	9.12.2020	
16.	Оформлення пояснювальної записки	13.12.2020	
17.	Захист курсової роботи	15.12.2020	

Студент _____ Яцук О.Д.

(підпис) (прізвище, ім’я, по батькові)

Керівник _____ Хмелюк М.С.

(підпис) (прізвище, ім’я, по батькові)

«____»_____2020р

АНОТАЦІЯ

Яцук О.Д. Розробка додатку для розпізнавання людей по голосу. КПІ ім. Ігоря Сікорського, Київ, 2020.

Робота містить 42 с. тексту, 14 рисунків, 1 таблицю, посилання на 11 літературних джерел, додатки.

Об’єктом розробки є додаток для розпізнавання людини по голосу. Основною метою програми є створення сервісу який дасть змогу додати ще один рівень безпеки, що буде базуватись на такому біометричному параметрі як голос, до будь-якого застосунку. Окрім цього, не менш важливим пунктом у роботі над курсовою було створення проекту, під час розробки якого можна було ознайомитись з нюансами все можливого тестування програмного застосунку.

У курсовій роботі розроблено бібліотеку для вирішення вище поставленої задачі, а також інтерфейс для взаємодії з нею. Взаємодіяти зі створеними функціями можна як і через публічний WEB API, так і використовуючи, розроблену за допомогою Angular Framework, front-end частину.

Основний функціонал програми написаний на мові програмування C#, яка є частиною .NET Framework та з використанням бази даних MS SQL Server. Щодо використання створеного сервісу, то він може бути корисним у тих моментах коли необхідно додати ще один рівень захисту у застосунку. Наявність публічного WEB API дає гнучкість представленому рішенню, що дозволяє необмежено використовувати розроблений функціонал у різного типу застосунках, наприклад, мобільні додатки, програми для ПК (персональний комп’ютер) і веб застосунки.

ЗМІСТ

ВСТУП

Кожного дня через сучасні засоби зв’язку проходить неймовірна кількість інформації. Якась її частина може знаходитись у вільному доступі, а якась в свою чергу потребує певний рівень захисту.

Досить тривалий час у світі саме пароль використовувався, як міра обмеження доступу до тих чи інших ресурсів, проте технології розвиваються, кількість інформації неупинно росте, а пароль не може забезпечити належного ступеню захисту. Це і призвело до виникнення біометричних засобів безпеки, адже вони надають, як надзвичайно точну ідентифікацію людини, так і неймовірно високий рівень захисту від злому.

Дана курсова робота створена з метою розробки сервісу, який би надав своєму користувачу можливість розпізнавати людину по голосу. Оскільки даний тип ідентифікації не вважається, поки що, достатньо надійним, адже не розроблено ще досконалого алгоритму який би враховував усі мінливі аспекти людського голосу, курсова є маленьким кроком у розвитку цієї технології.

Окрім цього, під час виконання курсової роботи, були вдосконалені навички розробки плану реалізації додатку, вміння покривати створений функціонал різноманітними видами тестів, а також були освоєні основні принципи забезпечення стабільної роботи програмного застосунку.

Програма розроблена з використанням сучасних підходів та технологій по розробці програмного забезпечення. Основними підходами є правильне планування очікуваного функціоналу з використанням різного роду діаграм. Що до технологій то у проекті було задіяно Angular, .Net Framework та велика кількість додаткових бібліотек для тестування. Документація для роботи з цими технологіями відповідно представлена у списку джерел [1, 2].

Курсова робота складається з наступних розділів: вступ, основні розділи, висновки, список використаних джерел та додатків. Графічна частина включає в себе діаграми необхідні для розробки програмного забезпечення, а також скріншоти готового продукту. Загальний обсяг 42 сторінки.

1 ПОСТАНОВКА ЗАДАЧІ

У межах даного курсового проекту основною задачею була розробка та тестування програмного продукту, який повинен забезпечувати біометричну ідентифікацію людини на основі її голосу, а також забезпечення стабільної роботи додатку та можливості безперешкодного його розширення на основі знань отриманих з курсу «Компоненти програмної інженерії».

Розроблений продукт повинен правильно реагувати на дії користувача, відповідати усім поставленим функціональним вимогам.

Перед початком створення програми були чітко виділені основні вимоги до функціоналу:

- реалізація ідентифікації користувача сервісу;
- можливість завантажити зразки голосу з метою створення бази даних голосів;
- можливість завантажити зразок голосу з метою розпізнавання людини основуючись на створеній базі голосів.

2 ПРОЕКТУВАННЯ СИСТЕМИ

Перед безпосереднім написанням коду застосунку слід попрацювати над коректним та повним описом очікуваного функціоналу. Для цього у наступних підрозділах буде розроблено діаграми, які в повній мірі дозволять описати задачі, що повинна виконувати програма, це в свою чергу спростить її подальшу розробку.

2.1 Use-Case діаграма

Діаграма застосувань дозволяє на поверхневому рівні описати основні аспекти функціоналу застосунку (дивись рисунок 2.1.1).

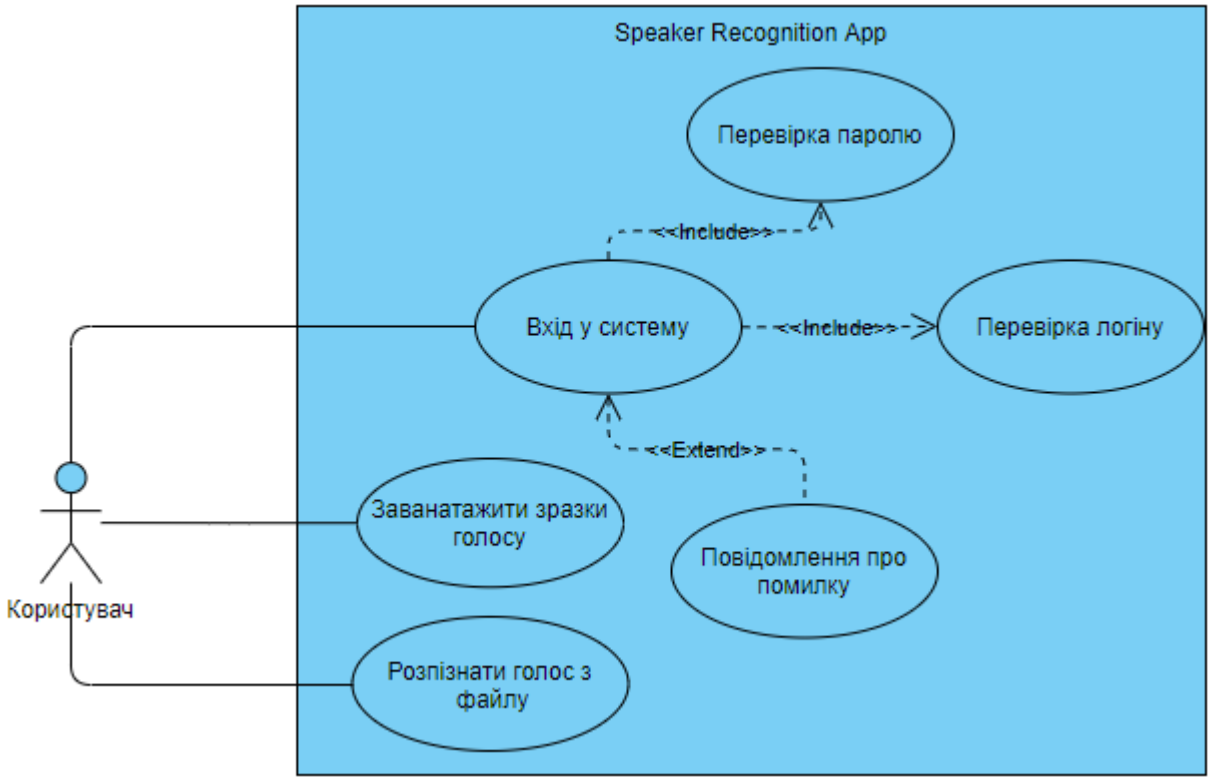

Рисунок 2.1.1 – Use-Case діаграма.

Відповідно до рисунку додаток має ідентифікувати користувача з перевіркою паролю та логіну, а в разі не правильного введення цих даних програма сповіщає користувача про помилку. Окрім цього після входу у систему користувачу доступні такі функції як розширення бази даних голосів шляхом завантаження нових зразків голосу, а також розпізнавання людини зі завантаженого файлу записаного голосу.

2.2 Діаграма послідовності

Діаграма послідовності являє собою певне пониження рівня абстракції представленого у діаграмі застосувань, а саме вона уособлює в собі більш чіткий та детальний опис сценаріїв описаних у попередньому розділі.

Таким чином для сервісу аутентифікації розроблений наступний хід подій (дивитись рисунок 2.2.1). Користувач приходить на сайт та хоче виконати вхід у систему, відповідно система видає користувачу форму для введення необхідних даних для його аутентифікації. Після заповнення даних та їх відправлення до сервісу, він звертається до бази даних формуючи запит основуючись на переданому логіні та паролі. По результатах запиту сервіс визначає чи присутній даний користувач у системі. У разі позитивного результату сервіс генерує токен який і передає користувачеві, а у випадку негативного результату створюється повідомлення про помилку.

Сценарії роботи з сервісом розпізнавання по голосу знаходяться у додатку А рисунки А.1 та А.2. Робота з функціями цього сервісу не значним чином відрізняється від вище описаної послідовності дій аутентифікації. Основну різницю представляє тип даних що передається до сервісу, а саме це аудіо файли, а також сховище даних, яке у цьому випадку представлено файловим сервером.

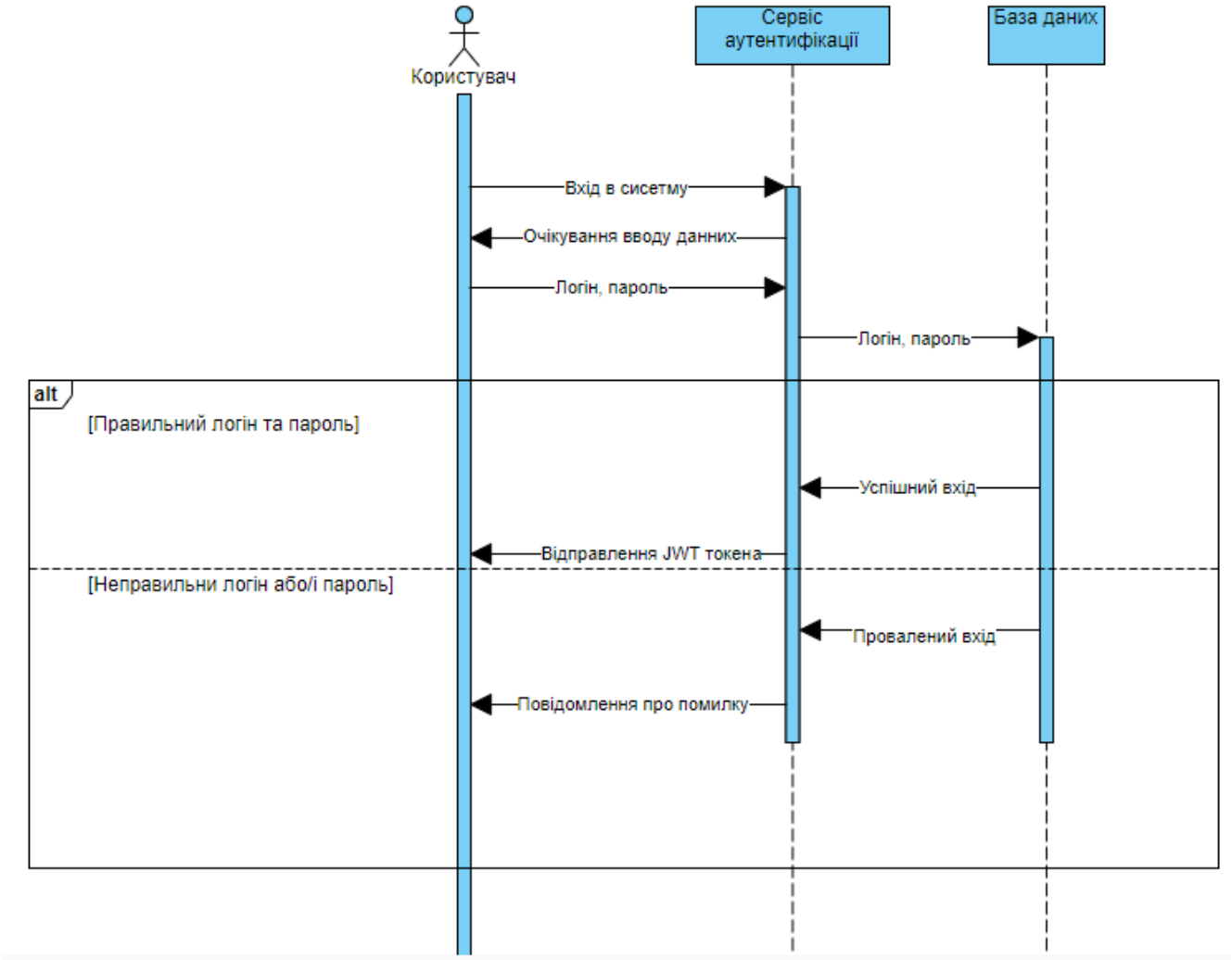

Рисунок 2.2.1 – Діаграма послідовності для сервісу аутентифікації.

На рисунку 2.1.1 частково зображено і те як опрацьовуються дані у системі, проте для більш детальнішого аналізу потоку інформації через програму використовуються data-flow діаграми. Розроблені діаграми зображені на рисунку А.7 у додатку А. Окрім цього з метою глибокого аналізу майбутньої системи хорошою практикою є розробка діаграми компонентів та діаграми станів, які представлені на рисунках А.8 та А.9 у додатку А відповідно.

2.3 BDD сценарії

Основною ідеєю даної методології є поєднання в процесі розробки чисто технічних інтересів та інтересів бізнесу, дозволяючи тим самим описати простими словами складний програмний функціонал. Для опису використовується предметно-орієнтована мова, основу якого становлять конструкції з природної мови, зрозумілі неспеціалісту, які зазвичай виражають поведінку програмного продукту і очікувані результати. Окрім цього даний підхід спрощує покриття застосунку тестами, адже надає розробнику уже готові сценарії тестування. В межах курсової роботи було розроблено наступні сценарії:

«Сценарій: Користувач виконує аутентифікацію з правильними даними
Дано користувач знаходиться на сторінці аутентифікації
Коли користувач водить правильний логін і пароль
І натискає кнопку «Log In»
Тоді користувач переходить на основну сторінку додатку
Сценарій: Користувач виконує аутентифікацію з неправильними даними
Дано користувач знаходиться на сторінці аутентифікації
Коли користувач водить неправильний логін і/або пароль
І натискає кнопку «Log In»
Тоді система відображає повідомлення про помилку
Сценарій: Користувач завантажує зразки голосу у систему
Дано користувач знаходиться на сторінці завантаження зразків голосу
Коли користувач натискає кнопку «Завантажити»
Тоді користувач переходить у діалог вибору файлів
Коли користувач вибирає необхідні файли
І натискає кнопку «Ok»
Тоді обрані файли завантажуються у систему
Сценарій: Користувач розпізнає голос з файлу
Дано користувач знаходиться на сторінці розпізнавання голосу

Коли користувач натискає кнопку «Розпізнати»

Тоді користувач переходить у діалог вибору файлу

Коли користувач обирає необхідний файл для розпізнавання

І натискає кнопку «Ok»

Тоді система відображає у повідомленні результат розпізнавання голосу».

3 АРХІТЕКТУРА ПРОЕКТУ

Відповідно до поставлених перед додатком задач back-end частина обох сервісів повинна працювати з БД, а у випадку з сервісом розпізнавання голосу це файлова система, мати можливість покриватись різного типу тестами (наприклад unit-тести, інтеграційні), мати шар WEB API в якості користувацького інтерфейсу, а також бути реалізованою за допомогою .Net Framework. Всі ці можливості, а на додачу ще й легку масштабованість проекту, може забезпечити onion-архітектура.

По-перше, цей тип архітектури додатку надає гнучкість проекту, що і дозволяє легко розширювати його у майбутньому. Дана властивість надається додатку в результаті організації коду за основними принципами запропонованої архітектури:

- усі основні сутності програми описані у центральному (найнижчому) шарі;
- шар не може посилатись на елементи шару, що знаходяться вище;
- між шарами застосунку повинна бути низька зв’язність яка забезпечується; описом основного функціоналу застосунку за допомогою інтерфейсів.

По-друге, onion архітектура гарантує можливість покриття розробленого додатку різноманітними тестами, адже шар тестів знаходиться найвище в ієрархії і в результаті це надає можливість добавляти та редагувати покриття тестами не міняючи структуру шарів самого додатку. Отже, опираючись на вище перераховані принципи, архітектура окремого мікро сервісу представлена на рисунку 3.1.

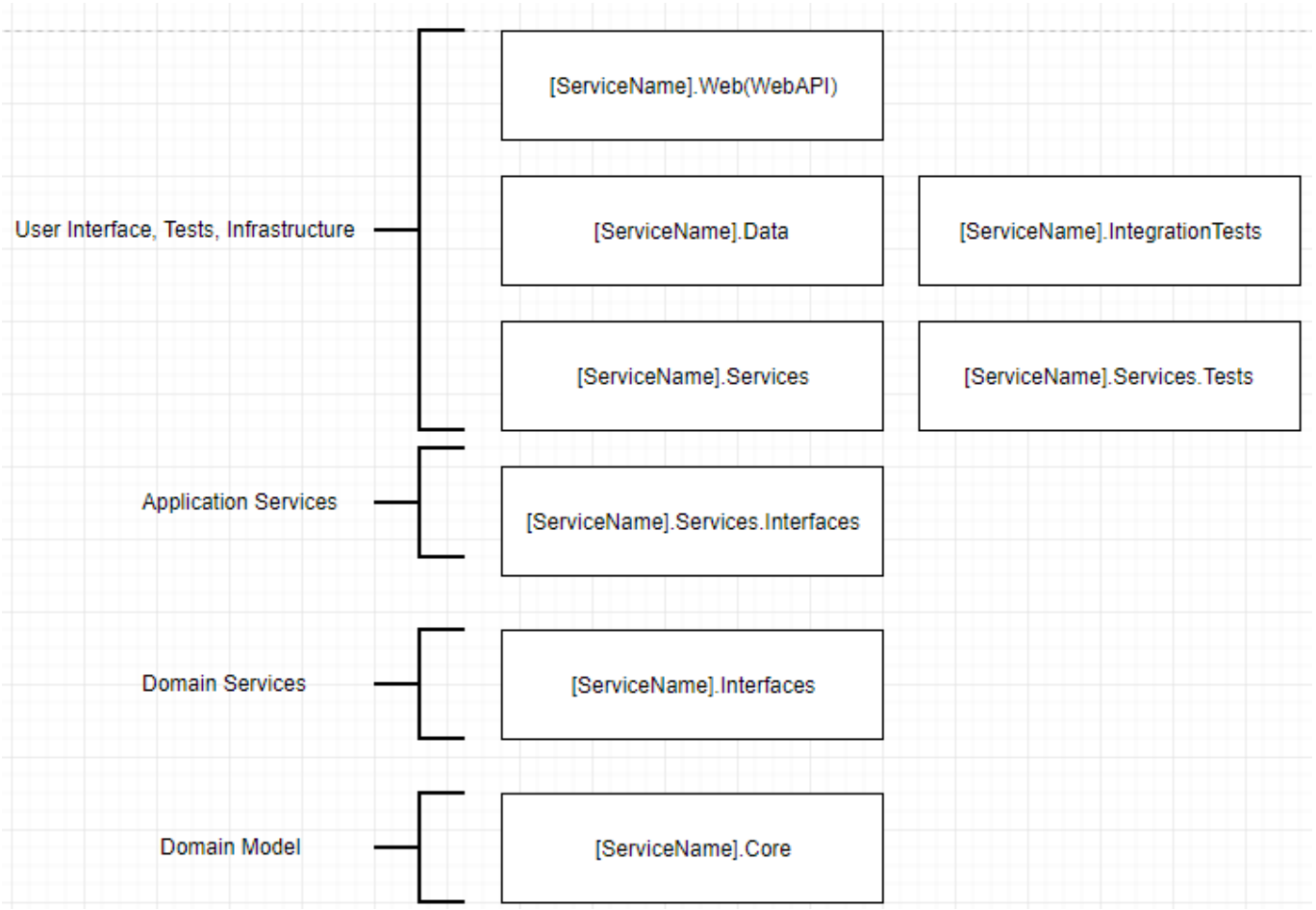

Рисунок 3.1 – Схема архітектури сервісу.

Основні бізнес сутності розташовані у шарі «Domain Model», який і є ядром архітектури. Наступний шар «Domain Services» описує інтерфейси доступу до даних, в той час як «Application Services» описує інтерфейси роботи з даними, яку можна виконати в межах додатку. На найвищому рівні знаходиться проект WEB API, реалізація усіх вище перелічених інтерфейсів, а також проекти тестів для даного сервісу.

4 СТРУКТУРА ДІАЛОГУ

Результатом виконання даної курсової роботи є сервіс аутентифікації, та сервіс розпізнавання людини по голосу, а тому вся взаємодія з розробленим функціоналом виконується через WEB API кожного з перелічених сервісів. Проте для зручності представлення результатів роботи також було розроблено front-end частину за допомогою Angular Framework. Наприклад основний функціонал сервісу аутентифікації представлений на рисунку 2.1.

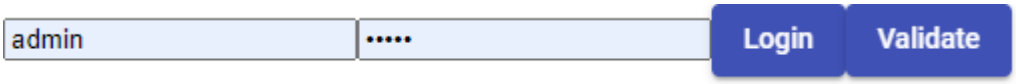

Рисунок 2.1 – Інтерфейс взаємодії зі сервісом аутентифікації.

Використовуючи представлені компоненти інтерфейсу користувача можна виконати основні функції WEB API сервісу:

- POST /token, що надає сформований токен аутентифікації конкретного користувача;
- POST /api/validate, що виконує перевірку переданого токена на достовірність.

В свою чергу основними WEB API функціями сервісу по розпізнаванню голосу є:

- POST /api/recognition?sampleAuthorName=[name], що дає змогу додати в базу зразки голосу відповідної людини ім’я якої вказано у параметрі «sampleAuthorName».
- POST api/recognition/identify, що дає змогу розпізнати людину на основі переданого у тілі запиту запису голосу.

5 ОРГАНІЗАЦІЯ РОБОТИ ПО СПРІНТАХ

Важливим аспектом перед початком роботи над застосунком, після окреслення усіх вимог та прийняття основних рішень пов’язаних з архітектурою додатку, є планування робочого процесу. В межах курсової роботи планування було виконано за допомогою розбиття по спрінтах усіх завдань по розробці програми (дивитись таблицю 1).

Таблиця 4.1 – Розбивка роботи по спрінтах.

26.10 – 8.11	1	Розробка діаграм
	2	Побудова архітектури додатку
	3	Написання BDD сценаріїв
9.11 – 22.11	1	Написання сервісу аутентифікації
	2	Покриття юніт тестами сервісу аутентифікації
	3	Покриття інтеграційними тестами сервісу аутентифікації
23.11 – 6.12	1	Написання сервісу аутентифікації
	2	Покриття юніт тестами сервісу аутентифікації
	3	Покриття інтеграційними тестами сервісу розпізнавання по голосу
7.12 – 14.12	1	Розробка веб інтерфейсу
	2	Розробка BDD тестів
	3	Розробка автоматизованих тестів Selenium
	4	Налаштування CI

6 ТЕСТУВАННЯ

6.1 BDD та Selenium тестування

В силу того що розробка додатку в основному проходить з використанням C# та .Net Framework, BDD тести реалізовані на основі можливостей саме цих технологій. Для роботи з тестуванням додатку використовувалась SpecFlow бібліотека, яка і надає змогу опису сценаріїв [3]. BDD сценарій тестування сервісу аутентифікації виглядає наступним чином:

```
Feature: LogIn

@mytag
Scenario: LogIn with correct credentials
    Given Launch Firefox
    And Navigate to Web Frontend
    When Enter admin for login and password
    And Click on LogIn button
    And Click on Validation button
    Then Token and valdation result should be visible.
```

За допомогою SpecFlow бібліотеки з даного сценарію генеруються відповідний клас який і виконує тестування, у випадку сервісу аутентифікації це клас LogInSteps. В середині класу використовуються сутності описані в таких бібліотеках як:

- Selenium.Support;
- Selenium.WebDriver;
- Selenuim.WebDriverBackedSelenium.

Вище перераховані бібліотеки використовуються для запуску браузера, переходу на відповідну сторінку та взаємодію з нею з метою тестування [4].

6.2 Юніт-тести

Юніт-тести – це важлива частина даної курсової, адже саме вони є одним з пунктів, який гарантує безперебійну роботу застосунку протягом тривалого часу. В силу обраної архітектури сервісів, юніт-тести без будь-яких проблем покривають увесь розроблений функціонал. Основною задачею такого тестування є модульне покриття різного виду тестовими сценаріями усіх аспектів роботи розробленого коду. Таким чином для обох сервісів, було розроблено тести, які перевіряють наявність очікуваних помилок, в ході виконання тої чи іншої функції, а також чи видає програмний метод очікуваний результат при певних вхідних даних. Не менш важливим аспектом під час розробки юніт-тестів є правильний підхід до їх найменування. В межах даної роботи усі тести називались за наступним зразком: «[Метод який тестується]_[Опис умов тестування слово 1]_[Опис умов тестування слово 2]_..._[Очікуваний результат тесту]». Саме такий підхід до організації назв сценаріїв тестування дозволить, швидко, в разі необхідності, зрозуміти який з аспектів функціоналу тестується в яких умовах це відбувається, а також який результат очікується.

Для прикладу можна розглянути декілька тестів з сервісу аутентифікації, які відповідно перевіряють створення сервісу та коректність відпрацювання його методів:

- Construct_UnitOfWork_Is_Null_Throws;
- Construct_UnitOfWork_Is_Set_Correctly_Constructs_Succesfully;
- Autheticate_Not_Existing_User_Email_Returns_Null;
- Autheticate _Incorrect_Password_Returns_Null;
- Autheticate_Data_Is_Correct_Returns_User;
- Authenticate_Correct_Data_Performs_Less_Than_One_Second.

З найменування «Autheticate_Not_Existing_User_Email_Returns_Null» випливає, що тестується метод «Autheticate» за умови, що вводиться не існуючий email користувача і очікуваним результатом виконання є null. Відпрацювання розроблених юніт-

тестів можна розглянути на рисунках А.4 та А.5 у додатку А для сервісу аутентифікації та сервісу розпізнавання по голосу відповідно.

Щодо технічної сторони тестування то основною бібліотекою для виконання усіх тестів була MsTest в комбінації з Moq бібліотекою, у функціонала бібліотек досить низький рівень входу, що дозволяє швидко розібратись з основними аспектами роботи з ними і покрити увесь необхідний код тестовими сценаріями [5, 6]. Окрім цього для спрощення роботи по порівнянню очікуваних результатів відпрацювання методу з фактичними, використовувалась бібліотека FluentAssertions, яка дозволяє більш читабельним способом оформити усі необхідні порівняння [7]. Приклад юніт-тесту з використанням вище перерахованих технологій можна знайти у додатку Б.

6.3 Інтеграційні тести

Інтеграційне тестування це уже наступний крок у забезпеченні стабільності роботи застосунку після юніт-тестів. Адже в той час коли юніт-тести перевіряють роботу окремих модулів коду, інтеграційні тести ж перевіряють взаємодію цих модулів між собою. У поточній курсовій роботі тестами покривались WEB API контролери відповідних сервісів. Для організації роботи контролерів в межах тестового проекту використовувався веб сервер iisexpress. Він дозволив запустити застосунок для виконання усіх тестових сценаріїв. Для сервісу аутентифікації був розроблений наступний сценарій:

- отримання токена аутентифікації «http://localhost:8080/token»
- перевірка статусу відповіді, який має дорівнювати 200
- перевірка наявності токену
- перевірка токену аутентифікації «http://localhost:8080/api/validation»
- перевірка статусу відповіді, який має дорівнювати 200
- перевірка відповіді на рівність true.

Описаний вище сценарій дозволить перевірити коректність обробки запитів на WEB API контролері сервісу, а саме дозволить упевнитись, що сервіс видає правильні токени для аутентифікації користувача. Приклад реалізації цього сценарію є у додатку Б.

6.4 Автоматичні тести при коміті

Автоматичне виконання тестів при коміті є частиною такої практики розробки як Continuous Integration. У звичайному проекті, де над різними частинами системи розробники трудяться незалежно, стадія інтеграції є заключною та виконується при додаванні кожним розробником своєї частини до загального проекту, а тому автоматичне виконання усіх розроблених тестів є невід'ємною частиною цього процесу, воно запобігає потраплянню коду який може зламати існуючий функціонал застосунку.

В межах курсової роботи було використано GitHub Actions для виконання усіх необхідних операцій при коміті. Вони включають в себе:

- Налаштування MsBuild;
- Налаштування VsTest;
- Відновлення Nuget пакетів для кожного сервісу;
- Build та запуск тестів для кожного мікро сервісу.

Приклад роботи GitHub Actions є на рисунку А.3 у додатку А.

6.5 Стрес-тестування

Для стрес тестування застосунку використовувалась програма JMeter [8]. Для спрощення розробки тестових сценаріїв в межах курсової роботи було використано «Recording template». Цей підхід після певного налаштування дозволить записати усі дії, які повинні бути виконані для стрес тестування. Для сервісу аутентифікації, це будуть дії переходу на сторінку відправлення даних користувача з метою отримання токену аутентифікації та відправлення токену для його перевірки. Отриманий результат запису зображено на рисунку 5.5.1.

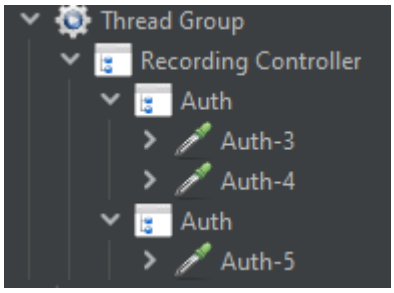

Рисунок 5.1.1 – Записані дії для стрес тестування сервісу аутентифікації.

Записані дії за допомогою програми JMeter позволяють проводити стрес тестування зі завчасно налаштованими умовами, такими як кількість потоків, користувачів та циклів. Така гнучкість конфігурації дозволяє проводити стрес тестування в залежності від очікуваної кількості користувачів сервісу. Результати стрес тестування зображені на рисунку А.6 у додатку А.

7 ІНСТРУКЦІЯ ПРОГРАМІСТА

7.1 Загальні відомості

Веб-додаток було реалізовано на основі мікро сервісної архітектури. У застосунку не має єдиної точки входу на серверній частині, кожен сервіс має свій набір WEB API контролерів з певним функціоналом.

Сервіс аутентифікації реалізовує функціонал допуску користувача у систему. І працює таким чином, що користувачу слід відправити свій логін та пароль до відповідного WEB API контролера а у відповідь отримати токен аутентифікації який буде необхідний для виконання подальших дій з додатком. Ще однією задачею сервісу є перевірка правильності токенів.

Сервіс розпізнавання по голосу включає такі функції як завантаження зразків голосу для розширення бази даних, а також розпізнавання людини з записаного зразка голосу. Відповідні задачі можна виконати WEB API сервісу.

Для збереження даних використовується база MSSql Server у зв’язку з Entity Framework [9]. Сервісна частина додатку, розроблена з використанням шаблону розробки MVC, детальніше про який можна почитати зі списку джерел [10], та запускається за допомогою IIS веб сервера в той час як front-end написаний на Angular Framework в межах розробки запускається на webpack-dev-server. Увесь код програми знаходиться за посиланням представленим у списку джерел [11].

7.2 Умови проектування програми

Мікро сервіси написані з використанням технології .NET Framework 4.5.2 у зв’язці з EntityFramework та WebApi2, клієнтська частина застосунку використовує Angular Framework. Усі компоненти розгортаються вручну за допомогою IIS сервера та з використанням Angular CLI.

Цільова операційна система, під яку було розроблено дане програмне забезпечення – Windows 10. Мінімальні системні вимоги:

- процесор armv7l; amd64/x86 з такотовою частотою 1.5Ghz;
- оперативна пам`ять: 4 Гб;
- мінімум 4Гб вільної пам`яті на жорсткому диску;
- операційна система Windows 10 з Angular CLI.

8 КЕРІВНИЦТВО КОРИСТУВАЧА

8.1 Загальні відомості

Метою даного веб-додатку є реалізація системи розпізнаванню людини по голосу. Користувач має змогу створювати та розширювати базу даних голосів, а також виконувати дії по розпізнаванню людини основуючись на записі її голосу. Усі функції додатку захищені за допомогою сервісу аутентифікації тому для доступу до них користувачу слід надати коректний логін та пароль, отримати відповідний.

8.2 Умови застосування програми

Для використання можливостей даного веб-додатку необхідно мати доступ до сервера де його розгорнуто. Не потрібно встановлювати ніяких додаткових програм, тільки браузер з підтримкою усіх стандартів для роботи веб-застосунків (зазвичай навіть стандартний браузер операційних систем для настільного комп’ютера має змогу впоратися з поставленою задачею). Рекомендованим браузером буде Google Chrome останньої версії.

8.3 Послідовність дій користування програмою

При роботі з додатком першим кроком буде аутентифікація для цього можна використати як розроблений web інтерфейс так і на пряму WEB API сервісу аутентифікації. Запит буде виглядати наступним чином «{адреса серверу}/token» у тілі запиту

слід відправити логін і пароль за зразком: «grant_type=password&username=admin &password=admin». У відповідь сервер наддасть токен який і буде використовуватись для виконання подальших дії у застосунку. В свою чергу дії, пов’язані зі розпізнаванням голосу, також можна виконати за допомогою запитів до сервісу. Доступні функції це додавання нового зразка голосу та розпізнавання людини по зразку голосу. Приклад оформлення запитів можна розглянути у Додатку Б.

ВИСНОВКИ

В результаті роботи над курсовою було спроектовано та розроблено веб-застосунок, який виконує поставлені задачі та в силу своєї архітектури підлягає подальшому розширенню та використанню на різних платформах.

Результуючий продукт виконаний з застосуванням все можливих сучасних підходів та технологій по розробці веб-застосунків, а саме він має розподілену архітектуру клієнт-сервер, де клієнт розгортається віддалено та доступний з любого пристрою не залежно від операційної системи. Окрім цього серверна частина програми має шар WEB API, який відв’язує її від конкретної реалізації front-end частини, що у майбутньому дозволить використовувати розроблений функціонал для різного виду користувацьких інтерфейсів.

Що до можливого покращення програми, то воно включає в себе розробку більш функціонального та зручного web інтерфейсу, що з легкістю реалізовується за допомогою використаного Angular Framework, а також в силу вибору onion-архітектури є здатність легко покращувати роботу по розпізнаванню голосу, додаючи нові модулі та шари, з досконалішими алгоритмами, до back-end частини сервісу.

Створена програма є повністю функціональною та надає змогу ідентифікувати людину на основі такої біометричної характеристики як голос. Що в свою чергу дозволить забезпечити ще один рівень захисту доступу до того чи іншого ресурсу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Angular [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/.
2. .NET Framework documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/framework/.
3. SpecFlow documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.specflow.org/projects/specflow/en/latest/Integrations/MsTest.html.
4. Selenium [Електронний ресурс] – Режим доступу до ресурсу: https://www.selenium.dev/.
5. Unit testing fundamentals [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2019.
6. Moq [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/Moq/moq4/wiki/Quickstart.
7. Fluent Assertions [Електронний ресурс] – Режим доступу до ресурсу: https://fluentassertions.com/.
8. JMeter [Електронний ресурс] – Режим доступу до ресурсу: https://jmeter.apache.org/.
9. MS SQL Server documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15.
10. MVC – Model View Controller [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/181772/.
11. Репозиторій з вихідним кодом [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/Majardom/CourseWork.

ДОТАТОК А

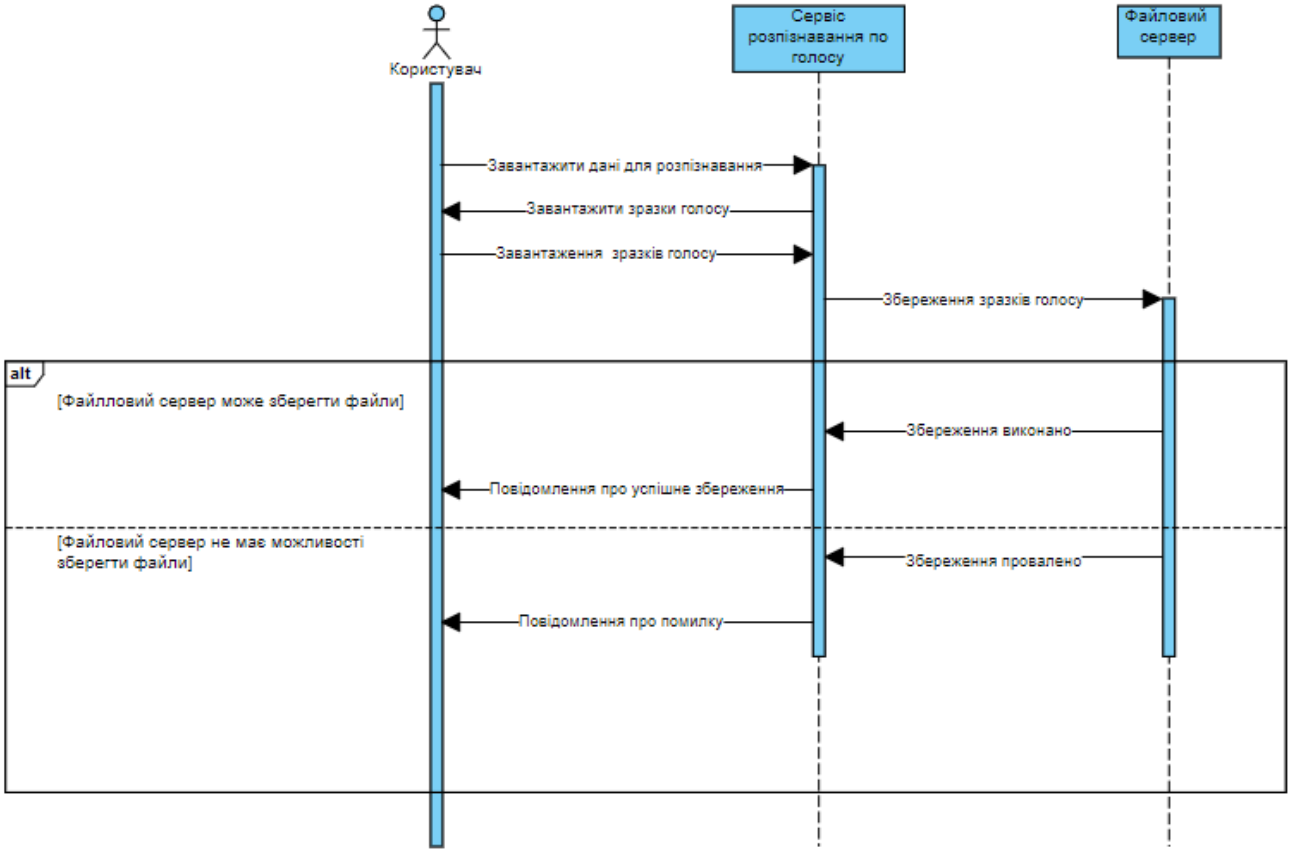

Рисунок А.1 – Діаграма послідовності для сервісу розпізнавання по голосу.

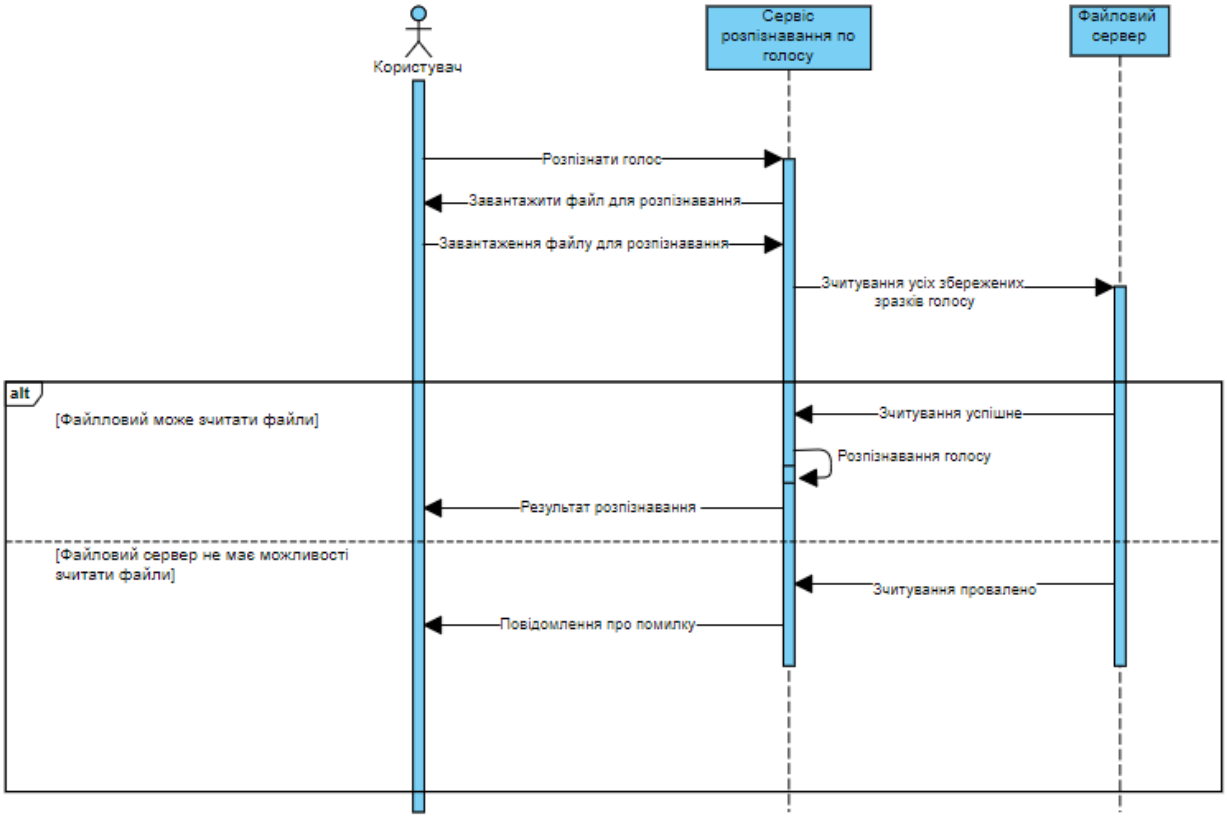

Рисунок А.2 – Діаграма послідовності для сервісу розпізнавання по голосу.

Рисунок А.3 – Приклад роботи GitHub Actions.

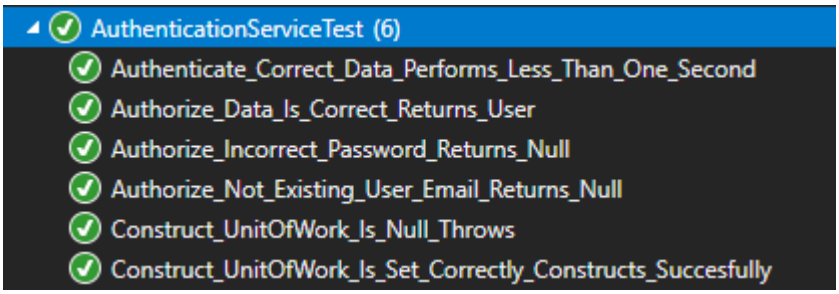

Рисунок А.4 – Відпрацювання юніт-тестів для сервісу аутентифікації.

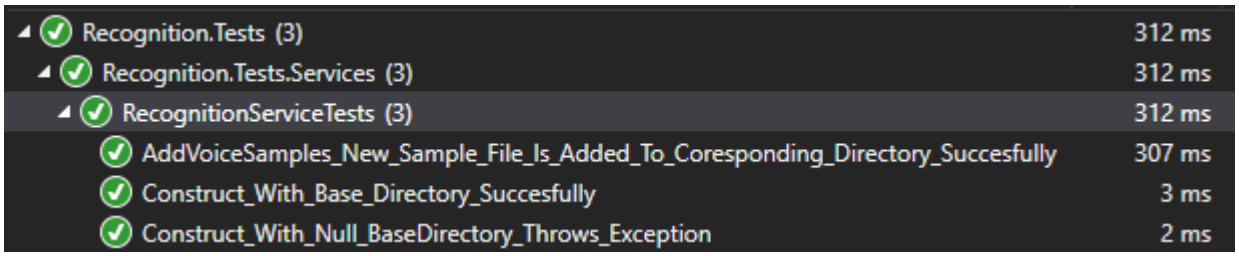

Рисунок А.5 – Відпрацювання юніт-тестів для сервісу аутентифікації.

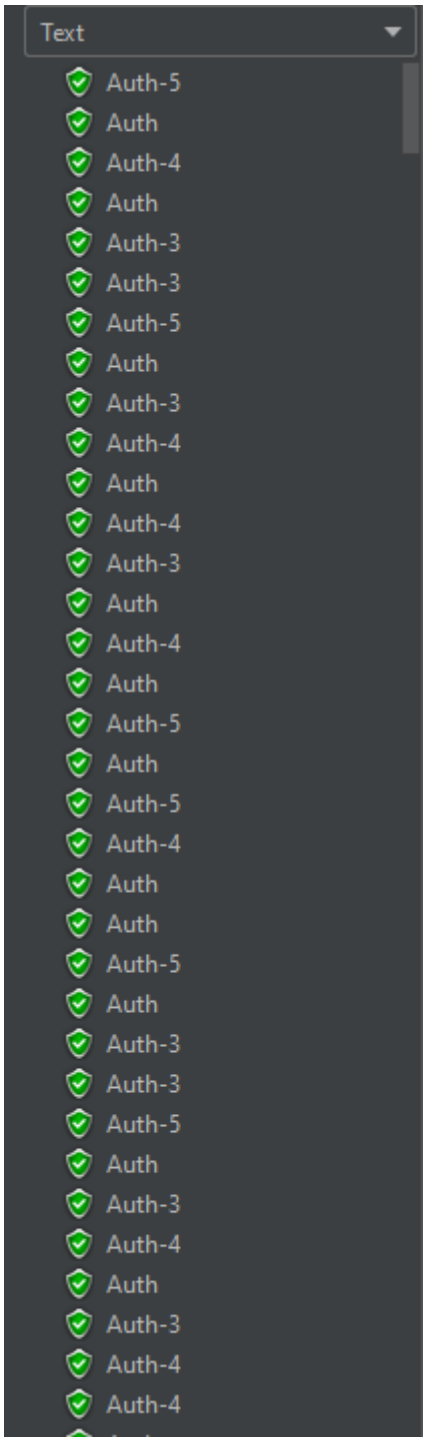

Рисунок А.6 – Результати стрес тестування застосунку.

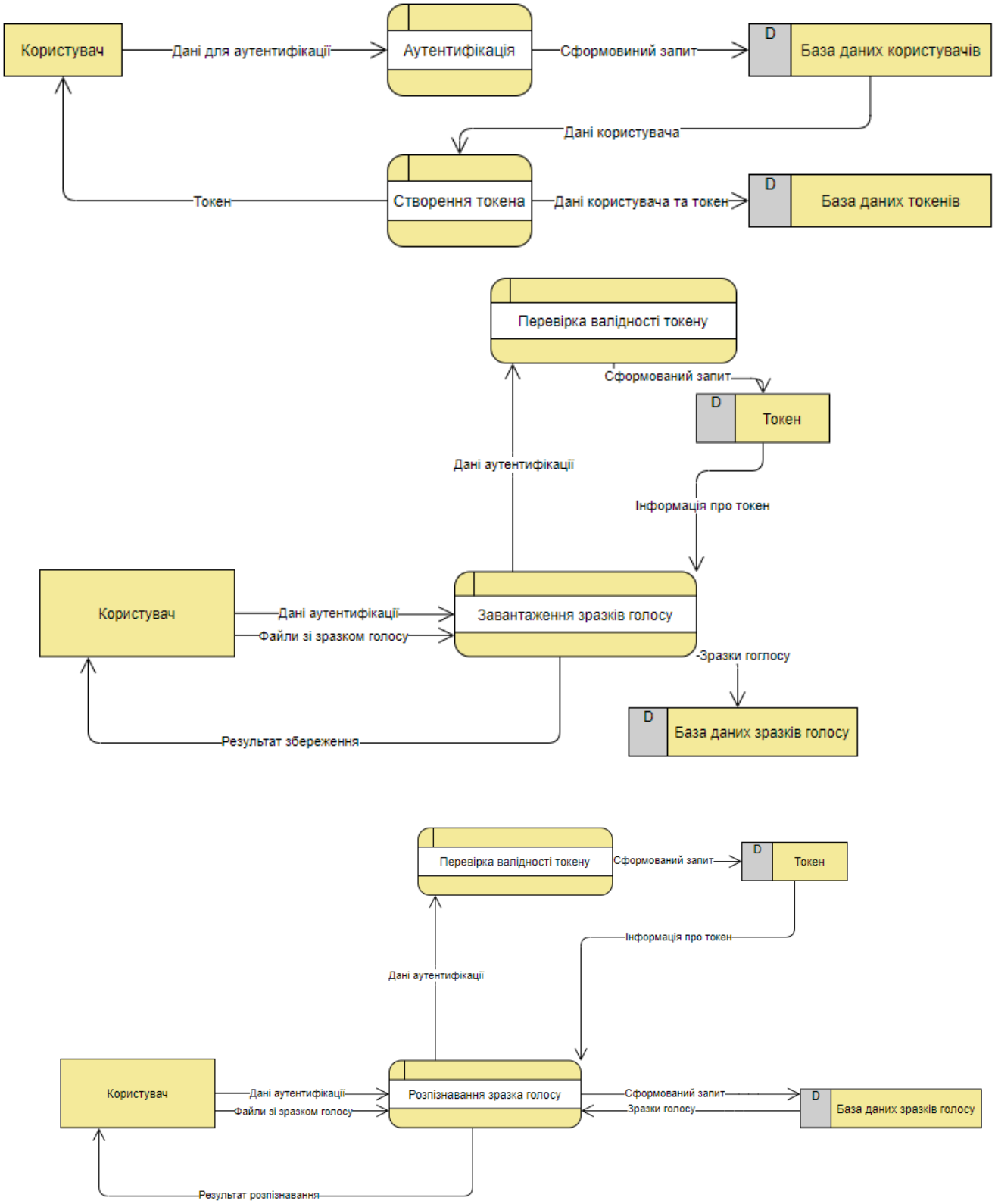

Рисунок А.7 – Data flow діаграма для сервісів застосунку.

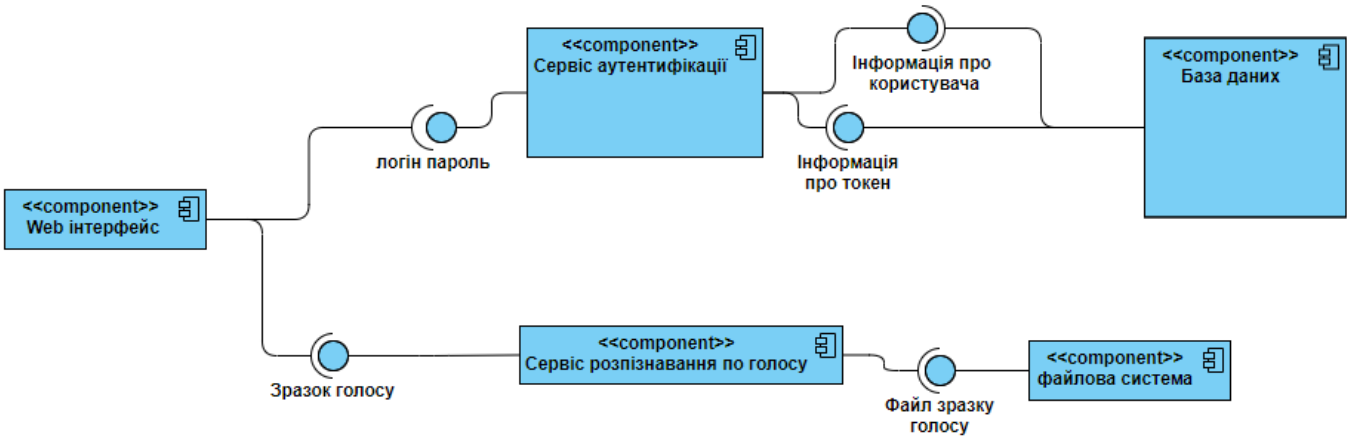

Рисунок А.8 – Діаграма компонентів.

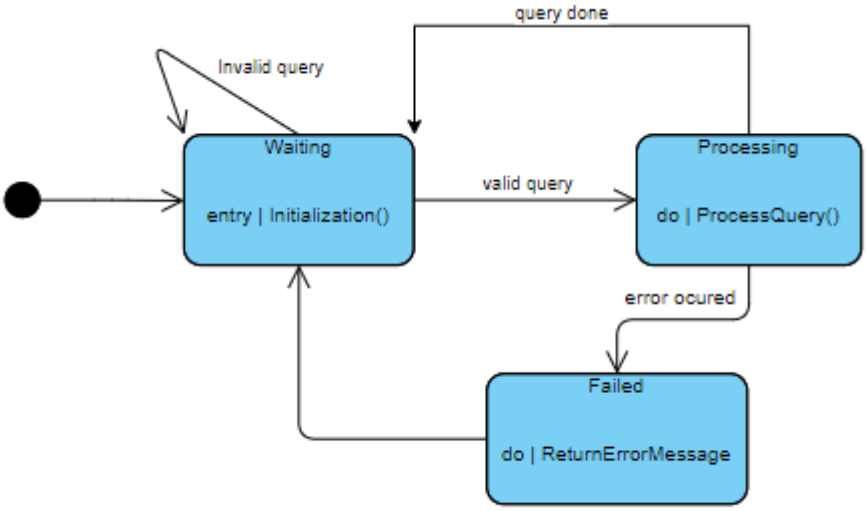

Рисунок А.9 – Діаграма станів сервісу.

ДОДАТОК Б

Приклад юніт тесту

MSTest

```
[TestMethod]
public void Construct_UnitOfWork_Is_Null_Throws()
{
    //arrange
    IUnitOfWork uow = null;

    //act
    Action act = () => { var sut = new AuthenticationService(uow); };

    //assert
    act.Should()
        .Throw<ArgumentNullException>();
}
```

Інтеграційний тест сценарій перевірки сервісу аутентифікації

```
[TestMethod]
public async Task LogIn_When_Credentials_Are_Correct_Issues_Valid_Token()
{
    //arange
    var stringContent = new StringContent("grant_type=password&username=admin&password=admin");
```

```
var tokenURL = "http://localhost:8080/token";
TokenResponseDto tokenResponse = null;
HttpResponseMessage httpTokenResponse = null;

var validationURL = "http://localhost:8080/api/validation";
bool validationResponce = false;
HttpResponseMessage httpValidationResponse = null;

//act
using (var client = new HttpClient())
{
    httpTokenResponse = await client.PostAsync(tokenURL, stringContent);
    var jsonResonce = await httpTokenResponse.Content.ReadAsStringAsync();
    tokenResponse = JsonConvert.DeserializeObject<TokenResponseDto>(jsonResonce);

    var validationDto = new TokenValidationDto { Token = tokenResponse.access_token };

    stringContent = new StringContent(JsonConvert.SerializeObject(validationDto), Encoding.UTF8, "application/json");
    httpValidationResponse = await client.PostAsync(validationURL, stringContent);
    var validationStringResponce = await httpValidationResponse.Content.ReadAsStringAsync();
    validationResponce = bool.Parse(validationStringResponce);
}
```

```
        //Assert
        httpTokenResponse
            .Should()
            .NotBeNull();
        httpValidationResponse
            .Should()
            .NotBeNull();
        httpTokenResponse.StatusCode
            .Should()
            .Be(HttpStatusCode.OK);
        httpValidationResponse.StatusCode
            .Should()
            .Be(HttpStatusCode.OK);
        tokenResponse.Should()
            .NotBeNull();
        tokenResponse.access_token
            .Should()
            .NotBeNull();
        tokenResponse.token_type
            .Should()
            .Be("bearer");
        validationResponce
            .Should()
            .BeTrue();
}
```

Сервіс аутентифікації

```
using System.ComponentModel.DataAnnotations;

namespace Authentication.Core
{
	public class BaseObject
	{
		[Key]
		public string Id { get; set; }
	}
}
namespace Authentication.Core
{
	public class TokenIdentity : BaseObject
	{
		public string Token { get; set; }
		public string UserId { get; set; }
	}
}
namespace Authentication.Core
{
	public class User : BaseObject
	{
		public string Name { get; set; }
		public string Email { get; set; }

		public string Password { get; set; }
	}
```

```
}


using System;
using System.Linq;
using Authentication.Core;
using Authentication.Interfaces;
using Authentication.Services.Intefaces;

namespace Authentication.Services
{
	public class AuthenticationService : IAuthenticationService
	{
		private readonly IUnitOfWork _uow;

		public AuthenticationService(IUnitOfWork uow)
		{
			if (uow == null)
				throw new ArgumentNullException(nameof(uow));

			_uow = uow;
		}

		public User Autheticate(string email, string password)
		{
			if (string.IsNullOrEmpty(email))
				throw new ArgumentNullException(nameof(email));
			if (string.IsNullOrEmpty(password))
				throw new ArgumentNullException(nameof(password));
```

```
                var authentificatedUser = _uow.Users.GetAll()
                        .FirstOrDefault(x => x.Email == email && x.Password ==
password);

                if (authentificatedUser != null)
                {
                        var ids = _uow.TokenIdentities.GetAll().Where(x => x.UserId ==
authentificatedUser.Id).Select(x => x.Id).ToList();
                        ids.ForEach(x => _uow.TokenIdentities.Delete(x));
                }

                return authentificatedUser;
        }

        public void SaveTokenIdentity(TokenIdentity tokenIdentity)
        {
                _uow.TokenIdentities.Create(tokenIdentity);
                _uow.TokenIdentities.Save();
        }

        public bool Validate(string token)
        {
                return _uow.TokenIdentities.GetAll().Where(x => x.Token ==
token).Any();
        }
    }
}
```

Сервіс розпізнавання по голосу

Фрагменти коду для додавання нового зразку голосу у систему

```
public VoicePrint CreateVoicePrint(T userKey, Stream voiceSampleFile)
    {
        var audioSample = ConvertFileToDoubleArray(voiceSampleFile);

        return CreateVoicePrint(userKey, audioSample);
    }

public VoicePrint CreateVoicePrint(T userKey, double[] voiceSample)
    {
        lock (_lock)
        {
            if (userKey == null)
            {
                throw new ArgumentNullException(nameof(userKey), "The userKey is null");
            }

            if (store.ContainsKey(userKey))
            {
                throw new ArgumentException("The userKey already exists: [{userKey}");
            }

            double[] features = audioProcessor.ProcessAndExtract(voiceSample);
            VoicePrint voicePrint = new VoicePrint(features);

            if (!universalModelWasSetByUser)
            {
                if (universalModel == null)
                {
                    universalModel = new VoicePrint(voicePrint);
                }
```

```
            else
            {
                universalModel.Merge(features);
            }
        }

        store.Add(userKey, voicePrint);

        return voicePrint;
    }
}
```

Запити до сервісу розпізнавання голосу

POST http://localhost:60754/api/recognition?sampleAuthorName=test
HEADERS
token: {токен аутентифікації}
BODY "application/json"
{
    “samplesBase64”: [“” , “”, “”]
}