**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Сумський державний університет**

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

_____ Оксана ШОВКОПЛЯС
(підпис)

06 грудня 2024 р.

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня магістр**

зі спеціальності 122 «Комп'ютерні науки»

освітньо-професійної програми «Інформатика»

на тему: Інформаційна технологія створення навчального додатку для вивчення іноземної мови з можливістю голосового введення слів

здобувача групи ІН.м-33 Лисянського Захара Володимировича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Захар ЛИСЯНСЬКИЙ
(підпис)

Керівник,

старший викладач кафедри комп'ютерних наук

кандидат технічних наук

Олег БЕРЕСТ _____
(підпис)

**Суми - 2024**

**Сумський державний університет**
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Оксана ШОВКОПЛЯС
(підпис)

# ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

**на здобуття освітнього ступеня магістра**

зі спеціальності 122 «Комп'ютері науки», освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-33 Лисянського Захара Володимировича

1. Тема роботи: Інформаційна технологія створення навчального додатку для вивчення іноземної мови з можливістю голосового введення слів
затверджую наказом по СумДУ від *«03» грудня 2024 року № 1257-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 06 грудня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
*1) Літературний огляд. 2) Постановка задачі 3) Методика вирішення поставлених задач. 4) Програмна реалізація. 5) Висновки*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проєкту, що стосується їх

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |

7. Дата видачі завдання «18» серпня 2024 р.

Завдання прийняв до виконання _____ (підпис)
Керівник _____ (підпис)

# КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів кваліфікаційної роботи | Термін виконання | Примітка |
|---|---|---|---|
| *1* | *Літературний огляд* | | |
| *2* | *Постановка задачі* | | |
| *3* | *Методика вирішення поставлених задач* | | |
| *4* | *Програмна реалізація* | | |
| *5* | *Висновки* | | |

Здобувач вищої освіти _____ (підпис)
Керівник _____ (підпис)

# АНОТАЦІЯ

**Записка**: 80 стор., 15 рис., 1 табл., 21 використаних джерел, 3 додатки.

**Обґрунтування актуальності теми роботи –** тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої практичної задачі розробки навчальної технології з використанням голосових технологій. Такий підхід дозволяє підвищити ефективність навчального процесу та забезпечити зручну й адаптивну взаємодію користувача з додатком, що відповідає сучасним тенденціям цифрової освіти.

**Об'єкт дослідження** – процес розробки інформаційної технології створення навчального додатку для вивчення іноземної мови з можливістю голосового введення слів.

**Мета роботи** — проектування та розробка інформаційної технології створення навчального додатку для вивчення іноземної мови з можливістю голосового введення слів.

**Методи дослідження** — аналіз технологій голосового введення та відтворення тексту, проектування інтуїтивного користувацького інтерфейсу, алгоритми реалізації голосових технологій у мобільних додатках.

**Результати** — розроблено інформаційну технологію, що дозволяє користувачам вводити слова голосом, прослуховувати їх правильну вимову, організовувати слова в тематичні папки, здійснювати пошук за допомогою текстового та голосового введення, а також відстежувати прогрес через систему статистики. Проведено тестування додатку, яке підтвердило стабільність роботи, відповідність функціональності та зручність використання.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, JAVASCRIPT, TYPESCRIPT, REACT NATIVE.

**ЗМІСТ**

# ВСТУП

Люди, що живуть у сучасному цифровому суспільстві, постійно прагнуть спростити своє повсякденне життя. Інтернет став невід'ємною частиною їхнього існування, а це, своєю чергою, спричинило високий попит на розробку мобільних додатків. Сьогодні мобільний інтернет дозволяє здійснювати широкий спектр завдань незалежно від наявності домашньої Wi-Fi мережі — користувачі можуть підключатися до мережі через смартфони, планшети чи навіть годинники. Проте без відповідних додатків для мобільних пристроїв, що інтегруються з цим підключенням, можливості мобільного інтернету залишалися б обмеженими. Ці додатки дозволяють виконувати різноманітні завдання, включаючи роботу з даними користувачів, їх передачу на сервери та подальшу обробку.

Сьогодні багато розробників займаються створенням програм для платформ Android та iOS, деякі з яких служать для комунікації, навчання, продажу, ігор, читання та пошуку інформації. Мобільні додатки можуть слугувати як для робочих цілей, так і для розваг.

Об’єктом цього дослідження є процес розробки мобільного крос-платформного додатку для навчальної інформаційної системи вивчення іноземної мови з можливістю голосового введення слів. У цій роботі буде детально описано створення додатку, який надає користувачам змогу вивчати іноземну лексику, вводити слова за допомогою голосу, отримувати зворотний зв'язок, а також знаходити нові слова відповідно до власних інтересів.

Дана робота складається зі вступу, огляду літератури, методики вирішення поставлених завдань, програмної реалізації, висновків та додатків.

# 1. ЛІТЕРАТУРНИЙ ОГЛЯД

## 1.1. Огляд аналогів додатка для вивчення іноземної мови з можливістю голосового введення слів

У процесі розробки навчальної інформаційної системи для вивчення іноземної мови з можливістю голосового введення слів важливо дослідити існуючі рішення на ринку. Існує безліч мобільних додатків для вивчення мов, які пропонують різні методи навчання, функціональні можливості та інтерактивні елементи. У цьому огляді розглянуто основні аналоги, їхні функції, переваги та недоліки, що дозволить визначити ключові вимоги для створення нового додатку [1]:

Duolingo — один із найпопулярніших додатків для вивчення мов. Він базується на ігровому підході до навчання, пропонуючи користувачам короткі заняття у вигляді квестів та завдань. У Duolingo є інтерактивні вправи на читання, писання, прослуховування та вимову. Ключовою особливістю є можливість виконувати вправи з використанням голосу, однак розпізнавання голосу обмежене і не завжди точно розуміє користувача. Крім того, додаток часто пропонує однотипні завдання, що може знижувати мотивацію у довгостроковій перспективі. До переваг можна віднести простоту використання, ігрові елементи та широкий вибір мов. Недоліками є обмежене розпізнавання голосу, одноманітність завдань.

Memrise — додаток, орієнтований на запам'ятовування нових слів за допомогою асоціативного мислення та інтерактивних карток. Він також включає функцію голосового введення, що дозволяє користувачам вимовляти слова для закріплення навичок. Memrise активно використовує асоціації та відео з носіями мови, що допомагає покращити вимову та розуміння мови на слух. Однак, додаток не пропонує достатньо інтерактивних вправ на закріплення граматики та побудову речень. Перевагами є ефективне вивчення лексики, асоціативне навчання, відео з носіями мови. Недоліками є

обмежені можливості для вивчення граматики та інтеграції в реальне спілкування.

Busuu пропонує комплексний підхід до вивчення мови, охоплюючи лексику, граматику, читання, прослуховування та вимову. Система також має функцію спілкування з носіями мови, що дозволяє користувачам тренуватися в реальному діалозі. Голосове введення в Busuu реалізоване на базі технології розпізнавання мови, але доступно лише на платних рівнях. Незважаючи на широкий функціонал, для ефективного використання додатку потрібен стабільний інтернет, що може бути недоліком у деяких ситуаціях. Перевагами є комплексний підхід, спілкування з носіями мови, функції граматичних уроків. Недоліками є обмежений доступ до голосового введення, залежність від інтернету.

HelloTalk відрізняється від інших додатків акцентом на спілкуванні з носіями мови. Він є платформою для обміну мовами, де користувачі можуть писати повідомлення, надсилати голосові повідомлення та коригувати одне одного. Додаток підтримує голосове введення, яке дозволяє тренувати навички вимови та розуміння мови на слух. Однак HelloTalk більше підходить для користувачів із певним рівнем знань, оскільки додаток не пропонує структурованих уроків чи граматичних завдань. Перевагами є спілкування з носіями мови, підтримка голосових повідомлень. Недоліками є відсутність структурованих уроків, орієнтація на користувачів із базовими знаннями.

Rosetta Stone пропонує методику повного занурення в мову, де навчання відбувається без використання рідної мови. Додаток включає функцію розпізнавання голосу, яка аналізує вимову користувача і надає зворотний зв’язок. Rosetta Stone вважається однією з найефективніших програм для вивчення вимови та інтонації, але його методика може бути складною для початківців через відсутність перекладів. Перевагами є ефективна методика для вивчення вимови, функція повного занурення. Недоліками є відсутність перекладів, висока вартість підписки.

Аналіз аналогів показав, що більшість існуючих додатків зосереджені на окремих аспектах навчання: лексика, вимова, граматика або спілкування з носіями. Проте жоден із розглянутих додатків не пропонує повноцінного інтегрованого підходу до вивчення іноземної мови з акцентом на голосове введення для тренування запам'ятовування нових слів. Це свідчить про потенціал для створення мобільного додатку, який би об'єднував переваги різних платформ і був би зручним як для початківців, так і для користувачів із середнім рівнем знань.

## 1.2. Етапи створення мобільних додатків

Процес розробки мобільного додатка можна поділити на такі основні етапи:

1. Аналіз продукту. Спершу визначаються мета продукту та його основні задачі. Проводиться дослідження ринку і конкурентів, включаючи непрямих. На цьому етапі формується відповідь на запитання "Як цей продукт допоможе користувачам?", а також визначаються вигляд, принципи та функціонал продукту.
2. Специфікація. Створюється детальний документ для розробників, що містить повний опис функціоналу мобільного додатка.
3. Оцінка і планування. На основі специфікації здійснюється оцінка витрат і термінів реалізації проекту. Враховуються обсяг робіт, витрати, потенційні ризики та заходи для їх мінімізації.
4. Дизайн. Розробляється візуальне оформлення додатка, включаючи загальну дизайн-концепцію, компоненти інтерфейсу, макети та інтерактивні прототипи.
5. Програмування. Реалізується функціонал додатка, основний код і логіка роботи програми.
6. Тестування. Додаток проходить перевірку на стійкість, надійність і безпеку, що забезпечує високу якість продукту перед релізом.

7. Реліз. Завершений проект завантажується в магазини додатків, такі як App Store для iOS і Google Play для Android, для доступу користувачам.

Ці етапи забезпечують надійний і ефективний процес розробки мобільного додатка.

## 1.3. Технології створення мобільних додатків

Існує кілька технологій і підходів, які використовуються для розробки мобільних додатків. Ось деякі з них:

Нативна розробка — це підхід, при якому додатки створюються для конкретних платформ, таких як iOS (з використанням Swift або Objective-C) чи Android (за допомогою Java або Kotlin). Цей метод забезпечує високу продуктивність і доступ до всіх можливостей платформи, але потребує окремого коду для кожної операційної системи.

Крос-платформна розробка — це стратегія, за якої одна кодова база використовується для створення додатка, що працює на різних платформах. Найпопулярнішими фреймворками для такої розробки є React Native, Xamarin і Flutter. Вони дозволяють спільно використовувати код, що значно прискорює процес розробки, хоча можуть виникнути обмеження щодо доступних можливостей кожної окремої платформи.

Веб-додатки — це програми, що працюють через веб-браузер на мобільних пристроях і не потребують інсталяції окремого додатку. Вони створюються за допомогою веб-технологій, таких як HTML, CSS та JavaScript. Для їх розробки можна використовувати фреймворки, наприклад, React, Angular або Vue.js.

Гібридна розробка — це комбінований підхід, що поєднує елементи нативної і крос-платформної розробки. Гібридні додатки використовують веб-технології для створення інтерфейсу, який працює вбудованим браузером у мобільному додатку. Прикладом таких фреймворків є Apache Cordova і Ionic.

Кожен з цих підходів має свої сильні та слабкі сторони. У таблиці 1.1 можна порівняти переваги та обмеження різних технологій для створення мобільних додатків [2].

Таблиця 1.1 Порівняння технологій розробки мобільних застосунків

| Технологія | Переваги | Недоліки |
|---|---|---|
| Нативна розробка | 1. Нативні додатки забезпечують найвищу продуктивність та функціональність, оскільки розробляються спеціально для певної платформи.<br>2. Вони мають повний доступ до всіх функцій та можливостей операційної системи, що дозволяє реалізовувати складні та специфічні функції.<br>3. Нативні додатки часто демонструють кращу інтеграцію з апаратним забезпеченням пристрою, що дозволяє оптимізувати роботу з камерами, сенсорами, GPS та іншими елементами. | 1. Для кожної платформи (iOS та Android) необхідно писати окремий код, що подовжує час розробки і збільшує витрати на створення додатку.<br>2. Вимагає спеціалізованих знань мов програмування для кожної платформи, таких як Swift або Objective-C для iOS та Java або Kotlin для Android. |
| Крос-платформна розробка | 1. Один код можна використовувати для створення додатків на різних платформах, що значно спрощує процес розробки.<br>2. Забезпечує швидке розгортання проекту та спрощує підтримку додатків на різних операційних системах.<br>3. Використання загальних | 1. Має обмеження в продуктивності та доступності функцій порівняно з нативними додатками, оскільки не завжди повністю використовує специфічні можливості кожної платформи.<br>2. Залежність від фреймворків |

| | | |
|---|---|---|
| | ресурсів і бібліотек дозволяє ефективно зменшити витрати на розробку та прискорити створення додатка. | може бути обмеженням, оскільки вони можуть не підтримувати найновіші функції платформ або мати обмежену гнучкість у порівнянні з нативними інструментами. |
| Веб-додатки | 1. Веб-технології є широко поширеними та знайомими для більшості розробників, що полегшує процес розробки.<br>2. Один код може працювати на різних платформах, без необхідності створювати окремі версії для кожної операційної системи.<br>3. Оновлення та розгортання додатку спрощуються завдяки використанню веб-технологій. | 1. Веб-додатки мають обмежений доступ до апаратного забезпечення пристрою та операційної системи, що знижує їх функціональність порівняно з нативними додатками.<br>2. Веб-додатки не можуть використовувати всі можливості платформи, такі як повний доступ до API та специфічних функцій. |
| Гібридна розробка | 1. Використання веб-технологій для розробки інтерфейсу дозволяє пришвидшити процес створення додатку.<br>2. Можливість запуску одного коду на різних платформах знижує витрати часу і ресурсів.<br>3. Швидкий цикл розробки та оновлення. | 1. Обмежена доступність функцій і можливостей платформи порівняно з нативною розробкою.<br>2. Залежність від фреймворків та інструментів розробки. |

Ключова відмінність між нативними та крос-платформними додатками полягає в їх продуктивності та здатності виконувати низькорівневі задачі. Нативні додатки зазвичай забезпечують кращу швидкість та доступ до специфічних функцій кожної платформи, що робить їх ідеальним вибором для великих проектів, які вимагають високих вимог до продуктивності.

Однак для малого та середнього бізнесу, а також IT-стартапів, крос-платформна розробка є ефективною альтернативою, оскільки дозволяє значно зекономити час і ресурси, розробляючи один додаток, що працює на обох основних платформах — iOS та Android.

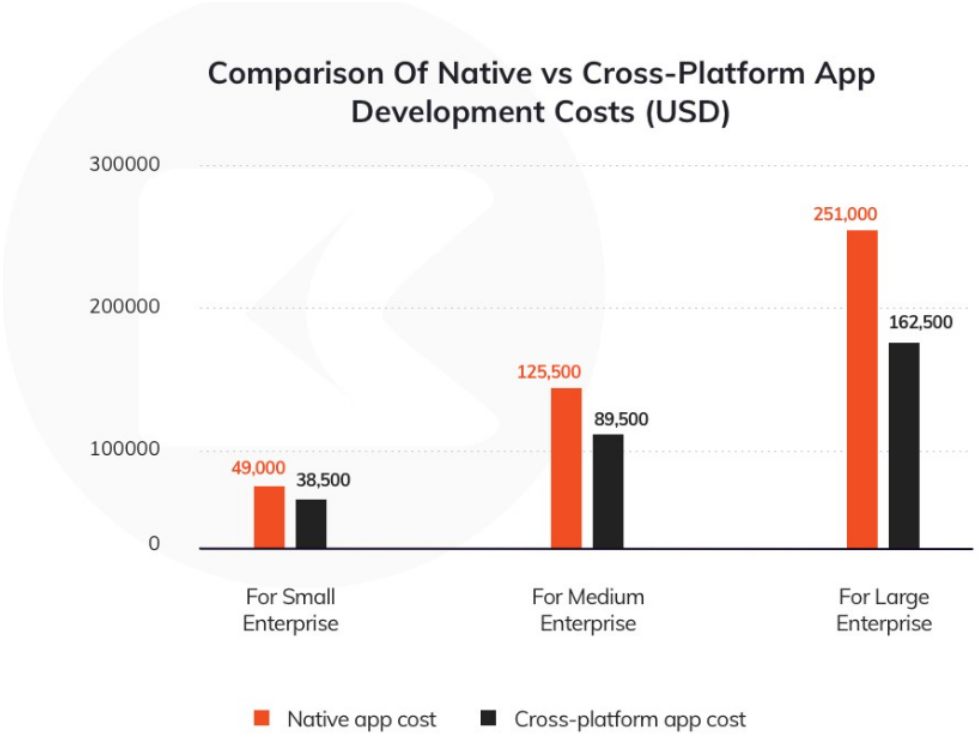


Рисунок 1.1 — Порівняння ціни нативної та крос-платформної [3]

На рисунку 1.1 представлено детальне порівняння затрат, що виникають при виборі конкретної технології. Для малого бізнесу та стартапів найбільше значення мають швидкість виходу на ринок, вартість та простота розробки. Ці фактори роблять крос-платформну розробку вигідним варіантом, оскільки вона дозволяє швидше реалізувати продукт, зменшуючи витрати на створення окремих додатків для кожної платформи.

### 1.4 Постановка задачі

В рамках даної роботи поставлено завдання проектування та розробка інформаційної технології створення навчального додатку для вивчення іноземної мови з можливістю голосового введення слів. Для досягнення цієї мети необхідно виконати наступні кроки:

1. Провести аналіз існуючих технологій створення мобільних додатків, їх особливостей, переваг та недоліків, а також огляд аналогів додатків для вивчення іноземних мов.
2. Розробити архітектуру клієнтської частини мобільного додатку, яка включає функції голосового введення (Speech-to-Text) та відтворення тексту (Text-to-Speech).
3. Створити макет мобільного додатку, який враховує особливості платформ iOS та Android, використовуючи принципи адаптивного дизайну для забезпечення зручності та інтуїтивності користувацького інтерфейсу.
4. Виконати програмну реалізацію мобільного додатку, впровадивши функціонал пошуку, створення та перегляду слів і перекладів, організації даних у папки та виведення статистики (рис 1.2).
5. Виконати мануальне та автоматизоване тестування, для перевірки коректності функціоналу та стабільності роботи.

Реалізація даних задач забезпечить створення сучасного мобільного додатку з впровадженням голосових технологій та інтуїтивно зрозумілого інтерфейсу, який сприятиме ефективному вивченню іноземної мови.

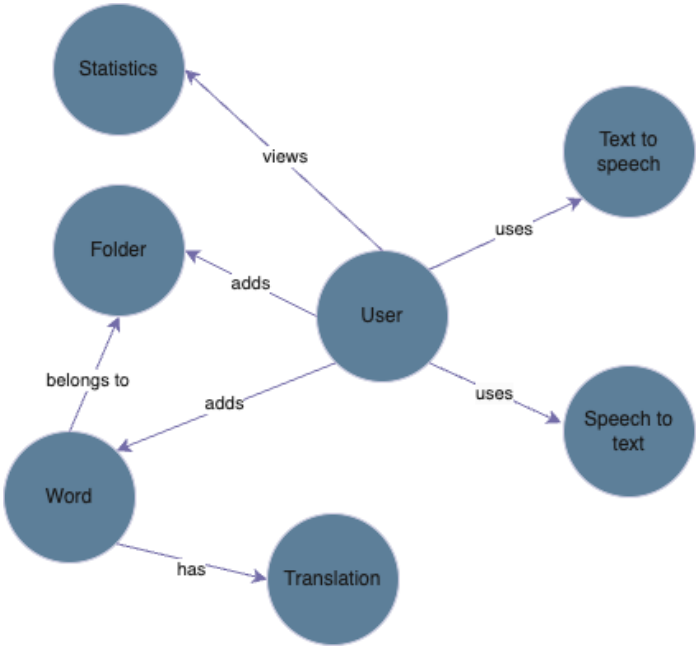


Рисунок 1.2 — ERD-діаграма функціоналу системи

# 2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

## 2.1 Дизайн мобільного додатку

### 2.1.1 Принципи побудови дизайну

При створенні дизайну мобільного додатку слід враховувати ряд основних принципів, що забезпечують інтуїтивність, зручність та привабливість інтерфейсу. Ці принципи спрямовані на підвищення ефективності навчання користувачів та створення комфортної взаємодії з додатком. Основні принципи включають [4]:

1. Простота та мінімалізм. Простий інтерфейс дозволяє користувачам легко орієнтуватися в додатку, без перевантаження зайвими елементами. Мінімалістичний дизайн допомагає зосередитися на головних функціях, уникаючи зайвих відволікаючих деталей. Це особливо важливо для освітніх додатків, де користувач має можливість концентруватися на навчанні, а не на освоєнні інтерфейсу.
2. Ієрархія та структура. Добре організована структура інтерфейсу дозволяє користувачам легко знаходити потрібну інформацію та функції. Використання чіткої ієрархії елементів, таких як заголовки, підзаголовки, іконки та розділи, допомагає покращити орієнтацію та сприяє інтуїтивному використанню додатку. Наприклад, вкладка для збережених слів та вкладка для статистики мають бути чітко розмежовані.
3. Гамма та контрастність. Використання збалансованої кольорової гами з акцентами на ключових елементах дозволяє направляти увагу користувача та створювати приємне візуальне враження. Контрастність між текстом та фоном забезпечує читабельність, що є особливо важливим для додатків, де користувачі працюють із текстовою

інформацією. Пастельні або нейтральні тони з акцентними кольорами можуть сприяти сприйняттю додатку як навчального інструменту.

4. Узгодженість елементів. Важливо, щоб усі елементи інтерфейсу, такі як кнопки, значки, шрифти, дотримувались єдиного стилю. Це допомагає створити візуальну гармонію і покращує сприйняття додатку, роблячи його більш цілісним. Узгодженість також стосується відступів, розмірів елементів і стилю анімацій.
5. Візуальні підказки та навчальні елементи. Враховуючи, що додаток є освітнім, корисно включити візуальні підказки та інструкції для нових користувачів. Це можуть бути спливаючі повідомлення, що пояснюють основні функції або поради щодо ефективного навчання. Це дозволяє користувачам швидко освоїти додаток і скористатися його можливостями для кращих результатів у вивченні мови.
6. Адаптивність під різні платформи та розміри екранів. Додаток повинен бути адаптивним, тобто підлаштовуватися під різні розміри екранів, забезпечуючи зручне використання як на смартфонах, так і на планшетах. Врахування особливостей Android та iOS під час дизайну дозволить створити максимально ефективний інтерфейс для користувачів обох платформ.
7. Ефективне використання жестів та голосових команд. З огляду на мету додатку — вивчення іноземної мови — важливо передбачити підтримку жестів для спрощення навігації та функціональності, а також інтеграцію голосових команд. Це дозволить користувачам додавати нові слова або шукати переклади за допомогою голосу, що підвищує зручність використання і додає інтерактивності.

Загалом, дотримання цих принципів сприяє створенню дизайну, який є водночас функціональним і привабливим для користувачів, сприяючи ефективному процесу навчання [5].

### 2.1.2 Відмінності в розробці дизайну для Android та iOS

Хоч сучасні версії Android та iOS мають багато схожих функцій, розробка дизайну для цих операційних систем має свої унікальні особливості. Далі розглянемо ключові відмінності [6].

Стиль та компоненти інтерфейсу. У дизайні для Android використовується концепція матеріального дизайну (Material You), яка імітує фізичні об'єкти та їх взаємодію. Основні характеристики матеріального дизайну включають використання тіней, градієнтів, багатошарових елементів, заокруглених форм, іконок та анімацій. Цей стиль створює враження глибини та об’ємності. Натомість iOS зосереджується на мінімалізмі та чистоті плоского дизайну, де переважає простота, відсутність зайвих деталей і акцент на чітких формах та однотонних елементах, що робить інтерфейс лаконічним і візуально легким.

Навігація та способи взаємодії. В Android інтерфейс включає три основні кнопки в нижній частині екрану: "Назад", "Додому" та "Огляд відкритих додатків". Ці кнопки дозволяють користувачам легко переміщатися між екранами, виходити на головний екран і відкривати список запущених додатків. У iOS управління здебільшого відбувається за допомогою жестів, як-от свайпи для повернення назад або перемикання між додатками, що дозволяє уникнути фізичних або віртуальних кнопок. У старіших моделях iPhone використовувалась фізична кнопка Home, яка виконувала схожі функції, але в нових моделях вона замінена на повноекранну навігацію за допомогою жестів. Ці різні підходи до управління безпосередньо впливають на те, як дизайнери підходять до побудови інтерфейсу.

Розміри та роздільна здатність дисплеїв. Ще однією відмінністю є різноманітність пристроїв. Екосистема Android включає безліч пристроїв з різними розмірами екранів та роздільною здатністю, що коливається від

невеликих телефонів до великих планшетів. Щільність пікселів (DPI) також сильно варіюється, від низького DPI у бюджетних моделях до дуже високого у флагманських пристроях. Це створює додаткові вимоги до дизайну, який має адаптуватися до широкого спектру екранів і забезпечувати гарний вигляд на будь-якому пристрої. В iOS цей процес дещо спрощується, оскільки Apple має обмежений асортимент моделей iPhone з визначеними розмірами та роздільними здатностями. Це дозволяє дизайнерам створювати більш уніфікований інтерфейс, який не потребує масштабної адаптації для великої кількості пристроїв.

Таким чином, розробка дизайну для Android і iOS вимагає врахування різних підходів та особливостей кожної платформи, щоб забезпечити користувачам максимально зручний і естетичний інтерфейс.

### 2.1.3 Дизайн мобільного додатку

На основі викладених вище принципів та специфічних відмінностей у підходах до розробки дизайну для платформ Android та iOS, у цьому розділі буде представлено дизайн мобільного додатку, адаптований до особливостей кожної системи. Враховуючи матеріальний дизайн Android та плоский мінімалістичний стиль iOS, мобільний інтерфейс розроблений таким чином, щоб забезпечити інтуїтивну взаємодію та зручність користувача незалежно від обраної операційної системи.

**1.** Головний екран

Головний екран — це стартова сторінка програми, на якій користувач може побачити основні розділи для вивчення та організації вивчених слів. Він включає різні папки з категоріями слів, такі як «My Words», «Archive», «University», що дозволяє структуровано зберігати та систематизувати нову лексику. Завдяки цьому інтерфейсу користувач отримує можливість швидкого доступу до часто використовуваних слів і фраз.

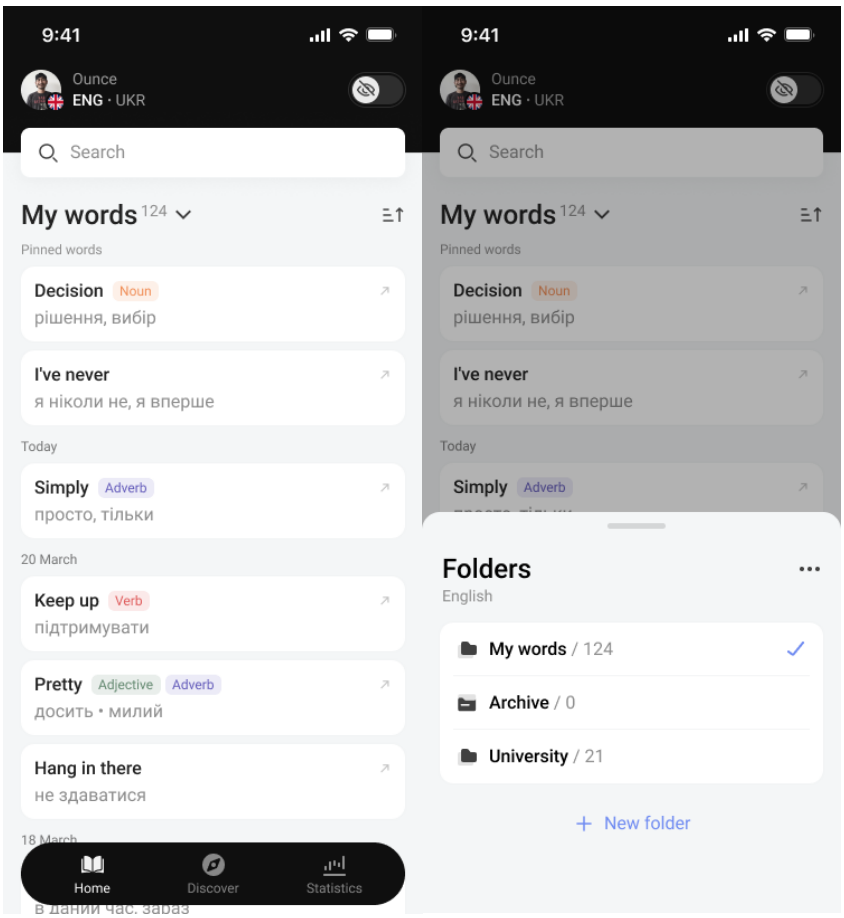


Рисунок 2.1 — Дизайн головного екрану

Функції:

- Пошук: Зручна функція, що дозволяє швидко знайти слово в базі програми або серед уже збережених користувачем слів. Це зручно для повторення або коли необхідно швидко знайти переклад.
- Додавання слів у папки: Користувач може створювати нові папки для класифікації лексики за темами чи рівнем складності.
- Система організації: Можливість створювати та редагувати папки дозволяє легко зберігати нові слова за тематичними категоріями, наприклад, для університетського навчання або повсякденного використання.

2. Деталі слова

При виборі конкретного слова користувач потрапляє на екран з детальною інформацією про це слово. Тут відображаються різні переклади, граматичні категорії, приклади використання в реченнях, синоніми та інші

корисні деталі. Це допомагає користувачу глибше зрозуміти значення слова та його вживання в різних контекстах.

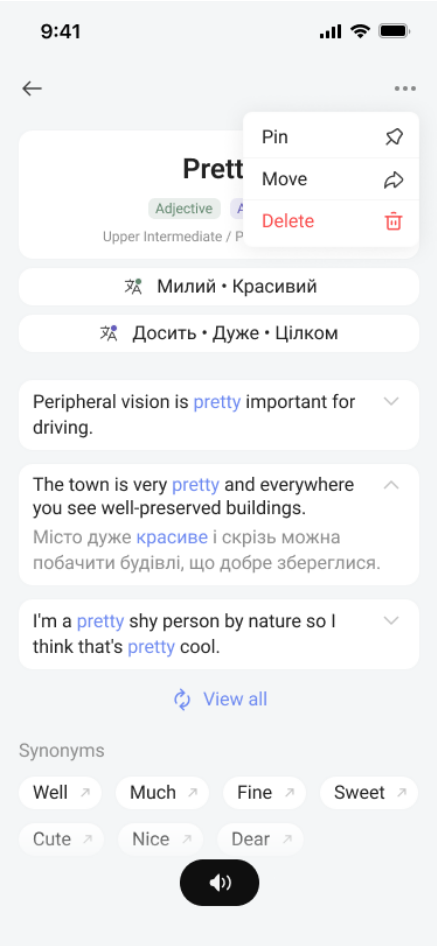


Рисунок 2.2 — Дизайн екрану слова

Функції:

- Приклади речень: Містить приклади використання слова в різних контекстах, що полегшує запам’ятовування та застосування на практиці.
- Синоніми та значення: Допомагає користувачеві краще зрозуміти суть слова, надаючи список синонімів для більш глибокого засвоєння.
- Функція голосового відтворення: Дозволяє прослухати вимову слова, що особливо корисно для тих, хто вивчає мову самостійно.

3. Пошук слова

Розділ для пошуку слів є одним із найважливіших компонентів системи. Користувач може використовувати як текстовий ввід, так і голосовий, що дозволяє шукати слова в будь-який момент і без зайвих зусиль. Пошуковий інтерфейс відображає результати з декількома варіантами перекладу, а також дає можливість додавати власні значення, якщо слово не було знайдено.

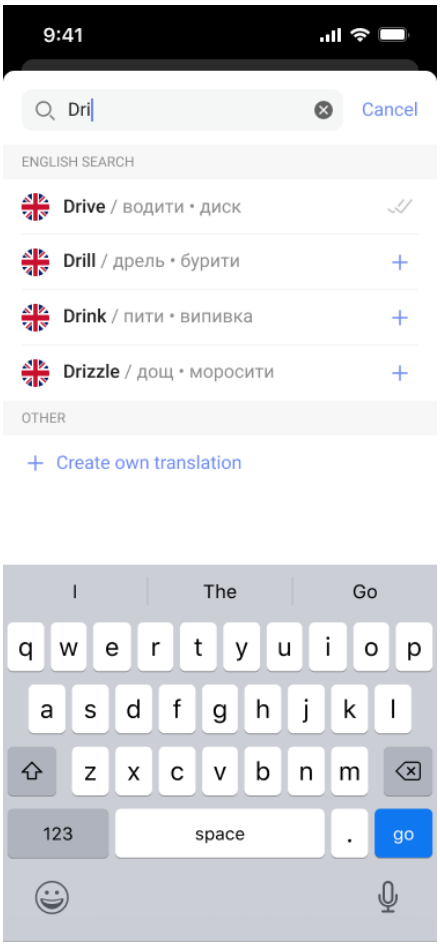


Рисунок 2.3 — Дизайн екрану пошуку слова

Функції:

- Голосовий пошук: Дає змогу користувачеві скористатися голосовим введенням для пошуку нових слів. Це зручна функція, особливо коли потрібно дізнатися значення слова «на ходу».
- Перегляд результатів: Показує список слів із різними варіантами перекладу, що дозволяє вибрати найбільш відповідний контекст.

- Створення власного перекладу: Якщо слово відсутнє у базі даних, користувач може додати власний переклад або значення, що дозволяє адаптувати додаток під індивідуальні потреби.

4. Додавання власного перекладу

У цьому розділі користувач може додати свій переклад слова або фрази, що корисно, якщо базовий словник програми не містить специфічних термінів чи слів. Така можливість особливо важлива для професійного лексикону, наприклад, у науковій або технічній сфері.

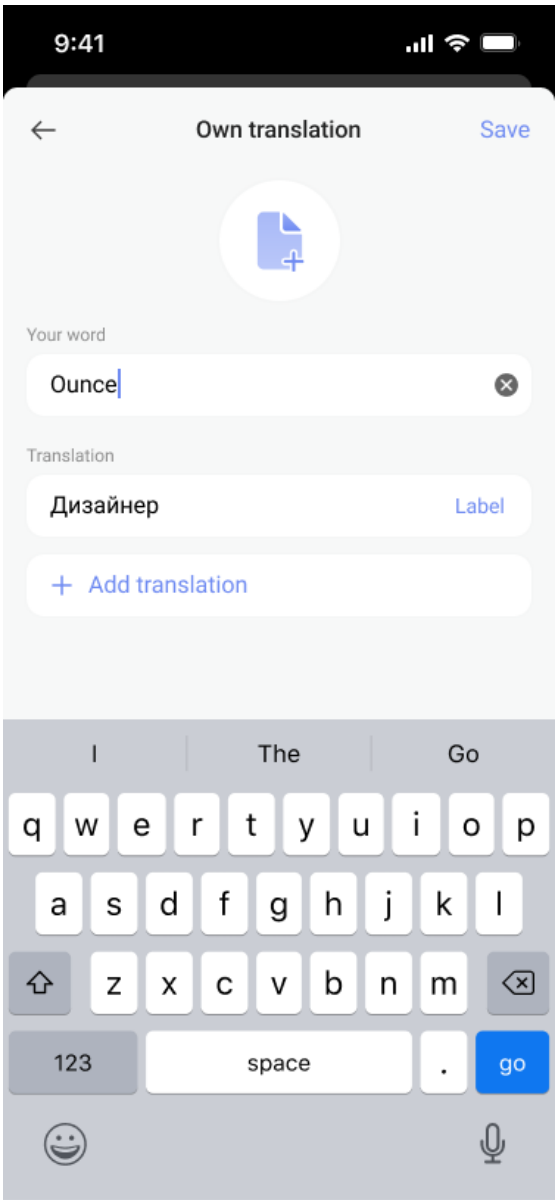


Рисунок 2.4 — Дизайн екрану додавання власного перекладу

Функції:

- Введення слова та перекладу: Користувач може ввести слово, його переклад або кілька варіантів значення, що робить додаток ще більш гнучким.
- Категоризація слова: Дозволяє додати мітки або категорії для кожного доданого перекладу, що полегшує організацію та пошук в майбутньому.

Важливі компоненти дизайну

- Чітка типографія: Великий розмір шрифтів і виділення важливих елементів роблять додаток зручним для читання.
- Інтуїтивна навігація: Простий інтерфейс і мінімалістичний дизайн дозволяють легко орієнтуватись.
- Голосовий ввід: Спрощує процес додавання та пошуку нових слів, особливо для користувачів, які вивчають вимову.

Цей дизайн підкреслює простоту використання і велику кількість можливостей для ефективного вивчення іноземної мови.

## 2.2 Клієнтська частина мобільного додатку

### 2.2.1 JavaScript у мобільній розробці

JavaScript є одним із найпопулярніших мов програмування, яка широко застосовується в мобільній розробці. Завдяки своїй гнучкості та універсальності, JavaScript дозволяє створювати інтерактивні інтерфейси та динамічні функції у веб та мобільних додатках. У контексті мобільної розробки JavaScript використовується у фреймворках, таких як React Native, який дозволяє будувати кросплатформенні додатки для iOS та Android на основі єдиного коду [7].

Основні особливості JavaScript у мобільній розробці [8]:

- Динамічна типізація — JavaScript підтримує динамічну типізацію, що спрощує роботу з даними, але може призводити до помилок на етапі виконання.
- Широке розповсюдження — JavaScript має величезну кількість бібліотек та фреймворків, що полегшують розробку.
- Кросплатформенність — JavaScript працює на різних платформах, що дозволяє використовувати його для створення додатків для різних операційних систем.

JavaScript є основою для фреймворків на зразок React Native, що дозволяє створювати мобільні додатки з використанням веб-технологій, не вдаючись до нативного коду для кожної платформи. Однак динамічна типізація JavaScript іноді призводить до ускладнень у виявленні помилок, що є одним із недоліків мови.

### 2.2.2 TypeScript: надбудова над JavaScript для надійності

TypeScript — це надбудова над JavaScript, яка дозволяє розробникам використовувати статичну типізацію та інші корисні інструменти для покращення надійності та стабільності коду. TypeScript підтримує всі функції JavaScript, але додає можливість задавати типи для змінних, функцій та об'єктів, що дозволяє уникати багатьох поширених помилок [9].

Основні переваги TypeScript у розробці мобільних додатків [10]:

- Статична типізація — дає змогу виявляти помилки ще на етапі компіляції, що значно знижує ймовірність помилок під час виконання.
- Покращена підтримка інструментів — компілятор TypeScript (tsc) перетворює код на чистий JavaScript, сумісний з усіма сучасними браузерами та середовищами.
- Сумісність з JavaScript — будь-яка коректна програма на JavaScript є коректною програмою на TypeScript, що спрощує перехід до використання TypeScript у вже існуючих проєктах.

Завдяки компілятору tsc можна виявити численні дефекти ще до випуску застосунку. Він перетворює TypeScript-код на JavaScript і аналізує програму для виявлення потенційних помилок. TypeScript дозволяє створювати файли js для будь-якої версії JavaScript, починаючи з ES3, і підтримує можливість встановлювати різні правила перевірки коду, вмикаючи або вимикаючи їх за потреби.

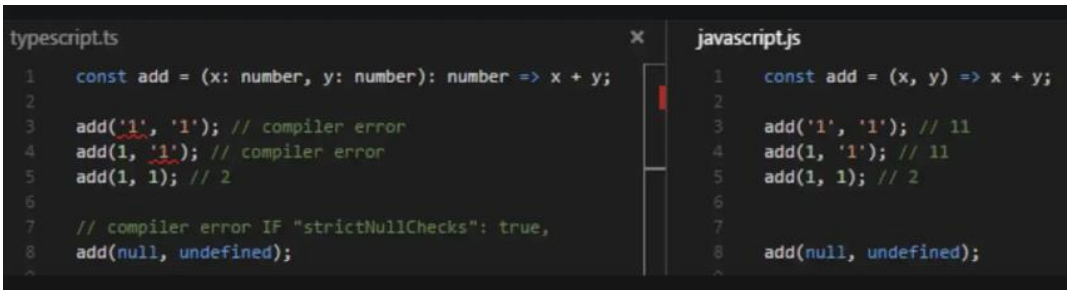


Рисунок 2.5 – TypeScript vs JavaScript [11]

TypeScript дозволяє уникнути багатьох помилок, пов'язаних із динамічною типізацією JavaScript, і тим самим сприяє створенню більш стабільного та надійного додатка. Завдяки використанню TypeScript, розробники можуть створювати коди, які легше підтримувати та розвивати, а також значно знижувати кількість помилок, викликаних людським фактором.

### 2.2.3 React Native: Кросплатформенний фреймворк

React Native є сучасним фреймворком, який дозволяє розробникам швидко створювати мобільні додатки для різних платформ. Його головна перевага полягає в можливості розробити один проєкт, який працює як на iOS, так і на Android. Завдяки цьому великі компанії, як Facebook, Instagram, Skype, Pinterest, Uber, Tesla, SoundCloud та інші, ефективно застосовують React Native у своїх продуктах [12].

Основні переваги React Native [13]:

- Підтримується Facebook та великою спільнотою розробників.

- Ліцензія з відкритим кодом.
- Кросплатформенність, що дозволяє створювати додатки для iOS і Android, значно скорочуючи час і ресурси, адже немає потреби писати код окремо для кожної платформи.
- Архітектура, що використовує нативні компоненти, забезпечує високу продуктивність додатків, що близька до нативних рішень.
- Широка спільнота розробників, які постійно підтримують та розвивають фреймворк, пропонує безліч корисних ресурсів, документацій, бібліотек та інструментів для спрощення процесу розробки.

Недоліки React Native:

- Існують певні обмеження у керуванні пам’яттю, що може впливати на продуктивність додатків.
- Використання JavaScript може спричиняти проблеми з безпекою.
- Не всі модулі та API доступні, що може потребувати додаткових зусиль для розробки певних функцій.

Загалом, React Native — це гарний вибір для створення мобільних додатків завдяки зручності використання, швидкості розробки та мінімальним втратам продуктивності.

## 2.3 Голосове введення та відтворення тексту у мобільних додатках

Голосове введення та відтворення тексту — це інноваційні технології, що значно розширюють можливості мобільних додатків для вивчення іноземних мов. Вони дозволяють користувачам вводити текст за допомогою голосу та прослуховувати правильну вимову слів і фраз, що особливо важливо для закріплення навичок усного мовлення. Цей розділ розгляне ключові аспекти реалізації голосового введення і відтворення тексту, їхні переваги, виклики, а також особливості інтеграції цих технологій у мобільні додатки.

### 2.3.1 Принципи роботи голосового введення

Голосове введення є сучасною технологією, яка дозволяє користувачеві взаємодіяти з мобільним додатком без використання клавіатури, здійснюючи пошук слів, введення тексту, створення нотаток або навіть керування додатком за допомогою голосових команд. Ця функція стає можливою завдяки технологіям автоматичного розпізнавання мовлення (ASR — Automatic Speech Recognition), які забезпечують перетворення аудіосигналу на текст. Цей процес включає кілька основних етапів:

1. Захоплення голосового сигналу. Користувач активує функцію голосового введення, натискаючи на відповідну кнопку або даючи спеціальну голосову команду. Додаток починає записувати його голосовий сигнал, забезпечуючи прийом аудіоінформації в реальному часі.
2. Попередня обробка звукових хвиль. Записаний аудіосигнал обробляється для виділення найважливіших характеристик звуку. Цей етап включає фільтрацію шумів і фонового звуку, аналіз частотних спектрів, а також виділення інтонації та інших параметрів. Попередня обробка дозволяє покращити якість сигналу, роблячи його більш чітким і зрозумілим для подальшого аналізу.
3. Аналіз та розпізнавання мовних одиниць. На цьому етапі оброблений сигнал порівнюється з базою даних мовних зразків і фонем — основних звукових одиниць. Звукові хвилі розбиваються на короткі відрізки, які система аналізує, визначаючи, які саме звуки і слова були вимовлені. Сучасні системи використовують глибокі нейронні мережі для розпізнавання мовлення, що дозволяє їм краще інтерпретувати складні звукові комбінації, враховуючи контекст і фонетичні закономірності мови.

4. Перетворення в текст. Після того, як мовні одиниці розпізнані, система об’єднує їх у слова і фрази, формуючи текстову форму мовлення користувача. Цей текст відображається на екрані або використовується додатком для виконання певних команд.

Сучасні мобільні додатки зазвичай інтегрують функцію голосового введення через хмарні сервіси, такі як Google Speech-to-Text API або Apple Speech Framework. Використання хмарних технологій дозволяє значно підвищити точність розпізнавання, оскільки обробка аудіоданих відбувається на віддалених потужних серверах, які мають доступ до великих обсягів обчислювальних ресурсів та складних нейронних моделей. Це дозволяє застосовувати більш точні алгоритми обробки сигналів і зменшує потребу у використанні ресурсів пристрою користувача.

### 2.3.2 Відтворення тексту (TTS)

Технологія відтворення тексту, або TTS (Text-to-Speech), дозволяє конвертувати текстовий контент у звукову форму, завдяки чому користувачі можуть прослуховувати інформацію, а не читати її. Ця технологія особливо корисна у мовних додатках, оскільки забезпечує можливість слухати правильну вимову слів, фраз та виразів, що сприяє розвитку навичок сприйняття іноземної мови на слух і покращенню вимови. Завдяки TTS користувач може повторювати за диктором, слухати інтонаційні особливості мовлення, що дуже важливо для практичного засвоєння нової мови. Технологія TTS активно застосовується у додатках для навчання, інтерактивних системах та мобільних додатках для людей з обмеженими можливостями.

Принцип роботи TTS включає декілька етапів:

1. Синтаксичний аналіз тексту. Перший етап полягає в аналізі структури тексту, зокрема розподілу речень, фраз та частин мови (іменників, дієслів тощо). Програма визначає синтаксичні особливості тексту, такі

як розділові знаки, паузи, інтонаційні наголоси, що дозволяє створити правильну структуру мовного потоку. Важливим елементом цього етапу є врахування контексту, щоб забезпечити точну інтерпретацію і уникнути неоднозначності в промові.

2. Перетворення в фонетичні одиниці. Після аналізу синтаксичної структури текст розбивається на фонетичні одиниці — звуки, які відповідають конкретній мові. Кожен звук або фонема обирається з урахуванням правильного наголосу та тональності. У деяких мовах це також передбачає додаткову обробку для відображення специфічних звукових елементів, таких як відтінки вимови у діалектах або інтонаційні особливості. На цьому етапі система виконує свого роду "переклад" тексту на фонетичні символи, який і буде озвучений.
3. Синтез звуку. Після перетворення тексту у фонетичні одиниці відбувається процес синтезу звуку. Фонетичні одиниці перетворюються на звуковий сигнал за допомогою мовного синтезатора. Сучасні системи TTS, зокрема Google Text-to-Speech, Microsoft Azure Speech та інші, використовують глибокі нейронні мережі або штучний інтелект для створення максимально природного звучання голосу. Такі технології дозволяють враховувати інтонацію, тембр та емоційний забарвлення мовлення, роблячи його більш живим і природним. Синтезований голос може мати різну тональність, залежно від налаштувань, що дозволяє адаптувати його для різних цільових груп.
4. Формування готового звукового файлу. У деяких системах, після синтезу голосу, створюється звуковий файл, який можна прослухати або зберегти для подальшого використання. Наприклад, у навчальних додатках для вивчення мов відтворення тексту у звуковому форматі може бути записане і збережене, щоб користувач міг прослуховувати матеріал навіть в автономному режимі.

У мовних додатках особливу увагу приділяють використанню спеціалізованих голосових моделей, адаптованих під різні мови і діалекти. Це забезпечує високу якість озвучування, яка враховує фонетичні і культурні особливості мови, що вивчається, і дозволяє користувачам чути правильну вимову навіть для рідкісних або складних мов. Наприклад, у випадку з китайською мовою важливо зберегти точність тонів, а для французької — інтонацію. Деякі системи також дозволяють вибирати між чоловічими і жіночими голосами, що надає додаткову гнучкість користувачам і сприяє підвищенню інтересу до навчання.

Переваги TTS у мобільних додатках

Технологія TTS має ряд значних переваг для користувачів. Вона допомагає людям з порушенням зору або труднощами читання, дозволяючи їм прослуховувати текстовий контент. Це робить інформацію доступнішою і більш зручною для сприйняття. Також TTS сприяє розвитку навичок слухання, що особливо корисно для людей, які вивчають нову мову. Окрім того, технологія TTS робить використання додатків більш універсальним, оскільки користувачі можуть отримувати інформацію у будь-який момент, навіть коли читання неможливе — під час водіння або спортивних тренувань.

Виклики у застосуванні TTS

Незважаючи на значні переваги, TTS має і свої недоліки. Наприклад, відтворення тексту може бути складним для мов з багатозначністю та складною фонетикою. Система також може мати труднощі з відтворенням складних інтонаційних структур або зі специфічними акцентами. Крім того, для досягнення максимально природного звучання потрібні значні обчислювальні ресурси, особливо якщо використовуються складні нейронні мережі.

Загалом, TTS є потужним інструментом, який сприяє підвищенню доступності мобільних додатків, а також є корисним для навчання мов. Він дозволяє створювати інтерактивні та інклюзивні додатки, які забезпечують новий рівень користувацького досвіду.

### 2.3.3 Переваги використання голосових технологій

Інтеграція технологій голосового введення (ASR) і відтворення тексту (TTS) у мобільні додатки забезпечує низку вагомих переваг, які підвищують ефективність, доступність та зручність використання таких додатків для різних категорій користувачів. Нижче наведені основні переваги використання цих технологій [14]:

1. Зручність і доступність. Завдяки голосовому введенню користувачі можуть швидко та зручно вводити текстові дані, просто вимовляючи слова або фрази. Це особливо корисно, коли потрібно знайти інформацію чи ввести текст в умовах, де неможливо використовувати клавіатуру, наприклад, під час руху або в транспорті. Голосове введення значно прискорює взаємодію з додатком, що робить процес навчання більш зручним і менш трудомістким. Користувач може просто продиктувати потрібний текст, а система розпізнає його й виведе у текстовій формі, що знижує потребу в постійному введенні інформації вручну.
2. Покращення вимови та розуміння мови. Відтворення тексту дозволяє користувачам чути правильну вимову слів і фраз, що є надзвичайно важливим для вивчення іноземної мови. Прослуховуючи вимову, користувач може імітувати її, покращуючи власну вимову й інтонацію, що сприяє розвитку усного мовлення та навичок аудіювання. Крім того, системи TTS можуть відтворювати різні варіанти вимови, наприклад, для різних діалектів або акцентів, що дозволяє користувачам ознайомитися з мовною різноманітністю. Це також допомагає краще розуміти іноземну мову на слух, що є важливим для повсякденного спілкування.
3. Інтерактивність та залучення. Можливість взаємодії з додатком через голосові команди робить процес навчання більш захопливим та інтерактивним. Замість звичайного використання клавіатури,

користувач може активувати різні функції додатку голосом, наприклад, почати нове завдання або повторити фразу для покращення вимови. Це створює ефект "живого" спілкування з додатком, роблячи навчальний процес більш природним і цікавим. Голосові команди забезпечують новий рівень інтерактивності, дозволяючи користувачам безпосередньо взаємодіяти з програмою та отримувати миттєвий зворотний зв’язок, що мотивує їх продовжувати навчання.

4. Доступність для людей з обмеженими можливостями. Голосові технології суттєво покращують доступність додатків для користувачів з інвалідністю. Зокрема, вони відкривають доступ до навчання для людей із порушеннями зору або з труднощами у наборі тексту. Завдяки голосовому введенню такі користувачі можуть без зайвих труднощів вводити текст, а з використанням TTS вони можуть прослуховувати текстову інформацію. Це розширює можливості для навчання і робить освітні додатки інклюзивнішими. Відтворення тексту у вигляді звуку дозволяє людям з вадами зору повноцінно взаємодіяти з додатком і використовувати його функції нарівні з іншими користувачами.
5. Ефективність навчання. Голосові технології надають користувачам додаткові можливості для більш інтенсивного і різностороннього навчання. Наприклад, слухаючи текст, користувачі краще запам’ятовують матеріал, оскільки задіяні кілька сенсорних каналів — слух і зір. Це підсилює ефект запам’ятовування і робить навчання більш результативним. Крім того, TTS може озвучувати слова чи речення з різною швидкістю, що дозволяє користувачам пристосовувати навчальний процес до своїх потреб і комфортно засвоювати новий матеріал у власному темпі.

Таким чином, інтеграція голосового введення та відтворення тексту в мобільні додатки для вивчення мов не лише розширює функціонал, але й підвищує ефективність навчання, забезпечуючи користувачів зручними та доступними інструментами для вивчення нових мов у будь-яких умовах.

### 2.3.4 Виклики та обмеження

Незважаючи на значні переваги, інтеграція технологій голосового введення (ASR) і відтворення тексту (TTS) у мобільні додатки пов’язана з низкою викликів та обмежень, які можуть впливати на якість і зручність їх використання. Розглянемо основні з них [15]:

1. Якість розпізнавання мови. Точність роботи ASR може залежати від різних факторів, таких як акценти, індивідуальні особливості вимови та фоновий шум. Наприклад, ASR-системи можуть мати труднощі з розпізнаванням мови користувачів з сильними акцентами або незвичною інтонацією. Фонові шуми, особливо в умовах громадських місць або на вулиці, можуть суттєво знижувати точність розпізнавання, що створює незручності для користувачів. Враховуючи ці фактори, забезпечення високої якості розпізнавання мови залишається важливим викликом для розробників голосових технологій.
2. Обмежена мовна підтримка. Не всі мови та діалекти однаково добре підтримуються сучасними сервісами ASR та TTS. Більшість існуючих технологій добре справляються з основними міжнародними мовами, такими як англійська, іспанська або французька, але можуть мати труднощі з розпізнаванням менш розповсюджених мов або локальних діалектів. Це обмеження особливо актуальне для мовних додатків, орієнтованих на вивчення мов, де користувачі можуть стикатися з необхідністю працювати з рідкісними або регіональними мовами. Щоб вирішити цю проблему, розробникам необхідно постійно розширювати базу підтримуваних мов і діалектів, що є складним і ресурсозатратним процесом.
3. Споживання ресурсів мобільного пристрою. Використання голосових технологій може значно збільшити споживання батареї, а також обсяг переданих даних. Голосові сервіси, особливо ті, що працюють на основі хмарних обчислень, вимагають постійного інтернет-з’єднання

для передавання аудіоданих та обробки запитів на сервері. Це може призводити до високих витрат мобільного трафіку, що є небажаним для користувачів з обмеженими інтернет-пакетами. Крім того, постійна обробка звуку та передача даних може швидко розряджати батарею пристрою, що знижує автономність мобільних додатків і робить їх менш зручними для тривалого використання.

4. Питання конфіденційності та безпеки даних. Багато голосових технологій використовують хмарні сервіси для обробки даних, що може викликати побоювання щодо конфіденційності та безпеки персональної інформації користувачів. Для розпізнавання мови або синтезу голосу додатки часто надсилають аудіозаписи користувачів на віддалені сервери для обробки, що може містити ризик несанкціонованого доступу або витоку даних. Це особливо важливо для освітніх додатків, де користувачі можуть ділитися персональними даними або конфіденційною інформацією. Розробники мають докладати зусиль для забезпечення надійного шифрування і захисту даних, а також надавати користувачам можливість контролювати, які дані вони готові ділити з сервісом.
5. Складність локалізації та культурних адаптацій. Голосові технології потребують врахування культурних і мовних особливостей кожного регіону, де використовується додаток. Системи ASR і TTS повинні не тільки розпізнавати мову, але й коректно інтерпретувати культурно залежні фрази, діалекти і вимови. Цей виклик вимагає від розробників адаптації голосових технологій під потреби різних культур і мовних груп, що додає складнощів у процес інтеграції та підтримки цих систем у додатках.

Таким чином, хоча голосові технології відкривають нові можливості для мобільних додатків, їх впровадження потребує ретельного опрацювання, врахування потенційних викликів і постійного удосконалення. Розробникам слід знаходити баланс між зручністю та ефективністю роботи голосових

функцій, забезпечуючи при цьому високий рівень конфіденційності й доступності для всіх користувачів.

### 2.3.5 Особливості реалізації в мобільних додатках

Розробка мобільних додатків з функціями голосового введення та відтворення тексту (TTS) вимагає врахування специфічних технічних і користувацьких аспектів, що забезпечують якісну інтеграцію цих технологій. Важливим є комплексний підхід до архітектури системи, яка б могла підтримувати точність розпізнавання мови та синтезу звуку, враховуючи обмеження мобільних пристроїв і мережі. Основні аспекти реалізації такі [16]:

1. Інтеграція API або SDK для розпізнавання мови та синтезу. Використання спеціалізованих API та SDK для розпізнавання мови (ASR) та синтезу тексту (TTS) є ключовим кроком у реалізації голосових технологій. Наприклад, для Android можна застосовувати Google Speech API, а для iOS — Apple Speech Framework. Ці сервіси забезпечують високу точність розпізнавання та дозволяють адаптувати голосові технології до потреб користувачів. Розробники можуть також інтегрувати додаткові сторонні рішення (наприклад, Microsoft Azure Speech або Amazon Polly) для підтримки кросплатформенності або забезпечення більш широкої мовної підтримки.
2. Налаштування параметрів мови та акценту. Врахування мовної різноманітності користувачів є важливим для точного розпізнавання голосу та природного відтворення тексту. Бажано надати користувачеві можливість обирати мову, акцент і швидкість синтезу, що покращує індивідуальний підхід до користування додатком. Деякі TTS-системи навіть дозволяють обирати між чоловічими і жіночими голосами або варіювати інтонацію, що може сприяти кращому сприйняттю тексту, особливо у додатках для вивчення мов.

3. Оптимізація для роботи в умовах обмеженої пропускної здатності або відсутності підключення до Інтернету. Більшість сучасних ASR і TTS-систем працюють через хмарні сервіси, що потребує постійного інтернет-з’єднання. Однак це може створювати незручності для користувачів, які перебувають в умовах слабкого або нестабільного зв'язку. Щоб вирішити цю проблему, розробники можуть передбачити локальні моделі для офлайн-розпізнавання мови та синтезу, які будуть працювати навіть без підключення до мережі. Наприклад, Google пропонує офлайн-пакети для Android, що дозволяють певний базовий рівень розпізнавання мовлення без інтернету. Хоча ці моделі можуть бути менш точними, вони забезпечують необхідний мінімум для зручності користувача.
4. Забезпечення продуктивності та економії ресурсів. Оскільки голосові технології можуть бути досить вимогливими до ресурсів пристрою, оптимізація продуктивності є важливою складовою розробки. Голосові функції можуть збільшувати використання оперативної пам’яті, центрального процесора і навіть споживання батареї. Розробники мають прагнути до ефективного управління ресурсами, використовуючи оптимізовані алгоритми обробки аудіо та зниження навантаження на систему, щоб не погіршувати продуктивність мобільного додатку.
5. Захист конфіденційності користувачів. Оскільки голосові команди та текст для синтезу часто обробляються на сторонніх серверах, важливо забезпечити конфіденційність і безпеку особистих даних користувачів. Розробники повинні використовувати шифрування аудіо та текстових даних під час передачі, дотримуватися стандартів конфіденційності, таких як GDPR, та надавати користувачам контроль над тим, які дані вони готові надавати додатку.
6. Покращення інтерактивності та зворотного зв’язку. Використання голосових технологій дозволяє створювати інтерактивні навчальні

сценарії, де додаток не лише розпізнає мовлення, але й відповідає користувачеві, коригує вимову або пропонує додаткові приклади. Це особливо важливо для мовних додатків, оскільки інтерактивність покращує мотивацію користувача та сприяє глибшому засвоєнню матеріалу. Наприклад, додаток може відтворювати правильну вимову слів або фраз і дозволяти користувачу повторювати їх до досягнення потрібного рівня точності.

Таким чином, інтеграція голосових технологій у мобільні додатки вимагає ретельного планування та врахування різних аспектів, які можуть вплинути на досвід користувача. Голосові технології значно підвищують функціональність додатків, роблячи процес навчання більш інтерактивним і доступним. Водночас важливо оптимізувати продуктивність та гарантувати конфіденційність користувачів, щоб забезпечити високу якість послуг і задовольнити потреби сучасних користувачів.

# 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

## 3.1 Структура проекту

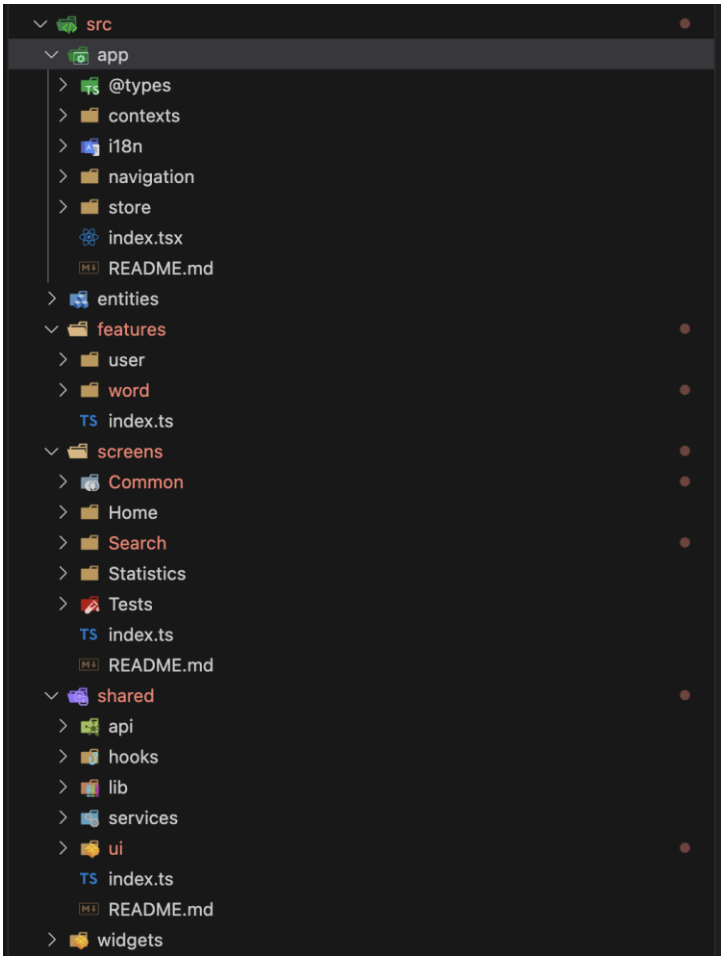


Рисунок 3.1 – Структура клієнтської частини

У процесі розробки мобільного додатк було дотримано модульного підходу для організації структури проекту. Кожен модуль відповідає за певний функціональний аспект, що забезпечує легкість у підтримці, розширюваності та тестуванні коду [17].

Проект розділено на кілька основних каталогів:

1. app - цей каталог відповідає за головну конфігурацію додатка та його початкові налаштування. Його вміст включає:
   - @types – містить глобальні типи, необхідні для “TypeScript”, що забезпечує типізацію в усьому проекті.

- contexts – реалізує контекст API для управління станом (наприклад, багатомовність).
- i18n – забезпечує багатомовність додатка через інтернаціоналізацію.
- navigation – відповідає за навігацію між екранами (Stack, Tabs, Drawer).
- store – зберігання глобального стану додатка.
- index.tsx – точка входу в додаток.

2. Каталог entities містить моделі даних (сутності) додатка. Кожна сутність має свої атрибути й методи для роботи із бізнес-логікою. Наприклад:
   - user – дані про користувача.
   - word – дані про слова в додатку.
3. В каталозі features зосереджено функціональні частини додатка, що реалізують ключові можливості. Структура побудована за принципом поділу за сутностями чи сценаріями використання:
   - user – функціонал, пов’язаний із користувачем (автентифікація, профіль, управління обліковим записом).
   - word – функціонал для роботи з даними слів (наприклад, додавання нового слова, перегляд списку слів, пошук).

   Файл index.ts у цьому каталозі служить для експортів функцій або компонентів з усіх модулів у каталозі features.
4. Каталог screens включає всі екрани додатка, які відображаються користувачеві. Кожен екран логічно розділений за призначенням:
   - Common – спільні компоненти або базові екрани, що можуть використовуватись у різних частинах додатка.
   - Home – головний екран додатка.
   - Search – екран для пошуку даних.
   - Statistics – екран, що відображає статистику.

   Файл index.ts збирає й експортує всі екрани для подальшого використання в маршрутизації чи функціональності.

5. Каталог shared призначений для спільних утиліт, які можуть бути використані в різних частинах додатка:
   - api – модулі для роботи з API (наприклад, запити до серверів чи бази даних).
   - hooks – кастомні React-хуки для повторно використовуваного функціонала (наприклад, обробка станів чи запитів).
   - lib – бібліотеки або сторонні залежності, налаштовані для використання в проекті.
   - services – сервіси для обробки бізнес-логіки або взаємодії з інфраструктурою (наприклад, авторизація, зберігання даних).
   - ui – бібліотека компонентів інтерфейсу користувача (кнопки, модальні вікна тощо).
   - index.ts – файл для централізованого експорту спільних модулів.
6. Каталог widgets містить комплексні компоненти, які поєднують у собі логіку й інтерфейс. Наприклад, swipeable кнопки, каруселі, багаторівневі меню. Цей підхід забезпечує модульність і можливість повторного використання.

Розглянемо деякі особливості архітектури:

1. Масштабованість – легко додавати нові модулі, екрани чи функції.
2. Покращена підтримка коду – чіткий поділ функцій і зон відповідальності спрощує внесення змін.
3. Повторне використання коду – компоненти, утиліти та сервіси створюються універсальними, що зменшує дублювання коду.
4. Інтеграція з TypeScript – забезпечує безпечну типізацію та раннє виявлення помилок під час розробки.

Структура проекту побудована за принципами модульності та масштабованості, що є ключовими у сучасній розробці мобільних додатків. Такий підхід забезпечує оптимальну організацію коду, сприяє злагодженій роботі команди й полегшує підтримку додатка в довгостроковій перспективі.

### 3.2 Основні використані бібліотеки

У розробці мобільного додатка використовувалися різноманітні бібліотеки для забезпечення функціональності, зручності та продуктивності:

1. @react-native-async-storage/async-storage - використовується для зберігання невеликих обсягів даних на пристрої, таких як токени, налаштування користувача, тощо. Простий у використанні й підтримує асинхронні операції.
2. @reduxjs/toolkit - бібліотека для управління станом на основі Redux. Спрощує налаштування, має вбудовану підтримку інструментів для розробки, таких як createSlice і createAsyncThunk.
3. redux-persist - дозволяє зберігати стан Redux у сховище (AsyncStorage), щоб дані зберігалися навіть після перезапуску додатка.
4. redux-saga - інструмент для обробки побічних ефектів (наприклад, API-запити) у Redux за допомогою генераторних функцій.
5. @react-navigation/native - основна бібліотека для навігації в React Native. Забезпечує реалізацію стекової, табуляційної та інших типів навігації.
6. @react-navigation/bottom-tabs - додає підтримку навігації з нижньою панеллю вкладок.
7. axios - HTTP-клієнт для виконання запитів до серверів API. Підтримує конфігурацію запитів, інтерсептори та обробку помилок.
8. i18next - потужна бібліотека для інтернаціоналізації, яка дозволяє додатку підтримувати кілька мов і зберігати локалізовані тексти.
9. react-native-splash-screen - дозволяє реалізувати splash-екран, який відображається під час запуску додатка.
10. react-native-svg - додає підтримку SVG-графіки для створення масштабованих іконок і графічних елементів.
11. react-native-tts - бібліотека для тексту в мовлення (Text-to-Speech), що дозволяє озвучувати текст.

12. @wdragon/react-native-voice - використовується для перетворення голосу в текст (Voice-to-Text).
13. react-native-reanimated - потужний фреймворк для створення анімацій у додатку.
14. react-native-responsive-screen - зручний спосіб створення адаптивного інтерфейсу на основі розмірів екрану.
15. react-hook-form -бібліотека для роботи з формами, яка мінімізує повторне рендерення та забезпечує зручну обробку даних форми.
16. zod - потужний інструмент для схем і валідації даних, що дозволяє визначати структуру об’єктів.
17. @sentry/react-native - платформа для відстеження помилок і логування, яка допомагає виявляти й виправляти баги в реальному часі.

Застосування цих бібліотек значно полегшило розробку додатка, забезпечуючи інтеграцію з API та багатоплатформову підтримку, зручне створення адаптивного та локалізованого інтерфейсу, легке управління станом і обробку форм та високу продуктивність і приємний досвід взаємодії з додатком.

### 3.3 Text-to-speech та speech-to-text функціонал

Розглянемо реалізацію функціоналу перетворення тексту в мовлення (Text-to-Speech, TTS) та мовлення в текст (Speech-to-Text, STT), що були інтегровані в мобільний додаток. Ці технології забезпечують підвищення доступності додатка, особливо для користувачів з обмеженими можливостями або тих, хто віддає перевагу голосовій взаємодії з інтерфейсом [18].

### 3.3.1 Text-to-Speech (TTS)

Функціонал “Text-to-Speech” реалізовано за допомогою бібліотеки “react-native-tts”, яка дозволяє озвучувати текст з зазначеними налаштуваннями, такими як швидкість відтворення, гучність та вибір голосу. Основна логіка цього модуля виглядає наступним чином:

1. Основні особливості реалізації:
   - Використовується метод “Tts.speak()”, який приймає текстовий вхідний параметр та налаштування, що включають вибір голосу на iOS (iosVoiceId), швидкість відтворення тексту (rate) і специфічні параметри для Android, такі як гучність (KEY_PARAM_VOLUME) та аудіопотік (KEY_PARAM_STREAM).
   - Реєстрація обробників подій через методи “Tts.addEventListener” для відстеження початку, прогресу, завершення або скасування озвучення.
1. Переваги функціоналу TTS:
   - Покращує сприйняття текстової інформації для користувачів.
   - Забезпечує інтерактивність через можливість відтворення тексту в реальному часі.
2. Фрагмент коду реалізації:

```
const onPress = useCallback(() => {
 if (text) {
  Tts.speak(text, {
   iosVoiceId: 'com.apple.ttsbundle.Moira-compact',
   rate: 0.5,
   androidParams: {
    KEY_PARAM_PAN: 0,
    KEY_PARAM_VOLUME: 1,
    KEY_PARAM_STREAM: 'STREAM_MUSIC',
   },
  })
 }
```

```
}, [text])
```

### 3.3.2 Speech-to-Text (STT)

Функціонал “Speech-to-Text” реалізовано з використанням бібліотеки @wdragon/react-native-voice, яка дозволяє розпізнавати мовлення та перетворювати його на текст.

3. Основні особливості реалізації:
   - Розпізнавання голосу запускається через метод “Voice.start()”, який приймає параметр мови (наприклад, en-US для англійської).
   - Обробка результатів розпізнавання здійснюється за допомогою події “onSpeechResults”. У цьому методі виділяється останнє розпізнане слово або фраза, які потім передаються у батьківський компонент через метод “onChange”.

Для зупинки прослуховування використовується метод “Voice.stop()”, а для очищення ресурсів — “Voice.destroy()”.

4. Обробка помилок:

Метод “onSpeechError” дозволяє відстежувати та логувати помилки, які виникають у процесі розпізнавання мовлення.

5. Переваги функціоналу STT:
   - Полегшує введення тексту, особливо на мобільних пристроях.
   - Робить додаток зручним для користувачів з фізичними обмеженнями.
6. Фрагмент коду реалізації:

```
const onSpeechResults = (event: { value?: string[] }) => {
  const lastValue = event?.value?.slice(-1)?.[0]
  if (lastValue) {
    onChange?.(lastValue)
  }
}
```

```
Voice.start('en-US')
  .catch(err => console.error('Voice start error:', err))
```

### 3.3.3 Інтеграція функціоналу

Функціонал “Text-to-Speech” та “Speech-to-Text” працюють як окремі компоненти в додатку, які користувач може активувати за допомогою натискання на відповідні іконки. Наприклад:

- Натискання кнопки "Sound" активує функцію TTS.
- Натискання кнопки "SpeechToText" запускає STT для початку запису голосу.

Ці компоненти є інтерактивними та динамічно змінюють стан, залежно від дій користувача. Це є зручним для використання та сприяє кращій взаємодії з користувача з додатком.

У підсумку, функціонал TTS і STT є важливими елементами сучасних мобільних додатків, що підвищують доступність і зручність взаємодії з користувачами. Впровадження цих технологій демонструє, як інноваційні рішення можуть покращити користувацький досвід.

## 3.4 Аналіз результатів.

Розглянемо головний екран додатку, що включає поле для пошуку, яке дозволяє швидко знайти необхідне слово серед збережених. Список слів відображається у вигляді окремих рядків, де для кожного слова показується його оригінальний запис англійською мовою, переклад українською та іконка переходу до перегляду детальної інформації. У нижній частині екрана розташована панель навігації з вкладками: "Home" для повернення до головного екрана, "Search" для пошуку нових слів, а також "Statistics" для перегляду статистики навчального прогресу (рис 3.2).

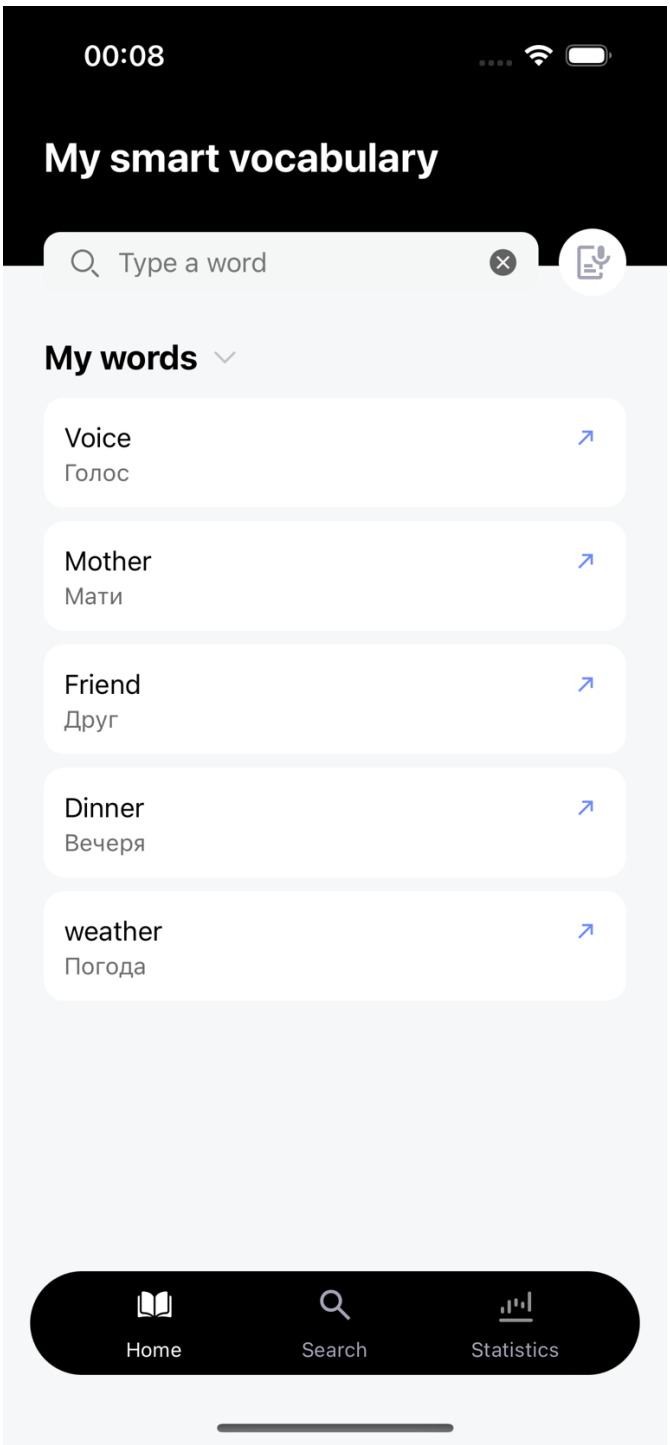


Рисунок 3.2 – Головний екран

Також додаток підтримує функцію організації слів за папками. Модальне вікно "Folders" дозволяє користувачеві переглядати наявні папки, кількість слів у кожній із них, обирати папки для фільтрації та створювати нові для групування слів за темами чи іншими критеріями. Зміни, внесені

через цю функцію, автоматично синхронізуються з головним списком, забезпечуючи динамічне оновлення даних (рис. 3.3).

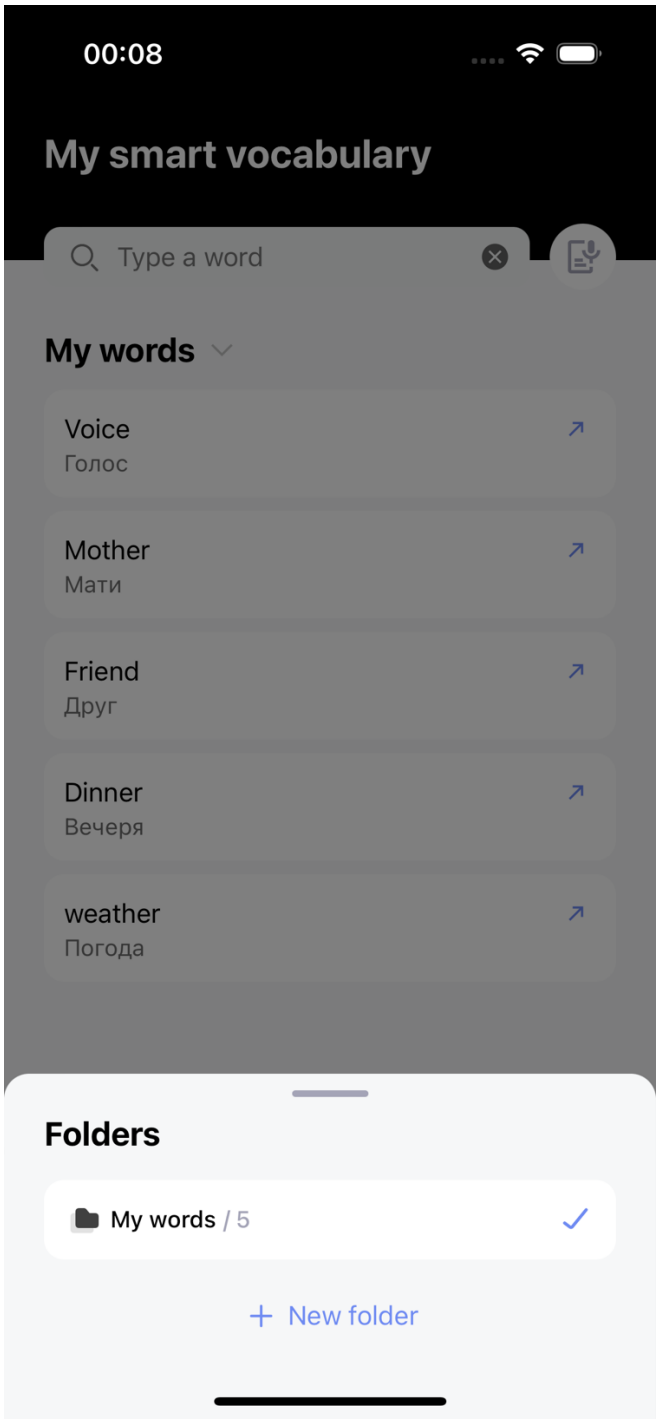


Рисунок 3.3 – Модальне вікно "Folders"

Екран пошуку дозволяє користувачам швидко знаходити слова, використовуючи зручне пошукове поле, розташоване у верхній частині інтерфейсу. Користувач може вводити будь-яке слово англійською мовою,

після чого система миттєво відображає результати. У випадках, коли необхідного слова немає у базі, додаток пропонує створити власний переклад за допомогою кнопки "Create own translation". Це дозволяє користувачам доповнювати словник власними словами, що робить систему гнучкішою та персоналізованою (рис 3.4).

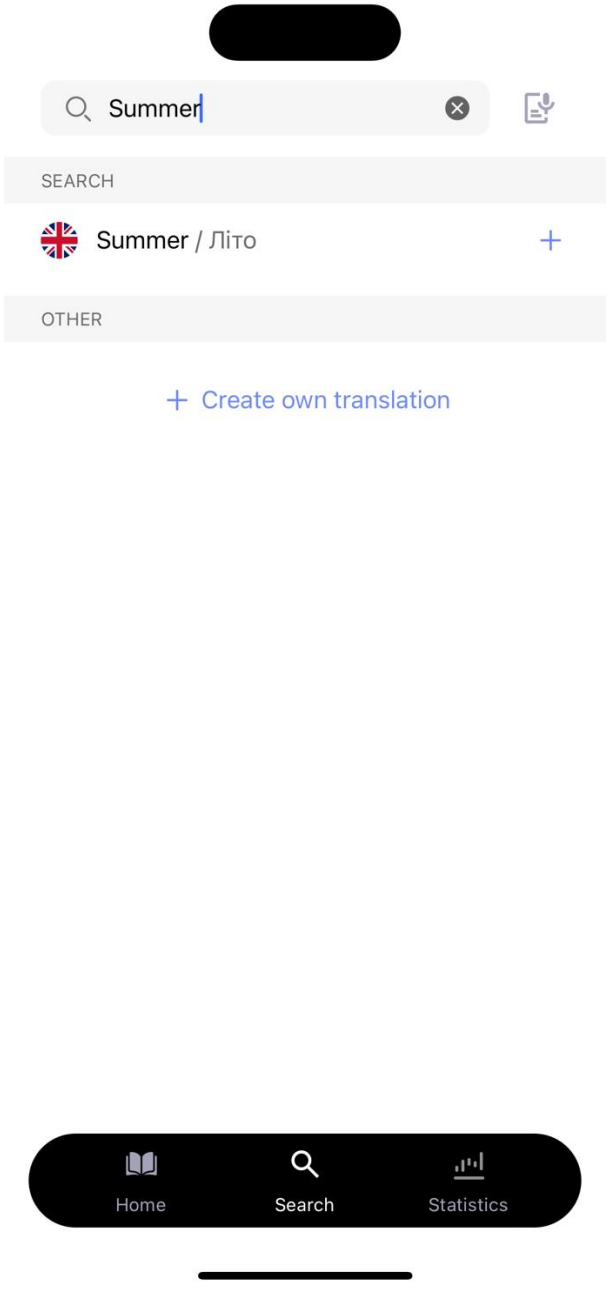


Рисунок 3.4 – Екран пошуку слів

Створення власного перекладу реалізоване через екран, який дозволяє вводити слово англійською мовою та додавати до нього один або кілька перекладів. Усі введені переклади можуть бути відредаговані або видалені завдяки інтерактивним кнопкам, що супроводжують кожне текстове поле. Для спрощення вводу тексту передбачена функція голосового введення, активується через іконку мікрофона, розташовану біля текстового поля. Голосовий ввід дозволяє легко додавати слова, зберігаючи час та зусилля користувача. Фінальний крок у створенні перекладу — це натискання кнопки "Save", яка зберігає додану інформацію до персонального словника, забезпечуючи синхронізацію даних між різними розділами додатку (рис 3.5).

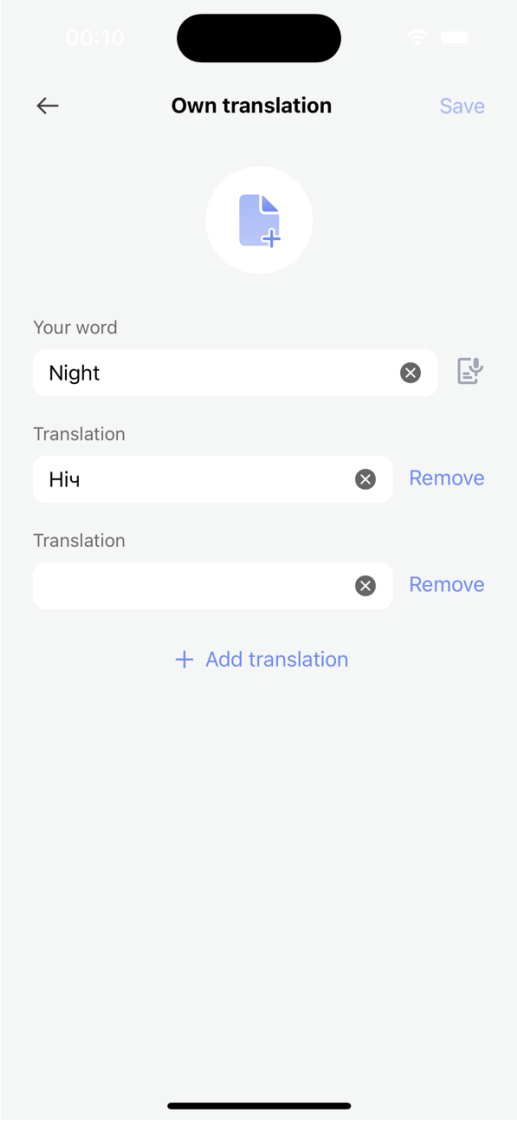


Рисунок 3.5 – Екран створення власного перекладу

Екран деталей слова надає користувачеві поглиблену інформацію. У верхній частині екрана відображається оригінальне слово, виділене великим шрифтом, що робить його легко помітним. Під ним розташований переклад, представлений у вигляді списку, якщо до слова додано декілька перекладів. Крім того, екран містить блок із синонімами, які представлені у вигляді інтерактивних елементів. Це дозволяє користувачам розширити свій словниковий запас, досліджуючи схожі за значенням слова. Інтерактивність цього екрана підсилюється за рахунок кнопки прослуховування вимови слова (функція Text-to-Speech), яка дозволяє відтворювати правильну вимову слова англійською мовою. Додатково передбачена кнопка "Add", що дозволяє додати слово до обраного списку або до конкретної папки, допомагаючи користувачеві організовувати свій словник у логічні категорії (рис. 3.6).

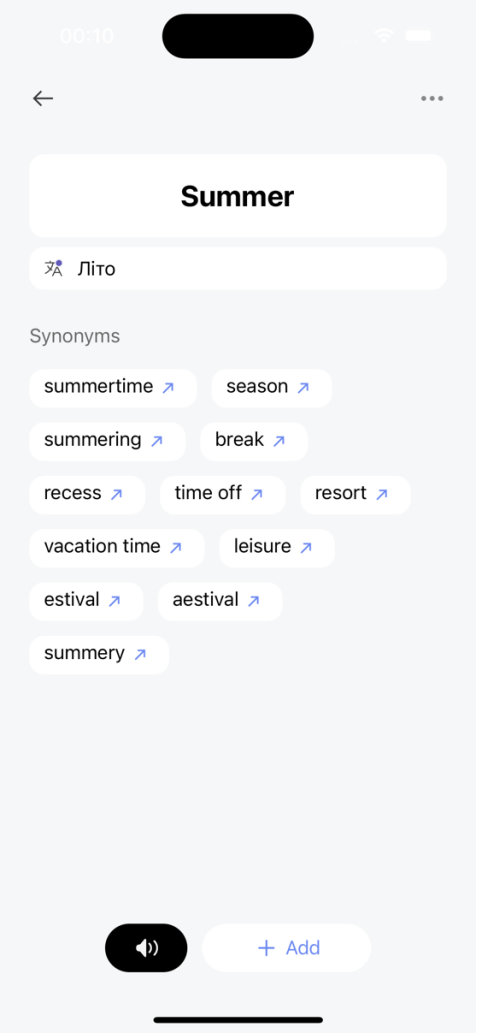


Рисунок 3.6 – Екран деталей слова

Екран деталей слова змінюється при збереженні цього слова у власний словник. У правому верхньому куті екрана присутнє меню додаткових дій, яке дозволяє перемістити слово до іншої папки, або видалити його зі словника. Цей екран є ключовим для детального аналізу та управління записами у словнику (рис. 3.7).

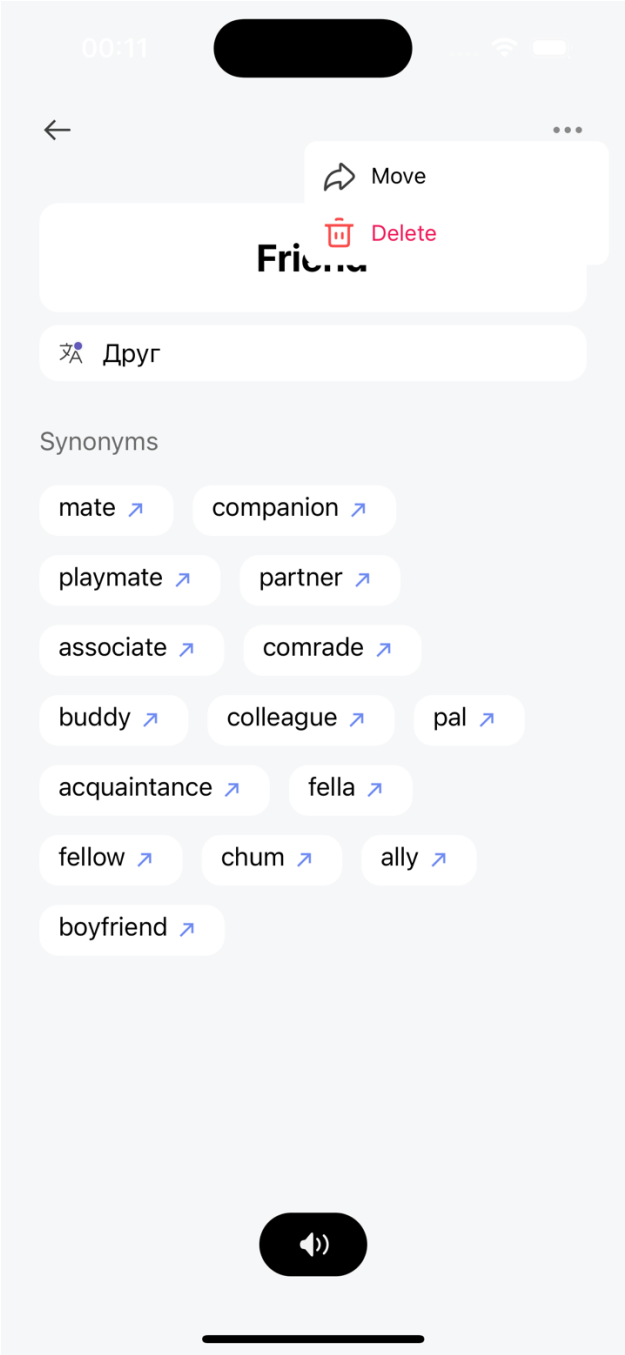


Рисунок 3.7 – Екран деталей збереженого слова

Екран статистики надає користувачеві можливість відстежувати прогрес у роботі з додатком. Він містить інформацію про загальну кількість доданих слів, дату останнього додавання нового слова та кількість створених папок. Ці дані дозволяють користувачеві оцінити, наскільки активно він використовує додаток для навчання. Простота цього екрана зосереджує увагу на ключових метриках, не відволікаючи від основної мети (рис. 3.8).

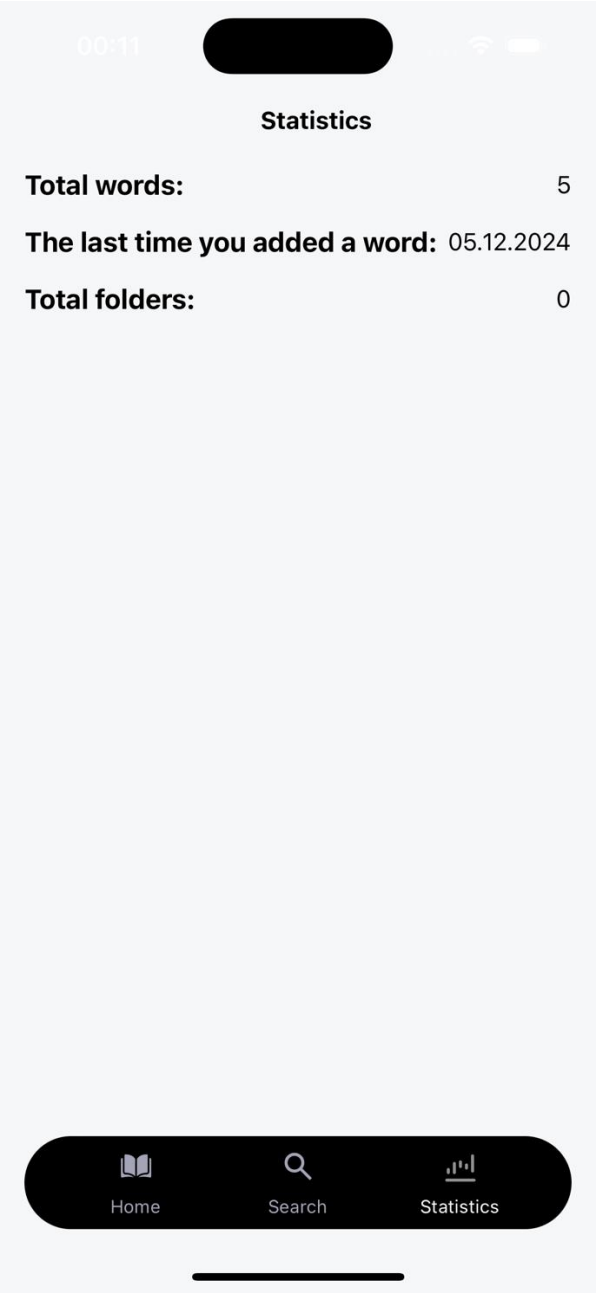


Рисунок 3.8 – Екран статистики

Дизайн усіх екранів мінімалістичний і зручний. Чітка типографія та інтуїтивно зрозумілі елементи управління забезпечують користувачеві легку навігацію між різними функціями. Інтерактивні кнопки та можливість голосової взаємодії спрощують процес роботи зі словником, роблячи його максимально ефективним та комфортним. Додаток орієнтований на персоналізацію досвіду користувача, дозволяючи йому не лише додавати нові слова, але й організовувати їх у папки, відстежувати статистику та зосереджуватися на розширенні свого словникового запасу за допомогою синонімів та інших інструментів.

### 3.5 Тестування додатку

У процесі розробки мобільного додатку було проведено комплексне тестування, що включало як ручні (мануальні) тести, так і автоматизоване тестування за допомогою бібліотеки “Detox”. Основна мета тестування полягала у забезпеченні стабільної роботи функціоналу, перевірці відповідності вимогам та зручності користувацького інтерфейсу [19].

Мануальне тестування охоплювало перевірку основного функціоналу додатку, включаючи [20]:

1. Пошук слів:
    - Перевірено коректність відображення результатів пошуку за різними запитами.
    - Тестувалися як слова, що вже є в базі, так і випадки, коли слово відсутнє (з пропозицією створення власного перекладу).
2. Створення власного перекладу:
    - Введення нового слова англійською мовою та одного або кількох перекладів українською.
    - Перевірка функцій видалення перекладу, додавання нового поля для перекладу та збереження.
    - Тестування функції голосового введення через іконку мікрофона.

3. Детальний перегляд слова:
    - Перевірка відображення оригінального слова, перекладів і синонімів.
    - Тестування функції відтворення вимови слова через “Text-to-Speech”.
    - Перевірка роботи інтерактивних елементів, таких як синоніми та меню для переміщення або видалення слова.
4. Робота зі статистикою:
    - Перевірено коректне відображення загальної кількості слів, дати останнього доданого слова та кількості папок.
5. Навігація між екранами:
    - Тестувалися плавність і коректність переходу між вкладками "Home", "Search" і "Statistics".

Результати мануального тестування дозволили виявити та усунути дрібні помилки у відображенні інтерфейсу та логіці роботи додатку.

Для підвищення надійності тестування та забезпечення стабільної роботи додатку в різних умовах було використано бібліотеку Detox — популярний інструмент для тестування React Native додатків. Detox дозволяє автоматично перевіряти функціонал додатку на реальних або віртуальних пристроях [21].

Автоматизоване тестування включало:

1. Тести на запуск додатку:
    - Перевірка, чи запускається додаток без помилок на різних пристроях і в різних середовищах.
2. Тести на взаємодію з елементами інтерфейсу:
    - Автоматична перевірка роботи пошукового поля, додавання слів і перегляду синонімів.
    - Тести на коректність переходів між екранами.
3. Тести функціоналу “Text-to-Speech” та “Speech-to-Text”:

- Перевірка активації функцій голосового введення та відтворення через інтерактивні кнопки.

4. Перевірка стабільності:
    - Стрес-тести, що моделювали багаторазове додавання та видалення слів для виявлення можливих збоїв або витоків пам’яті.

# ВИСНОВКИ

В ході виконання кваліфікаційної роботи було розглянуто основні аспекти створення сучасних мобільних додатків з використанням голосових технологій. У результаті було створено функціональний додаток, що сприяє ефективному вивченню іноземної мови з інтеграцією інноваційних методів взаємодії. Робота включала наступні етапи:

1. Проведено аналіз існуючих мобільних додатків для вивчення мов, що дало змогу визначити ключові функціональні можливості, такі як голосове введення, відтворення тексту (TTS), адаптивний дизайн, персоналізація словника та інтерактивність.
2. Виконано огляд технологій мобільної розробки з акцентом на кросплатформенні інструменти. Обрано React Native як основний фреймворк завдяки його перевагам: можливості розробки для iOS та Android з єдиною кодовою базою, високій продуктивності та широкій спільноті розробників.
3. Детально описано етапи створення додатку, починаючи від аналізу потреб користувачів до впровадження функціоналу. Було враховано специфіку дизайну для платформ Android та iOS із застосуванням принципів адаптивного дизайну.
4. Виконано дизайн додатку з урахуванням мінімалістичних принципів, що забезпечують інтуїтивність та естетичну привабливість інтерфейсу.
5. Розроблено клієнтську частину додатку за допомогою React Native і TypeScript.
6. Інтегровано функції голосового введення (Speech-to-Text) та відтворення тексту (Text-to-Speech), що дозволяють користувачам вводити слова голосом і прослуховувати їх вимову.
8. Проведено комплексне тестування додатку, включаючи мануальні перевірки та автоматизоване тестування за допомогою бібліотеки Detox. Це дозволило забезпечити стабільну роботу функціоналу,

перевірити коректність реалізації інтерфейсу та виявити можливі помилки.

У результаті виконання роботи створено повноцінний мобільний додаток для вивчення іноземних мов з наступними можливостями:

- здійснювати голосовий ввід слів та переглядати їх переклади;
- організовувати слова у тематичні папки для зручності використання;
- прослуховувати вимову слів за допомогою функції TTS;
- відстежувати власний прогрес через інтегровану статистику.

Додаток реалізовано із застосуванням сучасних технологій та має значний потенціал для подальшого вдосконалення, зокрема:

- додавання функціоналу для створення інтерактивних навчальних вправ;
- впровадження підтримки мультимовності для розширення користувацької бази;
- реалізація платного доступу до розширених функцій, таких як преміум-вправи, персоналізовані рекомендації, інтеграція з професійними мовними курсами, або відсутність реклами, що дозволить забезпечити фінансову підтримку подальшого розвитку додатку.

Враховуючи сучасні тенденції, розроблений додаток є актуальним і перспективним інструментом для інтерактивного вивчення мови, який може бути легко адаптований до потреб різних категорій користувачів.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. The Best Language Learning Apps [2024 Edition]. URL: https://www.fluentu.com/blog/learn/best-language-learning-apps/ (дата звернення: 01.09.2024).
2. Katruk M. Mobile App Development: Native vs Cross-Platform. URL: https://trm.solutions/archive/2018/09/10/mobile-app-development-native-vs-cross-platform (дата звернення: 01.09.2024).
3. Native vs Hybrid vs Cross Platform - Best option for Mobile App. URL: https://citrusbug.com/blog/native-vs-hybrid-vs-cross-platform-mobile-application (дата звернення: 05.09.2024).
4. Дизайн мобільних додатків: 7 кроків створення з прикладами. URL: https://wezom.com.ua/ua/blog/dizajn-mobilnyh-prilozhenij-pochemu-on-vazhen-i-gde-zakazat (дата звернення: 12.09.2024).
5. Mobile App Design Best Practices 2024. URL: https://www.designstudiouiux.com/blog/mobile-app-design-best-practices/ (дата звернення: 20.09.2024).
6. Посібник із дизайну мобільних додатків під iOS та Android. Основні відмінності платформ. URL: https://ux.pub/editorial/posibnik-iz-dizainu-mobilnikh-dodatkiv-pid-ios-ta-android-osnovni-vidminnosti-platform-1a5d (дата звернення: 01.10.2024).
7. MDN JavaScript Documentation. URL: https://developer.mozilla.org/ru/docs/Web/JavaScript (дата звернення: 07.10.2024).
8. JavaScript for Mobile Apps: Choose the Perfect Framework. URL: https://leobit.com/blog/javascript-for-mobile-apps-choose-the-perfect-framework/ (дата звернення: 12.10.2024).
9. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 23.10.2024).

10. Mastering TypeScript: Benefits and Best Practices. URL: https://www.telerik.com/blogs/mastering-typescript-benefits-best-practices (дата звернення: 28.10.2024).
11. Pang A. TypeScript vs. JavaScript. URL: https://medium.com/geekculture/typescript-vs-javascript-e5af7ab5a331 (дата звернення: 29.10.2024).
12. React Native Introduction. URL: https://reactnative.dev/docs/getting-started (дата звернення: 01.11.2024).
13. Cordenne B. 15 Examples of Successful Companies Using React Native in 2022. URL: https://www.trio.dev/blog/companies-use-react-native (дата звернення: 01.11.2024).
14. From Text to Speech: Revolutionizing How Your Business Connects with React Native TTS. URL: https://startup-house.com/blog/react-native-tts-business-communication (дата звернення: 08.11.2024).
15. Переваги синтезу мовлення в різних галузях. URL: https://uk.shaip.com/blog/benefits-of-text-to-speech/ (дата звернення: 08.11.2024).
16. React Native Text To Speech. URL: https://dev.to/ajmal_hasan/react-native-text-to-speech-3a7g (дата звернення: 08.11.2024).
17. FSD Documentation. URL: https://feature-sliced.design/ru/docs (дата звернення: 09.11.2024).
18. Comparison of Text to Speech Solutions for React Native. URL: https://www.netguru.com/blog/react-native-text-to-speech (дата звернення: 12.11.2024).
19. Ensuring the Quality of Your App with Testing in React Native. URL: https://dev.to/paulocappa/ensuring-the-quality-of-your-app-with-testing-in-react-native-o6g (дата звернення: 16.11.2024).
20. Mastering React Native App Testing: A Comprehensive Guide. URL: https://tanmayshende007.medium.com/mastering-react-native-app-testing-a-comprehensive-guide-9a3377216de0 (дата звернення: 20.11.2024).

21. Detox Getting Started. URL: https://wix.github.io/Detox/docs/introduction/getting-started (дата звернення: 21.11.2024).

# ДОДАТОК А. ЛІСТИНГ ОСНОВНИХ ЕКРАНІВ

Головний компонент додатку (вхідна точка):

```
import React, { useEffect } from 'react'

import { SENTRY_DNS } from '@env'
import SplashScreen from 'react-native-splash-screen'

import { Navigator } from '@/app/navigation'

import { Sentry } from '@/shared/lib'

Sentry.init({
  dsn: SENTRY_DNS,
  tracesSampleRate: 1.0,
})

const App = () => {
  useEffect(() => {
    setTimeout(() => {
      SplashScreen.hide()
    }, 1000)
  }, [])

  return <Navigator />
}

export default Sentry.wrap(App)
```

Головний екран:

```
import React, { useState } from 'react'

import { useTranslation } from 'react-i18next'

import { useTheme } from 'styled-components'

import { WordFeature } from '@/features/word'
```

```
import { Typography } from '@/shared'
import { Background } from '@/shared/ui/background'

import * as UI from './components'
import * as S from './styles'

export const Main = () => {
  const { COLORS } = useTheme()
  const { t } = useTranslation()
  const [searchTerm, setSearchTerm] = useState('')

  return (
    <Background.Container top={0} bottom={0} color={COLORS.background}>
      <S.Header>
        <Typography.H1 color="white">{t('app.name')}</Typography.H1>
        <S.SearchContainer>
          <WordFeature.SearchInput
            value={searchTerm}
            onChange={setSearchTerm}
          />
        </S.SearchContainer>
      </S.Header>

      <UI.MyWordsList searchTerm={searchTerm} />
    </Background.Container>
  )
}
```

Компонент списку збережених слів:

```
import React, { useMemo, useState } from 'react'

import { FlatList } from 'react-native'

import { useTranslation } from 'react-i18next'

import { useTypedSelector } from '@/app/store'

import { WordFeature } from '@/features/word'

import { getWordSelector, TFolder, WordEntity } from '@/entities/word'
```

```
import {
  appPadding,
  Icon,
  Styled,
  Typography,
  useBottomSheetRef,
} from '@/shared'

import { TMyWordsListProps } from './types'

export const MyWordsList = ({ searchTerm }: TMyWordsListProps) => {
  const { t } = useTranslation()
  const { words } = useTypedSelector(getWordSelector)
  const foldersBSRef = useBottomSheetRef()

  const [selectedFolder, setSelectedFolder] = useState<TFolder | null>(null)

  const filteredWords = useMemo(
    () =>
      words.filter(
        word =>
          (!selectedFolder || word.forderId === selectedFolder._id) &&
          (!searchTerm ||
            word.text.toLowerCase().includes(searchTerm.toLowerCase())),
      ),
    [words, searchTerm, selectedFolder?._id],
  )

  return (
    <>
      <FlatList
        key={selectedFolder?._id}
        style={{ paddingHorizontal: appPadding }}
        ListHeaderComponent={() => (
          <Styled.Touchable
            mTop="22px"
            mBottom="12px"
            justify="flex-start"
            onPress={() => foldersBSRef.current?.open()}
            width="auto">
            <Typography.H2 mRight="4px">
              {selectedFolder?.name || t('folder.my_words')}
            </Typography.H2>
```

```
            <Icon name="ExpandDown" />
          </Styled.Touchable>
        )}
        data={filteredWords}
        keyExtractor={item => item._id}
        renderItem={({ item }) => <WordEntity.MyCard word={item} />}
        ItemSeparatorComponent={() => <Styled.Divider height={8} />}
        ListFooterComponent={() => <Styled.Divider height={100} />}
      />
      <WordFeature.FoldresBS  ref={foldersBSRef}  onChange={setSelectedFolder}
/>
    </>
  )
}
```

Екран пошуку слів:

```
import React, { useCallback, useState } from 'react'

import { useTranslation } from 'react-i18next'
import { useTheme } from 'styled-components'

import { EScreens } from '@/app/navigation'

import { WordFeature } from '@/features/word'

import { TranslateService } from '@/entities/word'

import { formatTranslationToWord } from '@/entities/word/utils/format'

import {
  Background,
  Button,
  Icon,
  Styled,
  Typography,
  useNavigation,
  useQuery,
} from '@/shared'

import * as UI from './components'
```

```
import * as S from './styles'

export const Main = () => {
  const { COLORS } = useTheme()
  const { t } = useTranslation()
  const navigation = useNavigation()
  const [searchText, setSearchText] = useState('')

  const { data: translateData } = useQuery(TranslateService.postTranslate, {
    toLang: 'ukrainian',
    fromLang: 'english',
    text: searchText,
    disableInitialCall: !searchText,
  })
  console.log('translateData: ', translateData)
  const word =
    translateData?.translation && !!searchText
      ? formatTranslationToWord(translateData.translation)
      : null

  const onPressCreateOwn = useCallback(() => {
    navigation.navigate(EScreens.SearchOwnTranslation)
  }, [])

  const onPressWord = useCallback(() => {
    navigation.navigate(EScreens.WordMain, { word })
  }, [word])

  return (
    <Background.Container>
      <Styled.Padding mBottom="12px">
        <WordFeature.SearchInput onChange={setSearchText} />
      </Styled.Padding>

      <S.TitleSection>
        <Typography.Caption1R
          textTransform="uppercase"
          color={COLORS.neutral_500}>
          {t('search.title')}
        </Typography.Caption1R>
      </S.TitleSection>

      {!!word && <UI.WordItem data={word} onPress={onPressWord} />}
```

```
      {!word && (
        <Styled.FlexWrapper mTop="16px">
          <Icon name="SearchOff" size={62} />
        </Styled.FlexWrapper>
      )}

      <S.TitleSection mTop="12px">
        <Typography.Caption1R
          textTransform="uppercase"
          color={COLORS.neutral_500}>
          {t('common.other')}
        </Typography.Caption1R>
      </S.TitleSection>

      <Button.Standard
        text={t('own_translation.create')}
        type="tertiary"
        icon="Plus"
        mTop="12px"
        onPress={onPressCreateOwn}
      />
    </Background.Container>
  )
}
```

Екран деталей слова:

```
import React from 'react'

import { useRoute } from '@react-navigation/native'

import _ from 'lodash'
import { useTranslation } from 'react-i18next'
import { useTheme } from 'styled-components'

import { EScreens } from '@/app/navigation'
import { TScreenQueryProps } from '@/app/navigation/types'

import { useTypedSelector } from '@/app/store'

import { Header } from '@/widgets/header'
```

```
import { WordFeature } from '@/features/word'

import {
  getWordSelector,
  TranslateService,
  useWordControl,
} from '@/entities/word'

import { Background, Icon, Styled, Typography, useQuery } from '@/shared'

import { Common } from '@/shared/ui/common'

import * as S from './styles'

export const Word = () => {
  const { COLORS } = useTheme()
  const { t } = useTranslation()
  const { folders } = useTypedSelector(getWordSelector)
  const { existInAnyFolder } = useWordControl()

  const {
    params: { word },
  } = useRoute<TScreenQueryProps<EScreens.WordMain>>()

  const { data: synonimsData } = useQuery(TranslateService.postSynonyms, {
    lang: 'english',
    word: word.text,
  })

  const synonims = synonimsData?.synonyms?.synonyms || []

  const folderName = folders?.find(item => item._id === word.forderId)?.name
  return (
    <Background.Container color={COLORS.background}>
      <Header.Standard
        goBack
        color={COLORS.background}
        rightAction={
          <WordFeature.HeaderActions
            wordId={word._id}
            folderId={word?.forderId || null}
          />
        }
```

```
/>

<Background.Scroll>
  <S.WordContainer mTop="12px">
    <Typography.H1>{word.text}</Typography.H1>
    {!!folderName && (
      <Typography.Body2R color="neutral_500" mTop="8px">
        {folderName}
      </Typography.Body2R>
    )}
  </S.WordContainer>

  <S.TranslationContainer mTop="8px" mBottom="24px">
    <Icon name="Translate" size={16} />
    <Typography.Body1R mLeft="12px" style={{ flex: 1 }}>
      {word.transaltions?.map(item => _.capitalize(item))?.join(' • ')}
    </Typography.Body1R>
  </S.TranslationContainer>

  {word?.examples?.map(item => (
    <S.ExampleContainer key={item.id.toString()} mBottom="12px">
      <Common.ColoredText
        highlightWords={item.source_phrases.map(val => val.phrase)}>
        {item.source}
      </Common.ColoredText>

      <Common.ColoredText
        color="neutral_500"
        mTop="4px"
        highlightWords={item.target_phrases.map(val => val.phrase)}>
        {item.target}
      </Common.ColoredText>
    </S.ExampleContainer>
  ))}

  {!!synonims?.length && (
    <>
      <Typography.Body1R color="neutral_500" mTop="4px" mBottom="12px">
        {t('word.synonyms')}
      </Typography.Body1R>

      <Styled.FlexWrapper wrap="wrap" justify="flex-start">
        {synonims.map(item => (
```

```
                <S.SynonymContainer
                  mRight="12px"
                  mBottom="12px"
                  key={item.id.toString()}>
                  <Typography.Body1R>{item.synonym}</Typography.Body1R>
                  <Icon name="ArrowUpRight" />
                </S.SynonymContainer>
              ))}
            </Styled.FlexWrapper>
          </>
        )}

        <Styled.Divider height={100} />
      </Background.Scroll>

      <S.Footer>
        <WordFeature.TextToSpeech text={word.text} />

        {!existInAnyFolder(word.text) && (
          <WordFeature.SaveButton word={word} mLeft="12px" />
        )}
      </S.Footer>
    </Background.Container>
  )
}
```

Екран статистики:

```
import React from 'react'

import { format } from 'date-fns'
import { useTranslation } from 'react-i18next'

import { useTheme } from 'styled-components'

import { useTypedSelector } from '@/app/store'

import { Header } from '@/widgets/header'

import { getWordSelector, TFolder, TWord } from '@/entities/word'
```

```
import { Background, Styled, Typography } from '@/shared'

export const Main = () => {
  const { t } = useTranslation()
  const { COLORS } = useTheme()
  const { words, folders } = useTypedSelector(getWordSelector)

  const latestWord = words.reduce((latest, current) => {
    return new Date(current.createdAt || '') > new Date(latest.createdAt ||
'')
      ? current
      : latest
  }, {} as TWord)

  const getFoldersWords = (id: string) => {
    return words.filter(item => item.forderId === id)
  }

  const renderFolders = (item: TFolder) => {
    return (
      <Styled.FlexWrapper mTop={'16px'} justify={'space-between'}>
        <Typography.H3Bold>{item.name}:</Typography.H3Bold>
        <Typography.H3Bold>
          {getFoldersWords(item._id).length}
        </Typography.H3Bold>
      </Styled.FlexWrapper>
    )
  }

  return (
    <Background.Container color={COLORS.background}>
      <Header.Standard
        title={t('statistics.title')}
        color={COLORS.background}
      />

      <Background.Standard pHorizontal={16}>
        <Styled.FlexWrapper justify={'space-between'}>
          <Typography.H3Bold>{t('statistics.total')}:</Typography.H3Bold>
          <Typography.Body1R>{words.length}</Typography.Body1R>
        </Styled.FlexWrapper>

        <Styled.FlexWrapper mTop={'16px'} justify={'space-between'}>
```

```
<Typography.H3Bold>{t('statistics.last_adding')}:</Typography.H3Bold>
          <Typography.Body1R>
            {format(latestWord.createdAt || new Date(), 'dd.MM.yyyy')}
          </Typography.Body1R>
        </Styled.FlexWrapper>

        <Styled.FlexWrapper mTop={'16px'} justify={'space-between'}>

<Typography.H3Bold>{t('statistics.total_folder')}:</Typography.H3Bold>
          <Typography.Body1R>{folders.length}</Typography.Body1R>
        </Styled.FlexWrapper>

        {!!folders.length && (
          <Styled.FlexWrapper mTop={'16px'} justify={'space-between'}>
            <Typography.H3Bold>{t('folder.folders')}:</Typography.H3Bold>
          </Styled.FlexWrapper>
        )}

        {folders.map(renderFolders)}
      </Background.Standard>
    </Background.Container>
  )
}
```

Модальне вікно зі списком папок:

```
import React, { forwardRef, useCallback, useEffect, useState } from 'react'

import { Alert } from 'react-native'

import { BottomSheetFlatList } from '@gorhom/bottom-sheet'
import { useTranslation } from 'react-i18next'

import { useSafeAreaInsets } from 'react-native-safe-area-context'

import uuid from 'react-native-uuid'
import { useDispatch } from 'react-redux'

import { useTypedSelector } from '@/app/store'

import { getWordSelector, TFolder, wordActions } from '@/entities/word'
```

```
import {
  appPadding,
  Button,
  hp,
  Icon,
  Styled,
  Typography,
  useBottomSheetRef,
} from '@/shared'
import { BottomSheet } from '@/shared/ui/bottomSheet'
import { TBottomSheetModalRef } from '@/shared/ui/bottomSheet/Modal'

import * as S from './styles'
import { TFoldersBSProps } from './types'

export const FoldresBS = forwardRef<TBottomSheetModalRef, TFoldersBSProps>(
  ({ onChange, value = null }, ref) => {
    const bsRef = useBottomSheetRef(ref)
    const { t } = useTranslation()
    const { bottom } = useSafeAreaInsets()
    const dispatch = useDispatch()
    const { folders, words } = useTypedSelector(getWordSelector)
    const [selectedFolder, setSelectedFolder] = useState<string | null>(value)

    useEffect(() => {
      setSelectedFolder(value)
    }, [value])

    const onPressAddFolder = useCallback(() => {
      Alert.prompt(t('folder.enter_folder_name'), '', text => {
        !!text &&
          dispatch(
            wordActions.createFolder({
              _id: uuid.v4(),
              name: text,
            }),
          )
      })
    }, [])

    const onPressSelect = (folder: null | TFolder) => {
```

```
      setSelectedFolder(folder?._id || null)
      onChange?.(folder)

      bsRef.current?.close()
    }

    const onPressDeleteFolder = useCallback((folderId: string) => {
      dispatch(wordActions.removeFolder(folderId))
    }, [])

    return (
      <BottomSheet.Modal
        ref={bsRef}
        enableDynamicSizing
        maxDynamicContentSize={hp(80)}>
        <BottomSheetFlatList
          style={{ paddingHorizontal: appPadding }}
          data={folders}
          keyExtractor={item => item._id}
          ListHeaderComponent={() => (
            <>
              <Typography.H2 mBottom="16px">
                {t('folder.folders')}
              </Typography.H2>

              <S.FolderWrapper
                mBottom="2px"
                onPress={() => onPressSelect(null)}>
                <Styled.FlexWrapper width="auto">
                  <Icon name="FolderDublicate" />
                  <Typography.H4 mLeft="4px">
                    {t('folder.my_words')}
                  </Typography.H4>
                  <Typography.H4           color="neutral_300">{`           /
${words.length}`}</Typography.H4>
                </Styled.FlexWrapper>

                {!selectedFolder && <Icon name="Done" />}
              </S.FolderWrapper>
            </>
          )}
          renderItem={({ item }) => (
```

```
            <S.FolderWrapper        mBottom="2px"        onPress={()        =>
onPressSelect(item)}>
              <Styled.FlexWrapper width="auto">
                <Icon name="FolderDublicate" />
                <Typography.H4 mLeft="4px">{item.name}</Typography.H4>
                <Typography.H4 color="neutral_300">{` / ${words.reduce(
                  (st, val) => st + +(val.forderId === item._id),
                  0,
                )}`}</Typography.H4>
              </Styled.FlexWrapper>

              {selectedFolder === item._id && <Icon name="Done" />}

              {!(selectedFolder === item._id) && (
                <Styled.Touchable
                  width="auto"
                  onPress={() => onPressDeleteFolder(item._id)}>
                  <Icon name="Trash" />
                </Styled.Touchable>
              )}
            </S.FolderWrapper>
          )}
          ListFooterComponent={() => (
            <Styled.FlexWrapper mBottom={`${bottom + 16}px`} mTop="20px">
              <Button.Text
                leftIcon="Plus"
                text={t('folder.new_folder')}
                onPress={onPressAddFolder}
              />
            </Styled.FlexWrapper>
          )}
        />
      </BottomSheet.Modal>
    )
  },
)
```

# ДОДАТОК Б. ЛІСТИНГ TTS ТА STT ЛОГІКИ

Перетворення голосу в текст:

```
import React, { useState, useCallback, useEffect } from 'react'

import { useIsFocused } from '@react-navigation/native'
import Voice from '@wdragon/react-native-voice'
import { useTheme } from 'styled-components'

import { Icon, Styled } from '@/shared'

import { TSpeechToTextProps } from './types'

export const SpeechToText = ({
  onChange,
  disabled,
  ...props
}: TSpeechToTextProps) => {
  const { COLORS } = useTheme()
  const isFocused = useIsFocused()

  const [listen, setListen] = useState(false)

  const onStart = useCallback(() => {
    setListen(true)

    const onSpeechResults = (event: { value?: string[] }) => {
      const lastValue = event?.value?.slice(-1)?.[0]
      if (lastValue) {
        console.log('onSpeechResults: ', lastValue)
        onChange?.(lastValue) // Pass the recognized speech to the parent
      }
    }

    // Register voice handlers
    Voice.onSpeechResults = onSpeechResults
    Voice.onSpeechError = error => {
      console.error('Speech error:', error)
      setListen(false)
    }
```

```
    Voice.start('en-US') // Set language as needed
      .catch(err => console.error('Voice start error:', err))
  }, [])

  const onEnd = useCallback(() => {
    setListen(false)

    Voice.stop().catch(err => console.error('Voice stop error:', err))

    Voice.destroy().catch(err => console.error('Voice destroy error:', err))
  }, [])

  useEffect(() => {
    if (isFocused) {
      if (listen) {
        onStart()
      }
    } else {
      onEnd()
    }
  }, [isFocused])

  return (
    <Styled.Touchable
      {...props}
      width="auto"
      disabled={disabled}
      onPress={listen ? onEnd : onStart}>
      <Icon
        name="SpeechToText"
        fill={
          disabled
            ? COLORS.primary_300
            : listen
            ? COLORS.primary_500
            : COLORS.neutral_300
        }
      />
    </Styled.Touchable>
  )
}
```

Перетворення тексту в голос:

```
import React, { useCallback, useEffect } from 'react'

import Tts from 'react-native-tts' // Import the TTS library

import { Icon } from '@/shared'

import * as S from './styles'
import { TTextToSpeechProps } from './types'

export const TextToSpeech = ({ text, ...props }: TTextToSpeechProps) => {
  const onPress = useCallback(() => {
    if (text) {
      Tts.speak(text, {
        iosVoiceId: 'com.apple.ttsbundle.Moira-compact',
        rate: 0.5,
        androidParams: {
          KEY_PARAM_PAN: 0,
          KEY_PARAM_VOLUME: 1,
          KEY_PARAM_STREAM: 'STREAM_MUSIC',
        },
      })
    }
  }, [text])

  useEffect(() => {
    Tts.addEventListener('tts-start', event => console.log('start', event))
    Tts.addEventListener('tts-progress', event =>
      console.log('progress', event),
    )
    Tts.addEventListener('tts-finish', event => console.log('finish', event))
    Tts.addEventListener('tts-cancel', event => console.log('cancel', event))
  }, [])

  return (
    <S.Conatiner {...props} onPress={onPress}>
      <Icon name="Sound" />
    </S.Conatiner>
  )
}
```

# ДОДАТОК В. ЛІСТИНГ СЕРВІСУ ПОШУКУ СЛІВ

Сервіс для отримання перекладів та інформації про слова:

```
import { apiPrivate, TResponse } from '@/shared/api'

import * as Types from './types'

export class TranslateService {
  static async postTranslate(
    payload: Types.TPostTranslateApi['payload'],
  ): TResponse<Types.TPostTranslateApi['response']> {
    return apiPrivate.post(`/translate`, payload)
  }

  static async postContext(
    payload: Types.TPostContextApi['payload'],
  ): TResponse<Types.TPostContextApi['response']> {
    return apiPrivate.post(`/context`, payload)
  }

  static async postSpellcheck(
    payload: Types.TPostSpellcheckApi['payload'],
  ): TResponse<Types.TPostSpellcheckApi['response']> {
    return apiPrivate.post(`/spellcheck`, payload)
  }

  static async postSynonyms(
    payload: Types.TPostSynonymsApi['payload'],
  ): TResponse<Types.TPostSynonymsApi['response']> {
    return apiPrivate.post(`/synonyms`, payload)
  }
}
```

Моделі даних:

```
import { TTranslationExample } from './translate'

export type TWord = {
  _id: string
```

```
  text: string
  transaltions: Array<string>
  forderId?: string | null
  createdAt?: string

  examples?: Array<TTranslationExample>
  voice?: string
}

export type TTranslationExample = {
  id: number
  source: string
  target: string
  source_phrases: Array<{
    phrase: string
    offset: number
    length: number
  }>
  target_phrases: Array<{
    phrase: string
    offset: number
    length: number
  }>
}

export type TTranslationContext = {
  examples: TTranslationExample[]
  rude: boolean
}

export type TTranslation = {
  ok: boolean
  text: string
  source: string
  target: string
  translations: string[]
  detected_language: string
  voice: string
  context: TTranslationContext
}

export type TSynonym = {
  id: number
```

```
  synonym: string
}

export type TSynonyms = {
  ok: boolean
  text: string
  source: string
  synonyms: TSynonym[]
}

export type TFolder = {
  _id: string
  name: string
}
```