НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Мобільний застосунок для забезпечення комунікації людей з глухотою та сліпотою

Виконав студент IV курсу, групи ІП-02
(шифр групи)

Мойсол Андрій Олексійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник асистент, Ясенова А. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ОТ, к.т.н., доц., Волокита А. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії
Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 121 Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

ЗАВДАННЯ
на дипломний проєкт студенту
Мойсолу Андрію Олексійовичу

(прізвище, ім'я, по батькові)

1. Тема проєкту Мобільний застосунок для забезпечення комунікації людей з глухотою та сліпотою

керівник проєкту Ясенова Анна Вадимівна, асистент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27» травня 2024 р. №2112-с
2. Термін подання студентом проєкту « 17 » червня 2024 року
3. Вихідні дані до проєкту: технічне завдання
4. Зміст пояснювальної записки
1) Дослідження предметної області: проведення аналізу предметної області, вивчення існуючих рішень, опис бізнес-процесів, формулювання задачі.
2) Визначення вимог до програмного забезпечення: розробка сценаріїв використання, створення функціональних вимог, формулювання нефункціональних вимог.
3) Проєктування програмного забезпечення: визначення архітектури, обґрунтування вибору інструментів розробки, створення програмного забезпечення, оцінка безпеки даних.
4) Оцінка якості та тестування: аналіз якості програмного забезпечення, опис тестових процесів, розробка контрольного прикладу.
5) Впровадження та підтримка програмного забезпечення: встановлення програмного забезпечення, його підтримка та супровід.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна контекстна

3) Схема структурна контейнерна

4) Схема структурна компонентна

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «11» березня 2024 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	16.03.2024	
3	Постановка та формалізація задачі	27.03.2024	
4	Розробка інформаційного забезпечення	05.04.2024	
5	Алгоритмізація задачі	09.04.2024	
6	Обґрунтування вибору використаних технічних засобів	13.04.2024	
7	Розробка програмного забезпечення	01.05.2024	
8	Налагодження програми	07.05.2024	
9	Виконання графічних документів	16.05.2024	
10	Оформлення пояснювальної записки	29.05.2024	
11	Подання ДП на попередній захист	03.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____ Андрій МОЙСОЛ

(підпис) (ініціали, прізвище)

Керівник _____ Анна ЯСЕНОВА

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 24 таблиць, 17 рисунків та 10 джерел – загалом 50 сторінки.

Дипломний проєкт присвячений розробці мобільного додатка для забезпечення комунікації людей з глухотою та сліпотою за допомогою азбуки Морзе та сучасних технологій.

Мета дипломного проєкту полягає в полегшенні життя людей з глухотою та сліпотою, створивши ефективний інструмент для незалежного спілкування та взаємодії з оточуючим світом за допомогою коду Морзе, розпізнавання зображень та голосових команд.

Об'єкт дослідження: Мобільні додатки для комунікації людей з особливими потребами.

Предмет дослідження: Методи та технології розробки мобільних додатків для забезпечення комунікації глухосліпих користувачів, включаючи азбуку Морзе, розпізнавання зображень та голосові команди.

У розділі “Передпроєктне обстеження предметної області” розглянуто сучасні технології комунікації для глухосліпих людей, проаналізовано їхні переваги та недоліки

Розділ “Розроблення вимог до програмного забезпечення” присвячений визначенню функціональних та нефункціональних вимог до мобільного додатка, на основі аналізу потреб користувачів та сучасних технологій комунікації для глухосліпих людей.

Розділ “Конструювання та розроблення програмного забезпечення” присвячений розробці архітектури, реалізації основних функцій для забезпечення ефективної комунікації глухосліпих користувачів.

Розділ “Аналіз якості та тестування програмного забезпечення” присвячений оцінці продуктивності, надійності та зручності використання додатка, а також виявленню та усуненню можливих проблем шляхом проведення ретельного тестування.

Розділ “Розгортання та супровід програмного забезпечення” присвячений покроковому опису процесу розгортання додатка та забезпеченню його

стабільної роботи і подальшого вдосконалення через моніторинг, оновлення та технічну підтримку.

Програмне забезпечення впроваджено на платформі iOS, доступне для завантаження в AppStore, та забезпечується стабільна робота через регулярні оновлення і технічну підтримку.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ДОДАТОК, ГЛУХОТА, СЛІПОТА, АЗБУКА МОРЗЕ, КОМУНІКАЦІЯ, IOS, SWIFT, SWIFTUI, OPENAI, GOOGLE CLOUD

ABSTRACT

The explanatory note of the diploma project consists of five chapters, 24 tables, 17 figures, and 10 sources - a total of 50 pages.

The diploma project is dedicated to the development of a mobile application to provide communication for people with deafness and blindness using Morse code and modern technologies.

The goal of the diploma project is to make life easier for people with deafness and blindness by creating an effective tool for independent communication and interaction with the world around them using Morse code, image recognition, and voice commands.

Object of research: Mobile applications for communication of people with special needs.

Subject of research: Methods and technologies for developing mobile applications to provide communication for deafblind users, including Morse code, image recognition, and voice commands.

In the section “Pre-project survey of the subject area”, modern communication technologies for deafblind people are considered, their advantages and disadvantages are analyzed

The section “Development of software requirements” is devoted to the definition of functional and non-functional requirements for a mobile application, based on the analysis of user needs and modern communication technologies for deafblind people.

The section “Software Design and Development” is devoted to the development of architecture, implementation of basic functions to ensure effective communication for deafblind users.

The section “Quality Analysis and Software Testing” is dedicated to assessing the performance, reliability and usability of the application, as well as identifying and eliminating possible problems through thorough testing.

The section “Software Deployment and Maintenance” is devoted to a step-by-step description of the application deployment process and ensuring its stable operation and further improvement through monitoring, updates, and technical support.

The software is implemented on the iOS platform, is available for download in the AppStore, and is ensured to work stably through regular updates and technical support.

KEYWORDS: MOBILE APPLICATION, DEAFNESS, BLINDNESS, MORSE CODE, COMMUNICATION, IOS, SWIFT, SWIFTUI, OPENAI, GOOGLE CLOUD

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-9614.045430.01.91	Технічне завдання	12	
3	А4	КПІ.ІП-9614.045430.02.81	Пояснювальна записка	57	
4	А4	КПІ.ІП-9614.045430.03.12	Текст програми	68	
5	А4	КПІ.ІП-9614.045430.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-9614.045430.05.34	Керівництво користувача	6	
7	А3	КПІ.ІП-9614.045430.06.99	Графічний матеріал	5	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-9614.045430.00.90		
Розроб.		Мойсол А.О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Ясенова А.В.					
Н.контр.		Шулькевич Т.В.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-02	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Мобільний застосунок для забезпечення комунікації людей з глухотою та сліпотою

Технічне завдання

КПІ.ІП-9614.045430.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Анна ЯСЕНОВА

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Андрій МОЙСОЛ

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: MorseBridge

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного забезпечення MorseBridge, котре використовується для забезпечення ефективної комунікації людей з глухотою та сліпотою за допомогою азбуки Морзе, розпізнавання зображень та голосових команд. Це програмне забезпечення призначене для використання у різних галузях, включаючи охорону здоров'я, освіту та соціальну підтримку. Можливими користувачами є медичні працівники, вчителі, соціальні працівники, а також самі люди з глухотою та сліпотою, які потребують надійного інструменту для комунікації та взаємодії з оточуючим світом. MorseBridge надає користувачам можливість незалежного спілкування та отримання інформації, що значно покращує якість їхнього життя.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки MorseBridge є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для автоматизації та полегшення комунікації людей з глухотою та сліпотою за допомогою мобільного додатка.

Метою розробки є забезпечення функціональної та експлуатаційної можливості незалежного спілкування для людей з глухотою та сліпотою, полегшення взаємодії з оточуючим світом за допомогою азбуки Морзе, розпізнавання зображень та голосових команд.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– Підтримка інтуїтивно зрозумілих жестів для введення, редагування та видалення символів.

– Можливість розпізнавання голосових команд та перетворення їх у код Морзе.

– Функція розпізнавання зображень та перетворення опису в код Морзе.

– Відображення та озвучування введеного тексту.

Рисунок 4.1 – Макет інтерфейсу з розпізнаванням картинок

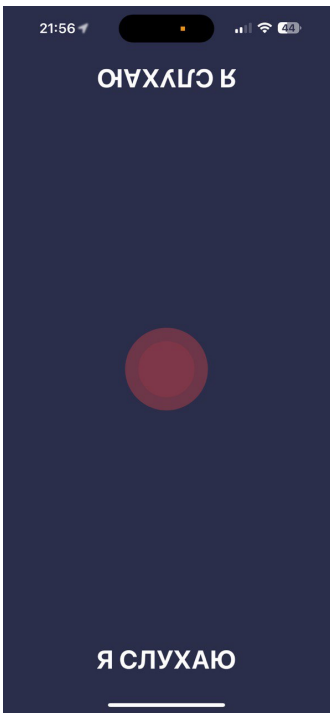


Рисунок 4.2 – Макет інтерфейсу з розпізнаванням голосу

4.1.2 Для користувача:

– Можливість введення повідомлень у форматі азбуки Морзе за допомогою жестів.

– Видалення тексту за допомогою жестів.

– Надсилання введених повідомлень.

– Прийом та відтворення отриманих повідомлень у форматі азбуки Морзе.

– Розпізнавання голосових команд та перетворення їх у текстові повідомлення.

– Використання камери для розпізнавання зображень та отримання опису в форматі азбуки Морзе.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення MorseBridge повинно функціонувати на iOS-пристроях.

Мінімальна конфігурація технічних засобів:

- тип пристрою: iPhone 8 або новіший;
- процесор: A11 Bionic;
- оперативна пам'ять: 2 ГБ;
- операційна система: iOS 16.0 або новіша;
- підключення до мережі Інтернет зі швидкістю від 10 мегабіт;

Рекомендована конфігурація технічних засобів:

- тип пристрою: iPhone 12 або новіший;
- процесор: A14 Bionic;
- оперативна пам'ять: 4 ГБ;
- операційна система: iOS 17.0 або новіша;
- підключення до мережі Інтернет зі швидкістю від 50 мегабіт;

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційної системи iOS.

4.5.1 Вимоги до мови розробки

Розробку виконати на мові програмування Swift

4.5.2 Вимоги до середовища розробки

Розробку виконати за допомогою середовища розробки Xcode

4.5.3 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді посилання на репозиторій на платформі GitHub, а також у додатках до пояснювальної записки.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використань;
– схема структурна контекстна;
– схема структурна контейнерна;
– схема структурна компонентна.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: **Мобільний застосунок для забезпечення комунікації людей з глухотою та сліпотою**

КПІ.ІП-9614.045430.02.81

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE - Integrated Development Environment – інтегроване середовище розробки.

ШІ – Штучний інтелект

GCP – Google Cloud Platform

SSL – Secure Socket Layer

ВСТУП

У сучасному суспільстві технологічний прогрес значно впливає на якість життя людей з різними потребами та можливостями. Незважаючи на численні досягнення, питання забезпечення ефективної комунікації для людей з глухотою та сліпотою залишається актуальним. Ці люди часто стикаються з значними труднощами у повсякденному житті через обмежений доступ до інформації та комунікаційних можливостей. Глухота та сліпота, особливо коли вони поєднані, створюють серйозні бар'єри для інтеграції в суспільство, освіти, працевлаштування та соціальної взаємодії.

Люди з глухотою та сліпотою зазвичай використовують різноманітні методи комунікації, такі як мова жестів, шрифт Брайля, дактильна абетка, тактильні символи та інші. Однак ці методи мають свої обмеження та не завжди є зручними. Наприклад, мова жестів вимагає наявності зорового контакту, а шрифт Брайля потребує спеціальних пристроїв або друкованих матеріалів. Окрім того, далеко не всі люди з оточення особи з інвалідністю володіють цими методами комунікації, що ускладнює взаємодію.

На сьогоднішній день існує кілька технологічних рішень, спрямованих на поліпшення комунікації для людей з глухотою та сліпотою. Одним із таких рішень є Брайлівські дисплеї, які дозволяють людям з глухотою та сліпотою читати текст, який виводиться у вигляді тактильних символів. Проте ці пристрої зазвичай дорогі та громіздкі, що обмежує їх доступність. Іншим варіантом є тактильні комунікатори – спеціальні пристрої, які використовують тактильні символи для передачі інформації. Вони також мають обмежений функціонал і високу вартість, що робить їх менш популярними серед користувачів.

Метою цієї дипломної роботи є розробка інноваційного мобільного застосунку "MorseBridge", який забезпечить ефективну та зручну комунікацію для людей з глухотою та сліпотою. Основна ідея полягає у

використанні коду Морзе, який дозволяє передавати інформацію через вібрації. Такий підхід є універсальним, оскільки не вимагає зорового чи звукового сприйняття інформації.

Запропонований застосунок є важливим кроком до інклюзивності та покращення якості життя для людей з глухотою та сліпотою. Використання мобільних технологій дозволяє створити доступне, функціональне та економічно вигідне рішення, яке може бути впроваджене у повсякденне життя.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Традиційні методи комунікації для людей з глухотою та сліпотою включають використання шрифту Брайля, мови жестів, тактильної абетки та тактильних символів. Шрифт Брайля, винайдений Луї Брайлем у 19 столітті, є системою письма, що дозволяє сліпим людям читати і писати за допомогою тактильних точок. Мова жестів, яка використовується людьми з глухотою, є візуальною мовою, що базується на жестах, виразах обличчя та рухах рук.

У сучасних умовах розвиток інформаційних технологій надав нові можливості для покращення комунікації для людей з глухотою та сліпотою. Одним із таких інноваційних рішень є використання технології Морзе. Код Морзе, винайдена Семюелем Морзе у 1830-х роках, дозволяє передавати інформацію за допомогою коротких та довгих сигналів, що можуть бути закодовані як точки та тире. Це робить її універсальним засобом комунікації, який може бути адаптований для використання в різних середовищах, включаючи тактильні та звукові сигнали.

На поточний момент розвитку ІТ-технологій існує кілька рішень, спрямованих на інтеграцію технологій для покращення комунікації людей з глухотою та сліпотою. Наприклад, Брайлівські дисплеї, що дозволяють відображати текст у вигляді тактильних символів, є важливим інструментом для багатьох користувачів. Проте ці пристрої мають певні обмеження, такі як висока вартість та громіздкість, що ускладнює їх широке впровадження.

Спеціалізовані додатки для перекладу тексту в мову жестів або шрифт Брайля також мають свої недоліки. Вони часто потребують спеціального обладнання та не завжди є зручними у використанні в різних ситуаціях. Це обмежує їх ефективність та доступність для користувачів.

Одним із основних недоліків поточного стану речей є висока вартість та складність використання існуючих технологічних рішень. Більшість наявних пристроїв та програмного забезпечення потребують спеціалізованого обладнання або мають обмежений функціонал. Крім того, багато рішень не є універсальними і потребують певного рівня навчання та адаптації.

Для покращення ситуації у сфері ІТ необхідно розробляти більш доступні та універсальні рішення, що можуть бути легко інтегровані у повсякденне життя людей з глухотою та сліпотою. Одним із таких шляхів є використання коду Морзе для розробки мобільних застосунків, які можуть забезпечити ефективну комунікацію без потреби у дорогому обладнанні або складному навчанні.

В рамках цього дипломного проєкту було обрано шлях розробки мобільного застосунку "MorseBridge", який використовує код Морзе для забезпечення комунікації людей з глухотою та сліпотою. Застосунок дозволяє передавати та отримувати повідомлення за допомогою тактильних сигналів, забезпечуючи зручний та ефективний спосіб спілкування. Основні функції включають переклад голосу в код Морзе, зворотний переклад з Морзе на текст, а також розпізнавання зображень та їх опис кодом Морзе. Інтуїтивний інтерфейс з підтримкою жестів робить використання застосунку простим та зручним, навіть без попереднього досвіду роботи з подібними технологіями.

1.2 Аналіз існуючих рішень

У даному підрозділі проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, які допоможуть у реалізації вашої розробки. Нижче наведено опис готових програмних продуктів, що частково чи повністю реалізують функціонал, описаний у технічному завданні.

Aipoly Vision

Aipoly Vision - це мобільний додаток, що використовує технології штучного інтелекту для розпізнавання об'єктів у реальному часі. Додаток може описувати об'єкти та сцени голосом, що робить його корисним для людей зі слабким зором або повною сліпотою.

Be My Eyes

Be My Eyes - це мобільний додаток, який з'єднує людей зі слабким зором або повною сліпотою з добровольцями через відеодзвінок. Волонтери можуть допомогти користувачам з повсякденними завданнями, такими як читання етикеток чи навігація в новому середовищі.

TapTapSee

TapTapSee - це мобільний додаток, який дозволяє користувачам зі слабким зором або повною сліпотою розпізнавати об'єкти за допомогою камери їхнього смартфона. Користувачі можуть робити фотографії об'єктів, а додаток описує, що на них зображено.

Voiceitt

Voiceitt - це додаток, який використовує технології розпізнавання мови для допомоги людям з вадами мовлення. Додаток може перетворювати мову користувача на зрозумілий текст або синтезований голос.

В таблиці 1.1 приведений порівняльний аналіз додатків.

Таблиця 1.1 – порівняння з аналогами

Функціонал / Продукт	**Aipoly Vision**	**Be My Eyes**	**TapTapSee**	**Voiceitt**	**Morse Bridge**
Інтеграція з кодом Морзе	Ні	Ні	Ні	Ні	Так
Розпізнавання зображень за допомогою ШІ	Так	Ні	Так	Ні	Так
Управління за допомогою жестів	Ні	Ні	Ні	Ні	Так
Використання тактильних сигналів	Ні	Ні	Ні	Ні	Так
Переклад мови в текст	Ні	Ні	Ні	Так	Так
Опис об'єктів голосом	Так	Ні	Так	Ні	Ні

Як видно з таблиці порівняння, жоден з наявних програмних продуктів не об'єднує функціонал розпізнавання об'єктів, опису їх голосом, використання коду Морзе та інтуїтивного інтерфейсу з підтримкою жестів. Це робить MorseBridge унікальним рішенням, яке може значно покращити комунікаційні можливості людей з глухотою та сліпотою.

1.3 Опис бізнес-процесів

Для опису бізнес процесів програмного забезпечення використовується BPMN модель (рисунок 1.1).

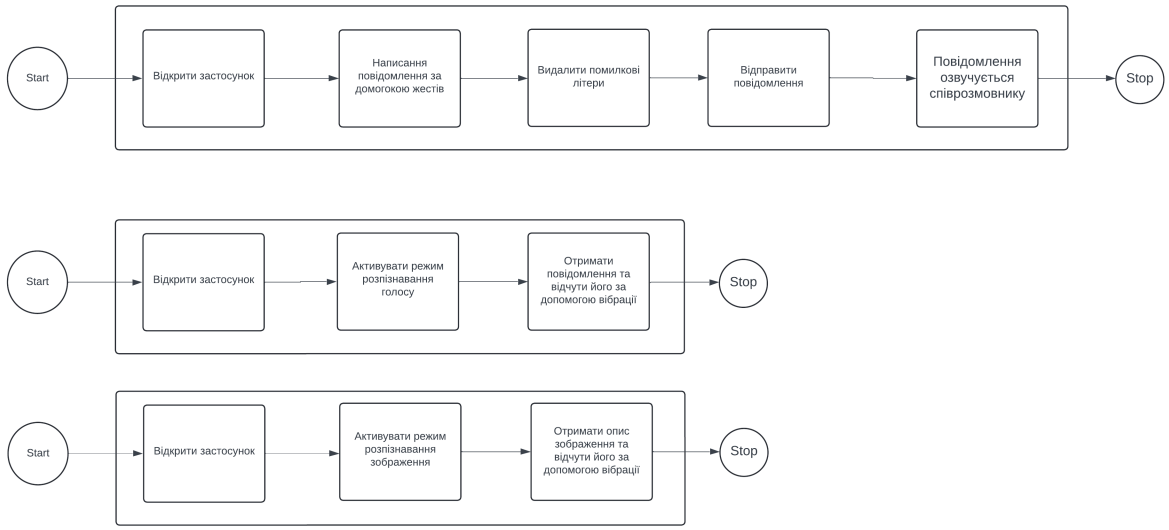


Рисунок 1.1 – Схема бізнес процесів

Бізнес процес 1:

- Користувач відкриває додаток MorseBridge.
- Користувач вводить повідомлення за допомогою жестів.
- Користувач видаляє неправильно написані літери.
- Користувач відправляє повідомлення.
- Повідомлення озвучується співрозмовнику.

Бізнес процес 2:

- Користувач відкриває додаток MorseBridge.
- Користувач торкається і утримує екран для активації функції розпізнавання голосу.
- Додаток слухає співрозмовника та перетворює його мову на текст.
- Текст перекладається в азбуку Морзе та передається користувачу через вібрацію.

Бізнес процес 3:

- Користувач відкриває додаток MorseBridge.
- Користувач трясе телефон для активації функції розпізнавання зображень.
- Додаток робить фотографію об'єкта перед користувачем.
- Штучний інтелект аналізує зображення та описує його у вигляді азбуки Морзе.
- Опис об'єкта передається користувачу через вібрацію.

1.4 Постановка задачі

Метою розробки MorseBridge є створення інноваційного мобільного додатка, який забезпечить ефективну та доступну комунікацію для людей з глухотою та сліпотою. Використовуючи азбуку Морзе та вібрації, додаток надає користувачам можливість спілкуватися без необхідності використання зору або слуху, що значно підвищує їх незалежність та якість життя.

Цілі розробки:

- Забезпечення доступності комунікації: Розробити інтуїтивно зрозумілий інтерфейс, який дозволяє людям з глухотою та сліпотою легко вводити та отримувати повідомлення за допомогою азбуки Морзе.
- Інтеграція сучасних технологій: Використовувати штучний інтелект для розпізнавання зображень та голосу, що дозволить користувачам отримувати детальну інформацію про навколишнє середовище.
- Оптимізація процесу введення даних: Розробити систему жестів, яка забезпечить зручний та швидкий спосіб введення повідомлень.

– Забезпечення надійності та безпеки: Гарантувати захист даних користувачів та надійну роботу додатка.

На вході:

– Дані користувача у вигляді жестів для введення повідомлень.
– Голосові дані для розпізнавання мови.
– Зображення для аналізу штучним інтелектом.

На виході:

– Перекладені в азбуку Морзе повідомлення, які можуть бути отримані та зрозумілі користувачем через вібрації.
– Текстові описи зображень, передані користувачеві у вигляді азбуки Морзе.
– Перекладені голосові повідомлення у вигляді тексту та азбуки Морзе.

Висновки до розділу

У даному розділі було проведено детальний аналіз існуючих рішень у сфері забезпечення комунікації для людей з глухотою та сліпотою, а також розглянуто бізнес-процеси та поставлені задачі для розробки мобільного додатка MorseBridge.

Було досліджено кілька наявних програмних продуктів, таких як Aipoly Vision, Be My Eyes, TapTapSee та Voiceitt. Кожен із цих продуктів має свої унікальні функції, такі як розпізнавання об'єктів, допомога через відеодзвінки, розпізнавання мови та опис об'єктів голосом. Проте, жоден з цих додатків не об'єднує всі необхідні функції для забезпечення повноцінної комунікації для людей з глухотою та сліпотою.

Було проведено порівняльний аналіз, який показав, що MorseBridge буде унікальним додатком, оскільки він включає в себе не тільки функції розпізнавання об'єктів і голосу, але й використання азбуки Морзе для введення та отримання повідомлень, що забезпечує інтуїтивно зрозуміле управління додатком.

Розглянуто бізнес-процеси, які включають введення та управління повідомленнями, отримання повідомлень, розпізнавання голосу та зображень. Основні жести та розширені жести, які використовуються в MorseBridge, дозволяють користувачам зручно вводити та керувати повідомленнями без необхідності візуального контролю. Процеси розпізнавання голосу та зображень забезпечують додаткову функціональність для отримання інформації про навколишнє середовище та комунікації з іншими людьми.

Визначено мету розробки MorseBridge – створення інноваційного мобільного додатка, який забезпечить ефективну та доступну комунікацію для людей з глухотою та сліпотою. Сформульовано основні цілі та задачі, які необхідно вирішити в рамках проекту. На вході система приймає дані користувача у вигляді жестів, голосових команд та зображень, а на виході забезпечує передані повідомлення у вигляді азбуки Морзе через вібрації або озвучення.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Основний функціонал додатка включає:

- Введення повідомлень: Користувачі можуть вводити повідомлення за допомогою інтуїтивно зрозумілих жестів, які перетворюються на код Морзе та озвучуються співрозмовнику.
- Розпізнавання голосу: Додаток може розпізнавати голос співрозмовника та перетворювати їх на текст, який потім перекладається в азбуку Морзе.
- Розпізнавання зображення: Використовуючи штучний інтелект, додаток аналізує зображення, зроблені користувачем, та описує їх у вигляді коду Морзе.

На рисунку 2.1 можна побачити діаграму варіантів використання.

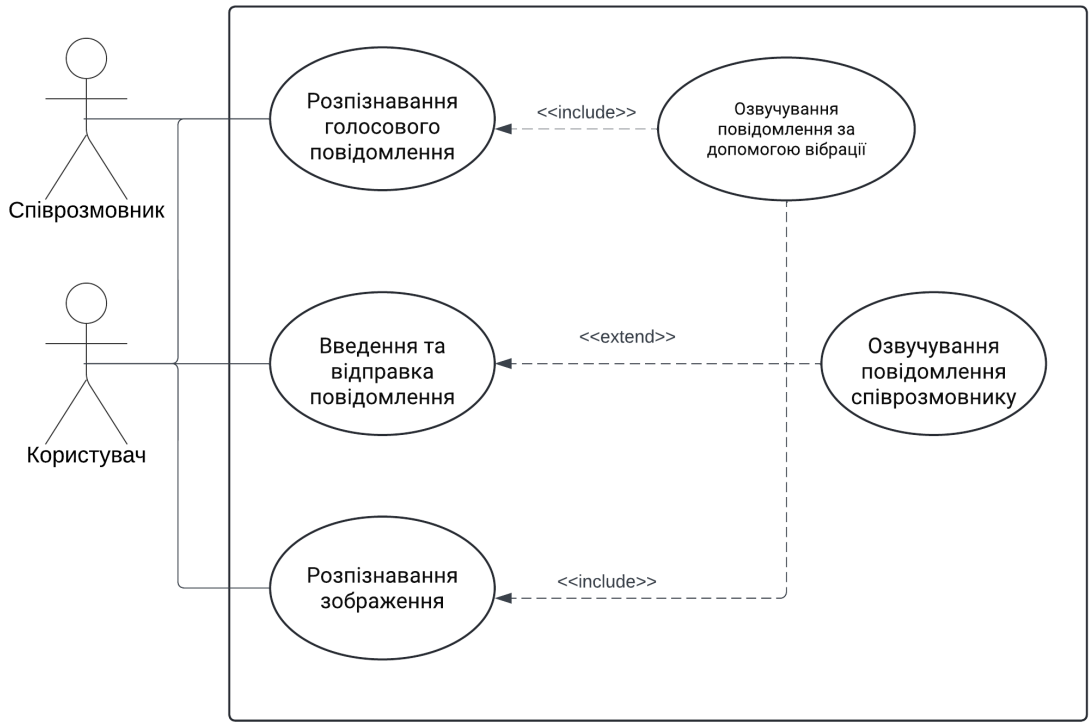


Рисунок 2.1 - Діаграма варіантів використання

В таблицях 2.1 - 2.6 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 - Варіант використання UC-01

Use case name	Введення та відправка повідомлення
Use case ID	UC-01
Goals	Користувач вводить повідомлення за допомогою жестів азбуки Морзе та відправляє його
Actors	Користувач
Trigger	Користувач хоче надіслати повідомлення
Pre-conditions	Користувач відкрив додаток MorseBridge і знаходиться в режимі введення повідомлень
Flow of Events	1. Користувач вводить повідомлення за допомогою жестів. 2. Додаток перетворює жести в символи азбуки Морзе. 3. Користувач підтверджує повідомлення. 4. Додаток відправляє повідомлення.
Extension	Озвучування повідомлення співрозмовнику (UC-04)
Post-Condition	Повідомлення успішно відправлено співрозмовнику

Таблиця 2.2 - Варіант використання UC-02

Use case name	Отримання повідомлення
Use case ID	UC-02
Goals	Користувач отримує повідомлення через вібрації
Actors	Користувач
Trigger	Надходить нове повідомлення
Pre-conditions	Додаток запущений і готовий до прийому повідомлень.

Продовження таблиці 2.2

Flow of Events	1. Система отримує повідомлення 2. Додаток перетворює повідомлення в азбуку Морзе. 3. Повідомлення передається користувачу через вібрацію.
Extension	Немає
Post-Condition	Користувач успішно отримав і зрозумів повідомлення.

Таблиця 2.3 - Варіант використання UC-03

Use case name	Розпізнавання голосового повідомлення
Use case ID	UC-03
Goals	Користувач промовляє повідомлення, яке перетворюється на текст і далі в азбуку Морзе.
Actors	Користувач, Співрозмовник
Trigger	Користувач активує функцію розпізнавання голосу.
Pre-conditions	Додаток запущений і готовий до розпізнавання голосу.
Flow of Events	1. Користувач активує функцію розпізнавання голосу. 2. Користувач промовляє повідомлення. 3. Додаток розпізнає промовлене повідомлення і перетворює його на текст. 4. Текст переводиться в код Морзе. 5. Повідомлення передається користувачу через вібрацію або озвучення.
Extension	Озвучування повідомлення за допомогою вібрації (UC-05).
Post-Condition	Користувач отримав повідомлення у вигляді вібрації або озвучення.

Таблиця 2.4 - Варіант використання UC-04

Use case name	Розпізнавання зображення
Use case ID	UC-04
Goals	Користувач використовує камеру для зйомки об'єкта, а додаток аналізує зображення і описує його за допомогою азбуки Морзе.
Actors	Користувач
Trigger	Користувач активує функцію розпізнавання зображення.
Pre-conditions	Додаток запущений і готовий до розпізнавання зображення.
Flow of Events	1. Користувач активує функцію розпізнавання зображення. 2. Додаток робить фотографію. 3. Штучний інтелект аналізує зображення. 4. Система описує об'єкт у вигляді азбуки Морзе. 5. Опис передається користувачу через вібрацію або озвучення.
Extension	Озвучування повідомлення за допомогою вібрації (UC-05).
Post-Condition	Користувач отримав опис зображення у вигляді вібрації або озвучення.

Таблиця 2.5 - Варіант використання UC-05

Use case name	Озвучування повідомлення за допомогою вібрації
Use case ID	UC-05
Goals	Повідомлення передається користувачу через вібрацію.
Actors	Користувач
Trigger	Користувач отримує повідомлення або результат розпізнавання голосу чи зображення.
Pre-conditions	Повідомлення готове до передачі.

Продовження таблиці 2.5

Flow of Events	1. Система готує повідомлення до передачі. 2. Додаток передає повідомлення через вібрацію. 3. Користувач отримує повідомлення.
Extension	Немає
Post-Condition	Користувач отримав повідомлення через вібрацію.

Таблиця 2.6 - Варіант використання UC-06

Use case name	Озвучування повідомлення співрозмовнику
Use case ID	UC-06
Goals	Повідомлення передається співрозмовнику через аудіо або текст.
Actors	Користувач
Trigger	Користувач відправляє повідомлення співрозмовнику.
Pre-conditions	Повідомлення готове до передачі.
Flow of Events	1. Система готує повідомлення до передачі. 2. Додаток передає повідомлення співрозмовнику через аудіо або текст. 3. Співрозмовник отримує повідомлення.
Extension	Немає
Post-Condition	Співрозмовник отримав повідомлення.

2.2 Розроблення функціональних вимог

Програмне забезпечення MorseBridge розділене на кілька модулів, кожен з яких виконує певний набір функцій. Загальна модель вимог наведена у таблиці 2.7. Опис функціональних вимог до кожного модуля наведений у таблицях 2.8 – 2.11. На рисунку 2.2 наведена матриця трасування вимог.

Таблиця 2.7 – Модель вимог

Введення повідомлень	Модуль для введення повідомлень за допомогою жестів азбуки Морзе.
Отримання повідомлень	Модуль для отримання повідомлень через вібрації або озвучення.
Розпізнавання голосу	Модуль для розпізнавання голосових повідомлень та їх конвертації в текст та азбуку Морзе.
Розпізнавання зображень	Модуль для розпізнавання зображень і опису їх за допомогою азбуки Морзе.

Таблиця 2.8 – Вимоги модуля “Введення повідомлень”

ID Вимоги	Опис	Пріоритет
FR-01	Користувач повинен мати можливість вводити повідомлення за допомогою жестів.	Високий
FR-02	Додаток повинен коректно інтерпретувати дотики та рухи пальців для введення точок і тире.	Високий
FR-03	Додаток повинен підтверджувати введені літери жестом вниз.	Високий
FR-04	Користувач повинен мати можливість видаляти останню літеру одним пальцем вліво.	Середній

Таблиця 2.9 – Вимоги модуля “Отримання повідомлень”

ID Вимоги	Опис	Пріоритет
FR-05	Додаток повинен підтримувати отримання повідомлень через вібрацію або озвучення.	Високий
FR-06	Користувач повинен мати можливість отримувати повідомлення від співрозмовника через азбуку Морзе.	Високий
FR-07	Повідомлення повинні бути передані через вібрацію.	Високий

Таблиця 2.10 – Вимоги модуля “Розпізнавання голосу”

ID Вимоги	Опис	Пріоритет
FR-08	Додаток повинен підтримувати отримання повідомлень через вібрацію або озвучення.	Високий
FR-09	Користувач повинен мати можливість отримувати повідомлення від співрозмовника через азбуку Морзе.	Високий
FR-10	Повідомлення повинні бути передані через вібрацію.	Високий

Таблиця 2.11 – Вимоги модуля “Розпізнавання зображень”

ID Вимоги	Опис	Пріоритет
FR-11	Користувач повинен мати можливість активувати розпізнавання зображення трясінням телефону.	Високий
FR-12	Додаток повинен робити фотографію об'єкта.	Високий
FR-13	Штучний інтелект повинен аналізувати зображення і описувати його у вигляді азбуки Морзе.	Високий

Продовження таблиці 2.11

FR-14	Опис зображення повинен передаватися користувачу через вібрацію.	Високий

Use Case ID	FR-01	FR-02	FR-03	FR-04	FR-05	FR-06	FR-07	FR-08	FR-09	FR-10	FR-11	FR-12	FR-13	FR-14
UC-01	X	X	X	X	X									
UC-02						X	X	X						
UC-03									X	X	X			
UC-04												X	X	X
UC-05								X	X			X		

Рисунок 2.2 – Матриця трасування вимог

2.2 Розроблення нефункціональних вимог

Список нефункціональних вимог до програмного забезпечення:

- Швидкість обробки:
  1. Додаток повинен обробляти вхідні жести і голосові команди з мінімальною затримкою.
  2. Розпізнавання та обробка зображень повинні виконуватись не довше ніж за 3 секунди.
- Масштабованість системи:
  1. Система повинна підтримувати збільшення кількості користувачів без зниження продуктивності.
  2. Архітектура додатка повинна дозволяти додавання нових функцій і можливостей без значних змін у існуючому коді.
- Надійність:
  1. Додаток повинен працювати без збоїв при інтенсивному використанні.
- Обробка помилок:
  1. Система повинна мати механізми для виявлення та обробки помилок, що виникають під час роботи.
- Зручність використання:

1. Інтерфейс додатка повинен бути інтуїтивно зрозумілим і зручним для користувачів з глухотою та сліпотою.
2. Всі функції повинні бути доступні через прості та зрозумілі жести.

– Офлайн режим:
1. Додаток повинен мати обмежену функціональність в офлайн режимі.

– Платформа:
1. Додаток повинен працювати на всіх сучасних версіях iOS.
2. Повинна бути забезпечена сумісність з основними моделями мобільних пристроїв, що підтримують iOS.

Висновки до розділу

У цьому розділі було детально розглянуто функціональні та нефункціональні вимоги до розробки мобільного додатка MorseBridge. Функціональні вимоги охоплюють основні можливості додатка, такі як введення, відправка та отримання повідомлень, розпізнавання голосу та зображень, а також озвучування повідомлень за допомогою вібрації. Було створено матрицю трасування вимог, яка забезпечує зв'язок між функціональними вимогами та відповідними use case-ами, що допомагає гарантувати повноту реалізації кожної функції.

Нефункціональні вимоги були визначені для забезпечення високої продуктивності, надійності, безпеки та зручності використання додатка. Вимоги щодо продуктивності забезпечують мінімальні затримки при обробці даних, а вимоги щодо надійності включають безперебійну роботу додатка та обробку помилок. Вимоги щодо безпеки гарантують захист даних користувачів, а зручність використання забезпечується інтуїтивно зрозумілим інтерфейсом та підтримкою користувачів. Крім того, були визначені вимоги щодо сумісності та мобільності, які забезпечують роботу

додатка на різних платформах і оптимальне використання ресурсів пристрою.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Для розробки мобільного додатка MorseBridge було обрано архітектурний паттерн MVVM (Model-View-ViewModel) у поєднанні з Coordinator. Ця архітектура забезпечує чітке розділення відповідальності між різними компонентами додатка, що дозволяє підвищити його масштабованість, тестованість та підтримуваність.

View Layer - Використовується для створення інтуїтивно зрозумілого та доступного інтерфейсу користувача, який дозволяє вводити та отримувати повідомлення за допомогою жестів, озвучення та вібрації.

ViewModel Layer - Забезпечує обробку даних та логіку представлення, забезпечуючи зв'язок між View і Model.

Model Layer - Визначає структуру даних, що використовуються в додатку. Містить основну логіку додатка, включаючи обробку повідомлень та взаємодію з API.

Coordinator Layer - Управляє навігацією між різними екранами додатка, забезпечуючи чистий та зрозумілий потік користувача.

У таблиці 3.1 наведено перелік основних класів та опис їх функцій.

Таблиця 3.1 – Класи та їх функції

Клас	Функції
AppDependencyContainer	Ініціалізація та надання залежностей для інших компонентів додатка
Haptic	Створення та управління тактильними сигналами за допомогою Core Haptic
CameraService	Управління доступом до камери пристрою
OpenAIService	Відправка запитів до OpenAI API через GCP Cloud Functions
SpeechRecognitionService	Використання вбудованих можливостей iOS для розпізнавання голосу
SpeechSynthesizerService	Використання вбудованих можливостей iOS для синтезу мови
InstructionsCoordinator	Управління навігацією для екранів інструкцій
InstructionsViewModel	Логіка представлення даних для екранів інструкцій
DictionaryCoordinator	Управління навігацією для словника азбуки Морзе
DictionaryViewModel	Логіка представлення даних для словника азбуки Морзе
MorsePlaygroundUIView	Користувацький інтерфейс для введення азбуки Морзе
MainCoordinator	Основне управління навігацією додатка
MainViewModel	Логіка представлення для головного екрану додатка
AppViewModel	Глобальна логіка представлення для додатка
AppCoordinator	Управління всією навігацією та координацією між різними частинами додатка

На рисунку 3.1 можна побачити контекстну діаграму (C4 Level 1).

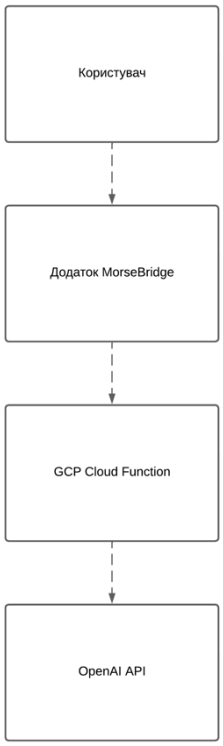


Рисунок 3.1 – Контекстна діаграма

На рисунку 3.2 можна побачити контейнерну діаграму (C4 Level 2)

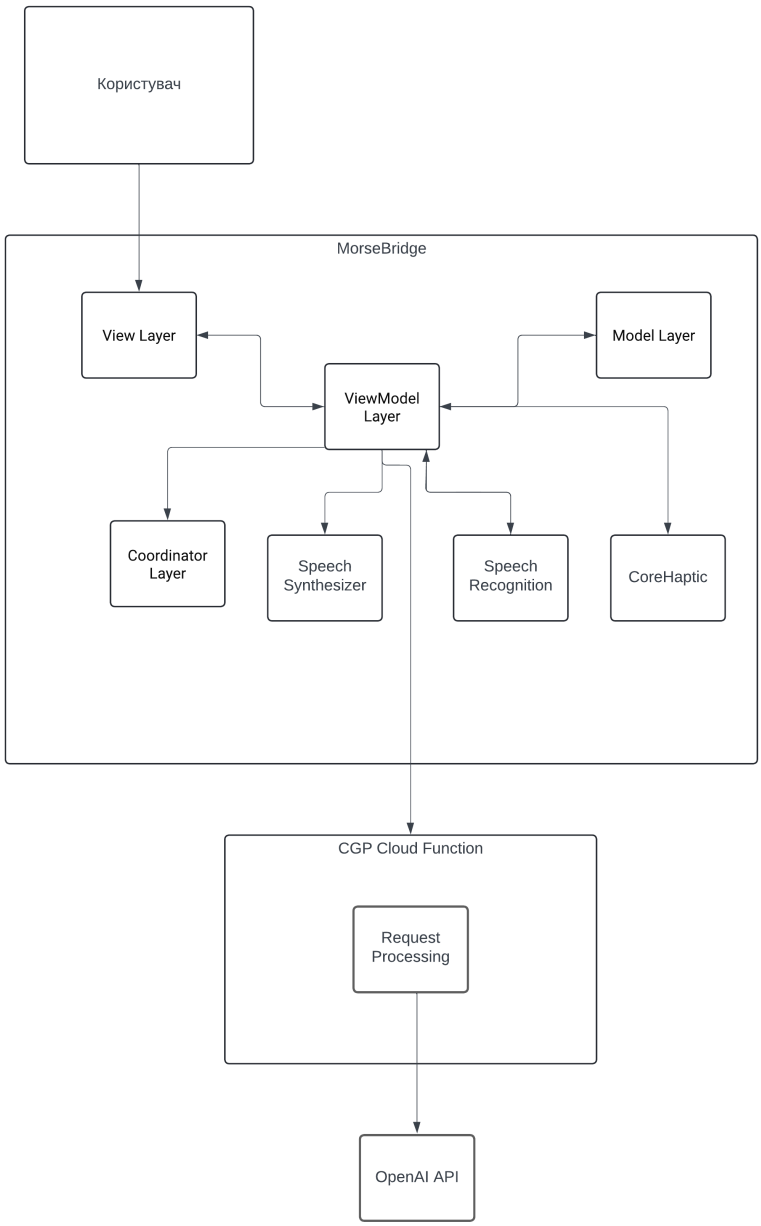


Рисунок 3.2 – Контейнерна діаграма

На рисунку 3.3 можна побачити діаграму компонентів (C4 Level 3).

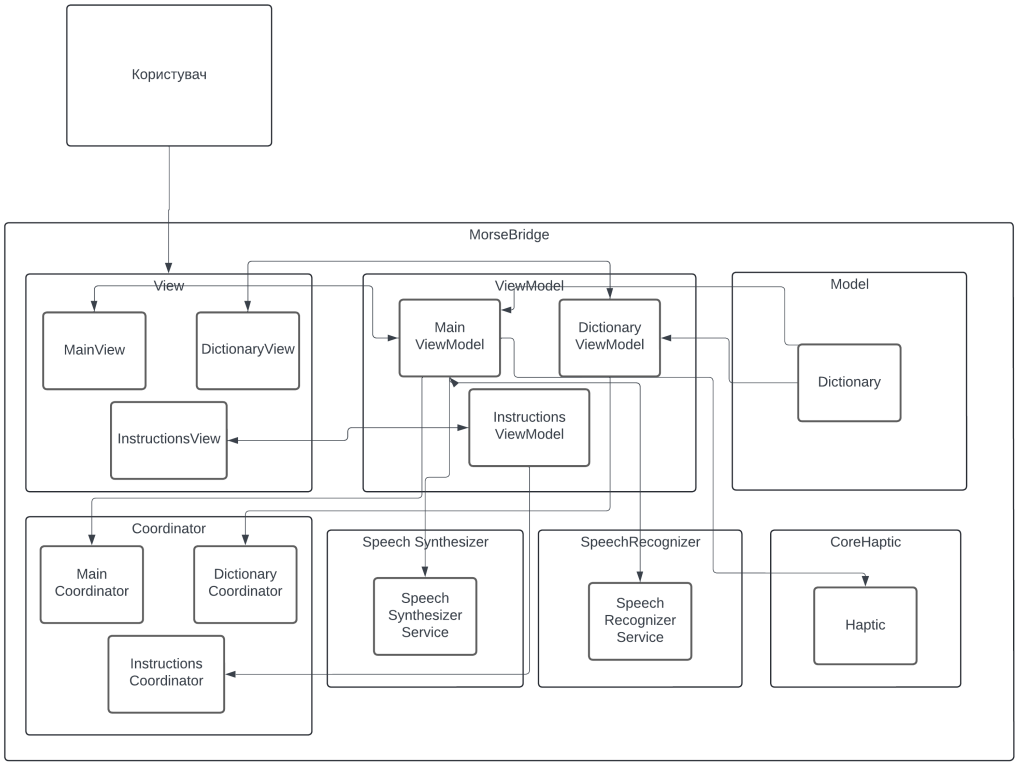


Рисунок 3.3 – Діаграма компонентів

3.2 Обґрунтування вибору засобів розробки

Swift був обраний як основна мова програмування для розробки додатку MorseBridge. Основні переваги Swift включають:

- Продуктивність: Swift забезпечує високу продуктивність, що особливо важливо для обробки даних та взаємодії з користувачем в режимі реального часу.
- Безпека: Swift має потужні механізми для запобігання помилок, таких як типобезпечність та управління пам'яттю.
- Сучасний синтаксис: Swift має зрозумілий та лаконічний синтаксис, що спрощує розробку та підтримку коду.
- Інтеграція з iOS: Swift є рідною мовою для розробки додатків під iOS, що дозволяє легко використовувати всі можливості платформи.

SwiftUI був обраний для розробки нових компонентів UI завдяки його декларативному підходу та підтримці живого попереднього перегляду, що прискорює процес розробки. UIKit використовується для компонентів, які ще не підтримуються або мають кращу реалізацію в UIKit.

Xcode був обраний як основна середовище розробки (IDE) для MorseBridge. Основні причини включають:

- Інтеграція з iOS: Xcode є офіційним IDE для розробки під iOS, забезпечуючи повну інтеграцію з усіма інструментами та бібліотеками Apple.
- Інструменти для відладки: Потужні інструменти для відладки, такі як LLDB, Instruments, та вбудовані тестові фреймворки.
- Підтримка SwiftUI та UIKit: Повна підтримка обох фреймворків, включаючи живий попередній перегляд для SwiftUI.

OpenAI API використовується для розпізнавання зображень та генерації описів. Він був обраний завдяки його потужним можливостям для генерації тексту та високій точності розпізнавання.

Core Haptic використовується для створення та управління тактильними сигналами. Основні переваги:

- Інтеграція з iOS: Core Haptic є рідною бібліотекою iOS для роботи з вібрацією, що забезпечує високу продуктивність та низькі затримки.
- Гнучкість: Підтримує створення складних вібраційних патернів, що важливо для передачі повідомлень через азбуку Морзе.

Вбудовані інструменти iOS використовуються для синтезу мови та розпізнавання голосу. Основні переваги:

- Інтеграція з iOS: Забезпечує високу продуктивність та простоту використання завдяки рідній підтримці.
- Точність: Висока точність розпізнавання та синтезу мови.

3.3 Конструювання програмного забезпечення

У MorseBridge було розроблено кілька оригінальних алгоритмів та модифіковано існуючі для забезпечення специфічних функцій додатка.

Для введення азбуки Морзе за допомогою жестів було створено алгоритм, який розпізнає різні типи жестів та перетворює їх у відповідні символи. Наприклад, дотик до екрана інтерпретується як точка, проведення вправо - як тире, проведення вниз підтверджує введення символу, а проведення вліво видаляє останній символ. Цей алгоритм дозволяє користувачам легко вводити текст без необхідності використання візуального інтерфейсу.

Алгоритм розпізнавання зображень та генерації описів використовує OpenAI API для розпізнавання зображень та генерації описів. Зображення,

зроблене за допомогою камери, надсилається до API, який повертає опис зображення у вигляді тексту. Потім цей текст перекладається в азбуку Морзе та передається користувачу через вібрацію або озвучення. Цей процес включає кілька етапів: захоплення зображення, надсилання запиту до API, обробка відповіді та передача повідомлення користувачу.

Структура даних для зберігання азбуки Морзе

Для зберігання відповідності символів азбуки Морзе використовується структура даних типу словник. Словник містить пари "символ - код Морзе", що дозволяє швидко знайти відповідний код для будь-якої літери або цифри.

Модель даних для зберігання повідомлень

Модель даних для зберігання повідомлень включає наступні атрибути: унікальний ідентифікатор повідомлення, текст повідомлення, та відправник (ми, співрозмовник або ШІ). Ця модель дозволяє зберігати і відслідковувати всі повідомлення, які були створені та відправлені користувачем.

Опис бази даних

MorseBridge не використовує локальної бази даних, оскільки всі дані зберігаються та обробляються в реальному часі за допомогою сервісів OpenAI API та внутрішніх механізмів обробки на пристрої. Замість цього додаток використовує локальні структури даних для тимчасового зберігання інформації про користувача та його повідомлення.

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення

OpenAI API використовується для розпізнавання зображень та генерації текстових описів. Цей сервіс забезпечує високоточне

розпізнавання об'єктів на зображеннях та створення описів, що дозволяє MorseBridge надавати користувачам точну інформацію про їхнє оточення.

Core Haptic використовується для створення та управління тактильними сигналами. Ця бібліотека дозволяє створювати складні вібраційні патерни, які використовуються для передачі повідомлень азбукою Морзе. Використання Core Haptic забезпечує високу точність і надійність тактильного зворотного зв'язку.

Вбудовані інструменти iOS для синтезу мови та розпізнавання голосу використовуються для забезпечення голосової взаємодії користувачів з додатком. Speech Synthesizer перетворює текст на мову, що дозволяє озвучувати повідомлення, а Speech Recognizer розпізнає голосові команди користувачів, що дозволяє вводити текст за допомогою голосу.

3.4 Аналіз безпеки даних

Передача даних є одним з критично важливих аспектів безпеки для додатку MorseBridge. У процесі взаємодії між клієнтським додатком та серверними компонентами, такими як OpenAI API, існує ризик перехоплення даних зловмисниками. Це стосується як текстових повідомлень, так і зображень, що обробляються для генерації описів. Однією з основних вразливостей є можливість перехоплення даних під час їх передачі. Зловмисники можуть скористатися цим для отримання конфіденційної інформації або для модифікації даних, що передаються.

Для захисту даних від перехоплення використовується протокол HTTPS, який забезпечує шифрування всіх даних, що передаються між клієнтом і сервером. HTTPS використовує SSL/TLS сертифікати для створення зашифрованого каналу зв'язку, що значно ускладнює можливість атак типу "людина посередині" (Man-in-the-Middle). Регулярне оновлення та перевірка SSL/TLS сертифікатів є необхідною умовою для підтримання високого рівня безпеки. Це дозволяє

гарантувати, що з'єднання залишаються захищеними та відповідають найновішим стандартам безпеки.

Крім того, використання сторонніх API, таких як OpenAI API, також може стати джерелом потенційних вразливостей. Недостатня безпека сторонніх сервісів може призвести до несанкціонованого доступу до даних або втрати конфіденційності інформації, що передається до або з API. Щоб мінімізувати ці ризики та захистити ключ доступу до OpenAI API, у MorseBridge використовується власний backend на основі GCP Cloud Functions. Це рішення дозволяє зберігати API ключі у безпечному середовищі на серверній стороні, виключаючи необхідність зберігання їх на клієнтському пристрої, де вони можуть бути вразливими для атак. GCP Cloud Functions обробляють всі запити до OpenAI API, виступаючи проміжною ланкою між клієнтським додатком і API. Це не лише забезпечує захист ключів доступу, але й дозволяє додатково фільтрувати і контролювати запити перед їх надсиланням до OpenAI.

Висновки до розділу

У розділі 3 було розглянуто важливі аспекти розробки програмного забезпечення MorseBridge, включаючи архітектуру, вибір засобів розробки, процес конструювання та аналіз безпеки даних. Архітектура додатка, заснована на патерні MVVM у поєднанні з координатором, забезпечує чітке розділення відповідальностей і підвищує масштабованість та підтримуваність системи. Вибір Swift як основної мови програмування, а також використання SwiftUI та UIKit для створення інтерфейсу користувача обґрунтовано їхніми перевагами в продуктивності, безпеці та сучасності. Конструювання програмного забезпечення включало розробку оригінальних алгоритмів для обробки жестів і розпізнавання зображень, що забезпечують специфічні функції додатка. Для захисту даних від перехоплення було впроваджено

використання HTTPS, регулярне оновлення SSL/TLS сертифікатів. Власний backend на базі GCP Cloud Functions забезпечує додатковий рівень безпеки для ключів доступу до OpenAI API. Усі ці заходи сприяють створенню надійного, безпечного та функціонального додатка, що відповідає потребам користувачів з глухотою та сліпотою.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

У цьому розділі виконується аналіз якості програмного забезпечення MorseBridge, створеного для покращення комунікації людей з глухотою та сліпотою. Основна мета тестування полягає у забезпеченні високої надійності, продуктивності та коректності роботи ПЗ відповідно до встановлених вимог.

Завдання тестування включають:

- перевірку правильності роботи додатка відповідно до функціональних вимог;
- перевірку сумісності додатка з різними версіями iOS;
- виявлення проблем, помилок і недоліків для їх усунення;
- оцінку зручності графічного інтерфейсу.

Для оцінки якості програмного забезпечення було обрано наступні метрики:

- Час відгуку додатка при виконанні основних функцій.
- Стабільність роботи додатка під час тривалого використання та здатність відновлюватися після помилок.
- Коректна робота додатка на різних версіях iOS.
- Інтуїтивність та доступність інтерфейсу для користувачів.

Цей підхід до аналізу якості не лише допомагає оцінити поточний стан програмного забезпечення, але й визначити основні напрямки для його подальшого вдосконалення та оптимізації.

4.2 Опис процесів тестування

Тестування виконується згідно з документом «Програма та методика тестування». Було виконане мануальне тестування програмного забезпечення MorseBridge, опис відповідних тестів наведено у таблицях 4.1-4.10.

Таблиця 4.1 – Тест 1

Початковий стан	Користувач знаходиться на головному екрані додатка.
Вхідні дані	Жест дотиків та проведень для введення точок та тире.
Опис проведення тесту	Користувач виконує дотики для введення точок та проведення вправо для тире. Після введення символу виконує проведення вниз для підтвердження букви.
Очікуваний результат	Введений текст у форматі Морзе відображається на екрані.
Фактичний результат	Збігається з очікуваним.
Початковий стан	Користувач знаходиться на головному екрані додатка.

Таблиця 4.2 – Тест 2

Початковий стан	Введено текст у форматі Морзе.
Вхідні дані	Проведення вліво одним пальцем.
Опис проведення тесту	Користувач виконує проведення вліво одним пальцем для видалення останнього введеного символу.
Очікуваний результат	Останній символ видаляється з тексту.
Фактичний результат	Збігається з очікуваним.
Початковий стан	Введено текст у форматі Морзе.

Таблиця 4.3 – Тест 3

Початковий стан	Введено текст у форматі Морзе.
Вхідні дані	Проведення вліво двома пальцями.
Опис проведення тесту	Користувач виконує проведення вліво двома пальцями для видалення всього введеного тексту.
Очікуваний результат	Весь введений текст видаляється.
Фактичний результат	Збігається з очікуваним.
Початковий стан	Введено текст у форматі Морзе.

Таблиця 4.4 – Тест 4

Початковий стан	Користувач знаходиться на головному екрані додатка.
Вхідні дані	Голосове повідомлення.
Опис проведення тесту	Користувач затискає екран та вимовляє фразу. Додаток перетворює голосове повідомлення в код Морзе.
Очікуваний результат	Голосове повідомлення успішно перетворюється в код Морзе і відображається на екрані.
Фактичний результат	Збігається з очікуваним.
Початковий стан	Користувач знаходиться на головному екрані додатка.

Таблиця 4.5 – Тест 5

Початковий стан	Введено текст у форматі Морзе.
Вхідні дані	Проведення вгору одним пальцем.
Опис проведення тесту	Користувач виконує проведення вгору одним пальцем для озвучування введеного тексту.
Очікуваний результат	Введений текст озвучується.
Фактичний результат	Збігається з очікуваним.

Продовження таблиці 4.5

Початковий стан	Введено текст у форматі Морзе.

Таблиця 4.6 – Тест 6

Початковий стан	Користувач знаходиться на головному екрані додатка.
Вхідні дані	Зображення, зроблене за допомогою камери.
Опис проведення тесту	Користувач трясе телефон, щоб активувати розпізнавання зображень. Додаток робить фотографію та надсилає її для розпізнавання.
Очікуваний результат	Опис зображення успішно перетворюється в код Морзе та передається користувачу через вібрацію.
Фактичний результат	Збігається з очікуваним.
Початковий стан	Користувач знаходиться на головному екрані додатка.

Таблиця 4.7 – Тест 7

Початковий стан	Введено текст у форматі Морзе.
Вхідні дані	Проведення вниз одним пальцем.
Опис проведення тесту	Користувач виконує проведення вниз одним пальцем для підтвердження введеного повідомлення та його надсилання.
Очікуваний результат	Повідомлення надсилається успішно.
Фактичний результат	Збігається з очікуваним.
Початковий стан	Введено текст у форматі Морзе.

Таблиця 4.8 – Тест 8

Початковий стан	Введено текст у форматі Морзе, отримано нове повідомлення.
Вхідні дані	Проведення вниз двома пальцями.
Опис проведення тесту	Користувач виконує проведення вниз двома пальцями для відтворення останнього отриманого повідомлення у вигляді коду Морзе через вібрацію.

Продовження таблиці 4.8

Очікуваний результат	Останнє отримане повідомлення передається користувачу через вібрацію у форматі коду Морзе.
Фактичний результат	Збігається з очікуваним.
Початковий стан	Введено текст у форматі Морзе, отримано нове повідомлення.

Таблиця 4.9 – Тест 9

Початковий стан	Введено текст у форматі Морзе.
Вхідні дані	Зведення пальців.
Опис проведення тесту	Користувач виконує жест зведення пальців для передачі всього введеного тексту.
Очікуваний результат	Весь введений текст успішно передається.
Фактичний результат	Збігається з очікуваним.
Початковий стан	Введено текст у форматі Морзе.

Таблиця 4.10 – Тест 10

Початковий стан	Введено текст у форматі Морзе.
Вхідні дані	Розведення пальців.
Опис проведення тесту	Користувач виконує жест розведення для передачі останньої введеної літери.
Очікуваний результат	Остання введена літера успішно передається.
Фактичний результат	Збігається з очікуваним.
Початковий стан	Введено текст у форматі Морзе.

4.3 Опис контрольного прикладу

Для демонстрації функціоналу MorseBridge ми розглянемо повний цикл використання додатка, включаючи введення повідомлення у форматі Морзе, видалення символів, передачу введеного тексту, розпізнавання зображень та перегляд отриманих повідомлень.

Користувач виконує дотики для введення точок та проведення вправо для тире. Для підтвердження введеної букви користувач проводить вниз одним пальцем.

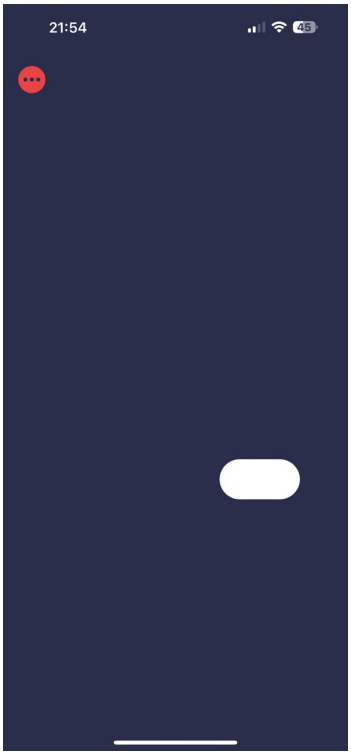


Рисунок 4.1 – Введення тире

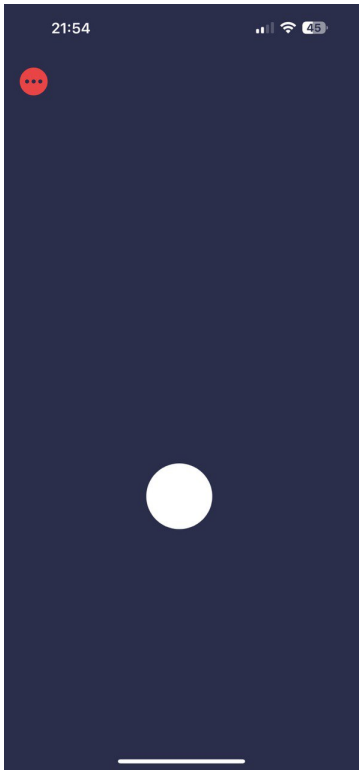


Рисунок 4.2 – Введення крапки

Користувач проводить вгору одним пальцем для виведення повідомлення.

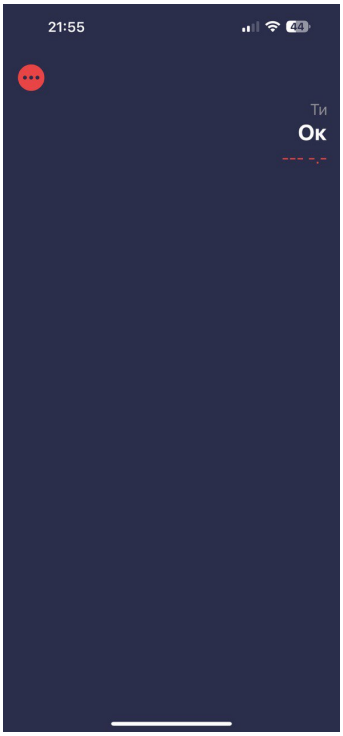


Рисунок 4.3 – Введення повідомлення

Користувач трясе телефон, щоб активувати розпізнавання зображень.

Рисунок 4.4 – Результат розпізнавання зображення

Користувач затискає екран для запису голосового повідомлення. Користувач вимовляє фразу, і додаток перетворює голосове повідомлення в код Морзе.

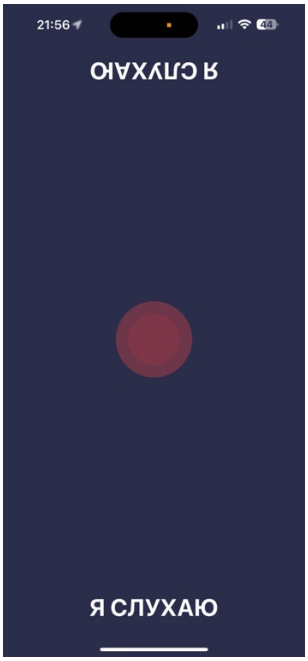


Рисунок 4.5 – Екран розпізнавання мови

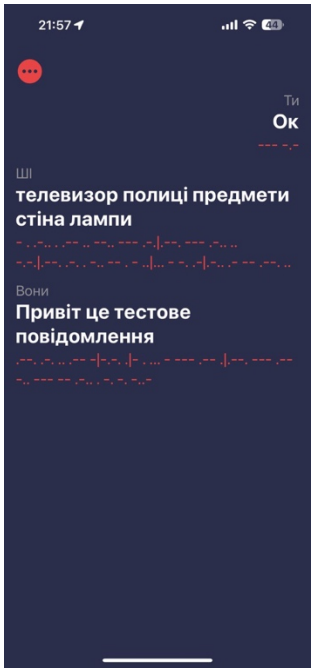


Рисунок 4.6 – Результат розпізнавання мови

Користувач натискає на кнопку для відкриття екрану з азбукою Морзе.

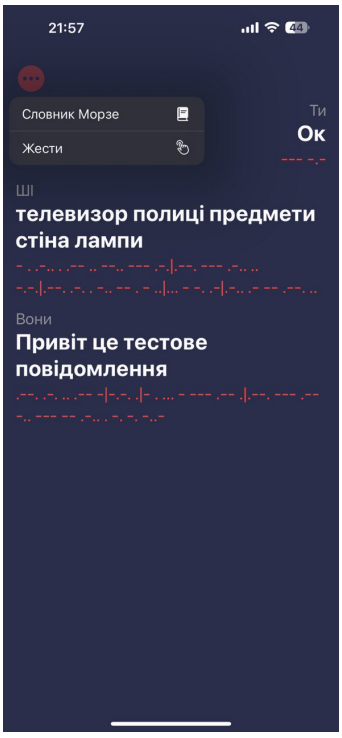


Рисунок 4.7 – Кнопка вибору словника Морзе

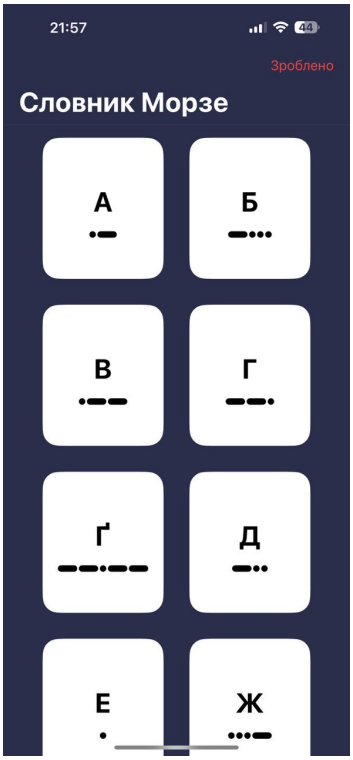


Рисунок 4.8 – Екран словника Морзе

Користувач натискає на кнопку для відкриття екрану з жестами.

Рисунок 4.9 – Екран інструкції жестів

Цей контрольний приклад включає всі можливі розгалуження взаємодії з додатком MorseBridge, з якими може стикатися користувач. Він забезпечує зрозуміле і повне представлення процесу введення, передачі та отримання повідомлень у форматі Морзе, а також використання додатка для розпізнавання зображень і голосових команд.

Висновки до розділу

У даному розділі було проведено детальний аналіз якості програмного забезпечення MorseBridge, визначено метрики для оцінки його продуктивності, надійності, сумісності та зручності використання. Розглянуто основні завдання тестування, серед яких перевірка правильності роботи відповідно до функціональних вимог, забезпечення сумісності з різними версіями iOS, виявлення та усунення проблем, а також оцінка зручності графічного інтерфейсу.

Тестування виконувалося вручну, що дозволило виявити та усунути помилки, забезпечити стабільну роботу додатка та підтвердити відповідність програмного забезпечення встановленим вимогам. Описані процеси тестування охоплюють всі ключові функції MorseBridge, включаючи введення та видалення тексту у форматі Морзе, передачу повідомлень, розпізнавання зображень та голосове введення.

Контрольний приклад демонструє повний цикл використання додатка, включаючи всі можливі розгалуження та особливості взаємодії. Користувачі можуть легко переходити між екранами, вводити та відправляти повідомлення, розпізнавати зображення та використовувати голосові команди. Це забезпечує зрозуміле і повне представлення функціональності MorseBridge та його здатність покращити комунікацію для людей з глухотою та сліпотою.

Загалом, проведений аналіз якості та тестування підтверджують, що MorseBridge є надійним, продуктивним та зручним у використанні додатком, який відповідає всім заявленим вимогам. Це дозволяє зробити висновок, що додаток готовий до використання та має високу практичну значимість для покращення якості життя користувачів з глухотою та сліпотою. Додаток забезпечує ефективну та незалежну комунікацію, розширюючи можливості взаємодії користувачів з оточуючим світом.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Для розгортання програмного забезпечення MorseBridge використовуються інструменти та сервіси, що забезпечують ефективне управління життєвим циклом додатка, включаючи створення релізів, тестування, публікацію та моніторинг. Основними засобами розгортання є:

1. Xcode – середовище розробки для iOS, яке використовується для створення, тестування та збірки додатка.
2. App Store Connect – платформа для управління додатками в App Store, яка використовується для підготовки та публікації MorseBridge.
3. TestFlight – інструмент для бета-тестування додатків перед їх випуском.
4. Firebase – аналітичний інструмент для моніторингу продуктивності додатка та збирання зворотного зв’язку.

Крок 1: Підготовка до розгортання

1. Налаштування облікового запису розробника
   - Зареєструватись в Apple Developer Program.
   - Налаштувати необхідні сертифікати та профілі підпису для випуску додатка.
2. Створення релізної версії додатка
   - Відкрити проект MorseBridge в Xcode.
   - Переконайтеся, що всі функціональні вимоги виконані, і проведено необхідні тести.
   - Створіти збірку додатка для випуску (release build).
3. Тестування релізної версії
   - Використовувати TestFlight для бета-тестування додатка.

- Запросити бета-тестерів для перевірки додатка на різних моделях iPhone та різних версіях iOS.

Крок 2: Випуск додатка

1. Підготовка до публікації
   - Підготувати опис додатка, скріншоти, відео-презентації та інші матеріали для App Store.
   - Заповніти інформацію про додаток у App Store Connect.
2. Відправка на перевірку
   - Відправити релізну версію додатка на перевірку до App Store через App Store Connect.
   - Внести зміни за результатами перевірки, якщо це необхідно.
3. Публікація
   - Після успішного проходження перевірки опублікуйте додаток у App Store.

Для візуалізації процесу розгортання програмного забезпечення MorseBridge надаються ілюстації:

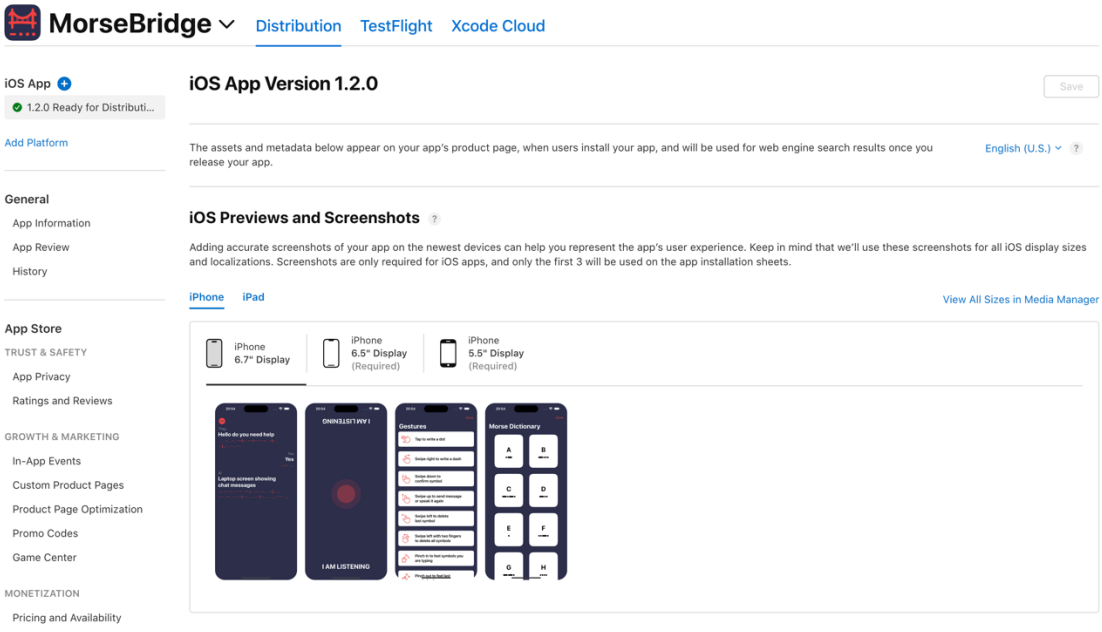


Рисунок 5.1 – Кабінет AppStoreConnect з опублікованими скріншотами додатка

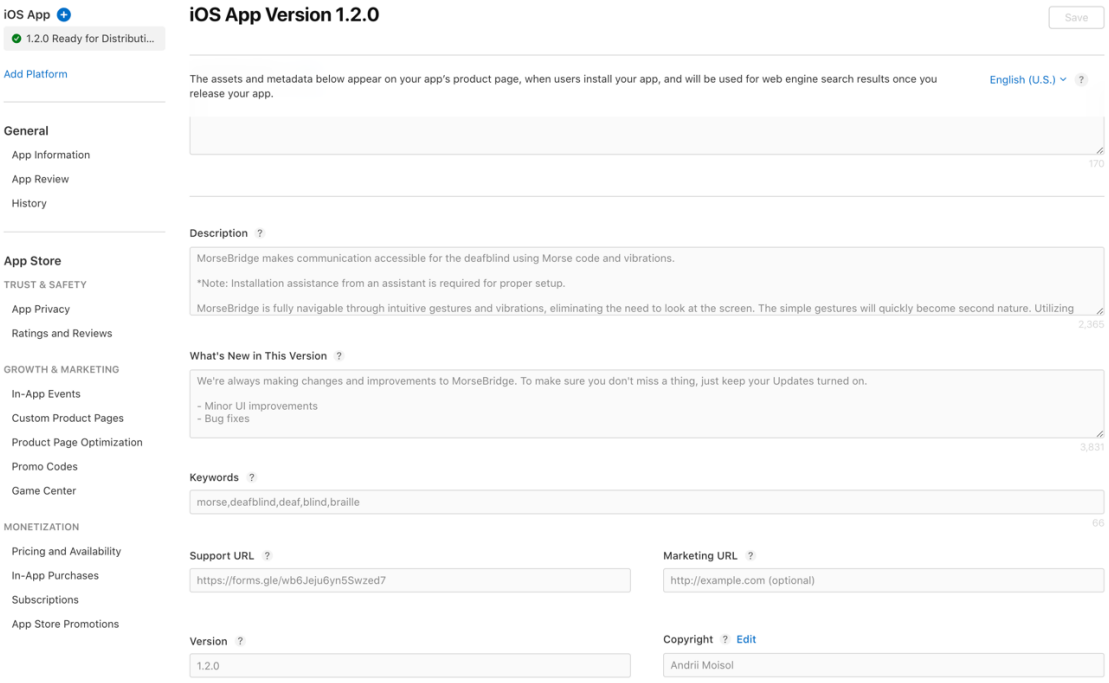


Рисунок 5.2 – Кабінет AppStoreConnect з введиними описами додатка

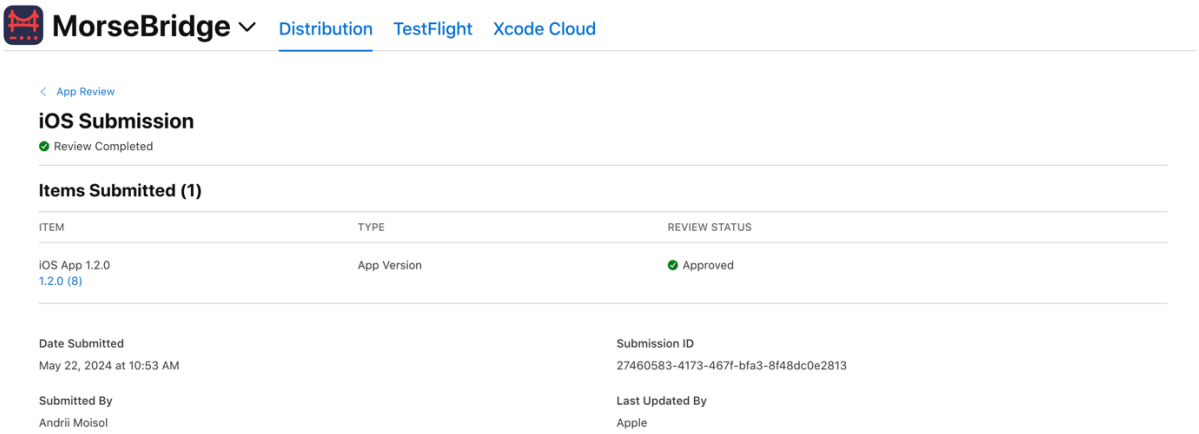


Рисунок 5.3 – Кабінет AppStoreConnect з результатами перевірки перед публікацією

5.2 Супровід програмного забезпечення

Підтримка програмного забезпечення MorseBridge включає кілька важливих аспектів, спрямованих на забезпечення стабільної роботи додатка, швидке реагування на виникаючі проблеми та постійне вдосконалення функціональності. Основні етапи підтримки описані нижче.

1. Моніторинг продуктивності:
   - Використання аналітичних інструментів, таких як Firebase Analytics, для відстеження продуктивності додатка та поведінки користувачів.
   - Регулярний аналіз даних для виявлення можливих проблем та покращення взаємодії користувачів з додатком.
2. Обробка зворотного зв'язку:
   - Збір відгуків користувачів через App Store, соціальні мережі та інші канали.
   - Аналіз зворотного зв'язку для виявлення основних проблем та запитів на нові функції.

3. Випуск оновлень:
   - Регулярне випускання оновлень для виправлення помилок, покращення продуктивності та додавання нових функцій.
   - Використання TestFlight для бета-тестування нових версій перед їх публікацією в App Store.

Висновки до розділу

У даному розділі було детально описано процеси розгортання та супроводу програмного забезпечення MorseBridge. Для забезпечення стабільної роботи та ефективного розгортання додатка було використано низку інструментів і сервісів, таких як Xcode, App Store Connect, TestFlight та Firebase. Описано покроковий процес розгортання, що включає підготовку інфраструктури, створення релізної версії, тестування, підготовку до публікації, відправку на перевірку та публікацію в App Store.

Було розглянуто основні аспекти супроводу програмного забезпечення, включаючи моніторинг продуктивності, обробку зворотного зв'язку, випуск оновлень та надання технічної підтримки. Для автоматизації процесу розгортання нових версій використовується сервіс GitHub Actions, який забезпечує швидке та ефективне впровадження оновлень.

Моніторинг та аналітика дозволяють відстежувати продуктивність додатка та виявляти можливі проблеми, що забезпечує високий рівень стабільності та задоволення користувачів. Обробка зворотного зв'язку допомагає вносити необхідні покращення та додавати нові функції на основі запитів користувачів.

Регулярне випускання оновлень та швидке виправлення помилок гарантують, що MorseBridge залишається актуальним та надійним інструментом для комунікації глухосліпих користувачів. Надання технічної

підтримки через різні канали зв'язку дозволяє оперативно вирішувати проблеми користувачів та підтримувати високий рівень їх задоволеності.

Загалом, виконані процеси розгортання та супроводу забезпечують високу якість програмного забезпечення MorseBridge, його стабільну роботу та постійне вдосконалення, що сприяє покращенню якості життя користувачів та забезпечує ефективну комунікацію для глухосліпих людей.

ВИСНОВКИ

В результаті виконання дипломного проєкту було спроєктовано та розроблено мобільний додаток MorseBridge, призначений для покращення комунікації людей з глухотою та сліпотою. У якості середовища розробки було обрано Xcode, а для створення інтерфейсу користувача використовувалися SwiftUI та UIKit. Розробка додатка виконувалася за архітектурою MVVM+Coordinator, що забезпечило високу гнучкість, підтримуваність та масштабованість.

Додаток MorseBridge був протестований на пристроях з різними версіями iOS, щоб переконатися, що він акуратно відображається та працює стабільно на всіх підтримуваних пристроях. Виконане тестування підтвердило відповідність додатка заявленим функціональним вимогам, його надійність та зручність використання.

Розроблений додаток відповідає сучасному рівню наукових і технічних знань, забезпечуючи ефективну комунікацію для глухосліпих людей за допомогою азбуки Морзе, розпізнавання зображень та голосових команд. Використання сучасних технологій, таких як OpenAI API для розпізнавання зображень та синтезу мови, дозволило створити додаток, який значно покращує якість життя користувачів.

Додаток MorseBridge готовий до впровадження та може бути розповсюджений через App Store для широкого кола користувачів з глухотою та сліпотою. Можливими галузями використання є сфери охорони здоров'я, освіти та соціальної підтримки, де важлива незалежна комунікація та взаємодія з оточуючим світом.

Розроблений додаток має високу науково-технічну та соціально-економічну значущість. Він забезпечує глухосліпих людей новими можливостями для комунікації, що сприяє їхній соціальній інтеграції та

підвищенню якості життя. Використання сучасних технологій у додатку також демонструє можливість ефективного застосування штучного інтелекту для вирішення соціально значущих задач.

Подальші дослідження у відповідній тематиці доцільні, оскільки є можливість розширити функціональність додатка, наприклад, шляхом додавання підтримки інших мов, вдосконалення алгоритмів розпізнавання мови та зображень, а також інтеграції з іншими сервісами для покращення користувацького досвіду.

Усі задачі, поставлені в дипломному проєкті, були успішно вирішені. Було спроєктовано та реалізовано мобільний додаток MorseBridge, забезпечено його стабільну роботу на різних пристроях, проведено детальне тестування та підготовлено додаток до публікації. Розроблений додаток забезпечує ефективну комунікацію для глухосліпих людей, відповідає сучасним стандартам якості та має високу соціальну значущість.

Наукова новизна роботи полягає у вперше реалізованій можливості комплексного використання азбуки Морзе, розпізнавання зображень та голосових команд у мобільному додатку для глухосліпих користувачів, що дозволило значно покращити їхню комунікацію та взаємодію з оточуючим світом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) ATIA | Assistive Technology Industry Association. *Assistive Technology Industry Association*. URL: https://www.atia.org (date of access: 19.05.2024).
2) Be My Eyes - See the world together. *Be My Eyes - See the world together*. URL: https://www.bemyeyes.com/ (date of access: 19.05.2024).
3) Freedom Scientific – High-quality video magnifiers, braille displays, screen magnification software, and #1 screen reader, JAWS. *Freedom Scientific – High-quality video magnifiers, braille displays, screen magnification software, and #1 screen reader, JAWS*. URL: https://www.freedomscientific.com (date of access: 19.05.2024).
4) ITDRC. *ITDRC*. URL: https://www.itdrc.org (date of access: 19.05.2024).
5) ITU: Committed to connecting the world. *ITU*. URL: https://www.itu.int (date of access: 19.05.2024).
6) NIDCD. *NIDCD*. URL: https://www.nidcd.nih.gov (date of access: 19.05.2024).
7) TapTapSee - Blind and Visually Impaired Assistive Technology - powered by CloudSight.ai Image Recognition API. *TapTapSee - Blind and Visually Impaired Assistive Technology - powered by CloudSight.ai Image Recognition API*. URL: http://taptapseeapp.com/ (date of access: 19.05.2024).
8) V7 Aipoly. *V7 Aipoly*. URL: https://aipoly.com/ (date of access: 19.05.2024).
9) Voiceitt - Inclusive Voice AI. *Voiceitt - Inclusive Voice AI*. URL: https://www.voiceitt.com/ (date of access: 19.05.2024).
10) World Blind Union. *World Blind Union*. URL: https://www.worldblindunion.org (date of access: 19.05.2024).

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK
by Turnitin

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016334653

Дата перевірки:
09.06.2024 02:30:46 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
09.06.2024 07:35:00 EEST

ID користувача:
76913

Назва документа: **ІП-02_Мойсол_Андрій_ПЗ**

Кількість сторінок: **51** Кількість слів: **6353** Кількість символів: **49842** Розмір файлу: **1.14 MB** ID файлу: **1016135095**

5.76%
Схожість

Найбільша схожість: **1.87%** з джерелом з Бібліотеки (ID файлу: **1016120532**)

1.5% Джерела з Інтернету	79	Сторінка 53
5.76% Джерела з Бібліотеки	232	Сторінка 53

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	11

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ЗАБЕЗПЕЧЕННЯ КОМУНІКАЦІЇ ЛЮДЕЙ З ГЛУХОТОЮ ТА СЛІПОТОЮ

Текст програми

КПІ.ІП-9614.045430.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Анна ЯСЕНОВА

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Андрій МОЙСОЛ

Київ – 2024

Посилання на репозиторій з повним текстом програмного коду
https://github.com/Arideno/MorseBridge

Файл main.swift

```
//
//  main.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 04.05.2024.
//

import UIKit

let isRunningTests = NSClassFromString("XCTestCase") != nil

let appDelegateClass = isRunningTests ? nil : NSStringFromClass(AppDelegate.self)

UIApplicationMain(CommandLine.argc, CommandLine.unsafeArgv, nil, appDelegateClass)
```

Файл AppCoordinator.swift

```
//
//  AppCoordinator.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 04.05.2024.
//


import UIKit


final class AppCoordinator: Coordinator<Void> {
    private unowned var window: UIWindow
    private let dependencyContainer: AppDependencyContainer

    init(dependencyContainer: AppDependencyContainer, window: UIWindow) {
        self.dependencyContainer = dependencyContainer
        self.window = window
    }
```

```
    override func start() {
        window.makeKeyAndVisible()
    }
}

extension AppCoordinator: AppNavigationDelegate {
    func openMain() {
        let container = dependencyContainer.resolve() as MainDependencyContainerType
        coordinate(to: container.makeCoordinator(window: window))
    }
}
```

Файл AppDelegate.swift

```
//
//  AppDelegate.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 04.05.2024.
//

import UIKit
import FacebookCore

class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow? = UIWindow(frame: UIScreen.main.bounds)
    private lazy var dependencyContainer = AppDependencyContainer.shared
    private lazy var coordinator: AppCoordinator = dependencyContainer.resolve(a1: window!)
    private lazy var viewModel: AppViewModel = dependencyContainer.resolve(a1: coordinator
as AppNavigationDelegate)
    private lazy var firebaseService: FirebaseServiceType = dependencyContainer.resolve()

    func application(
        _ application: UIApplication,
```

```
        didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey:
Any]?
    ) -> Bool {
        firebaseService.configure()

        let navBarAppearance = UINavigationBarAppearance()
        navBarAppearance.backgroundColor = .background
        navBarAppearance.titleTextAttributes = [.foregroundColor: UIColor.white]
        navBarAppearance.largeTitleTextAttributes = [.foregroundColor: UIColor.white]

        UINavigationBar.appearance().scrollEdgeAppearance = navBarAppearance
        UINavigationBar.appearance().standardAppearance = navBarAppearance
        UINavigationBar.appearance().compactAppearance = navBarAppearance
        UINavigationBar.appearance().compactScrollEdgeAppearance = navBarAppearance

        viewModel.appDidFinishLaunching(app: application, launchOptions: launchOptions)

        ApplicationDelegate.shared.application(application, didFinishLaunchingWithOptions:
launchOptions)

        coordinator.start()

        return true
    }

    func application(
        _ app: UIApplication,
        open url: URL,
        options: [UIApplication.OpenURLOptionsKey: Any] = [:]
    ) -> Bool {
        ApplicationDelegate.shared.application(app, open: url, options: options)
    }
}
```

Файл AppViewModel.swift

```
//
//  AppViewModel.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 04.05.2024.
//

import UIKit

final class AppViewModel {
    private weak var navigationDelegate: AppNavigationDelegate?

    init(navigationDelegate: AppNavigationDelegate) {
        self.navigationDelegate = navigationDelegate
    }

    func appDidFinishLaunching(
        app: UIApplication,
        launchOptions: [UIApplication.LaunchOptionsKey: Any]?
    ) {
        navigationDelegate?.openMain()
    }
}

// sourcery: AutoMockable
protocol AppNavigationDelegate: AnyObject {
    func openMain()
}
```

Файл DictionaryCoordinator.swift

```
//
//  DictionaryCoordinator.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 08.05.2024.
//

import UIKit

final class DictionaryCoordinator: Coordinator<Void> {
    private weak var rootViewController: DictionaryViewController!
    private let parentViewController: UIViewController
    private let dependencyContainer: DictionaryDependencyContainerType

    init(
        dependencyContainer: DictionaryDependencyContainerType,
        parentViewController: UIViewController
    ) {
        self.parentViewController = parentViewController
        self.dependencyContainer = dependencyContainer
    }

    override func start() {
        let vc = DictionaryViewController()
        vc.viewModel = dependencyContainer.makeViewModel(navigationDelegate: self)
        rootViewController = vc

        vc.modalPresentationStyle = .fullScreen
        parentViewController.present(vc, animated: true)
    }
}

extension DictionaryCoordinator: DictionaryNavigationDelegate {
    func done() {
```

```
        finish(())
    }

    func close() {
        rootViewController.dismiss(animated: true) { [weak self] in
            self?.finish(())
        }
    }
}
```

Файл DictionaryDependencyContainer.swift

```
//
//  DictionaryDependencyContainer.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 08.05.2024.
//

import UIKit

protocol DictionaryDependencyContainerType {
    func makeViewModel(navigationDelegate: DictionaryNavigationDelegate) ->
DictionaryViewModel
    func makeCoordinator(parent: UIViewController) -> DictionaryCoordinator
}

final class DictionaryDependencyContainer: DependencyContainerFacade,
DictionaryDependencyContainerType {
    func makeViewModel(navigationDelegate: DictionaryNavigationDelegate) ->
DictionaryViewModel {
        resolve(a1: navigationDelegate)
    }

    func makeCoordinator(parent: UIViewController) -> DictionaryCoordinator {
        DictionaryCoordinator(dependencyContainer: self, parentViewController: parent)
```

```
    }
}
```

Файл DictionaryView.swift

```
//
//  DictionaryView.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 08.05.2024.
//

import SwiftUI

struct DictionaryView: View {
    @ObservedObject private(set) var viewModel: DictionaryViewModel

    var body: some View {
        NavigationStack {
            ScrollView {
                LazyVGrid(columns: [GridItem(.fixed(150), spacing: 32), GridItem(.fixed(150))],
spacing: 32) {
                    ForEach(viewModel.elements, id: \.symbol) { element in
                        Button {
                            viewModel.playSymbol(element.symbol)
                        } label: {
                            VStack {
                                Text(String(element.symbol))
                                    .font(.largeTitle)
                                    .bold()
                                    .foregroundStyle(.black)
                                    .padding(.vertical, 8)

                                MorseCodeView(code: element.morseCode)
                            }
                            .frame(width: 150, height: 175)
```

```
                    .background(.white)
                    .cornerRadius(20)
                }
                .disabled(viewModel.isPlaying)
            }
        }
        .padding(.top, 16)
    }
    .background(Color.background)
    .navigationTitle(L10n.MorseDictionary.title)
    .navigationBarTitleDisplayMode(.large)
    .toolbar {
        ToolbarItem(placement: .topBarTrailing) {
            Button(L10n.Common.done, action: viewModel.closeAction)
        }
    }
  }
}

#if PREVIEW
#Preview {
  let viewModel = DictionaryViewModel(navigationDelegate:
DictionaryNavigationDelegateMock())
  return DictionaryView(viewModel: viewModel)
}
#endif
```

Файл DictionaryViewController.swift

```
//
//  DictionaryViewController.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 08.05.2024.
//
```

```
import UIKit

final class DictionaryViewController: UIViewController, ViewModelHolderType {
    typealias ViewModelType = DictionaryViewModel

    override func viewDidLoad() {
        super.viewDidLoad()
        add(DictionaryView(viewModel: viewModel), to: view)
    }
}
```

Файл DictionaryViewModel.swift

```
//
//  DictionaryViewModel.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 08.05.2024.
//

import Combine

final class DictionaryViewModel: ObservableObject {
    // MARK: - Outputs
    @Published private(set) var elements: [DictionaryElement] = []
    @Published private(set) var isPlaying = false

    // MARK: - Services
    private weak var navigationDelegate: DictionaryNavigationDelegate?

    private let dictionary = MorseDictionary.current

    init(navigationDelegate: DictionaryNavigationDelegate) {
        self.navigationDelegate = navigationDelegate
```

```
    let order = dictionary.getOrder()
    elements = dictionary.getMorseCodes().map {
        DictionaryElement(
            symbol: dictionary.getCharacter(for: $0)!,
            morseCode: $0
        )
    }.sorted(by: {
        if $0.symbol.isLetter && $1.symbol.isLetter {
            if let firstIndex = order.firstIndex(of: String($0.symbol)),
               let secondIndex = order.firstIndex(of: String($1.symbol)) {
                return firstIndex < secondIndex
            } else {
                return $0.symbol < $1.symbol
            }
        } else if $0.symbol.isLetter {
            return true
        } else if $1.symbol.isLetter {
            return false
        } else if $0.symbol.isNumber && $1.symbol.isNumber {
            return $0.symbol < $1.symbol
        } else if $0.symbol.isNumber {
            return true
        } else if $1.symbol.isNumber {
            return false
        } else {
            return $0.symbol < $1.symbol
        }
    })
}

deinit {
    navigationDelegate?.done()
}

func closeAction() {
```

```
        navigationDelegate?.close()
    }

    func playSymbol(_ symbol: Character) {
        guard !isPlaying else { return }
        isPlaying = true
        let morseCode = dictionary.morseFromText(String(symbol))
        Haptic.shared.playMorseCode(morseCode) {
            self.isPlaying = false
        }
    }
}

// MARK: - Helpers
struct DictionaryElement {
    let symbol: Character
    let morseCode: MorseCode
}

// sourcery: AutoMockable
protocol DictionaryNavigationDelegate: AnyObject {
    func done()
    func close()
}
```

Файл MorseCodeView.swift

```
//
//  MorseCodeView.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 16.05.2024.
//

import SwiftUI
```

```
struct MorseCodeView: View {
    let code: MorseCode

    var body: some View {
        HStack(spacing: 2) {
            ForEach(code.elements.indices, id: \.self) { index in
                switch code.elements[index] {
                case .dot:
                    Circle()
                        .fill(.black)
                        .frame(width: 8, height: 8)
                case .dash:
                    RoundedRectangle(cornerRadius: 4)
                        .fill(.black)
                        .frame(width: 24, height: 8)
                default: EmptyView()
                }
            }
        }
    }
}
```

Файл InstructionsCoordinator.swift

```
//
//  InstructionsCoordinator.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 17.05.2024.
//

import UIKit

final class InstructionsCoordinator: Coordinator<Void> {
    private weak var rootViewController: InstructionsViewController!
    private let parentViewController: UIViewController
```

```
    private let dependencyContainer: InstructionsDependencyContainerType

    init(
        dependencyContainer: InstructionsDependencyContainerType,
        parentViewController: UIViewController
    ) {
        self.parentViewController = parentViewController
        self.dependencyContainer = dependencyContainer
    }

    override func start() {
        let vc = InstructionsViewController()
        vc.viewModel = dependencyContainer.makeViewModel(navigationDelegate: self)
        rootViewController = vc

        vc.modalPresentationStyle = .fullScreen
        parentViewController.present(vc, animated: true)
    }
}

extension InstructionsCoordinator: InstructionsNavigationDelegate {
    func done() {
        finish(())
    }

    func close() {
        rootViewController.dismiss(animated: true) { [weak self] in
            self?.finish(())
        }
    }
}
```

Файл InstructionsDependencyContainer.swift

```
//
//  InstructionsDependencyContainer.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 17.05.2024.
//

import UIKit

protocol InstructionsDependencyContainerType {
    func makeViewModel(navigationDelegate: InstructionsNavigationDelegate) ->
InstructionsViewModel
    func makeCoordinator(parent: UIViewController) -> InstructionsCoordinator
}

final class InstructionsDependencyContainer: DependencyContainerFacade,
InstructionsDependencyContainerType {
    func makeViewModel(navigationDelegate: InstructionsNavigationDelegate) ->
InstructionsViewModel {
        resolve(a1: navigationDelegate)
    }

    func makeCoordinator(parent: UIViewController) -> InstructionsCoordinator {
        InstructionsCoordinator(dependencyContainer: self, parentViewController: parent)
    }
}
```

Файл InstructionsView.swift

```
//
//  InstructionsView.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 17.05.2024.
```

```
//

import SwiftUI

struct InstructionsView: View {
   @ObservedObject private(set) var viewModel: InstructionsViewModel

   var body: some View {
      NavigationStack {
         ScrollView {
            VStack(spacing: 24) {
               instructionsView(image: .icTap, text: L10n.Instructions.tap)
               instructionsView(image: .icSwipeOneRight, text: L10n.Instructions.swipeRight)
               instructionsView(image: .icSwipeOneDown, text: L10n.Instructions.swipeDown)
               instructionsView(image: .icSwipeOneUp, text: L10n.Instructions.swipeUp)
               instructionsView(image: .icSwipeOneLeft, text: L10n.Instructions.swipeLeft)
               instructionsView(image: .icSwipeTwoLeft, text: L10n.Instructions.swipeTwoLeft)
               instructionsView(image: .icPinchIn, text: L10n.Instructions.pinchIn)
               instructionsView(image: .icPinchOut, text: L10n.Instructions.pinchOut)
               instructionsView(image: .icSwipeTwoUp, text: L10n.Instructions.swipeTwoUp)
               instructionsView(image: .icSwipeTwoDown, text:
L10n.Instructions.swipeTwoDown)
               instructionsView(image: .icHold, text: L10n.Instructions.hold)
               instructionsView(image: .icShake, text: L10n.Instructions.shake)
            }
            .frame(maxWidth: .infinity)
            .padding(.horizontal, 16)
         }
         .background(Color.background)
         .navigationTitle(L10n.Instructions.title)
         .navigationBarTitleDisplayMode(.large)
         .toolbar {
            ToolbarItem(placement: .topBarTrailing) {
               Button(L10n.Common.done, action: viewModel.closeAction)
            }
```

```
            }
        }
    }

    private func instructionsView(image: ImageResource, text: String) -> some View {
        HStack(spacing: 24) {
            Image(image)
                .resizable()
                .frame(width: 48, height: 48)
                .foregroundStyle(.primaryMain)

            Text(text)
                .font(.title3)
                .bold()
                .foregroundStyle(.black)

            Spacer()
        }
        .padding(16)
        .frame(maxWidth: .infinity)
        .background(.white)
        .clipShape(RoundedRectangle(cornerRadius: 10))
    }
}

#if PREVIEW
#Preview {
    let viewModel = InstructionsViewModel(navigationDelegate:
InstructionsNavigationDelegateMock())
    return InstructionsView(viewModel: viewModel)
}
#endif
```

Файл InstructionsViewController.swift

```
//
//  InstructionsViewController.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 17.05.2024.
//

import UIKit

final class InstructionsViewController: UIViewController, ViewModelHolderType {
    typealias ViewModelType = InstructionsViewModel

    override func viewDidLoad() {
        super.viewDidLoad()
        add(InstructionsView(viewModel: viewModel), to: view)
    }
}
```

Файл InstructionsViewModel.swift

```
//
//  InstructionsViewModel.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 17.05.2024.
//

import Combine

final class InstructionsViewModel: ObservableObject {
    // MARK: - Outputs

    // MARK: - Services
    private weak var navigationDelegate: InstructionsNavigationDelegate?
```

```
    init(navigationDelegate: InstructionsNavigationDelegate) {
        self.navigationDelegate = navigationDelegate
    }

    deinit {
        navigationDelegate?.done()
    }

    func closeAction() {
        navigationDelegate?.close()
    }
}

// MARK: - Helpers

// sourcery: AutoMockable
protocol InstructionsNavigationDelegate: AnyObject {
    func done()
    func close()
}
```

Файл MainCoordinator.swift

```
//
//  MainCoordinator.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 05.05.2024.
//

import UIKit

final class MainCoordinator: Coordinator<Void> {
    private weak var rootViewController: MainViewController!
    private unowned var window: UIWindow
```

```
    private let dependencyContainer: MainDependencyContainerType

    init(
        dependencyContainer: MainDependencyContainerType,
        window: UIWindow
    ) {
        self.window = window
        self.dependencyContainer = dependencyContainer
    }

    override func start() {
        let vc = MainViewController()
        vc.viewModel = dependencyContainer.makeViewModel(navigationDelegate: self)
        rootViewController = vc

        window.rootViewController = vc
    }
}

extension MainCoordinator: MainNavigationDelegate {
    func done() {
        finish(())
    }

    func openDictionary() {
        let container = dependencyContainer.makeDictionaryDependencyContainer()
        coordinate(to: container.makeCoordinator(parent: rootViewController))
    }

    func openInstructions() {
        let container = dependencyContainer.makeInstructionsDependencyContainer()
        coordinate(to: container.makeCoordinator(parent: rootViewController))
    }
}
```

Файл MainDependencyContainer.swift

```
//
//  MainDependencyContainer.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 05.05.2024.
//

import UIKit

protocol MainDependencyContainerType {
    func makeViewModel(navigationDelegate: MainNavigationDelegate) -> MainViewModel
    func makeCoordinator(window: UIWindow) -> MainCoordinator
    func makeDictionaryDependencyContainer() -> DictionaryDependencyContainerType
    func makeInstructionsDependencyContainer() -> InstructionsDependencyContainerType
}

final class MainDependencyContainer: DependencyContainerFacade,
MainDependencyContainerType {
    func makeViewModel(navigationDelegate: MainNavigationDelegate) -> MainViewModel {
        resolve(a1: navigationDelegate)
    }

    func makeCoordinator(window: UIWindow) -> MainCoordinator {
        MainCoordinator(dependencyContainer: self, window: window)
    }

    func makeDictionaryDependencyContainer() -> DictionaryDependencyContainerType {
        resolve()
    }

    func makeInstructionsDependencyContainer() -> any InstructionsDependencyContainerType
{
        resolve()
    }
```

```
}
```

Файл MainView.swift

```
//
//  MainView.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 05.05.2024.
//

import SwiftUI

struct MainView: View {
    @ObservedObject private(set) var viewModel: MainViewModel

    var body: some View {
        ZStack(alignment: .trailing) {
            MorsePlaygroundView(
                addElementAction: viewModel.addElementAction,
                confirmAction: viewModel.confirmAction,
                deleteAction: viewModel.deleteAction,
                sendAction: viewModel.sendAction,
                repeatOrStopMyAction: viewModel.repeatOrStopMyAction,
                repeatOrStopTheirAction: viewModel.repeatOrStopTheirAction,
                feelAllAction: viewModel.feelAllAction,
                feelLastLetterAction: viewModel.feelLastLetterAction,
                speakAction: viewModel.speakAction
            )
            .onShake(perform: viewModel.onShakeAction)

            ScrollViewReader { proxy in
                ScrollView {
                    VStack {
                        LazyVStack(spacing: 16) {
                            ForEach(viewModel.sentMessages) { message in
```

```
            HStack {
                if message.sender == .you {
                    Spacer()
                }

                VStack(alignment: message.sender == .you ? .trailing : .leading) {
                    Text(message.sender.title)
                        .font(.title3)
                        .foregroundStyle(.gray)

                    Text(message.textString)
                        .font(.title)
                        .bold()
                        .foregroundStyle(.white)

                    Text(message.morseCode.string)
                        .font(.title2)
                        .foregroundStyle(.primaryMain)
                }

                if message.sender != .you {
                    Spacer()
                }
            }
            .id(message.id)
        }
    }

    Spacer(minLength: 0)
  }
  .padding(.horizontal, 16)
}
.topSafeAreaPadding(adding: 64)
.bottomSafeAreaPadding(adding: 16)
.allowsHitTesting(false)
```

```
    .onReceive(viewModel.scrollToEndPublisher) {
        proxy.scrollTo(viewModel.sentMessages.last?.id, anchor: .top)
    }
}

VStack {
    HStack {
        Menu("", systemImage: "ellipsis.circle.fill") {
            Button(action: viewModel.dictionaryAction) {
                Label(L10n.MorseDictionary.title, systemImage: "text.book.closed.fill")
            }

            Button(action: viewModel.instructionsAction) {
                Label(L10n.Instructions.title, systemImage: "hand.tap")
            }
        }
        .foregroundStyle(.primaryMain)
        .font(.largeTitle)

        Spacer()
    }

    Spacer()
}
.padding(.horizontal, 16)
.topSafeAreaPadding(adding: 16)

if viewModel.isRecognizingSpeech {
    VStack {
        Text(L10n.Common.iAmListening)
            .font(.largeTitle)
            .bold()
            .foregroundStyle(.white)
            .rotationEffect(.radians(.pi))
```

```
                Spacer()

                PulsatingView()

                Spacer()

                Text(L10n.Common.iAmListening)
                    .font(.largeTitle)
                    .bold()
                    .foregroundStyle(.white)
            }
            .topSafeAreaPadding(adding: 16)
            .bottomSafeAreaPadding(adding: 16)
            .frame(maxWidth: .infinity, maxHeight: .infinity)
            .background(Color.background)
        }
    }
    .ignoresSafeArea()
  }
}

#if PREVIEW
#Preview {
  let viewModel = MainViewModel(
    navigationDelegate: MainNavigationDelegateMock(),
    cameraService: CameraServiceTypeMock(),
    openAIService: OpenAIServiceTypeMock(),
    speechRecognitionService: SpeechRecognitionServiceTypeMock(),
    speechSynthesizerService: SpeechSynthesizerServiceTypeMock()
  )
  return MainView(viewModel: viewModel)
}
#endif
```

Файл MainViewController.swift

```
//
//  MainViewController.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 05.05.2024.
//

import UIKit

final class MainViewController: UIViewController, ViewModelHolderType {
    typealias ViewModelType = MainViewModel

    override func viewDidLoad() {
        super.viewDidLoad()
        add(MainView(viewModel: viewModel), to: view)
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)

        viewModel.viewDidAppear()
    }
}
```

Файл MainViewModel.swift

```
//
//  MainViewModel.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 05.05.2024.
//

import UIKit
```

```
import Combine

// swiftlint:disable type_body_length
final class MainViewModel: ObservableObject, SubscriptionContainable {
    // MARK: - Outputs
    @Published private(set) var sentMessages = [Message]()
    @Published private(set) var isRecognizingSpeech = false

    private(set) lazy var scrollToEndPublisher =
scrollToEndSubject.onMain().eraseToAnyPublisher()

    // MARK: - Services
    private weak var navigationDelegate: MainNavigationDelegate?
    private let cameraService: CameraServiceType
    private let openAIService: OpenAIServiceType
    private let speechRecognitionService: SpeechRecognitionServiceType
    private let speechSynthesizerService: SpeechSynthesizerServiceType
    private let notificationCenter: NotificationCenter

    private var currentBuffer = MorseCode()
    private var currentCode = MorseCode()
    private let dictionary = MorseDictionary.current
    private var isCapturingImage = false
    private var isPlayingMy = false
    private var isPlayingTheirs = false
    private var isSynthesizing = false
    private let scrollToEndSubject = PassthroughSubject<Void, Never>()

    private var canInteract: Bool {
        !isCapturingImage && !isPlayingMy && !isPlayingTheirs && !isSynthesizing
    }

    init(
        navigationDelegate: MainNavigationDelegate,
        cameraService: CameraServiceType,
```

```
    openAIService: OpenAIServiceType,
    speechRecognitionService: SpeechRecognitionServiceType,
    speechSynthesizerService: SpeechSynthesizerServiceType,
    notificationCenter: NotificationCenter = .default
) {
    self.navigationDelegate = navigationDelegate
    self.cameraService = cameraService
    self.openAIService = openAIService
    self.speechRecognitionService = speechRecognitionService
    self.speechSynthesizerService = speechSynthesizerService
    self.notificationCenter = notificationCenter

    notificationCenter.publisher(for: UIApplication.didBecomeActiveNotification)
        .weak(self) { $0.viewDidAppear }
}

deinit {
    navigationDelegate?.done()
}

func viewDidAppear() {
    Haptic.shared.playPlayground()
}

func addElementAction(_ element: MorseCodeElement) {
    guard canInteract else { return }
    currentBuffer.add(element)
    if element == .dot {
        Haptic.shared.playDot()
    } else if element == .dash {
        Haptic.shared.playDash()
    }
}

func confirmAction() {
```

```
    guard canInteract else { return }
    if dictionary.getCharacter(for: currentBuffer) != nil {
        if !currentCode.isEmpty && currentCode.last != .pipe {
            currentCode.add(.space)
        }
        currentCode.add(currentBuffer)
        currentBuffer.clear()
        Haptic.shared.play(.success)
    } else {
        currentBuffer.clear()
        Haptic.shared.play(.error)
    }
}

func deleteAction(all: Bool) {
    guard canInteract else { return }
    guard !currentCode.isEmpty || !currentBuffer.isEmpty else {
        Haptic.shared.play(.error)
        return
    }

    if all {
        currentCode.clear()
    } else {
        while !currentCode.isEmpty && currentCode.last != .space {
            currentCode.removeLast()
        }
    }
    currentBuffer.clear()

    Haptic.shared.play(.success)
}

func sendAction() {
    guard canInteract else { return }
```

```
        if currentCode.isEmpty {
            if let lastMessage = sentMessages.last(where: { $0.sender == .you }) {
                synthesizeMessage(lastMessage)
            } else {
                Haptic.shared.play(.error)
            }
        } else {
            let translatedString = dictionary.textFromMorse(currentCode).capitalized
            let message = Message(textString: translatedString, morseCode: currentCode,
sender: .you)
            addMessage(message)
            synthesizeMessage(message)
            currentCode.clear()
            currentBuffer.clear()
            Haptic.shared.play(.success)
        }
    }

    func dictionaryAction() {
        guard canInteract else { return }
        navigationDelegate?.openDictionary()
    }

    func onShakeAction() {
        guard canInteract else { return }
        isCapturingImage = true
        Haptic.shared.play(.heavy)
        DispatchQueue.main.asyncAfter(deadline: .now() + 1) {
            Haptic.shared.play(.heavy)
        }
        DispatchQueue.main.asyncAfter(deadline: .now() + 2) {
            Haptic.shared.play(.heavy)
            self.cameraService.capturePhoto()
                .mapToResult()
```

```
            .onMain()
            .weak(self) { $0.handleImageCapture }
    }
}

func repeatOrStopMyAction() {
    if isPlayingMy {
        isPlayingMy = false
        Haptic.shared.stopPlayingCode()
    } else if let lastMessage = sentMessages.last(where: { $0.sender == .you }), canInteract {
        isPlayingMy = true
        playHapticForMessage(lastMessage) {
            self.isPlayingMy = false
        }
    } else {
        Haptic.shared.play(.error)
    }
}

func repeatOrStopTheirAction() {
    if isPlayingTheirs {
        isPlayingTheirs = false
        Haptic.shared.stopPlayingCode()
    } else if let lastMessage = sentMessages.last(where: { $0.sender != .you }), canInteract {
        isPlayingTheirs = true
        playHapticForMessage(lastMessage) {
            self.isPlayingTheirs = false
        }
    } else {
        Haptic.shared.play(.error)
    }
}

func feelAllAction() {
    guard canInteract else { return }
```

```
    if !currentCode.isEmpty {
        isPlayingMy = true
        Haptic.shared.playMorseCode(currentCode) {
            self.isPlayingMy = false
        }
    } else {
        Haptic.shared.play(.error)
    }
}

func feelLastLetterAction() {
    guard canInteract else { return }
    let lastLetter = currentCode.lastLetter
    if !lastLetter.isEmpty {
        isPlayingMy = true
        Haptic.shared.playMorseCode(lastLetter) {
            self.isPlayingMy = false
        }
    } else {
        Haptic.shared.play(.error)
    }
}

func speakAction() {
    guard canInteract else { return }
    speechRecognitionService.requestPermissions()
        .onMain()
        .weak(self) { $0.handleSpeechRecognitionPermissions }
}

func instructionsAction() {
    guard canInteract else { return }
    navigationDelegate?.openInstructions()
}
```

```
private func handleSpeechRecognitionPermissions(isAuthorized: Bool) {
    guard isAuthorized else {
        Haptic.shared.play(.error)
        return
    }

    isRecognizingSpeech = true
    Haptic.shared.playRecognition()
    speechRecognitionService.startRecognition()
        .mapToResult()
        .onMain()
        .weak(self) { $0.handleSpeechRecognition }
}

func handleSpeechRecognition(result: Result<String, Error>) {
    isRecognizingSpeech = false
    Haptic.shared.stopRecognition()
    switch result {
    case .success(let text):
        let morseCode = dictionary.morseFromText(text)
        let message = Message(textString: text, morseCode: morseCode, sender: .they)
        addMessage(message)
        DispatchQueue.main.asyncAfter(deadline: .now() + 1) {
            self.isPlayingTheirs = true
            self.playHapticForMessage(message) {
                self.isPlayingTheirs = false
            }
        }
    case .failure:
        Haptic.shared.play(.error)
    }
}

private func handleImageCapture(result: Result<UIImage, Error>) {
    switch result {
```

```
        case .success(let image):
            openAIService.describeImage(image)
                .mapToResult()
                .onMain()
                .weak(self) { $0.handleDescribeImage }
        case .failure:
            isCapturingImage = false
            Haptic.shared.play(.error)
        }
    }

    private func handleDescribeImage(result: Result<String, Error>) {
        isCapturingImage = false
        switch result {
        case .success(let description):
            Haptic.shared.play(.success)
            let morseDescription = dictionary.morseFromText(description)
            let message = Message(textString: description, morseCode: morseDescription,
sender: .ai)
            addMessage(message)
            self.isPlayingTheirs = true
            playHapticForMessage(message) {
                self.isPlayingTheirs = false
            }
        case .failure: Haptic.shared.play(.error)
        }
    }

    private func synthesizeMessage(_ message: Message) {
        isSynthesizing = true
        speechSynthesizerService.speak(text: message.textString)
            .onMain()
            .sink { [weak self] in
                self?.isSynthesizing = false
                Haptic.shared.play(.success)
```

```
            }
            .store(in: &subscriptions)
    }

    private func playHapticForMessage(_ message: Message, completion: @escaping () -> Void)
{
        Haptic.shared.playMorseCode(message.morseCode, completion: completion)
    }

    private func addMessage(_ message: Message) {
        sentMessages.append(message)
        scrollToEndSubject.send()
    }
}

// MARK: - Helpers

// sourcery: AutoMockable
protocol MainNavigationDelegate: AnyObject {
    func done()
    func openDictionary()
    func openInstructions()
}

struct Message: Identifiable, Equatable {
    let id = UUID()

    let textString: String
    let morseCode: MorseCode
    let sender: MessageSender
}

enum MessageSender {
    case you
    case they
```

```
    case ai

    var title: String {
        switch self {
        case .you: L10n.Common.you
        case .they: L10n.Common.they
        case .ai: L10n.Common.ai
        }
    }
}
```

Файл MorsePlaygroudView.swift

```
//
//  MorsePlaygroundView.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 06.05.2024.
//

import SwiftUI

struct MorsePlaygroundView: UIViewRepresentable {
    let addElementAction: (MorseCodeElement) -> Void
    let confirmAction: () -> Void
    let deleteAction: (Bool) -> Void
    let sendAction: () -> Void
    let repeatOrStopMyAction: () -> Void
    let repeatOrStopTheirAction: () -> Void
    let feelAllAction: () -> Void
    let feelLastLetterAction: () -> Void
    let speakAction: () -> Void

    func makeUIView(context: Context) -> MorsePlaygroundUIView {
        let view = MorsePlaygroundUIView()
        view.addElementAction = addElementAction
```

```
        view.confirmAction = confirmAction
        view.deleteAction = deleteAction
        view.sendAction = sendAction
        view.repeatOrStopMyAction = repeatOrStopMyAction
        view.repeatOrStopTheirAction = repeatOrStopTheirAction
        view.feelAllAction = feelAllAction
        view.feelLastLetterAction = feelLastLetterAction
        view.speakAction = speakAction
        return view
    }

    func updateUIView(_ uiView: MorsePlaygroundUIView, context: Context) {
        uiView.addElementAction = addElementAction
        uiView.confirmAction = confirmAction
        uiView.deleteAction = deleteAction
        uiView.sendAction = sendAction
        uiView.repeatOrStopMyAction = repeatOrStopMyAction
        uiView.repeatOrStopTheirAction = repeatOrStopTheirAction
        uiView.feelAllAction = feelAllAction
        uiView.feelLastLetterAction = feelLastLetterAction
        uiView.speakAction = speakAction
    }
}

final class MorsePlaygroundUIView: UIView {
    private let maxTranslation: CGFloat = 50

    var addElementAction: ((MorseCodeElement) -> Void)?
    var confirmAction: (() -> Void)?
    var deleteAction: ((Bool) -> Void)?
    var sendAction: (() -> Void)?
    var repeatOrStopMyAction: (() -> Void)?
    var repeatOrStopTheirAction: (() -> Void)?
    var feelAllAction: (() -> Void)?
    var feelLastLetterAction: (() -> Void)?
```

```
    var speakAction: (() -> Void)?

    override init(frame: CGRect) {
        super.init(frame: frame)
        backgroundColor = .background
        setupGestures()
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    private func setupGestures() {
        let tapGesture = UITapGestureRecognizer(target: self, action: #selector(handleTap(_:)))
        let panGesture = UIPanGestureRecognizer(target: self, action: #selector(handlePan(_:)))
        panGesture.maximumNumberOfTouches = 2
        let pinchGesture = UIPinchGestureRecognizer(target: self, action:
#selector(handlePinch(_:)))
        let longPressGesture = UILongPressGestureRecognizer(target: self, action:
#selector(handleLongPress(_:)))
        longPressGesture.minimumPressDuration = 1

        addGestureRecognizer(tapGesture)
        addGestureRecognizer(panGesture)
        addGestureRecognizer(pinchGesture)
        addGestureRecognizer(longPressGesture)
    }

    @objc private func handleTap(_ gesture: UITapGestureRecognizer) {
        let location = gesture.location(in: self)
        addElement(.dot, at: location)
        addElementAction?(.dot)
    }

    // swiftlint:disable:next cyclomatic_complexity
```

```
    @objc private func handlePan(_ gesture: UIPanGestureRecognizer) {
        let location = gesture.location(in: self)
        let translation = gesture.translation(in: self)
        let horizontalTranslation = translation.x
        let verticalTranslation = translation.y

        let isComplete = abs(horizontalTranslation) >= maxTranslation || abs(verticalTranslation)
>= maxTranslation
        let direction: Direction

        if abs(horizontalTranslation) > abs(verticalTranslation) {
            direction = horizontalTranslation > 0 ? .right : .left
        } else {
            direction = verticalTranslation > 0 ? .down : .up
        }

        let numberOfTouches = gesture.numberOfTouches

        if isComplete {
            gesture.setTranslation(.zero, in: self)
            gesture.isEnabled = false
            gesture.isEnabled = true

            switch direction {
            case .right:
                if numberOfTouches == 1 {
                    addElement(.dash, at: location)
                    addElementAction?(.dash)
                }
            case .down:
                if numberOfTouches == 1 {
                    confirmAction?()
                } else if numberOfTouches == 2 {
                    repeatOrStopTheirAction?()
                }
```

```
        case .left:
            deleteAction?(numberOfTouches == 2)
        case .up:
            if numberOfTouches == 1 {
                sendAction?()
            } else if numberOfTouches == 2 {
                repeatOrStopMyAction?()
            }
        }
    }
}

@objc private func handlePinch(_ gesture: UIPinchGestureRecognizer) {
    if gesture.state == .ended {
        if gesture.scale < 1 {
            feelAllAction?()
        } else {
            feelLastLetterAction?()
        }
    }
}

@objc private func handleLongPress(_ gesture: UILongPressGestureRecognizer) {
    if gesture.state == .began {
        speakAction?()
    }
}

private func addElement(_ element: MorseCodeElement, at location: CGPoint) {
    let shapeView = MorseSignView(element: element)
    shapeView.center = location
    shapeView.alpha = 0.0

    addSubview(shapeView)
```

```
        UIView.animate(withDuration: 0.1, animations: {
            shapeView.alpha = 1.0
        }, completion: { _ in
            Timer.scheduledTimer(withTimeInterval: 0.1, repeats: false) { _ in
                UIView.animate(withDuration: 0.1, animations: {
                    shapeView.alpha = 0.0
                }, completion: { _ in
                    shapeView.removeFromSuperview()
                })
            }
        })
    }
}

final class MorseSignView: UIView {
    private let dotSize = CGSize(width: 80, height: 80)
    private let dashSize = CGSize(width: 100, height: 50)

    init(element: MorseCodeElement) {
        super.init(frame: .init(origin: .zero, size: (element == .dot) ? dotSize : dashSize))

        backgroundColor = .white
        layer.masksToBounds = true
        layer.cornerRadius = bounds.height / 2
        isUserInteractionEnabled = false
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
}

enum Direction {
    case left, right, up, down
}
```

Файл PulsatingView.swift

```
//
//  PulsatingView.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 16.05.2024.
//

import SwiftUI

struct PulsatingView: View {
    @State private var animate = false

    var body: some View {
        ZStack {
            Circle().fill(Color.secondaryMain.opacity(0.25)).frame(width: 160, height:
160).scaleEffect(animate ? 1 : 0)
            Circle().fill(Color.primaryMain.opacity(0.35)).frame(width: 120, height:
120).scaleEffect(animate ? 0.8 : 1)
        }
        .onAppear { animate = true }
        .animation(
            animate ? .easeInOut(duration: 1.5).repeatForever(autoreverses: true) : .default,
            value: animate
        )
    }
}
```

Файл CameraService.swift

```
//
//  CameraService.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 14.05.2024.
```

```
//

import UIKit
import Combine
import AVFoundation

// sourcery: AutoMockable
protocol CameraServiceType {
    func capturePhoto() -> AnyPublisher<UIImage, Error>
}

// Define CameraServiceError
enum CameraServiceError: Error {
    case permissionDenied
    case captureFailed
}

final class CameraService: NSObject, CameraServiceType {
    private var captureSession: AVCaptureSession?
    private var photoOutput: AVCapturePhotoOutput?
    private var photoCaptureCompletion: PassthroughSubject<UIImage, Error>?

    override init() {
        super.init()
        setupSession()
    }

    private func setupSession() {
        captureSession = AVCaptureSession()
        captureSession?.sessionPreset = .photo
        photoOutput = AVCapturePhotoOutput()

        guard let captureSession = captureSession, let photoOutput = photoOutput else {
            return
        }
```

```
        if let backCamera = AVCaptureDevice.default(for: .video) {
            do {
                let input = try AVCaptureDeviceInput(device: backCamera)
                if captureSession.canAddInput(input) {
                    captureSession.addInput(input)
                }
                if captureSession.canAddOutput(photoOutput) {
                    captureSession.addOutput(photoOutput)
                }
            } catch {
                print("Error setting up camera input: \(error)")
            }
        }
    }

    func capturePhoto() -> AnyPublisher<UIImage, Error> {
        photoCaptureCompletion = PassthroughSubject<UIImage, Error>()

        DispatchQueue.global(qos: .userInitiated).async {
            switch AVCaptureDevice.authorizationStatus(for: .video) {
            case .authorized:
                self.startCapture()
            case .notDetermined:
                AVCaptureDevice.requestAccess(for: .video) { granted in
                    if granted {
                        self.startCapture()
                    } else {

self.photoCaptureCompletion?.send(completion: .failure(CameraServiceError.permissionDenied
))
                    }
                }
            case .denied, .restricted:
```

```
self.photoCaptureCompletion?.send(completion: .failure(CameraServiceError.permissionDenied
))
            @unknown default:

self.photoCaptureCompletion?.send(completion: .failure(CameraServiceError.permissionDenied
))
            }
        }

        return photoCaptureCompletion!.onMain().eraseToAnyPublisher()
    }

    private func startCapture() {
        captureSession?.startRunning()
        let settings = AVCapturePhotoSettings()
        photoOutput?.capturePhoto(with: settings, delegate: self)
    }
}

// Extend CameraService to conform to AVCapturePhotoCaptureDelegate
extension CameraService: AVCapturePhotoCaptureDelegate {
    func photoOutput(_ output: AVCapturePhotoOutput, didFinishProcessingPhoto photo:
AVCapturePhoto, error: Error?) {
        captureSession?.stopRunning()

        if let error = error {
            photoCaptureCompletion?.send(completion: .failure(error))
        } else if let data = photo.fileDataRepresentation(), let image = UIImage(data: data) {
            photoCaptureCompletion?.send(image)
            photoCaptureCompletion?.send(completion: .finished)
        } else {
            photoCaptureCompletion?.send(completion: .failure(CameraServiceError.captureFailed))
        }
    }
```

```
}
```

Файл FirebaseService.swift

```
//
//  FirebaseService.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 04.05.2024.
//

import Foundation
import FirebaseCore
import FirebaseCrashlytics

protocol FirebaseServiceType: AnyObject {
    func configure()
}

final class FirebaseService: FirebaseServiceType {
    func configure() {
        guard let file = Bundle.main.path(forResource: BuildConfiguration.googleServiceInfo,
ofType: "plist"),
              let options = FirebaseOptions(contentsOfFile: file) else { fatalError("Can't create
firebase options") }
        FirebaseApp.configure(options: options)

        Crashlytics.crashlytics().setCrashlyticsCollectionEnabled(BuildConfiguration.current
== .production)
    }
}
```

Файл Haptic.swift

```
//
//  Haptic.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 06.05.2024.
//

import UIKit
import CoreHaptics
import AVFoundation
import Combine

final class Haptic: SubscriptionContainable {
    static let shared = Haptic()

    private let notificationCenter = NotificationCenter.default

    private var engine: CHHapticEngine?
    private let audioSession = AVAudioSession.sharedInstance()
    private var dotPatternPlayer: CHHapticPatternPlayer?
    private var dashPatternPlayer: CHHapticPatternPlayer?
    private var playgroundPatternPlayer: CHHapticPatternPlayer?
    private var recognitionPatternPlayer: CHHapticAdvancedPatternPlayer?
    private var codePlayer: CHHapticAdvancedPatternPlayer?

    private let timePerUnit = 0.1
    private let intraCharacterSpaceUnits = 2.0
    private let interCharacterSpaceUnits = 4.0
    private let wordSpaceUnits = 7.0
    private let dotUnits = 1.0
    private let dashUnits = 3.0

    init() {
        setupAudioSession()
```

```
    engine = try? CHHapticEngine(audioSession: audioSession)

    engine?.resetHandler = startEngine

    startEngine()

    notificationCenter.publisher(for: AVAudioSession.interruptionNotification)
        .weak(self) { $0.handleInterruption }
}

func playDot() {
    do {
        try dotPatternPlayer?.start(atTime: CHHapticTimeImmediate)
    } catch {
        print(error.localizedDescription)
    }
}

func playDash() {
    do {
        try dashPatternPlayer?.start(atTime: CHHapticTimeImmediate)
    } catch {
        print(error.localizedDescription)
    }
}

func playPlayground() {
    do {
        try playgroundPatternPlayer?.start(atTime: CHHapticTimeImmediate)
    } catch {
        print(error.localizedDescription)
    }
}

func playRecognition() {
```

```
    try? recognitionPatternPlayer?.start(atTime: CHHapticTimeImmediate)
}

func play(_ feedbackStyle: UIImpactFeedbackGenerator.FeedbackStyle) {
    UIImpactFeedbackGenerator(style: feedbackStyle).impactOccurred()
}

func play(_ feedbackStyle: UINotificationFeedbackGenerator.FeedbackType) {
    UINotificationFeedbackGenerator().notificationOccurred(feedbackStyle)
}

func playSelectionChanged() {
    UISelectionFeedbackGenerator().selectionChanged()
}

func playMorseCode(_ code: MorseCode, completion: @escaping () -> Void) {
    var events = [CHHapticEvent]()
    var currentUnits = 0.0

    for element in code.elements {
        switch element {
        case .dot:
            events.append(contentsOf: dotEvents(currentTime: currentUnits * timePerUnit))
            currentUnits += dotUnits
            currentUnits += intraCharacterSpaceUnits
        case .dash:
            events.append(contentsOf: dashEvents(currentTime: currentUnits * timePerUnit))
            currentUnits += dashUnits
            currentUnits += intraCharacterSpaceUnits
        case .space:
            currentUnits += interCharacterSpaceUnits
        case .pipe:
            currentUnits += wordSpaceUnits
        }
    }
```

```
    guard let pattern = try? CHHapticPattern(events: events, parameters: []) else { return }

    codePlayer = try? engine?.makeAdvancedPlayer(with: pattern)
    codePlayer?.completionHandler = { _ in
        DispatchQueue.main.async {
            completion()
        }
    }
    try? codePlayer?.start(atTime: CHHapticTimeImmediate)
}

func stopPlayingCode() {
    try? codePlayer?.stop(atTime: CHHapticTimeImmediate)
}

func stopRecognition() {
    try? recognitionPatternPlayer?.stop(atTime: CHHapticTimeImmediate)
}

private func handleInterruption(notification: Notification) {
    guard let userInfo = notification.userInfo,
          let typeValue = userInfo[AVAudioSessionInterruptionTypeKey] as? UInt,
          let type = AVAudioSession.InterruptionType(rawValue: typeValue) else {
        return
    }

    switch type {
    case .began: engine?.stop()
    case .ended: startEngine()
    @unknown default: break
    }
}

private func startEngine() {
```

```
        do {
            try engine?.start()
        } catch {
            print("Failed to start the engine: \(error)")
        }

        setupPlayers()
    }

    private func setupAudioSession() {
        do {
            try audioSession.setCategory(
                .playAndRecord,
                mode: .measurement,
                options: [
                    .defaultToSpeaker,
                    .mixWithOthers
                ]
            )
            try audioSession.setAllowHapticsAndSystemSoundsDuringRecording(true)
            try audioSession.setActive(true)
            print("Audio session set up successfully")
        } catch {
            print("Failed to set up audio session: \(error)")
        }
    }

    private func setupPlayers() {
        do {
            let dotPattern = try CHHapticPattern(events: dotEvents(currentTime: 0), parameters: [])
            let dashPattern = try CHHapticPattern(events: dashEvents(currentTime: 0), parameters:
[])
            let playgroundPattern = try pattern(for: "playground")
            let recognitionPattern = try CHHapticPattern(events: recognitionEvents(), parameters: [])
```

```
        dotPatternPlayer = try engine?.makePlayer(with: dotPattern)
        dashPatternPlayer = try engine?.makePlayer(with: dashPattern)
        playgroundPatternPlayer = try engine?.makePlayer(with: playgroundPattern)
        recognitionPatternPlayer = try engine?.makeAdvancedPlayer(with: recognitionPattern)
        recognitionPatternPlayer?.loopEnabled = true
        recognitionPatternPlayer?.loopEnd = 0
    } catch {
        print(error.localizedDescription)
    }
}

private func pattern(for name: String) throws -> CHHapticPattern {
    try CHHapticPattern(
        contentsOf: Bundle.main.url(forResource: name, withExtension: "ahap")!
    )
}

private func dotEvents(currentTime: TimeInterval) -> [CHHapticEvent] {
    [
        CHHapticEvent(
            eventType: .hapticContinuous,
            parameters: [
                CHHapticEventParameter(parameterID: .hapticSharpness, value: 0.6),
                CHHapticEventParameter(parameterID: .hapticIntensity, value: 0.9)
            ],
            relativeTime: currentTime,
            duration: 0.02
        ),
        CHHapticEvent(
            eventType: .audioContinuous,
            parameters: [
                CHHapticEventParameter(parameterID: .audioPitch, value: 0.4),
                CHHapticEventParameter(parameterID: .audioVolume, value: 1),
                CHHapticEventParameter(parameterID: .audioPan, value: 0)
            ],
```

```
            relativeTime: currentTime,
            duration: dotUnits * timePerUnit
        )
    ]
}

private func dashEvents(currentTime: TimeInterval) -> [CHHapticEvent] {
    [
        CHHapticEvent(
            eventType: .hapticContinuous,
            parameters: [
                CHHapticEventParameter(parameterID: .hapticSharpness, value: 0.6),
                CHHapticEventParameter(parameterID: .hapticIntensity, value: 0.6)
            ],
            relativeTime: currentTime,
            duration: dashUnits * timePerUnit
        ),
        CHHapticEvent(
            eventType: .audioContinuous,
            parameters: [
                CHHapticEventParameter(parameterID: .audioPitch, value: 0.4),
                CHHapticEventParameter(parameterID: .audioVolume, value: 1),
                CHHapticEventParameter(parameterID: .audioPan, value: 0)
            ],
            relativeTime: currentTime,
            duration: dashUnits * timePerUnit
        )
    ]
}

private func recognitionEvents() -> [CHHapticEvent] {
    [
        CHHapticEvent(
            eventType: .hapticContinuous,
            parameters: [
```

```
                CHHapticEventParameter(parameterID: .hapticSharpness, value: 0.6),
                CHHapticEventParameter(parameterID: .hapticIntensity, value: 0.6)
            ],
            relativeTime: 0,
            duration: 1
        )
    ]
  }
}
```

Файл OpenAIService.swift

```
//
//  OpenAIService.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 14.05.2024.
//

import UIKit
import Combine

// sourcery: AutoMockable
protocol OpenAIServiceType {
  func describeImage(_ image: UIImage) -> AnyPublisher<String, Error>
}

final class OpenAIService: OpenAIServiceType {
  func describeImage(_ image: UIImage) -> AnyPublisher<String, Error> {
    Publishers.Task { [weak self] promise in
      guard let self else { return }

      guard let base64Image = convertImageToBase64String(image: image) else {
        promise(.failure(OpenAIServiceError.failedToConvertImageToBase64))
        return
      }
```

```
        do {
            var request = URLRequest(
                url: URL(
                    string: "https://europe-west4-morsebridge.cloudfunctions.net/describe_image"
                )!
            )
            request.allHTTPHeaderFields = ["Content-Type": "application/json"]
            request.httpMethod = "POST"
            request.httpBody = try JSONSerialization.data(
                withJSONObject: [
                    "image": "\(base64Image)",
                    "language_code": MorseDictionary.current.language.rawValue
                ]
            )

            let (data, _) = try await URLSession.shared.data(for: request)
            let jsonData = try JSONSerialization.jsonObject(with: data) as? [String: Any]

            guard let description = jsonData?["description"] as? String else {
                promise(.failure(OpenAIServiceError.failedRequest))
                return
            }

            promise(.success(description))
        } catch {
            print(error.localizedDescription)
            promise(.failure(error))
        }
    }
    .onMain()
}

private func convertImageToBase64String(image: UIImage) -> String? {
    guard let resizedImage = resizeImage(image, scaleFactor: 0.3) else {
```

```
            return nil
        }
        guard let imageData = resizedImage.jpegData(compressionQuality: 0.8) else {
            return nil
        }
        let base64String = imageData.base64EncodedString()
        return base64String
    }

    private func resizeImage(_ image: UIImage, scaleFactor: CGFloat) -> UIImage? {
        let newSize = CGSize(width: image.size.width * scaleFactor, height: image.size.height *
scaleFactor)
        let rect = CGRect(origin: .zero, size: newSize)

        UIGraphicsBeginImageContextWithOptions(newSize, false, 1.0)
        image.draw(in: rect)
        let newImage = UIGraphicsGetImageFromCurrentImageContext()
        UIGraphicsEndImageContext()

        return newImage
    }
}

enum OpenAIServiceError: Error {
    case failedToConvertImageToBase64
    case failedRequest
}
```

Файл SpeechRecognitionService.swift

```
//
//  SpeechRecognitionService.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 16.05.2024.
//
```

```
import Foundation
import Combine
import Speech

// sourcery: AutoMockable
protocol SpeechRecognitionServiceType {
    func requestPermissions() -> AnyPublisher<Bool, Never>
    func startRecognition() -> AnyPublisher<String, Error>
}

final class SpeechRecognitionService: SpeechRecognitionServiceType {
    private var timerSub: AnyCancellable?
    private var resultSubject: PassthroughSubject<String, Error>?
    private var bestTranscription: String?
    private let audioEngine: AVAudioEngine
    private var task: SFSpeechRecognitionTask?
    private var recognizer: SFSpeechRecognizer?
    private var request: SFSpeechAudioBufferRecognitionRequest?

    init() {
        audioEngine = AVAudioEngine()
    }

    func requestPermissions() -> AnyPublisher<Bool, Never> {
        Future { promise in
            SFSpeechRecognizer.requestAuthorization { status in
                promise(.success(status == .authorized))
            }
        }
        .eraseToAnyPublisher()
    }

    func startRecognition() -> AnyPublisher<String, Error> {
        timerSub?.cancel()
```

```
        resultSubject = PassthroughSubject<String, Error>()

        recognizer = SFSpeechRecognizer(locale: Locale.current)

        if recognizer?.isAvailable == false {
            recognizer = SFSpeechRecognizer(locale: Locale(identifier: "en-US"))
        }

        recognizer?.defaultTaskHint = .dictation
        request = SFSpeechAudioBufferRecognitionRequest()

        let node = audioEngine.inputNode
        let recordingFormat = node.outputFormat(forBus: 0)

        node.installTap(onBus: 0, bufferSize: 1024, format: recordingFormat) { buffer, _ in
            self.request!.append(buffer)
        }

        audioEngine.prepare()

        do {
            try audioEngine.start()
        } catch {
            return Fail(error:
SpeechRecognitionServiceError.audioEngineError(error)).eraseToAnyPublisher()
        }

        resetTimer()
        task = recognizer!.recognitionTask(with: request!) { result, error in
            if let result = result {
                self.resetTimer()
                self.bestTranscription = result.bestTranscription.formattedString
                if result.isFinal {
                    self.audioEngine.stop()
                    self.audioEngine.inputNode.removeTap(onBus: 0)
```

```
                self.resultSubject?.send(result.bestTranscription.formattedString)
                self.resultSubject?.send(completion: .finished)
            }
        } else if let error = error {
            self.audioEngine.stop()
            self.audioEngine.inputNode.removeTap(onBus: 0)

self.resultSubject?.send(completion: .failure(SpeechRecognitionServiceError.recognitionError(er
ror)))
        }
    }

    return resultSubject!
        .handleCompletion { [weak self] in
            self?.task?.cancel()
            self?.task = nil
            self?.recognizer = nil
            self?.request = nil
            self?.timerSub = nil
        }
        .eraseToAnyPublisher()
  }

  private func resetTimer() {
    timerSub?.cancel()
    timerSub = Just(()).delay(for: .seconds(3), scheduler: RunLoop.main)
        .sink { [weak self] in
            self?.task?.finish()
        }
  }
}

enum SpeechRecognitionServiceError: Error {
  case audioEngineError(Error)
  case recognitionError(Error)
```

```
    case noTranscription
}
```

Файл SpeechSynthesizerService.swift

```
//
//  SpeechSynthesizerService.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 17.05.2024.
//

import Foundation
import Combine
import AVFoundation

// sourcery: AutoMockable
protocol SpeechSynthesizerServiceType {
    func speak(text: String) -> AnyPublisher<Void, Never>
}

final class SpeechSynthesizerService: NSObject, SpeechSynthesizerServiceType {
    private let sythesizer = AVSpeechSynthesizer()
    private var resultSubject: PassthroughSubject<Void, Never>?

    override init() {
        super.init()
        sythesizer.delegate = self
    }

    func speak(text: String) -> AnyPublisher<Void, Never> {
        resultSubject = PassthroughSubject()

        let utterance = AVSpeechUtterance(string: text)
        utterance.voice = AVSpeechSynthesisVoice()
```

```
        sythesizer.speak(utterance)

        return resultSubject!.eraseToAnyPublisher()
    }
}

extension SpeechSynthesizerService: AVSpeechSynthesizerDelegate {
    func speechSynthesizer(_ synthesizer: AVSpeechSynthesizer, didFinish utterance:
AVSpeechUtterance) {
        resultSubject?.send(())
        resultSubject?.send(completion: .finished)
    }
}
```

Файл MorseCode.swift

```
//
//  MorseCode.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 14.05.2024.
//

import Foundation

struct MorseCode {
    private(set) var elements: [MorseCodeElement]

    var string: String {
        elements.map(\.rawValue).joined()
    }

    var last: MorseCodeElement? {
        elements.last
    }

    var lastLetter: MorseCode {
        MorseCode(elements: Array(elements.split(separator: .space).last ?? []))
    }

    var isEmpty: Bool {
        elements.isEmpty
    }

    init(elements: [MorseCodeElement]) {
```

```
        self.elements = elements
    }

    init() {
        self.elements = []
    }

    init?(string: String) {
        guard string.allSatisfy({ MorseCodeElement(character: $0) != nil }) else { return nil }
        elements = string.compactMap { MorseCodeElement(character: $0) }
    }

    mutating func add(_ element: MorseCodeElement) {
        elements.append(element)
    }

    mutating func add(_ code: MorseCode) {
        elements.append(contentsOf: code.elements)
    }

    mutating func removeLast() {
        elements.removeLast()
    }

    mutating func clear() {
        elements.removeAll()
    }
}

extension MorseCode: Equatable {}

extension Array where Element == MorseCode {
    func joined(separator: MorseCodeElement) -> MorseCode {
        MorseCode(elements: flatMap { $0.elements + [separator] }.dropLast())
    }
}
```

Файл MorseCodeElement.swift

```
//
//  MorseCodeElement.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 14.05.2024.
//

import Foundation
```

```
enum MorseCodeElement: String {
    case dot = "."
    case dash = "-"
    case space = " "
    case pipe = "|"

    init?(character: Character) {
        switch character {
        case ".":
            self = .dot
        case "-":
            self = .dash
        case " ":
            self = .space
        case "|":
            self = .pipe
        default:
            return nil
        }
    }
}
```

Файл MorseDictionary.swift

```
//
//  MorseDictionary.swift
//  MorseBridge
//
//  Created by Andrii Moisol on 06.05.2024.
//

import Foundation

struct MorseDictionary {
    static let current = MorseDictionary(
        language: MorseDictionaryLanguage(
```

```
            rawValue: Locale.current.language.languageCode?.identifier ?? "en"
        ) ?? .en
    )

    let language: MorseDictionaryLanguage
    private let dictionary: [String: String]
    private let order: [String]

    // swiftlint:disable force_try force_cast
    private init(language: MorseDictionaryLanguage) {
        self.language = language

        let data = try! Data(
            contentsOf: Bundle.main.url(
                forResource: "\(language.rawValue)_morse",
                withExtension: "json"
            )!
        )

        let jsonData = try! JSONSerialization.jsonObject(with: data, options: []) as! [String: Any]
        dictionary = jsonData["morse"] as! [String: String]
        order = jsonData["order"] as! [String]
    }

    func getMorseCodes() -> [MorseCode] {
        dictionary.keys.map { MorseCode(string: $0)! }
    }

    func getOrder() -> [String] {
        order
    }

    func textFromMorse(_ code: MorseCode) -> String {
        code.elements
            .split(separator: .pipe)
```

```
            .map { Array($0) }
            .compactMap { wordFromMorse(MorseCode(elements: $0)) }
            .joined(separator: " ")
    }

    func morseFromText(_ text: String) -> MorseCode {
        text
            .split(separator: " ")
            .compactMap { morseFromWord(String($0)) }
            .joined(separator: .pipe)
    }

    private func wordFromMorse(_ code: MorseCode) -> String {
        code.elements
            .split(separator: .space)
            .map { Array($0) }
            .compactMap { getCharacter(for: MorseCode(elements: $0)) }
            .map { String($0) }
            .joined()
    }

    private func morseFromWord(_ word: String) -> MorseCode {
        word
            .compactMap { getCode(for: $0) }
            .joined(separator: .space)
    }

    func getCharacter(for code: MorseCode) -> Character? {
        dictionary[code.string]?.first
    }

    private func getCode(for character: Character) -> MorseCode? {
        guard let key = dictionary.first(where: { $0.value == "\(character.uppercased())" })?.key
else { return nil }
        return MorseCode(string: key)
```

```
    }
}

enum MorseDictionaryLanguage: String {
    case en
    case uk
    case fr
    case de
    case it
    case pt
    case es
    case el
}
```

Файл main.py

```
from openai import OpenAI
import functions_framework
from flask import jsonify



@functions_framework.http
def describe_image(request):
    client = OpenAI()

    try:
        image = request.get_json().get('image')
        language_code = request.get_json().get('language_code') or 'en'

        language = {
            'en': 'English',
            'es': 'Spanish',
            'fr': 'French',
            'de': 'German',
            'it': 'Italian',
            'pt': 'Portuguese',
```

```
            'uk': 'Ukrainian',
            'el': 'Greek',
        }[language_code]

        response = client.chat.completions.create(
            model='gpt-4o',
            messages=[
                {
                    'role': 'user',
                    'content': [
                        {
                            'type': 'text',
                            'text': f"'You are a assistant that describes image to deaf-blind person that
speaks only morse code.
    Give simple and short descriptions using ONLY ${language} language suitable for
transformation into morse code.
    Use only alphanumeric characters, do not use punctuation characters, you are limited to 15
tokens.
    What's on this image?'"
                        },
                        {
                            'type': 'image_url',
                            'image_url': {
                                'url': f'{image}'
                            }
                        }
                    ]
                }
            ],
            max_tokens=15
        )
    except Exception as e:
        return jsonify({
            'error': str(e)
        })
```

```
return jsonify({
    'description': response.choices[0].message.content,
})
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Мобільний застосунок для забезпечення комунікації людей з глухотою та сліпотою

Програма та методика тестування

КПІ.ІП-9614.045430.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Анна ЯСЕНОВА

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Андрій МОЙСОЛ

Київ – 2024

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є розроблене програмне забезпечення MorseBridge, призначене для забезпечення комунікації людей з глухотою та сліпотою за допомогою азбуки Морзе, розпізнавання зображень та голосових команд. Тестування проводиться на платформах iOS, включаючи різні версії операційної системи (iOS 16.0 і новіші) та пристрої, такі як iPhone 8 і новіші моделі.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
– знаходження проблем, помилок і недоліків з метою їх усунення;
– перевірка сумісності застосунку з різними версіями iOS;
– перевірка продуктивності та стабільності роботи на різних моделях iPhone;
– оцінка зручності та інтуїтивності графічного інтерфейсу для користувачів;
– перевірка коректної роботи функцій розпізнавання голосових команд та зображень;
– перевірка правильності передачі та отримання повідомлень у форматі азбуки Морзе.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;

- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення, так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- динамічне тестування на відповідність функціональним вимогам;
- тестування на мобільних пристроях з різною роздільною здатністю екрану;
- тестування на мобільних пристроях з різною версією операційної системи;
- тестування зміни орієнтації екрану;
- тестування працездатності програми у випадку відсутності з’єднання до мережі;
- тестування інтерфейсу користувача;
- тестування зручності використання;

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Мобільний застосунок для забезпечення комунікації людей з глухотою та сліпотою

Керівництво користувача

КПІ.ІП-9614.045430.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Анна ЯСЕНОВА

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Андрій МОЙСОЛ

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«MorseBridge» - це мобільний застосунок для комунікації людей з глухотою та сліпотою за допомогою азбуки Морзе, розпізнавання зображень та голосових команд. Додаток надає можливість введення тексту у форматі азбуки Морзе за допомогою жестів, розпізнавання голосових команд і перетворення їх у текст, а також розпізнавання зображень та передачі їхнього опису через вібрацію або текст.

Кінцева збірка програмного забезпечення MorseBridge включає:

- Інсталяційний пакет для платформи iOS;
- Додаток, що працює на пристроях iPhone 8 і новіших моделях;
- Підтримка iOS версій 16.0 і новіших;
- Графічний інтерфейс, оптимізований для зручного введення та отримання інформації через азбуку Морзе;
- Функції розпізнавання зображень, інтегровані з OpenAI API для забезпечення точного розпізнавання та опису.

Репозиторій містить:

- Вихідний код додатка, написаний на мові програмування Swift;
- Файли проекту Xcode.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Програмне забезпечення MorseBridge повинно функціонувати на iOS-пристроях.

Мінімальна конфігурація технічних засобів:

– тип пристрою: iPhone 8 або новіший;
– процесор: A11 Bionic;
– оперативна пам'ять: 2 ГБ;
– операційна система: iOS 16.0 або новіша;
– підключення до мережі Інтернет зі швидкістю від 10 мегабіт;

Рекомендована конфігурація технічних засобів:

– тип пристрою: iPhone 12 або новіший;
– процесор: A14 Bionic;
– оперативна пам'ять: 4 ГБ;
– операційна система: iOS 17.0 або новіша;
– підключення до мережі Інтернет зі швидкістю від 50 мегабіт;

2.2 Завантаження застосунку

Інсталяційна версія програмного забезпечення MorseBridge доступна для завантаження в AppStore. Для встановлення додатка користувачу необхідно:

– Відкрити AppStore на своєму пристрої;
– Знайти додаток MorseBridge через пошук;
– Натиснути кнопку "Завантажити" та встановити додаток на свій пристрій.

2.3 Перевірка коректної роботи

По завершенню встановлення додатка на робочому столі мобільного пристрою повинна відобразитись іконка даного застосунку. У разі, якщо дана іконка не з’явилась, то встановлення відбулось не успішно. Інакше користувач має змогу запустити додаток, клацнувши на його іконку. Після натискання повинна відобразитись початкова сторінка застосунку.

3 ВИКОНАННЯ ПРОГРАМИ

Користувач відкриває додаток MorseBridge на своєму iPhone, натиснувши на його іконку на головному екрані. Після завантаження додатка на головному екрані відображаються основні функції.

Для введення символів використовуються наступні жести: дотик для введення точки, проведення вправо для введення тире, проведення вниз для підтвердження введеної літери, проведення вліво одним пальцем для видалення останньої введеної літери, проведення вліво двома пальцями для видалення всього введеного тексту, зведення пальців для передачі всього введеного тексту, розведення пальців для передачі останньої введеної літери, проведення вгору для озвучування введеного тексту, проведення вгору двома пальцями для передачі останнього надісланого повідомлення, проведення вниз двома пальцями для передачі останнього отриманого повідомлення.

Для активації режиму розпізнавання голосу користувач торкається і утримує екран. Після вимовлення фрази, додаток перетворює голосове повідомлення у текст, який відображається на екрані та передається у форматі азбуки Морзе через вібрацію.

Для активації функції розпізнавання зображень користувач трясе телефон. Додаток робить фотографію та надсилає її для розпізнавання. Опис зображення перетворюється у код Морзе та передається через вібрацію або відображається на екрані.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Мобільний застосунок для забезпечення комунікації людей з глухотою та сліпотою

Графічний матеріал

КПІ.ІП-9614.045430.06.99

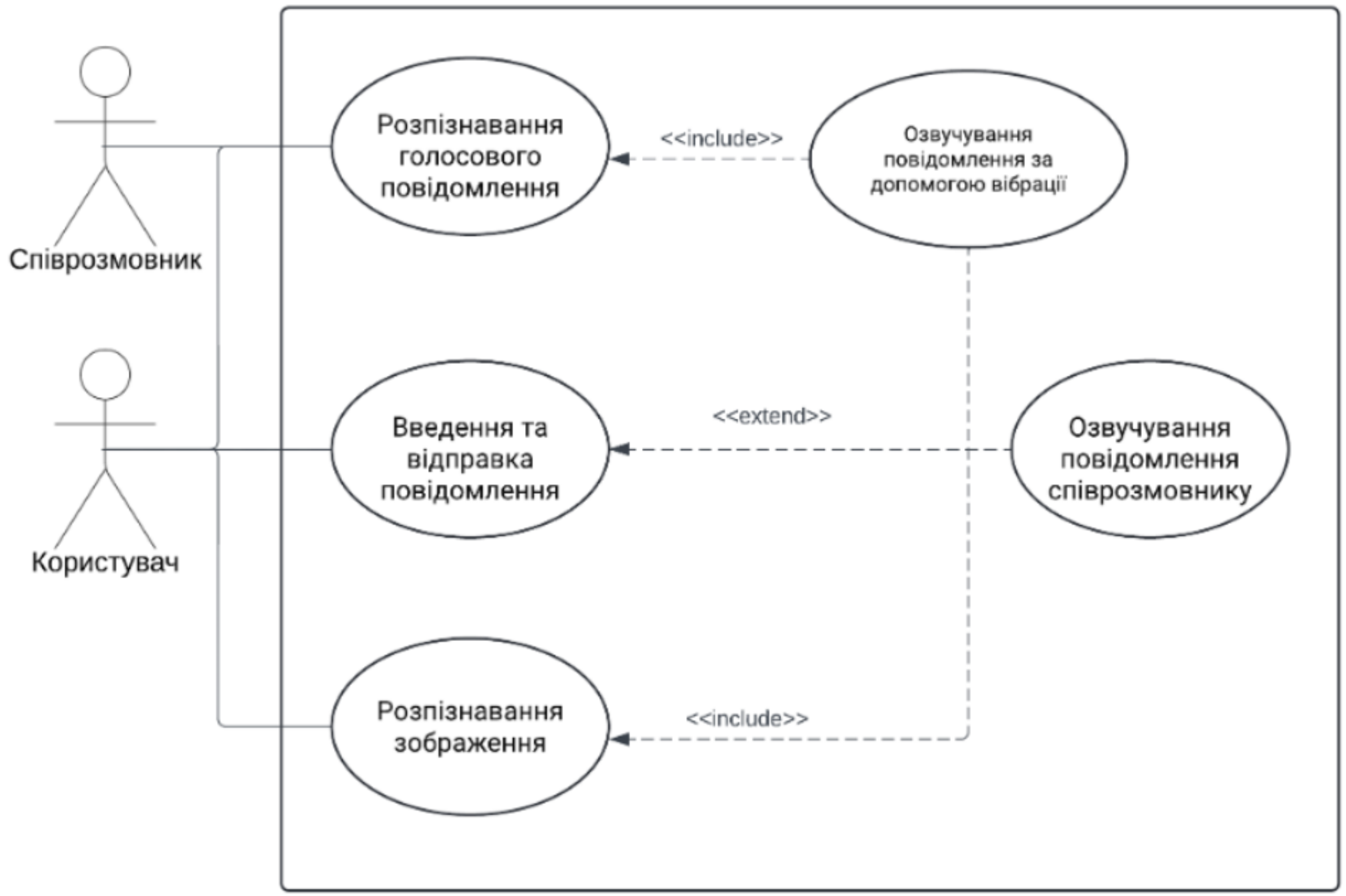
“ПОГОДЖЕНО”
Керівник проєкту:
_____ Анна ЯСЕНОВА

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Андрій МОЙСОЛ

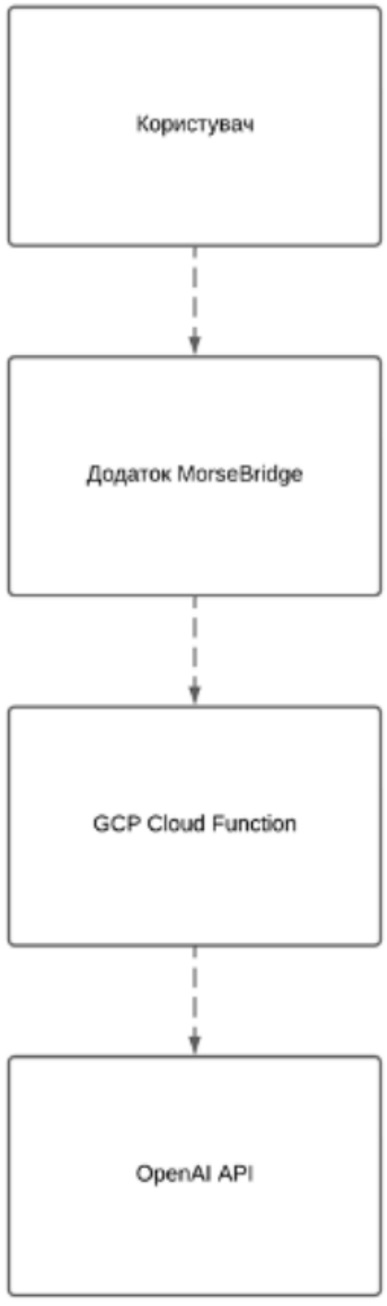
Київ – 2024

КПІ.ІП-9614.045430.06.99.СБД

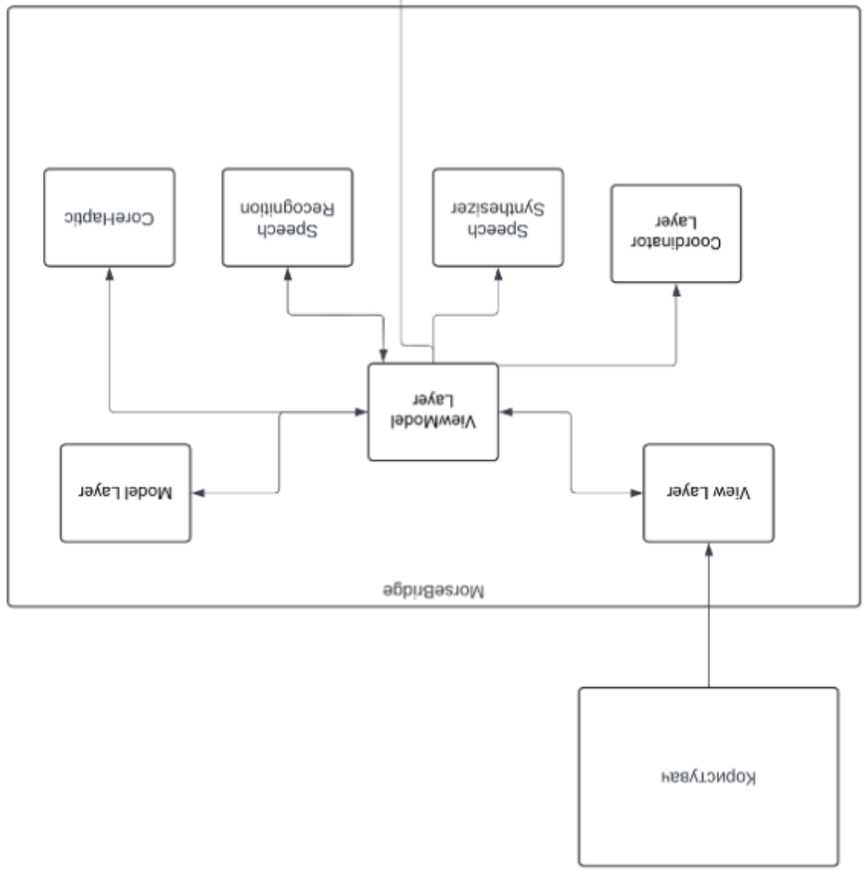


Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІП-9614.045430.06.99.СБД
Розробив		Мойсол А.О			Схема варіантів використання
Перевірив		Ясенова А.В.			Літера / Маса / Масштаб
Т. контр.					Аркуш 1 / Аркушів 4
Н. контр.		Шулькевич Т.В.			Мобільний застосунок для забезпечення комунікації людей з глухотою та сліпотою
Затвердив		Жаріков Е.В.			

КПІ.ІП-9614.045430.06.99.СБД



					КПІ.ІП-9614.045430.06.99.СБД		
					Схема контекстна	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Мойсол А.О				Аркуш 2	Аркушів 4
Перевірив		Ясенова А.В.					
Т. контр.							
					Мобільний застосунок для забезпечення комунікації людей з глухотою та сліпотою		
Н. контр.		Шулькевич Т.В.					
Затвердив		Жаріков Е.В.					

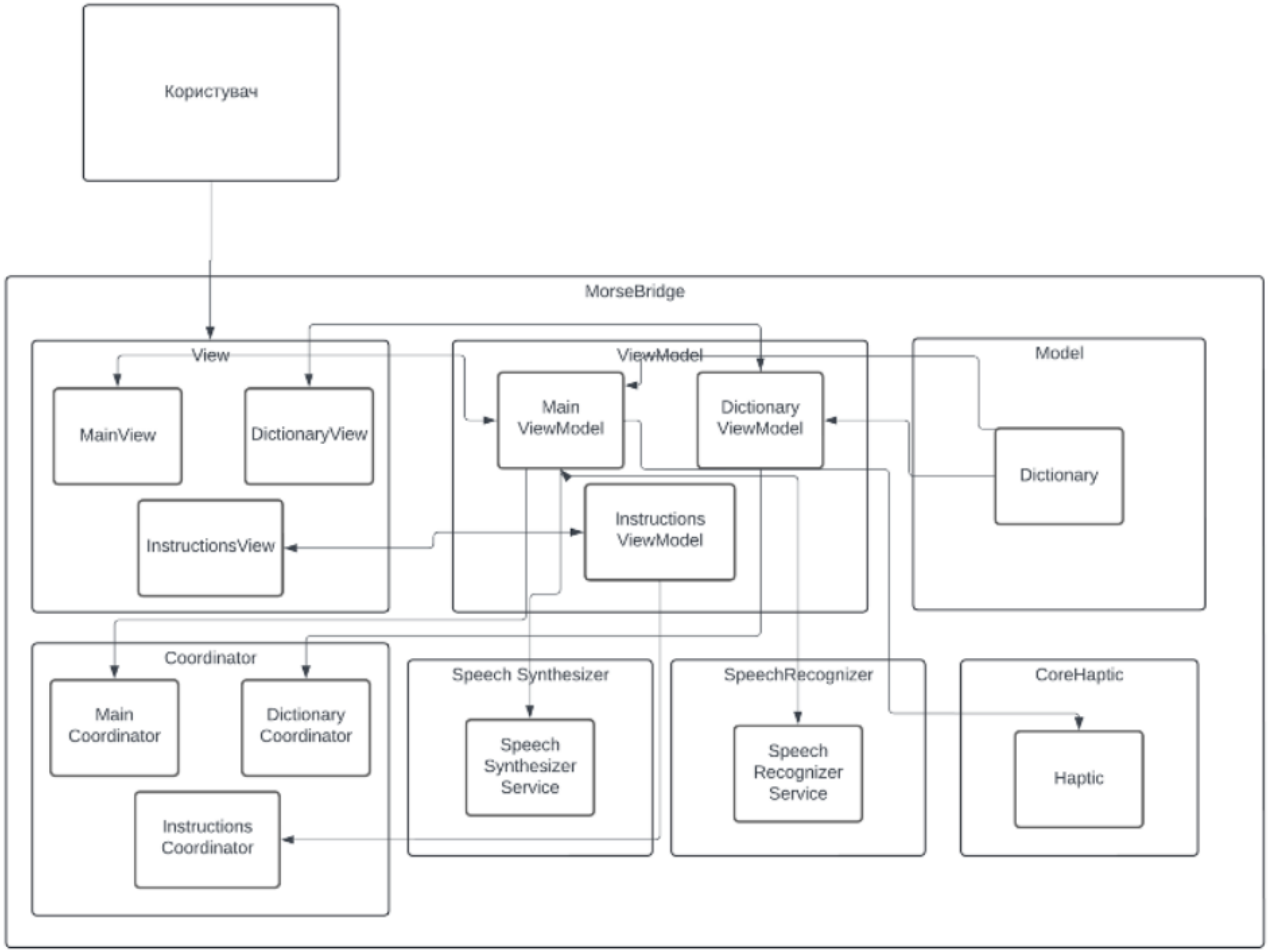


					КПІ.ІП-9614.045430.06.99.СБД
Зм.	Арк.	№ документа	Підпис	Дата	
Розробив		Мойсол А.О			Схема контейнера
Перевірив		Ясенова А.В.			
Т. контр.					
Н. контр.		Шульженич Т.В.			Мобільний застосунок для забезпечення комунікацій людей з глухотою та сліпотою
Затвердив		Жарікова Е.В.			

Літера	Маса	Масштаб

Аркуш 3	Аркушів 4

КПІ.ІП-9614.045430.06.99.СБД



Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІП-9614.045430.06.99.СБД
Розробив		Мойсол А.О			Схема компонентна
Перевірив		Ясенова А.В.			Літера / Маса / Масштаб
Т. контр.					Аркуш 4 / Аркушів 4
Н. контр.		Шулькевич Т.В.			Мобільний застосунок для забезпечення комунікації людей з глухотою та сліпотою
Затвердив		Жаріков Е.В.			