МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
**Національний університет «Запорізька політехніка»**

радіоелектроніки та телекомунікацій
(повне найменування інституту, факультету)

інформаційних технологій електронних засобів
(повне найменування кафедри )

# Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему: Розробка інтерфейсу та елементів програми керування роботом-гідом у крос-платформному фреймворку Xamarin

Виконав: студент(ка) 4 курсу, групи РТ-618сп

Спеціальності. "Автоматизація та комп'ютерно-інтегровані технології"
(код і найменування спеціальності)

Освітня програма (спеціалізація)
"Автоматизація, мехатроніка та робототехніка"

Остапенко В.Р.
(прізвище та ініціали)

Керівник Фарафонов О.Ю.
(прізвище та ініціали)

Рецензент Неласа Г.В.
(прізвище та ініціали)

2021

**Форма № 25**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
**Національний університет «Запорізька політехніка»**
(повне найменування закладу вищої освіти)

Інститут, факультет _Факультет радіоелектроніки та телекомунікацій___
Кафедра Інформаційних технологій електроних засобів
Ступінь вищої освіти __бакалавр_____
Спеціальність 151 «Автоматизація та комп'ютерно-інтегровані технології»
(код і найменування)
Освітня програма Автоматизація, мехатроніка та робототехніка
(назва освітньої програми (спеціалізації)

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри ІТЕЗ** Огнєвий Євген Вікторович
«26» квітня 2021 року

# З А В Д А Н Н Я
## НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Остапенко Владислав Романович
(прізвище, ім'я, по батькові)

1.Тема проєкту(роботи) Розробка інтерфейсу та елементів програми керування роботом-гідом у крос-платформному фреймворку Xamarin
керівник проєкту Фарафонов Олексій Юрійович, канд. тех. наук, доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «26» квітня 2021 року № 159
2. Строк подання студентом проєкту (роботи) 29 травня 2021 року
3. Вихідні дані до проєкту (роботи) робот-гід, STM32

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1 Загальна частина. 2 Спеціальна частина. 3 Розробка інтерфейсу з елементами керування.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
47 рисунки; 4 таблиці;

6. Консультанти розділів проєкту (роботи)

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | прий<br>викон<br>завда |
| Розділи 1-3 | Фарафонов О.Ю., доц. | 05.04 | |
| Нормоконтроль | Поспєєва І.Є., ст. викладач | | |

7. Дата видачі завдання «26» квітня 2021 року.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломного проєкту (роботи) | Строк виконання етапів проєкту ( роботи ) | Пр |
|---|---|---|---|
| 1 | Аналіз сучасних крос-платформних фреймворків | 26.04.21 | |
| 2 | Ознайомлення з системою керування роботом-гідом | 29.04.21 | |
| 3 | Постановка завдання | 01.05.21 | |
| 4 | Аналіз крос-платформного фреймворку Xamarin | 05.05.21 | |
| 5 | Розробка інтерфейсу роботу-гіду | 19.05.21 | |
| 6 | Оформлення ПЗ | 27.05.21 | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**Студент** _____ Остапенко В.Р.
( підпис ) (прізвище та ініціали)

**Керівник проєкту (роботи)** _____ Фарафонов О.Ю.
( підпис ) (прізвище та ініціали)

# РЕФЕРАТ

ПЗ: 62 с., 4 таблиці, 47 рисунка, 15 джерел, 13 слайдів

Об'єкт розробки: інтерфейс програми керування роботом-гідом.

Мета роботи: розробка інтерфейсу та елементів програми керування роботом-гідом у крос-платформному фреймворку Xamarin.

Для здійснення поставленої мети необхідно було вирішити наступні задачі:

- ознайомитись з системою керування роботом-гідом;

- ознайомитись з крос-платформним фреймворком Xamarin;

- провести аналіз Xamarin, у порівнянні, з іншими крос-платформними фреймворками;

- розробити інтерфейс з елементами керування роботом-гідом;

- розробити заходи щодо техніки безпеки при роботі за комп’ютером.

При проведенні робіт був зроблен аналіз Xamarin, у порівнянні, з іншими крос-платформними фреймворками. Були досліджені різні підходи, такі як Xamarin.Forms, Xamarin.iOS, Xamarin.Android, та підходи в розробці крос-платформних додатків. Був розроблений інтерфейс для керування роботом-гідом.

XAMARIN, ANDROID, C#, ІНТЕРФЕЙС, КЕРУВАННЯ, РОБОТ-ГІД, ЕЛЕМЕНТИ КЕРУВАННЯ, ФРЕЙМВОРК, КРОС-ПЛАТФОРМНА РОЗРОБКА.

**ЗМІСТ**

# ВСТУП

Xamarin — це платформа з відкритим кодом для створення сучасних та продуктивних додатків для iOS, Android та Windows за допомогою .NET. Xamarin — це абстракційний рівень, який управляє зв'язком спільного коду з базовим кодом платформи. Xamarin працює в керованому середовищі, яке забезпечує такі зручності, як виділення пам'яті та збір сміття.

Xamarin дозволяє розробникам розподіляти в середньому 90% своїх програм між платформами. Цей шаблон дозволяє розробникам писати всю свою бізнес-логіку однією мовою (або використовувати повторно існуючий код програми), але досягати природних характеристик, вигляду та відчуття на кожній платформі.

Xamarin розширює платформу для розробників .NET за допомогою інструментів та бібліотек, спеціально призначених для створення додатків для Android, iOS, та Windows.

Основною задачею даного дипломного проекту є написання інтерфейсу з елементами керування роботом-гідом використовуючи фреймворк Xamarin. Також проведемо аналіз фреймворку Xamarin, перелік усіх переваг та недоліків у порівнянні з іншими фреймворками. Оглянуті інші аналоги, що використовуються для написання програм для цих операційних систем – Android, iOS, Windows Phone.

В першому розділі буде розглянуте загальне поняття крос-платформного програмування. Будуть розглянуті різні рішення, для побудови програм під різні операційні системи. Будуть описані сучасні аналоги Xamarin, їх переваги та недоліки. Також будуть описані можливості фреймворку Xamarin, як працює його ядро, основні методи побудови додатків, такі як – Xamarin.Forms, Xamarin.iOS та Xamarin.Android. Будуть розглянуті принцип роботи та поняття в розробці додатків, використовуючи крос-платформний фреймворк Xamarin та мову C#.

В другому розділі будуть розглянуті основні підходи створення додатків використовуючи крос-платформний фреймворк Xamarin. Буде описане інтегроване середовища розробки Visual Studio, її переваги, буде проведено її аналіз, також буде виконано порівняння різних редакцій Visual Studio. Написані системні вимоги для роботи в середовищі розробки Visual Studio. Також будуть описані технології USB та WebSocket, які використовуються для роботи з роботом-гідом.

У третьому розділі розглянуто розробку інтерфейсу з елементами керування для роботу-гіду, використовуючи мову програмування XAML для створення інтерфейсу, наведені приклади створення елементу інтерфейсу Button. Також описана розробка та тестування інтерфейсу.

# 1 ЗАГАЛЬНА ЧАСТИНА

## 1.1 Поняття крос-платформної розробки

Для написання програм під кожну з операційну систему потрібно витрачати багато часу. Оскільки кожна операційна система має різні API інтерфейси, специфічну логіку роботи, нативну мову програмування. Для того, щоб створити програмний продукт, який буде доступний для різних операційних систем компанії-розробнику потрібно тримати штат програмістів, спеціалізованих для різних операційних систем. Тому є фреймворки, які дозволяють позбутися від такої залежності і писати код під різні операційні системи, використовуючи одну мову програмування.

Звісно, через це, є свої недоліки, такі як:

- програми написані мовою програмування для даної операційної системи працюють швидше;
- потрібні спеціалісти, які знають усі платформи, для цього фреймворку.

Проте, у такому підході є і переваги:

- розробникам не потрібно міняти логіку програмного продукту;
- також скорочується час на розробку продукту.

Крос-платформність – властивість програми працювати більш ніж на одній операційній системі, або апаратній платформі, та технології що дозволяють досягти такої властивості. Такий підхід дозволяє суттєво скоротити витрати на розробку нового або адаптацію існуючого програмного забезпечення. У табл. 1.1 зображено порівняння нативної та крос-платформної розробки додатків.

Технології досягнення крос-платформності:

- крос-платформність мов програмування;

Досягається шляхом забезпечення незалежності програмного коду від платформи. Більшість сучасних мов програмування можуть реалізуватись на різних системах.

— крос-платформність на рівні середовища виконання;

Забезпечується реалізацією в системах можливостей, необхідних програмам незалежно від платформи.

— крос-платформність на апаратному рівні;

Досягається реалізацією однакових машинних команд та форматом їх представлення, механізмів адресації пам’яті, тощо.

Таблиця 1.1 – Порівняння нативної та крос-платформної розробки

| Параметри | Нативні програми | Крос-платформні програми |
|---|---|---|
| Вартість | Висока вартість розробки | Відносно низька вартість розробки |
| Використання коду | Працює для однієї платформи | Один код можна використовувати на декількох платформах |
| Доступ до пристрою | Має безперешкодний доступ до API пристрою | Немає гарантованого доступу до всіх API пристрою |
| Послідовність інтерфейсу користувача | Відповідно до компонентів інтерфейсу користувача пристрою | Обмежена узгодженість з компонентами інтерфейсу користувача пристрою |
| Продуктивність | Бездоганна продуктивність для операційної системи, на яку було вироблено | Висока продуктивність, але відставання та проблеми сумісності апаратного забезпечення не рідкість |

## 1.2 Аналіз сучасних фреймворків

Xamarin – не єдиний фреймворк, який орієнтується на три ОС – WP, Android, iOS. Звичайно, є ще дуже багато аналогічних фреймворків, що дозволяють писати програми під ці ОС, деякі з них, звичайно, не підтримують усі ці три ОС, проте підтримують дві, проте навіть ці фреймворки слід описати, оскільки вони є альтернативою використанню C# та Xamarin. Досить популярними зараз є аналоги, що використовують Java та JavaScript у своїй роботі, оскільки ці мови популярні у всьому світі. У табл. 1.2 проводиться порівняння популярних аналогів фреймворку Xamarin.

Таблиця 1.2 – Порівняння популярних аналогів Xamarin

| | React Native | Flutter | Ionic | Phone Gap |
|---|---|---|---|---|
| Мова | JavaScript, React | Dart | JavaScript, HTML, CSS, Angular, React, Vue | JavaScript, HTML, CSS |
| Спільнота | Дуже велике | Невелике, але швидко розвивається | Велике | Велике |
| Платформи | Android, iOS, UWP | Android, iOS, Web, Desktop | Android, iOS, Web | Android, iOS, WP |
| Open source | Так | Так | Так | Так |
| Продуктивність | Висока, близька к нативній | Дуже велика | Середня | Середня |

Далі будуть більш детальне розписані усі аналоги фреймворку Xamarin, невелика інформація про них, та їх переваги і недоліки.

Найпопулярніші аналоги:

React Native

React Native заснований на бібліотеці JavaScript React від Facebook, яка надає власний інтерфейс платформи. React Native - чудовий вибір для простих додатків, де API мають чіткий міст між платформами. Однак для складних додатків рідний код часто необхідний для подолання будь-яких прогалин у функціональності.

Переваги:

- до 80% кодової бази можна розподіляти між платформами, залежно від складності програми;
- пропонує готові до застосування елементи продукту, тим самим скорочуючи час розробки;
- дозволяє бачити зміни, внесені в код, протягом декількох секунд, а не хвилин під час використання рідних технологій;
- фокусується на UI (user interface), значною мірою надаючи високочутливий інтерфейс.

Недоліки:

- не часті оновлення;
- покращує швидкість розробки, але збільшує тривалість процесу налагодження, особливо на Android. [1]

Flutter

Flutter розроблений та випущений Google у 2017 році. Він використовує мову Dart для розробки програм для Android, iOS, Mac, Windows та Linux з єдиної кодової бази.

Переваги:

- дозволяє бачити зміни, внесені в код, протягом декількох секунд, а не хвилин під час використання рідних технологій;
- знижує час розробки додатку на різні системи;

— Flutter базується на Dart, об’єктно-орієнтованій мові програмування, для якої розробникам було досить легко набути навичок.

Недоліки:

— бракує кількість бібліотек з готовим до впровадження функціоналом, в порівнянні з нативною розробкою;

— готовий додаток важить більше ніж в інших аналогів, через вбудовані віджети. [1]

Ionic

За останні кілька років, Ionic зарекомендував себе як лідер в області розвитку гібридних мобільних додатків. Команда Ionic зберігає структуру оновлення шляхом адаптації до останніх тенденцій, випереджаючи конкурентів. Його найближчі конкуренти мають комерційну ліцензію, в той час як Ionic безкоштовний для користування з відкритим вихідним кодом, крім того, його екосистема стала настільки велика, що ви можете легко знайти тонни ресурсів, щоб почати роботу в найкоротші терміни. Мова програмування – JavaScript.

Переваги:

— розроблений спеціально для мобільних операційних систем;

— має велику спільноту з багатьох країн світу.

Недоліки:

— знання AngularJS стає майже необхідним, якщо хочеться вийти за рамки базових програм;

— має складне проектування навігації, через не дуже простий у використанні UI-маршрутизатор. [1]

PhoneGap

Безкоштовний фреймворк, що дозволяє створити додатки для мобільних пристроїв використовуючи JavaScript, HTML5 та CSS3, без необхідності знання нативних мов програмування, під всі мобільні операційні системи.

Готовий додаток компілюється у вигляді пакетів для кожної мобільної операційної системи.

Переваги:

— велика кількість готових плагінів;

— можливість легко поділитися своєю програмою;

Недоліки:

— має низьку продуктивність та відсутність віджетів інтерфейсу користувача;

— PhoneGap залежить від iOS SDK для створення програми, а для завантаження цих SDK потрібен Mac. [1]

Native Script

Випущений у 2015 році, це фреймворк який дозволяє створювати власні програми для Android та iOS за допомогою JavaScript, TypeScript, Angular або Vue.js.

Переваги:

— виконується дуже швидко в порівнянні з іншими альтернативами;

— компілюється до нативного коду і працює як нативний код;

Недоліки:

— великий розмір файлу продукту;

— відсутність оновлень у документації.

RubyMotion

Випущений у 2012 році, RubyMotion - це частково відкритий фреймворк для створення крос-платформних нативних додатків для Android, iOS та MacOS за допомогою Ruby.

Мова програмування Ruby в основному використовується для веб-розробки. Однак за допомогою Ruby мобільні програми можна створювати за допомогою Ruby.

Переваги:

— багата кількість бібліотек;

— Легкий код.

Недоліки:

— складність налагодження.

## 1.3 Опис Xamarin

### 1.3.1 Технологія Xamarin

Xamarin - це компанія, що належить Microsoft у Сан-Франциско, і заснована в травні 2011 року інженерами, які створили Mono, Mono для Android та MonoTouch, які є крос-платформними реалізаціями Common Language Infrastructure та загальномовні специфікації (часто їх називають Microsoft .NET).

Історія заснування:

16 травня 2011 року Мігель де Ікаса оголосив у своєму блозі, що Mono буде розроблено та підтримано Xamarin, новоствореною компанією, яка планувала випустити новий набір мобільних продуктів. За словами де Ікаса, принаймні частина початкової команди Mono переїхала до нової компанії.

У грудні 2012 року Xamarin випустив Xamarin.Mac, плагін для існуючого інтегрованого середовища розробки MonoDevelop (IDE), який дозволяє розробникам створювати додатки на базі C # для операційної системи macOS від Apple та упаковувати їх для публікації через AppStore.

Наступного року було оголошено про випуск Xamarin 2.0. Випуск включав два основні компоненти: Xamarin Studio, ребрендинг свого відкритого коду IDE Monodevelop та інтеграцію з Visual Studio, IDE Microsoft для .NET Framework, що дозволяє використовувати Visual Studio для створення додатків Android , iOS та Windows.

28 травня 2014 була випущена третя версія інструментів розробки. Однією з головних новинок став інструмент Xamarin.Forms, що дозволяє створювати інтерфейс користувача з набору візуальних елементів, що

описуються мовою розмітки XAML. Інтерфейс показується у візуальні елементи відповідної операційної системи (Android, iOS і WinPhone).

У жовтні 2015 року компанія Xamarin оголосила, що придбала платформу для розробників RoboVM для Java. Але після придбання Xamarin Microsoft'ом, вони анонсували що не будуть підтримувати цей продукт.

У 2016 році Microsoft остаточно придбала компанію Xamarin, умови угоди не розголошуються, хоча Wall Street Journal повідомляє, що ціна становить від 400 до 500 мільйонів доларів.

Також на конференції Microsoft Build 2016 анонсували, що Xamarin тепер буде повністю безкоштовним фреймворком, який буде підтримуватися Microsoft.

Список продуктів, що були створені Xamarin:

— Xamarin Platform

  Дозволяє використовувати нативні API різних операційних систем.

— Xamarin.Forms

  Використовується для побудови UI (user interface).

— Xamarin Test Cloud

  Xamarin Test Cloud дає змогу тестувати мобільні додатки, написані будь-якою мовою на реальних, не розбиваних пристроях у хмарі.

— Xamarin Studio

  Основне середовище розробки для написання додатків під iOS чи Mac з використанням Xamarin

— Xamarin для Visual Studio

  Дозволяє писати програми не використовуючи іншї середовища розробки.

— Xamarin.Mac

  Фреймворк для написання програм на Mac. [2]

### 1.3.2 Платформа Mono

Mono – це безкоштовна платформа розробки з відкритим кодом, сумісна з .NET Framework. Зараз її очолює Xamarin, дочірня компанія Microsoft та .NET Framework. Mono можна запустити на багатьох програмних системах.

Mono дозволяє розробляти крос-платформні додатки з поліпшеною продуктивністю працівника. Заснована на стандартах ECMA для C# та Common Language Infrastructure. Включає в себе як інфраструктуру, так і засоби розробки, необхідні для запуску додатків .NET.

Mono складається з трьох груп компонентів:

- основні компоненти;
  - компілятор C#
  - віртуальну машину для CLI
  - основні бібліотеки
- стек розробки Mono/Linux/GNOME;

  Забезпечує інструменти для розробки додатків під час використання існуючих GNOME та безкоштовних бібліотек з відкритим кодом.
- стек сумісності Microsoft.

  Надає шлях для перенесення Windows

Mono було створено для запуску .NET програм на інших ОС, таких як OS X, Linux та інші. На рис. 1.1 зображена ієрархія Mono.

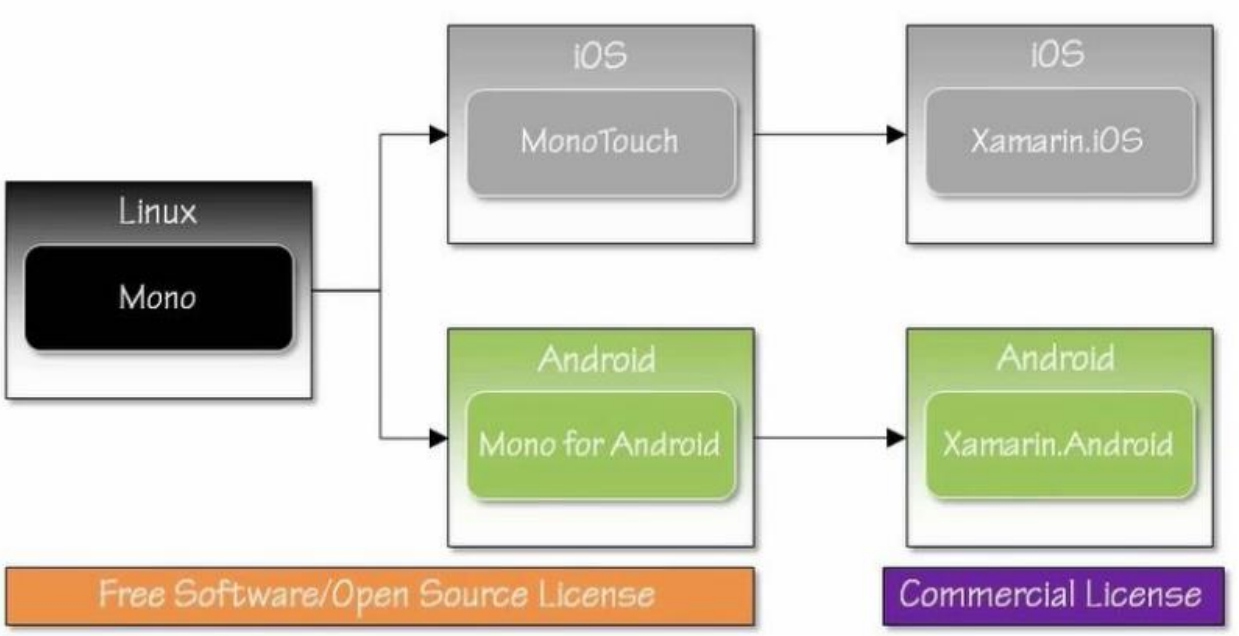


Рисунок 1.1 – Ієрархія Mono[4]

Як можна побачити, були створені реалізації CLR та CLI для ядра Linux, тобто усі додатки можна запустити на цій системі. Також було створено Mono під інші операційні системи – Mono for Android для Android, та MonoTouch для iOS, так як дані операційні системи працюють на ядрі Linux. Принцип роботи Mono схожий на принцип роботи .NET CLR, його можна побачити на рис. 1.2.

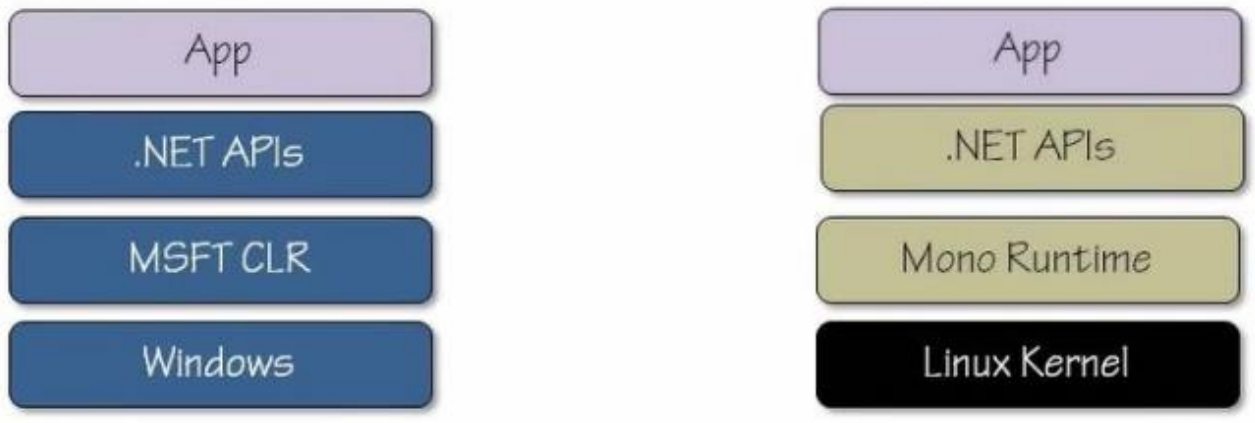


Рисунок 1.2 – Порівняння принципів роботи CLR та Mono[4]

### 1.3.3 Xamarin.Android

Додатки Xamarin.Android виконуються в середовищі Mono. Це середовище працює разом з віртуальною машиною Android Runtime. Обидва середовища працюють поверх ядра Linux і надають різні API для коду користувача, який дає розробникам отримати доступ до базової системи.

Для доступу к основним засобам операційної системи Linux використовуються бібліотеки класів .NET

В Android більшість систем недоступні у власних додатках, тому треба використовувати API-інтерфейси Java середовища Android. Архітектура має вигляд зображений на рис. 1.3

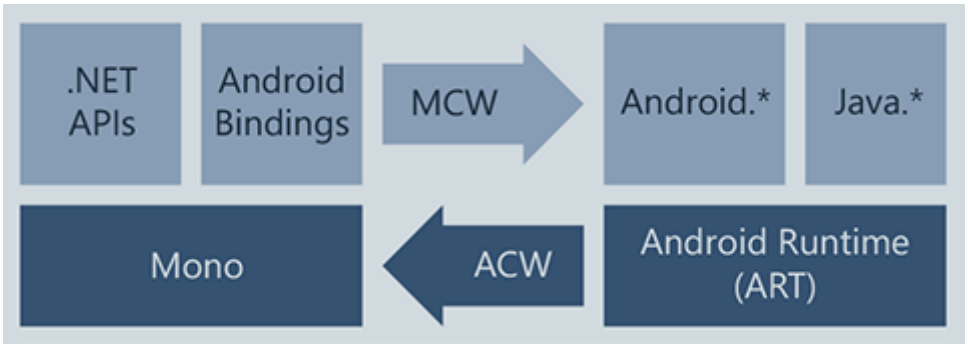


Рисунок 1.3 – Архітектура Xamarin.Android [5]

Додатки Xamarin.Android компілюються з мови C# в проміжну мову, котрий під час запуску програми перетворює Just-in-Time-компіляцію (JIT) в машинну збірку. Додатки Xamarin.Android працюють в середовищі Mono паралельно з віртуальною машиною середовища Android. Xamarin надає прив’язки .NET до просторів імен Android.* і Java.*. Середовище Mono звертається до цих просторів імен з використанням керованих викликаних оболонок (MCW) і надає середовищу APT викликані програми-оболонки Android (ACW), завдяки чому обидва середовища можуть викликати код один одного.

Додатки в Android працюють завдяки спеціальним об’єктам, так названих activity. Кожна сторінка має свою activity, а кожна activity у свою чергу мають свій lifecycle(цикл існування). Xamarin.Android також не є

виключенням, він дозволяє описувати Activity за допомогою мови програмування C#.

Activity - це незвичайна концепція програмування, характерна для Android. У традиційній розробці додатків зазвичай існує статичний основний метод, який виконується для запуску програми. Однак з Android справи йдуть інакше. Додатки Android можна запускати через будь-яку зареєстровану activity у програмі.

На практиці більшість програм матимуть лише певну activity, яка вказана як точка входу до програми. Однак, якщо програма аварійно завершує роботу або її завершує операційна система, операційна система може спробувати перезапустити програму під час останньої відкритої діяльності або де-небудь ще в межах попереднього стеку дій.

Крім того, операційна система може призупиняти дії, коли вони не активні, і повертати їх, якщо в ньому не вистачає пам’яті. Потрібно ретельно продумати, щоб програма могла правильно відновити свій стан у разі перезапуску діяльності, особливо якщо ця діяльність залежить від даних попередніх дій.

Життєвий цикл activity реалізований як сукупність методів, які ОС викликає протягом життєвого циклу activity. Ці методи дозволяють розробникам реалізовувати функціонал, необхідний для задоволення вимог до стану та управління ресурсами своїх додатків.

Операційна система Android виділяє activity залежно від їх стану. Це допомагає Android виявляти activity, яка більше не використовується, що дозволяє операційній системи повернути пам’ять та ресурси. На рис. 1.4 можна побачити стани, які існують протягом життєвого циклу activity.

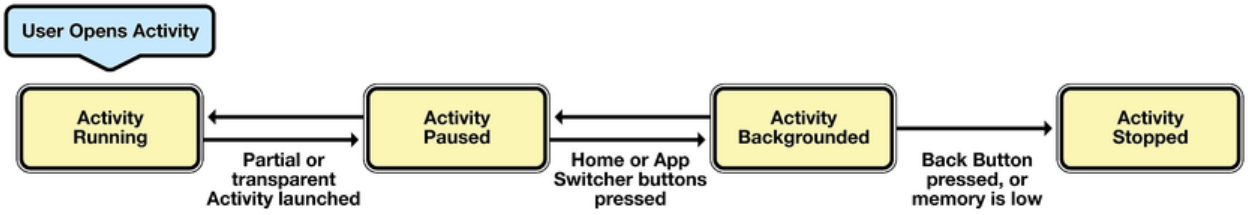


Рисунок 1.4 – Стани activity

Ці стани можна віднести у 4 основні групи:

- Active чи Running – activity вважаються активними або запущеними, якщо вони знаходяться на передньому плані, також відомому як вершина стеку activity. Це вважається найвищою пріоритетною діяльністю в Android.
- Paused – коли пристрій переходить у сплячий режим, або activity все ще видна, але частково прихована новим, не повнорозмірним activity. Paused activity все ще жива, тобто вона зберігає всю інформацію про стани та користувачів. Це вважається другою діяльність з найвищим пріоритетом в Android.
- Stopped/Backgrounded – activity повністю закрита іншою activity, вважається зупиненою або у фоновому режимі. Stopped activity все ще намагаються зберегти інформацію про свій стан якомога довше, але Stopped activity має найнижчий пріоритет з 3х станів, тому операційна система у першу чергу буде відключати activity як раз у цьому стані.
- Restarted – Android може видалити з пам’яті activity, яка перебуває від Paused чи Stopped життєвому циклі. Якщо користувач повертається до activity, її потрібно перезапустити, відновити до попередньо збереженого стану, а потім відобразити користувачеві.[3]

### 1.3.4 Xamarin.iOS

Додатки Xamarin.iOS виконуються в середовищі Mono і використовують компіляцію з повним часом (AOT) для компіляції коду C# на мові асемблера ARM. Це виконується паралельно з виконуючою середовища мети-C. Обидна середовища працюють поверх ядра, схожого на UNIX і надають доступ до різних інтерфейсів API для призначеного для користувача коду, що дозволяє

розробникам звертатися до базової машинної чи керованої системи. На рис. 1.5 зображено базовий обзір архітектури.

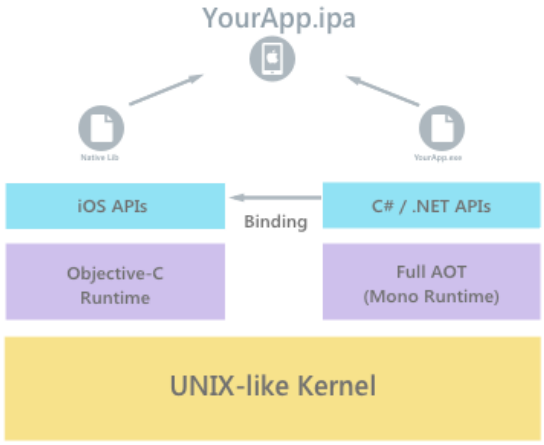


Рисунок 1.5 – Базова архітектура Xamarin.iOS

В iOS є обмеження безпеки, задане Apple, яке забороняє виконання динамічно створеного коду на пристроях. Щоб забезпечити дотримання цих протоколів безпеки, Xamarin.iOS замість цього використовує компілятор попереднього часу (AOT) для компіляції керованого коду. При цьому створюється власний двійковий файл iOS, який можна оптимізувати за допомогою LLVM для пристроїв, котрий може бути розгорнутий на процесорі Apple ARM. Приблизна схема спільного використання зображена на рис. 1.6.

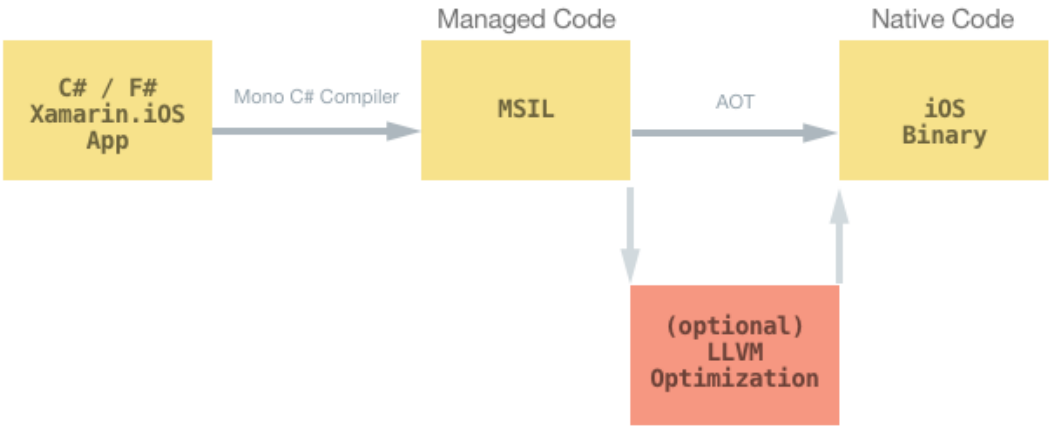


Рисунок 1.6 – Схема спільного використання [5]

Додатки Xamarin.iOS проходять повну Ahead-of-Time-компіляцію (AOT) з мови C # в власний код збірки ARM. Xamarin використовує селектори для надання коду Objective-C керованого коду C # і реєстратори для надання керованого коду C # кодом Objective-C. Селектори і реєстратори в сукупності називаються "прив'язками" і забезпечують взаємодію між Objective-C і C #.

### 1.3.5 Xamarin.Forms

Xamarin.Forms - це платформа для користувача інтерфейсу з відкритим кодом. За допомогою Xamarin.Forms розробники можуть створювати додатки для Xamarin.Android, Xamarin.iOS і Windows на основі загальної бази коду.

Xamarin.Forms дозволяє розробникам створювати призначені для користувача інтерфейси в XAML за допомогою коду програмної частини в C #. Ці інтерфейси на кожній платформі готуються до перегляду як власні елементи управління. На рис. 1.7 зображена загальна схема архітектури Xamarin.Forms.

Платформа Xamarin.Forms орієнтована на розробників, перед якими стоять наступні завдання:

- спільне використання макета призначеного для користувача інтерфейсу і розробка на різних платформах;
- спільне використання коду, тестів і бізнес-логіки на різних платформах;
- написання крос-платформних додатків на мові C # в Visual Studio.

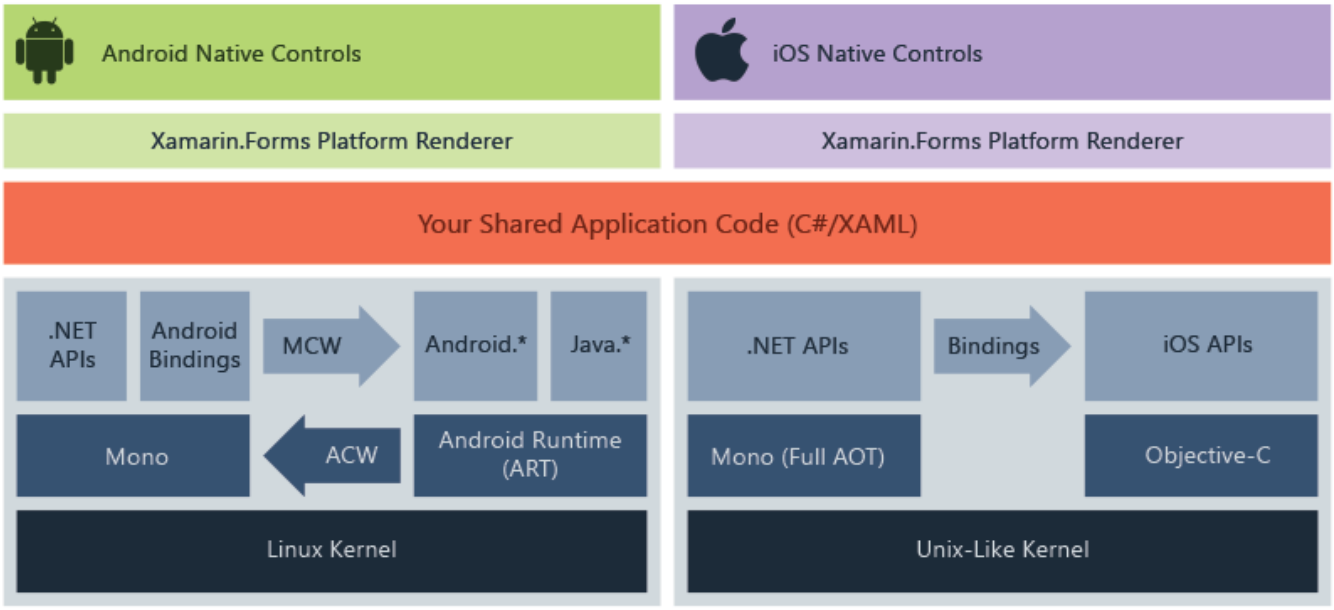


Рисунок 1.7 – Схема архітектури Xamarin.Forms

Xamarin.Forms надає узгоджений API для створення елементів призначеного для користувача інтерфейсу на різних платформах. Цей API може бути реалізований в XAML або C # і підтримує прив'язку даних для шаблонів, таких як "Модель - уявлення - модель вистави" (MVVM).

Під час виконання Xamarin.Forms використовує програму для відображення платформи для перетворення крос-платформних елементів призначеного для користувача інтерфейсу в власні елементи управління в Xamarin.Android, Xamarin.iOS і UWP. Дозволяє розробникам добитися звичного зовнішнього вигляду, поведінки і рівня продуктивності, а також реалізувати переваги спільного використання коду на різних платформах.

Додатки Xamarin.Forms зазвичай складаються із загальної бібліотеки .NET Standard і проектів окремих платформ. Загальна бібліотека містить уявлення XAML або C # і всю бізнес-логіку, наприклад служби, моделі або інший код. Проекти платформи містять всю залежну від платформи логіку або пакети, необхідні додатку. [5]

Основні типи елементів Xamarin.Forms:

— Xamarin.Forms Pages – являють собою крос-платформні екрани мобільних додатків.

- Xamarin.Forms Метка – використовуються для створення елементів управління користувальницького інтерфейсу в візуальних структурах.
- Xamarin.Forms Представлення – це стандартні блоки крос-платформних мобільних користувальницьких інтерфейсів.
- Xamarin.Forms Осередків – це спеціалізований елемент, який використовується для елементів в таблиці і описує, як повинен бути визуализирован кожен елемент в списку. Cell клас є похідним від Element, з якого VisualElement також успадковується. Осередок не є візуальним елементом; Замість цього використовується шаблон для створення візуального елемента. [6]

Xamarin. Forms Device Class

Клас Device містить ряд властивостей та методів, які допомагають розробникам налаштувати макет та функціональність на основі кожної платформи.

На додаток до методів та властивостей для націлювання коду на певні типи та розміри обладнання, клас Device включає методи, які можна використовувати для взаємодії з елементами керування інтерфейсом із фонових потоків. Для отримання додаткової інформації див. Взаємодія з інтерфейсом користувача з фонових потоків.

Методи Device class:

- Device.Idiom – властивість Device.Idiom може використовуватися для зміни макетів або функціональних можливостей залежно від пристрою, на якому запущена програма;
- Device.FlowDirection – отримує значення перерахування FlowDirection, яке представляє поточний напрямок потоку, що використовується пристроєм;
- Device.Styles – містить вбудовані визначення стилів, які можна застосувати до властивостей Style;

— Device.GetNamedSize – використовується коли встановлюються значення FontSize в C#;
— Device.GetNamedColor – метод може бути використаний для отримання іменованих кольорів на Android, iOS та UWP;
— Device.StartTimer – метод, який забезпечує простий спосіб ініціювати залежні від часу завдання, які працюють у загальному коді Xamarin.Forms, включаючи .NET Standard Library. [7]

# 2 СПЕЦІАЛЬНА ЧАСТИНА

## 2.1 Основні підходи створення додатків на Xamarin

Xamarin – це фреймворк в який входить різні бібліотеки, та різні бібліотеки-розширення, створенні іншими спеціалістами для використання в своїх проектах, вони знаходяться у вільному доступі. Інфраструктура крос-платформного фреймворку Xamarin постійно та широко вдосконалюється, розширюючи нові можливості. Переглядаючи певні технології потрібно дивитися не лише на структуру, та й на зручність розробки і поріг входження.

При розробці на крос-платформному фреймворку Xamarin виділяють два основних підходи для структурування коду:

— portable class library, тобто бібліотека, у якій код може виконуватися на різних операційних системах;

— shared project, проект у якого є спільний код для різних платформ.

### 2.1.1 Portable class library (PCL)

Коли ви створюєте проект програми або проект бібліотеки, отримана DLL обмежується роботою на певній платформі, для якої вона створена. Це заважає писати збірку для програми Windows, а потім повторно використовувати її на Xamarin.iOS та Xamarin.Android.

Однак, коли ви створюєте портативну бібліотеку класів, ви можете вибрати комбінацію платформ, на яких ви хочете запускати свій код. Вибір сумісності, який ви робите під час створення портативної бібліотеки класів, перетворюється в ідентифікатор “Профіль”, який описує, які платформи підтримує бібліотека.

У таблиці нижче наведено деякі функції, які відрізняються залежно від платформи .NET. Для написання збірки PCL, яка гарантовано працює на певних пристроях/платформах, ви просто вибираєте, яка підтримка потрібна під час створення проекту.

Переваги:

— централізоване спільне використання коду – запис і тестування коду в одному проекті, який може використовуватися іншими бібліотеками або програмами;

— на проект PCL можна легко посилатись іншими проектами у рішенні, або на вихідну збірку можна надавати спільний доступ для посилання іншим у своїх рішеннях.

Недоліки:

— оскільки одна і та ж портативна бібліотека класів є спільною для кількох програм, на бібліотеки, що відповідають платформі, не можна посилатися;

— портативна бібліотека класів може не включати класи, які інакше були б доступні як у MonoTouch, так і в Mono для Android.

Діаграма показує архітектуру крос-платформенного додатку, що використовує портативну бібліотеку класів для спільного використання коду, а також використовує ін’єкцію залежності для передачі функцій, залежних від платформи:

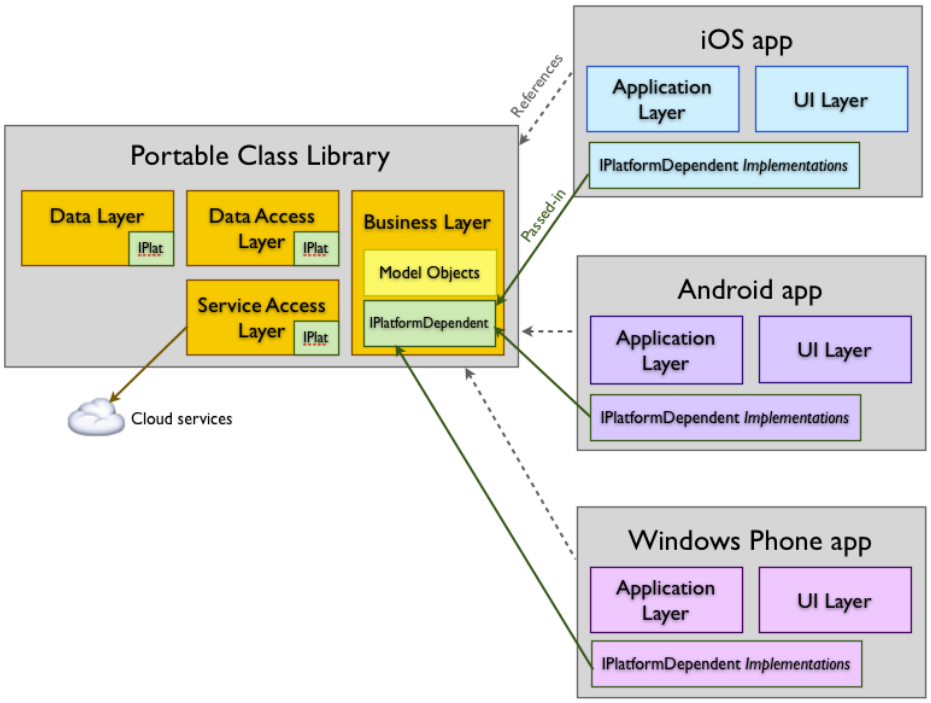


Рисунок 2.1 – Структура додатку з PCL [8]

### 2.1.2 Shared project

Іншим методом структурування коду є так званий «спільний проект». На відміну від більшості інших типів проектів, спільний проект не має вихідних даних (у формі DLL), натомість код компілюється в кожен проект, який посилається на нього. Це проілюстровано на рисунку 2.2 - концептуально весь вміст спільного проекту "копіюється" в кожен проект посилання та компілюється так, ніби він є частиною їх.

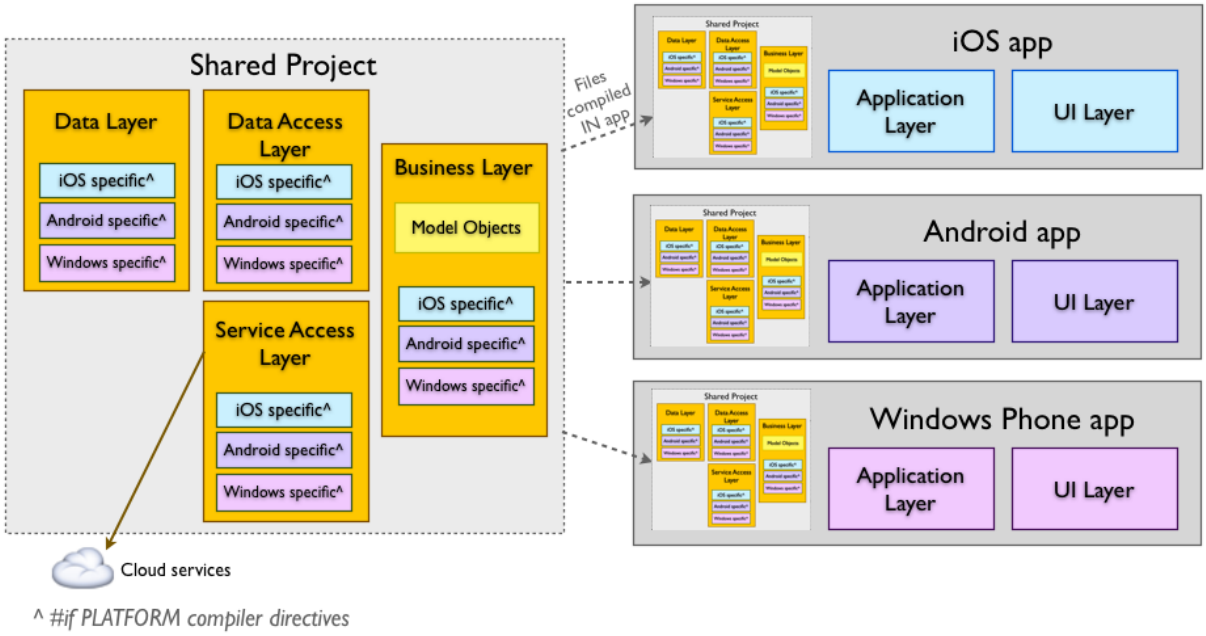


Рисунок 2.2 – Структура додатку використовуючи спільний проект

Код у спільному проекті може містити директиви компілятора, які дозволять або вимикати розділи коду залежно від того, який проект програми використовує код, що пропонується кольоровими вікнами платформи на схемі.

Спільний проект не компілюється самостійно, він існує виключно як групування файлів вихідного коду, які можна включити в інші проекти. Коли на нього посилається інший проект, код ефективно складається як частина цього проекту. Спільні проекти не можуть посилатися на будь-який інший тип проекту (включаючи інші спільні проекти). [9]

## 2.2 Інтегроване середовище розробки Visual Studio

Написання додатку на крос-платформному фреймворку Xamarin було виконано у інтегрованому середовищі розробки Visual Studio, компанії Microsoft. Через те що середовище розробки Visual Studio дозволяє досягти цієї цілі, має велику спільноту, крос-платформна, підходить для проектів любих розмірів.

Окрім стандартного редактора і відладчика, які існують в більшості середовищ IDE, Visual Studio включає в себе компілятори, засоби автозавершення коду, графічні конструктори і багато інших функцій для спрощення процесу розробки.

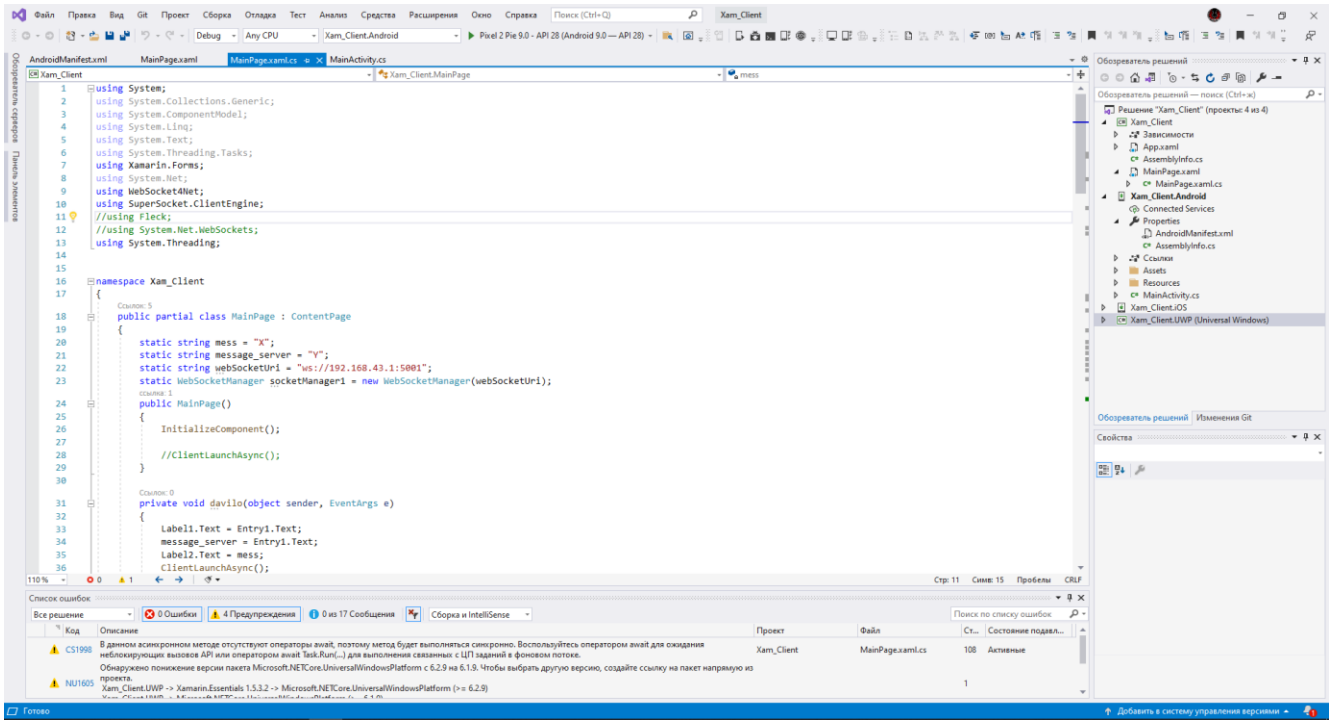


Рисунок 2.3 – Загальний вигляд середовища розробки Visual Studio

Visual Studio має різні способи індивідуальної настройки відповідно до особистого стилю та вимог до розробки. Цілий ряд параметрів можна використовувати в різних редакціях Visual Studio. Має можливість синхронізація параметрів у будь-якій редакції Visual Studio.

У Visual Studio можна індивідуально для себе налаштувати:

— загальні параметри середовища;

— кольорові теми середовища;

— регістр головного меню;

— меню і панель інструментів;

— макети вікон;

— зовнішні інструменти.

Для виконання поставленої задачі було обрано саме середовище розробки Visual Studio завдяки її можливостям:

— Live Share для спільного редагуванню коду та відладки у реальному часі не залежачи від типу програми або мови програмування;

— IntelliSense – набір функцій, що відображають відомості про код безпосередньо в редакторі і в деяких випадках автоматично створюють невеликі уривки коду;

— візуальні підказки, котрі дозволяють усувати проблеми негайно і не чекати, поки помилка буде виявлена під час збирання або запуску програми;

— CodeLens допомагає знаходити посилання на код, зміни коду, пов'язані помилки, робочі елементи, перевірки коду та модульні тести - все це не виходячи з редактора;

— перегляд визначень у якому показано визначення методу або типу, при цьому не потрібно відкривати окремий файл;

— для більш швидкого знаходження функцій інтегрованого середовища розробки та елементів коду, в Visual Studio представлений єдиний компонент пошуку.[10]

В ході виконання роботи були розглянуті Visual Studio Community - повнофункціональна, розширюється і безкоштовна інтегроване середовище

розробки для створення сучасних додатків Android, iOS і Windows, Visual Studio Professional – професіональні інструменти та служби для індивідуальних розробників або невеликих команд, Visual Studio Enterprise – інтегроване комплексне рішення для команд будь-якого розміру з високими вимогами до якості і масштабу проекту. В табл. 2.1 можна побачити порівняння редакцій Visual Studio.

Таблиця 2.1 Порівняння різних редакцій Visual Studio

| Функція | Visual Studio Community | Visual Studio Professional | Visual Studio Enterprise |
|---|---|---|---|
| Підтримка платформ розробки | + | + | + |
| Інтегрована середа розробки | + | + | + |
| IntelliTrace | - | - | + |
| Середа тестування | + | + | + |
| Крос-платформна розробка | + | + | + |
| Ручне тестування | - | - | + |

Після їх порівняння і аналізу було вирішено використовувати Visual Studio Community, через те що вона безкоштовна, та має увесь потрібний функціонал.

Для роботи в Visual Studio Community рекомендується використовувати комп'ютер з хоча б двохядерним процесором і з об'ємом оперативної пам'яті не менше 4 ГБ.

Мінімальна характеристика комп’ютера для роботи в Visual Studio Community має відповідати таким вимогам:

— процесор – двохядерний процесор з тактовою частотою не нижче 1,8 ГГц;
— об'єм оперативної пам'яті – 4 ГБ;
— тип оперативної пам’яті – DDR3;
— відеоадаптер з мінімальною роздільною здатністю 1280 на 720 пікселів;
— об’єм пам’яті – 210 ГБ;
— потужність блока живлення - 500 Вт.

Саме через безкоштовну версію, офіційну підтримку, велику кількість бібліотек, додаткову кількість все можливих вікон та вкладок, можливість гнучкого налаштування середовища та підтримку різних мов програмування Visual Studio є найкращим варіантом середовища розробки для спеціалістів будь якого рівня.

### 2.3 Протокол WebSocket

WebSocket — це комп'ютерний протокол зв'язку, що забезпечує повнодуплексні канали зв'язку через єдине TCP-з'єднання. Протокол WebSocket був стандартизований IETF як RFC 6455 в 2011 році, а API WebSocket в Web IDL стандартизований W3C.

WebSocket відрізняється від HTTP. Обидва протоколи розташовані на рівні 7 в моделі OSI і залежать від TCP на транспортному рівні. Хоча вони різні, RFC 6455 зазначає, що WebSocket «призначений для роботи над HTTP-портами 443 і 80, а також для підтримки HTTP-проксі-серверів і посередників», тим самим роблячи його сумісним з протоколом HTTP.

WebSockets починають життя як стандартний запит і відповідь HTTP. У межах цього ланцюжка відповідей на запит клієнт просить відкрити підключення WebSocket, і сервер відповідає (якщо це можливо). Якщо рукостискання було успішним, клієнт та сервер погоджуються

використовувати існуюче з'єднання TCP/IP, встановлене для запиту HTTP, як з'єднання WebSocket. Тепер дані можуть передаватися через це з’єднання за допомогою базового протоколу повідомлень у рамці. Як тільки обидві сторони визнають, що з'єднання WebSocket слід розірвати, з'єднання TCP розривається.

Для досягнення сумісності WebSocket “рукостискання” використовує заголовок HTTP Upgrade для переходу з протоколу HTTP на протокол WebSocket. Рукостискання — це процес, який забезпечує синхронізацію сервера зі своїми клієнтами. Рукостискання є основною концепцією протоколу WebSocket. На рис 2.4 зображена схема підключення по протоколу WebSocket.

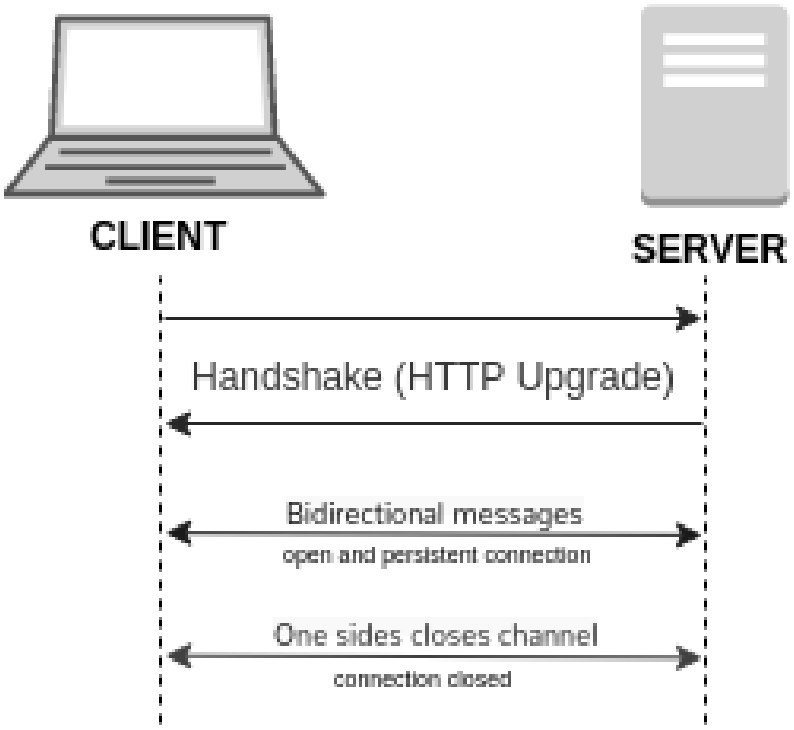


Рисунок 2.4 – Схема підключення використовуючи протокол WebSocket

Рукостискання починається із запиту/відповіді HTTP, що дозволяє серверам обробляти HTTP-з'єднання, а також з'єднання WebSocket на тому самому порту. Після встановлення з'єднання зв'язок переходить на двонаправлений двійковий протокол, який не відповідає протоколу HTTP.

На рис. 2.5 зображено рукостискання серверу з різними пристроями.

Основні особливості WebSocket такі:

- протокол WebSocket стандартизується, що означає, що за допомогою цього протоколу можливий зв’язок між веб-серверами та клієнтами в режимі реального часу.
- WebSocket перетворюються на крос-платформний стандарт для спілкування в реальному часі між клієнтом та сервером.
- цей стандарт дозволяє використовувати нові типи програм. Підприємства для веб-додатків у реальному часі можуть пришвидшитись за допомогою цієї технології.
- найбільша перевага WebSocket полягає в тому, що він забезпечує двосторонній зв'язок (повний дуплекс) над одним TCP-з'єднанням.

Протокол WebSocket забезпечує взаємодію між веб-браузером (або іншим клієнтським додатком) та веб-сервером із меншими накладними витратами, ніж напівдуплексні альтернативи, такі як опитування HTTP, полегшуючи передачу даних у реальному часі від і до сервера.

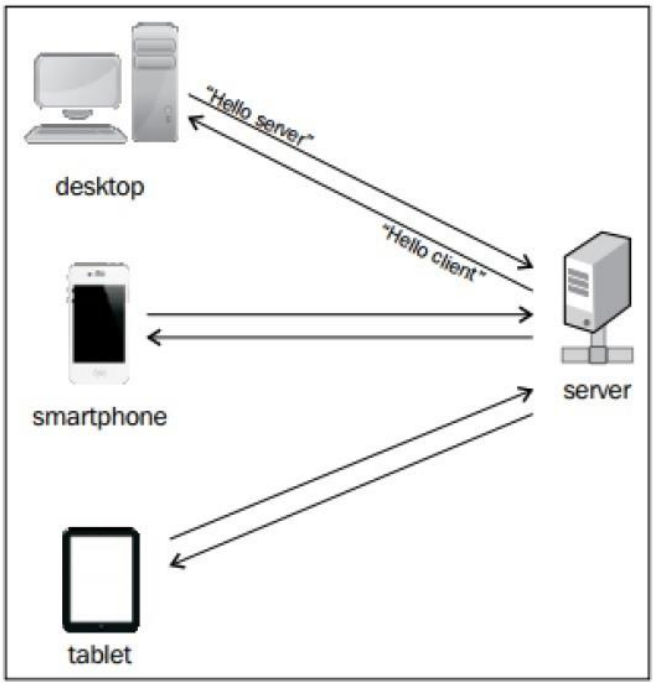


Рисунок 2.5 – Діаграма рукостискання

Це стає можливим завдяки забезпеченню стандартизованого способу відправки сервером вмісту клієнту без попереднього запиту клієнта та дозволяючи передавати повідомлення вперед і назад, залишаючи зв’язок відкритим. Таким чином, між клієнтом і сервером може відбуватися двостороння розмова, що триває.[11]

Як тільки WebSocket був створен, він починає встановлювати зв’язок. Протокол WebSocket працює над TCP.

Це означає, що при з'єднанні браузер відправляє по HTTP спеціальні заголовки, питаючи: «чи підтримує сервер WebSocket?». Якщо сервер в відповідних заголовках відповідає «так, підтримую», то далі HTTP припиняється і спілкування йде на спеціальному протоколі WebSocket, який вже не має з HTTP нічого спільного.

Передача даних у WebSocket складається з «фреймів», фрагментів даних, які можуть бути відправлені будь-якою стороною, і які можуть бути наступних видів:

— «текстові фрейми» - містять текстові дані, які надсилаються;

— «бінарні фрейми» - містять бінарні дані;

— «пінг-понг фрейми» використовується для перевірки з’єднання, відправляються з серверу, браузер реагує на них автоматично;

— також є «фрейм закриття з’єднання» і деякі інші службові фрейми.

Зазвичай, коли сторона хоче закрити з'єднання, вони відправляють «фрейм закриття з'єднання» з кодом закриття і вказують причину у вигляді тексту.

WebSockets дозволяють надсилати дані на основі повідомлень, подібні до UDP, але з надійністю TCP. WebSocket використовує HTTP як початковий транспортний механізм, але підтримує TCP-з'єднання живим після отримання відповіді HTTP, щоб його можна було використовувати для надсилання повідомлень між клієнтом та сервером. WebSockets дозволяють нам створювати програми в режимі реального часу без використання довгого опитування.

HTTP має власний набір схем, таких як http і https. Протокол WebSoket також має подібну схему, визначену у своєму шаблоні URL-адреси.

На рис 2.6 зображено URL-адресу WebSocket.

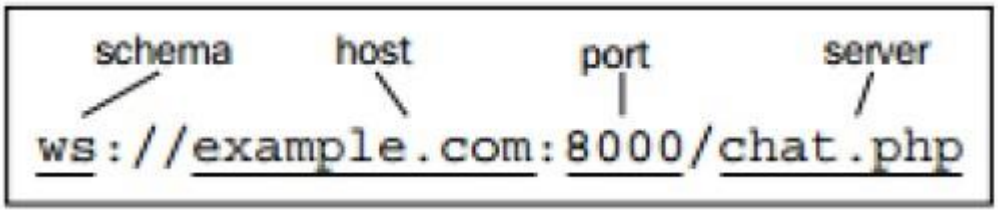


Рисунок 2.6 – Зображення URL-адресу[15]

У даному дипломного проекті є клієнтська частина, та серверна частина. Клієнтська частина відповідає за переправлення повідомлення до серверної частини, яка у свою чергу перенаправляє повідомлення, через USB, до STM32.

```
static string webSocketUri = "ws://192.168.43.1:5001";
static WebSocketManager socketManager1 = new WebSocketManager(webSocketUri);
```

Рисунок 2.7 – Визначення Uri

Як зображено на рисунку 2.7, спочатку ми маємо прописати Uri для визначення куди має підключатися клієнтський додаток.

Далі створюється з’єднання WebSocket, та створюються івенти, з’єднання відкрито, з’єднання закрито та похибка у з’єднанні.

```
ссылка: 1
public WebSocketManager(string webSocketUri)
{
    webSocket = new WebSocket(webSocketUri);
    webSocket.Opened += new EventHandler(websocket_Opened);
    webSocket.Closed += new EventHandler(websocket_Closed);
    webSocket.Error += new EventHandler<ErrorEventArgs>(websocket_Error);
    webSocket.MessageReceived += new EventHandler<MessageReceivedEventArgs>(websocket_MessageReceived);

    webSocket.Open();
    while (webSocket.State == WebSocketState.Connecting) { };   // by default webSocket4Net has AutoSendPing=true,
                                                                // so we need to wait until connection established
    if (webSocket.State != WebSocketState.Open)
    {
        throw new Exception("Connection is not opened.");
    }
}
```

Рисунок 2.8 – Перевірка з’єднання WebSocket

Далі статус з’єднання відправляється до повідомлення, яке буде висвічане на екрані додатку клієнта, це можна побачити на рисунку 2.9. При підключенні та відкритому з’єднанні «Websocket is opened», при похибці буде висвічуватися опис похибки, для її виправлення, та при закритому з’єднанні «Websocket is closed».

```
ссылка: 1
private void websocket_Opened(object sender, EventArgs e)
{
    mess = mess + "Websocket is opened.";
}
ссылка: 1
private void websocket_Error(object sender, ErrorEventArgs e)
{
    mess = mess + e.Exception.Message;
}
ссылка: 1
private void websocket_Closed(object sender, EventArgs e)
{
    mess = mess + "Websocket is closed.";
}
```

Рисунок 2.9 – Повідомлення до додатку клієнта про стан з’єднання

Після натискання клієнтом, наприклад, кнопки їзди уперед, буде виконуватися код, написаний на рисунку 2.10. Він присвоює строчці «message_server» літеру, у даному випадку «f». Яка далі потрапляє до функції на рисунку 2.11, через яку і передається повідомлення.

```
private void btnFwd(object sender, EventArgs e)
{
    message_server = "f";
    Label2.Text = mess;
    ClientLaunchAsync();
}
```

Рисунок 2.10 – Код кнопки їзди уперед

```
private static async void ClientLaunchAsync()
{
    var socketManager = socketManager1;
    socketManager.Send(message_server);
}
```

Рисунок 2.11 – Функція відправлення повідомлення

Після закінчення роботи та закриття зв’язку виконується функція Close. Яка закриває з’єднання між клієнтом та сервером, код зображено на рис. 2.12.

```
Ссылок: 0
public void Close()
{
    mess = mess + "Closing websocket...";
    webSocket.Close();
}
```

Рисунок 2.12 – Код закриття з’єднання

Зі сторони серверу все виглядає так. При створенні серверу ми прописуємо Uri. Далі додається цей сокет до листу з’єднань. Після обробки коду, якщо встановлюється з’єднання між клієнтом та сервером, то виводиться повідомлення о успішному приєднанні.

```
server.Start(socket =>
{
    client_message = client_message +"start+";

    socket.OnOpen = () =>
    {
        clients.Add(socket);
        client_message = client_message + "on_open+";
        foreach (IWebSocketConnection client in clients)
        {
            if (socket.ConnectionInfo.Id != client.ConnectionInfo.Id)
            {
                client.Send(socket.ConnectionInfo.Id + " connection proceed");
            }
            else
            {
                client.Send("Connected");
            }
        }
    };
```

Рисунок 2.13 – Код включення серверу

```
socket.OnBinary = data =>
{
    client_message = client_message + "on_binary+";
    foreach (IWebSocketConnection client in clients)
    {
        client.Send(data);
    }
};
```

Рисунок 2.14 – Код попередження

На рисунку 2.14 зображена частина коду, яка до даних підпис, що ці данні знаходяться у бінарному вигляді.

```
socket.OnMessage = message =>
{
    client_message = client_message + "on_message+" + socket.ConnectionInfo.Id + "_Say_" + message;
    client_message_USB = message;
    foreach (IWebSocketConnection client in clients)
    {
        client.Send(message);
    }
};
```

Рисунок 2.15 – Код отримання даних

На рисунку 2.15 можна побачити код який відповідає за отримання даних на сервері. Також він відправляє це повідомлення на USB. Для подальшого переправлення до STM32.

```
socket.OnClose = () =>
{
    clients.Remove(socket);
    client_message = client_message + "on_close+";
    foreach (IWebSocketConnection client in clients)
    {
        if (socket.ConnectionInfo.Id != client.ConnectionInfo.Id)
        {
            client.Send(socket.ConnectionInfo.Id + " disconnected");
        }

    };
};
```

Рисунок 2.16 – Закриття з’єднання

Як зображено на рис. 2.16, після закриття з’єднання на сервер приходить повідомлення о закритті.

## 2.4 Опис роботи з USB у Android: UsbManager, UsbDevice, UsbInterface, UsbEndpoint

У цьому дипломному проекті USB використовується для зв’язку між телефоном-сервером та платою STM32, яка виконує переправлення та перетворення отриманого повідомлення у зрозумілий для робота-гіда сигнал.

У цьому пункті буде описано загальна характеристика USB та як він працює з Android.

USB (Universal Serial Bus) – це галузевий стандарт, який встановлює специфікації кабелів та роз’ємів та протоколи для підключення, зв’язку та живлення (взаємодії) між комп’ютерами, периферійними пристроями та іншими комп’ютерами.

USB був розроблений для стандартизації підключення периферійних пристроїв до персональних комп’ютерів як для зв’язку, так і для подачі електроенергії.

Він значною мірою замінив такі інтерфейси, як послідовний порт та паралельний порт, і став звичним явищем для широкого кола пристроїв. Прикладами периферійних пристроїв, які підключаються через USB, є комп'ютерні клавіатури та миші, відеокамери, принтери, мобільні (портативні) цифрові телефони, дисководи та мережеві адаптери.

З точки зору користувача комп'ютера, інтерфейс USB покращує зручність використання кількома способами:

- інтерфейс USB самоконфігурується, усуваючи необхідність користувачеві регулювати налаштування пристрою для швидкості або формату даних, а також конфігурувати переривання, адреси вводу/виводу або прямі канали доступу до пам'яті;
- роз'єми USB стандартизовані на хості;
- USB використовує всі переваги додаткової обчислювальної потужності, яку можна економічно використовувати в периферійних пристроях, щоб вони могли управляти собою;

— невеликі пристрої можна живити безпосередньо від інтерфейсу USB, усуваючи необхідність у додаткових кабелях живлення;
— інтерфейс USB визначає протоколи для відновлення від поширених помилок, покращуючи надійність порівняно з попередніми інтерфейсами;
— встановлення пристрою, який покладається на стандарт USB, вимагає мінімальних дій оператора.

Але як і у інших стандартах, USB має ряд обмежень:

— кабелі USB мають обмежену довжину, оскільки стандарт був призначений для периферійних пристроїв на одній стільниці, а не між кімнатами чи будівлями;
— USB має сувору топологію деревоподібної мережі та протокол master/slave для адресації периферійних пристроїв;
— хост не може транслювати сигнали на всю периферію одночасно - кожен повинен вирішуватися окремо. [12]

Android-пристрій може мати різні режими, основні з яких: USB Device mode або USB Host mode.

Коли ми підключаємо пристрій до комп’ютера, то використовується режим USB Device mode. У цьому випадку живлення йде від комп’ютера до нашого пристрою і пристрій заряджається.

Якщо ми підключаємо який-небудь пристрій до телефону або планшету через USB-порт, то в такому випадку використовується режим USB Host. У цьому режимі вже телефон забезпечую живлення для підключеного пристрою. Такий режим ще називають OTG (On-The-Go). На рисунку 2.15 та 2.16 зображення відрізнення його від звичайного кабелю. Ця технологія призначена для з’єднання USB-пристроїв безпосередньо між собою без використання персонального комп’ютеру.

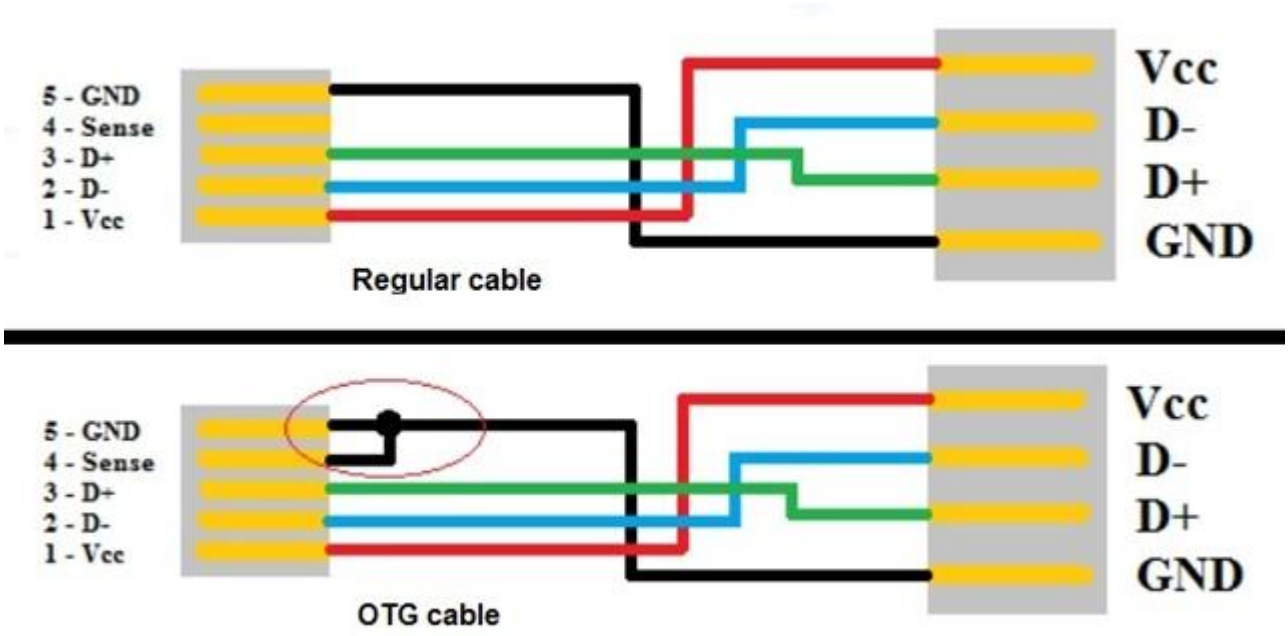


Рисунок 2.17 – Відрізнення між звичайним кабелем та OTG

USB OTG представляє концепцію пристрою, який виконує роль хоста та пристрою - коли два USB-пристрої підключені, а одним з них є USB-пристрій OTG, вони встановлюють лінію зв'язку. Прилад, що контролює зв'язок, називається хостом, а інший - пристроєм або периферією.


(а) (б)

Рисунок 2.18 – Порівняння USB OTG (а) та звичайного USB (б)

Пристрій OTG A є постачальником енергії, а пристрій OTG B - споживачем енергії. У конфігурації посилання за замовчуванням A-пристрій виконує функцію USB-хоста, а B-пристрій виконує функцію USB-пристрою або периферійного пристрою USB. У дипломному проекті підключення між

телефоном на базі Android, та STM32 яка оброблює отримане повідомлення, як раз проходить через технологію USB OTG.

Початкова роль кожного пристрою була визначена тим, який міні-штекер користувач вставляє у роз'їм.

Також треба відмітити USB Accessory mode, який з'явився з версією Android 3.0. Режим USB Accessory дозволяє користувачам підключати апаратне забезпечення USB-хосту, спеціально розроблене для пристроїв на базі Android. Аксесуари повинні відповідати протоколу Android accessory, зазначеному в документації до комплекту розробки аксесуарів Android. На рисунку 2.19 зображена ілюстрація роботи пристроїв андроїд у Host та Accessory Mode.

Це дозволяє пристроям на базі Android, які не можуть виконувати функції хоста USB, все одно взаємодіяти з обладнанням USB.

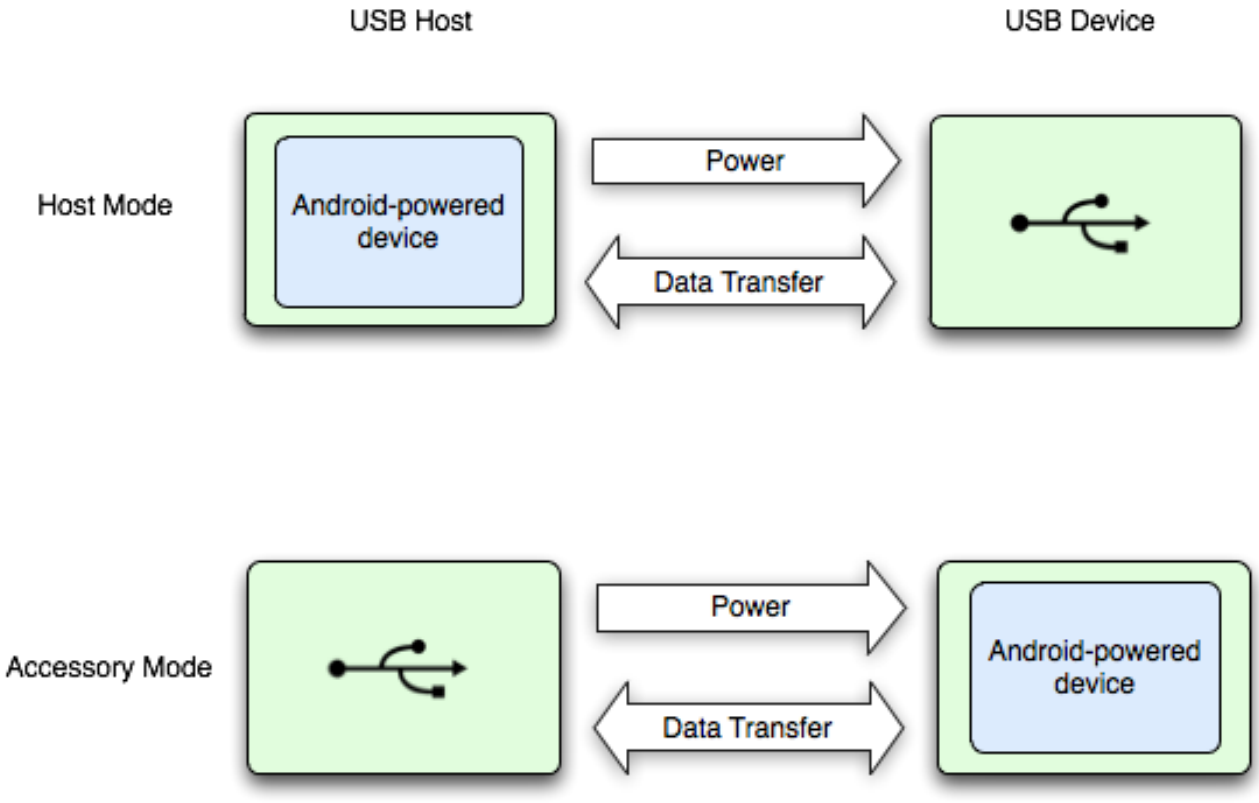


Рисунок 2.19 – Ілюстрація роботи пристроїв Android у описаних режимах

Щоб розібратися в режимі роботи USB Host для операційної системи Android, треба розуміти, з якими класами пристроїв USB він працює. Далі описані класи використовуються при взаємодії з пристроями USB Device.

В таблиці 2.2 описані класи USB Host API у пакеті android.hardware.usb.

Таблиця 2.2 – Класи USB Host API

| Клас | Опис |
|---|---|
| UsbManager | Дозволяє виробляти енумерації підключених пристроїв USB і організувати обмін з ними |
| UsbDevice | Являє собою підключений пристрій USB, і містить методи для доступу до ідентифікаційної інформації, інтерфейсів і кінцевих точок |
| UsbInterface | Являє собою інтерфейс пристрою USB визначаючий набір функціоналу пристрою. Пристрій може мати один або декілька інтерфейсів, з якими можна взаємодіяти. |
| UsbEndpoint | Являє кінцеву точку інтерфейсу (interface endpoint), яка дає канал зв'язку для цього інтерфейсу. Інтерфейс може мати 1 або більшу кількість кінцевих точок, і зазвичай має окремо кінцеву точку для введення і окремо кінцеву точку для виведення, щоб можна було вести обмін даними в двох напрямках. |
| UsbRequest | Представляє собою асинхронний запит для обміну з пристроєм через UsbDeviceConnection |
| UsbDeviceConnection | Представляє з'єднання з пристроєм, який переносить дані між Android і кінцевими точками. Цей клас дозволяє відправити блок даних туди і назад, синхронно або асинхронно. |

У більшості випадків потрібно використовувати всі ці класи (UsbRequest потрібно тільки якщо Ви здійснюєте асинхронну зв'язок), коли відбувається взаємодія з пристроєм USB device.

Зазвичай задіються UsbManager для отримання доступу до потрібного UsbDevice. Коли отримали пристрій USB, то для обміну даними потрібно знайти відповідний інтерфейс UsbInterface і кінцеву точку UsbEndpoint на цьому інтерфейсі. Як тільки Ви отримали правильну кінцеву точку, відкрийте з'єднання UsbDeviceConnection для обміну даними з пристроєм USB. [13]

Частина коду зображена на рис 2.20 потрібна для отримання дозволу роботи USB пристрою з Android.

```
private static String ACTION_USB_PERMISSION = "com.android.example.USB_PERMISSION";

ссылка: 1
public class MyBroadcastReceiver : BroadcastReceiver
{
    private static String ACTION_USB_PERMISSION = "com.android.example.USB_PERMISSION";
    private static String ACTION_USB_DEVICE_DETACHED = "";
    Ссылок: 0
    public override void OnReceive(Context context, Intent intent)
    {
        string action = intent.Action;
        if (ACTION_USB_PERMISSION == action)
        {
            lock (this)
            {
                UsbDevice device = (UsbDevice)intent.GetParcelableExtra(UsbManager.ExtraDevice);

                if (intent.GetBooleanExtra(UsbManager.ExtraPermissionGranted, false))
                {
                    if (device != null)
                    {
```

Рисунок 2.20 – Отримання дозволу для роботи з USB

На рис 2.20 зображена відключення від USB девайсу. Частина Intent.GetParcelableExtra потрібна для отримання розширених даних.

```
if (UsbManager.ActionUsbAccessoryDetached.Equals(action))
{
    UsbDevice device = (UsbDevice)intent.GetParcelableExtra(UsbManager.ExtraDevice);
    if (device != null)
    {
        //Вызовите Ваш метод, который сделает очистку и закроет обмен с устройством USB.
    }

}
```

Рисунок 2.20 – Код при відключенні USB

```
private static void SetTimer()
{
    aTimer = new System.Timers.Timer(10000);
    // Hook up the Elapsed event for the timer.
    aTimer.Elapsed += OnTimedEvent;
    aTimer.AutoReset = true;
    aTimer.Enabled = true;
}
```

Рисунок 2.21 – Створення таймеру

На рис 2.21 зображено код створення таймеру з десяти секундним інтервалом.

```
if (intf.EndpointCount == 0)
{
    lgView = "could not find endpoint";
    return;
}
else
{
    lgView = "^^^" + intf.EndpointCount.ToString();
}


UsbEndpoint epIN = null;
UsbEndpoint epOUT = null;
```

Рисунок 2.22 – Визначення кінцевих точок

На рис 2.22 зображено частина коду яка визначає кінцеві точки. Кінцеві точки (Endpoint - кінцева точка) – це один кінець каналу зв'язку. Коли API взаємодіє з іншою системою, точки дотику зв'язку з цим вважаються кінцевими точками. [14]

```
        public static string ConnectUSB(string message)
        {
            string page = message;
#if __ANDROID__

            page = DependencyService.Get<ITextToSpeech>().Speak(message);
#endif
            return page;
        }
```

Рисунок 2.23 – Частина коду для перетворення

```
public string Speak(string message)
{
    var context = MainActivity.Instance;
    //Intent intent_ = Intent;
    var intent_ = new Intent(Intent.ActionDefault);
    //context.StartActivity(intent_);
```

Рисунок 2.24 – Частина коду для перетворення

```
public interface ITextToSpeech
{
    Ссылок: 2
    string Speak(string text);
}
```

Рисунок 2.25 – Частина коду для перетворення

На рис. 2.23, 2.24, 2.25 зображені частини коду які використовуються для передачі та перетворення платформонезависимого коду в платформнозависимий код

Рис 2.26 показує пошук кінцевих точок для передачі даних по перериванням.

```
for (int i = 0; i < intf.EndpointCount; i++)
{
    UsbEndpoint endpoint = intf.GetEndpoint(i);
    if (endpoint.Type == UsbAddressing.XferBulk)
    {
        if (endpoint.Direction == UsbAddressing.In)
        {
            epIN = endpoint;
            lgView = lgView + "IN endpoint: " + intf.GetEndpoint(i);
        }
        else
        {
            epOUT = endpoint;
            lgView = lgView + "OUT endpoint: " + intf.GetEndpoint(i);
        }
    }
    else
    {
        lgView = lgView + "no endpoints for INTERRUPT_TRANSFER";
    }
}

mEndpointIntr = epOUT;
```

Рисунок 2.26 – Пошук кінцевих точок

Для APIs кінцева точка може включати в себе URL сервера або служби. Кожна кінцева точка - це місце, з якого APIs може отримати доступ до ресурсів, необхідних для виконання своєї функції. Кінцеві точки виконують конкретне завдання, приймають параметри і повертають дані клієнту.

API працюють із використанням «запитів» та «відповідей». Коли API запитує інформацію від веб-програми або веб-сервера, він отримає відповідь. Місце, куди API надсилають запити, і де живе ресурс, називається кінцевою точкою. [14]

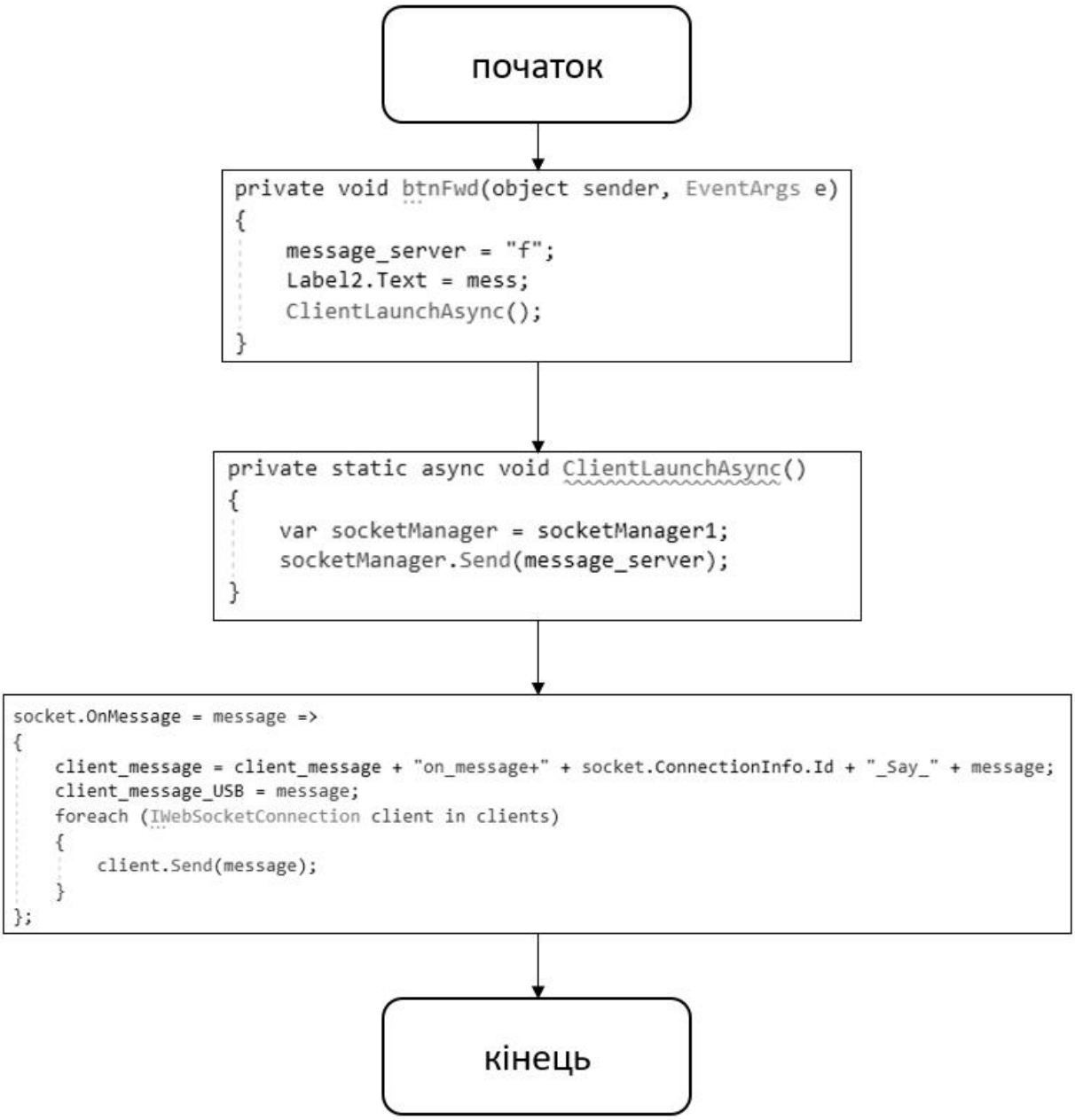


Рисунок 2.27 – Блок схема передання даних

На рисунку 2.27 зображено блок схему з клієнтського додатку до серверу і далі до STM32. У першому блоці ми маємо дані які далі, через message_server направляються далі до функції ClientLaunchAsync(), у якій виконується функція відправки даних по сокету до серверу. Після потрапляння до серверу виконується код у події OnMessage, і далі через client_message_USB відправляється до STM32.

# 3 РОЗРОБКА ІНТЕРФЕЙСУ З ЕЛЕМЕНТАМИ КЕРУВАННЯ

## 3.1 Створення інтерфейсу у Xamarin

Для створення інтерфейсу з елементами керування роботу-гіду, спочатку треба було визначити функціональні потреби цього інтерфейсу.

Розроблюючий інтерфейс робота-гіда має задовольняти наступні функціональні потреби:

— бути інтуїтивно зрозумілим;

— усі надписи мають бути читаємі;

— кнопки мають бути правильно названі;

— зручне розташування кнопок;

— розмір має автоматично підстроюватися для різних екранів;

— має бути зручним у використані;

Сам інтерфейсу написан XAML (eXtensible Application Markup Language, тобто розширювана мова розмітки), яка являє собою способом опису графічного інтерфейсу.

Розмітка XAML добре читається як людиною, так і комп’ютером, у контексті розробки на Android вона використовується для опису макетів використовуваних компонентів і їх атрибутів: розміру, позиції, кольору, розміру тексту, полів і відступів. Visual Studio розбирає розмітку XML, щоб відобразити макет в макетному редакторі і згенерувати код C#, який вже формує графічний інтерфейс на стадії виконання.

### 3.1.1 Основи XAML

Мова XAML - це мова на основі XML, створений корпорацією Майкрософт в якості альтернативи програмування коду для створення екземплярів і ініціалізації об'єктів, а також організації цих об'єктів в ієрархіях типу «батьки-нащадки».

XAML дозволяє розробникам визначати призначені для користувача інтерфейси в Xamarin.Forms додатках, використовуючи розмітку, а не код. Код XAML не потрібен в Xamarin.Forms програмі, але часто є більш стислим і більш наочним, ніж еквівалентний код.

У файлі XAML розробник Xamarin.Forms може визначати користувальницькі інтерфейси, використовуючи всі подання, макети та сторінки Xamarin.Forms, а також власні класи.

Файл XAML може бути скомпільований або вбудований у виконуваний файл. У будь-якому випадку, інформація XAML аналізується під час побудови, щоб знаходити іменовані об'єкти, і знову під час виконання, щоб створювати екземпляри, ініціалізувати об'єкти та встановлювати зв'язки між цими об'єктами та програмним кодом.

XAML має кілька переваг перед еквівалентним кодом:

— XAML часто є більш стислим і читабельним, ніж еквівалентний код;

— ієрархія «батьки-нащадки» елементів, властива XML, дозволяє XAML імітувати з більшою візуальною чіткістю ієрархію батьків-дочірніх об'єктів інтерфейсу користувача;

— XAML може бути легко написаний від руки програмістами, але також надає можливість перекладатись та створюватися засобами візуального дизайну.

Є також недоліки, здебільшого пов’язані з обмеженнями, властивими мовам розмітки:

— XAML не може містити код. Усі обробники подій повинні бути визначені у файлі коду;

— XAML не може містити циклів для повторної обробки;

— XAML не може містити умовної обробки;

— XAML не може викликати методи;

— XAML не може створювати екземпляри класів, які не визначають безпараметричний конструктор.

### 3.1.2 Приклад написання кнопки

Далі буде наведено приклад написання простішого додатку з однією кнопкою, яка буде міняти текст на цій же кнопці, з «Click on me» на «Click on me again».

На рис 3.1 можна побачити цей приклад, написання мовою XAML у крос-платформному фреймворку Xamarin.

```
<StackLayout>
    <Button
        Text="Click on me"
        Clicked="OnButtonClicked"
        />
</StackLayout>
```

Рисунок 3.1 – Код прикладу написання Button

У цьому прикладі можна побачити що параметр Text відповідає за текст, який в даному прикладі це «Click on me», він буде знаходитися на кнопці, а параметр Clicked описує назву події, «OnButtonClicked» яка буде виконуватися після натискання кнопки.

Далі, на рис 3.2, написана сама функція, яка буде заміняти текст на кнопці, з «Click on me» на «Click on me again».

```
Ссылок: 0
void OnButtonClicked(object sender, EventArgs e)
{
    (sender as Button).Text = "Click me again!";
}
```

Рисунок 3.2 – Функція яка виконує подію при натисканні на кнопку

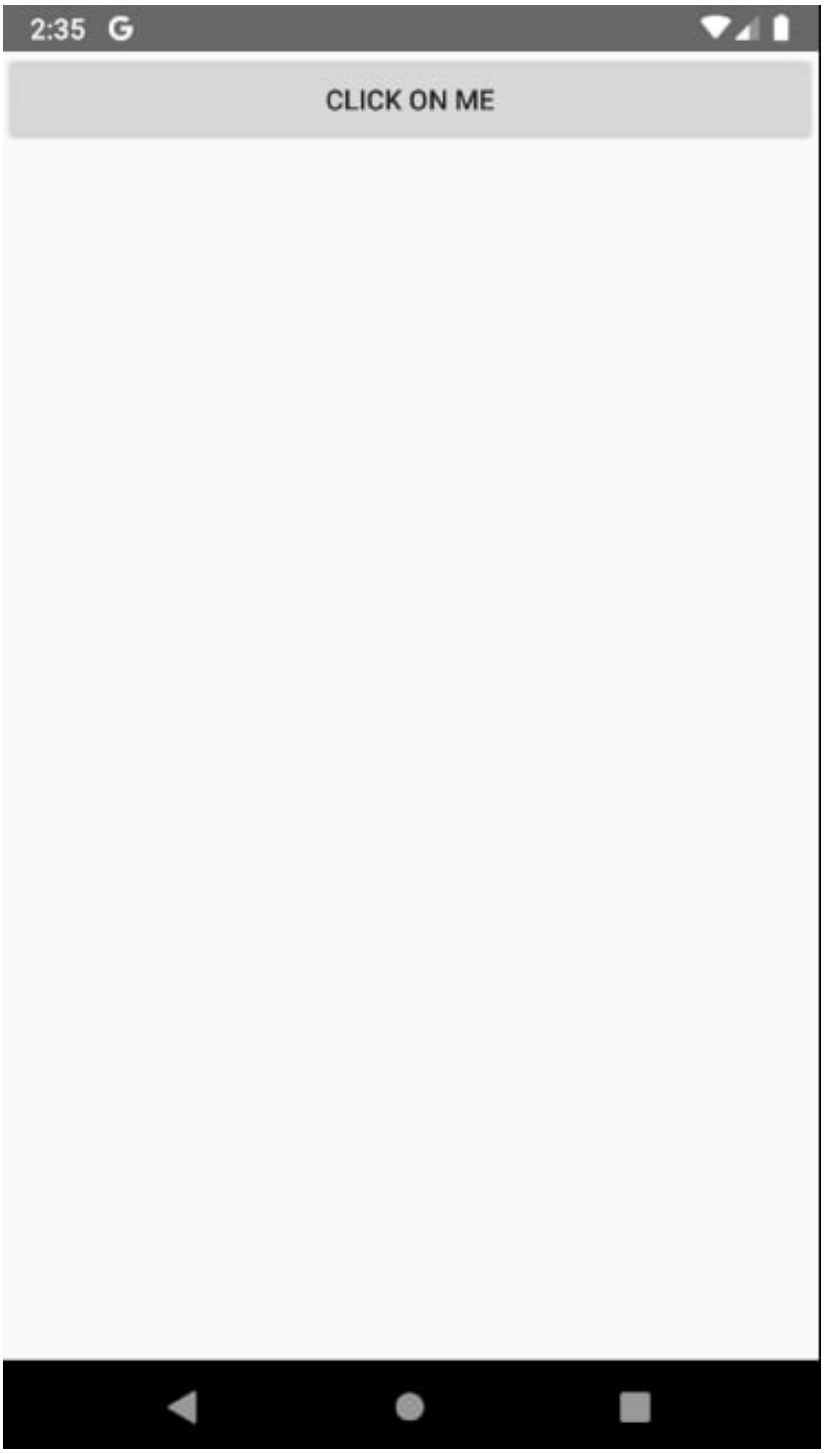


Рисунок 3.3 – Приклад кнопки до натискання на неї

На рис 3.3 ми бачимо нашу написану кнопку, у цьому прикладі в неї нема жодних стилей, але їх можна додатково прописати при описі Button. Можна обрати колір кнопки, її орієнтацію, як вертикальну так і горизонтальну, колір тексту, розміри текст та інше.

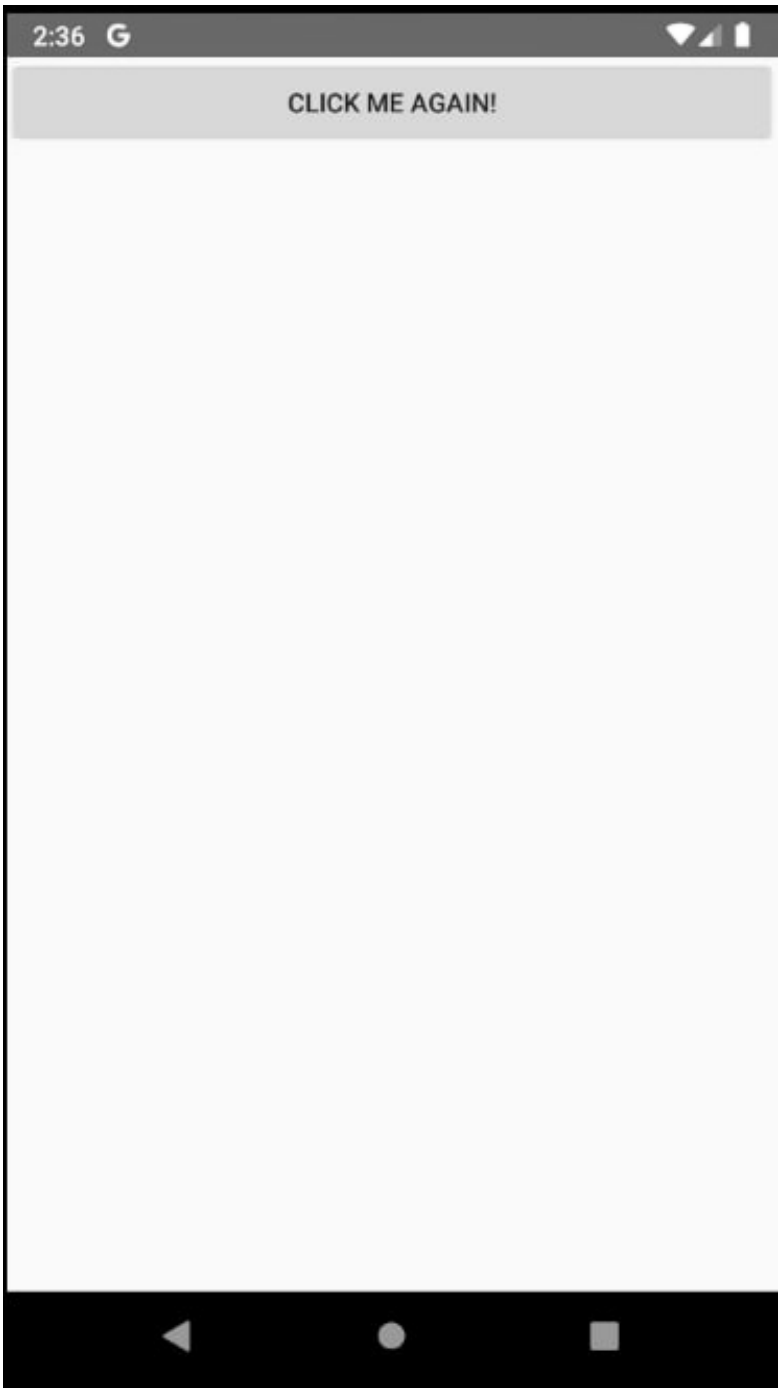


Рисунок 3.4 – Приклад кнопки після натискання на неї

Після натискання кнопки, виконується функція, яка написана на рис 3.2, при натисканні на неї текст на самій кнопці був замінено на «Click on me again!». Як ми бачимо на рис 3.4.

У кнопці ми можемо прописати будь яку функцію яку ми будемо використовувати, також можливо обрати після чого вона буде виконувати написану функцію.

У даному прикладі використовується варіант при написанні на кнопку Clicked, але також є ще інші варіанти, такі як Pressed та Released. Які будуть використовуватись під час виконання дипломного проекту.

### 3.2 Розробка інтерфейсу додатку

Перед побудовою любого додатку, спочатку треба спроектувати загальний вигляд додатку та його інтерфейсу. На рис 3.5 зображена загальна схема інтерфейсу додатку.

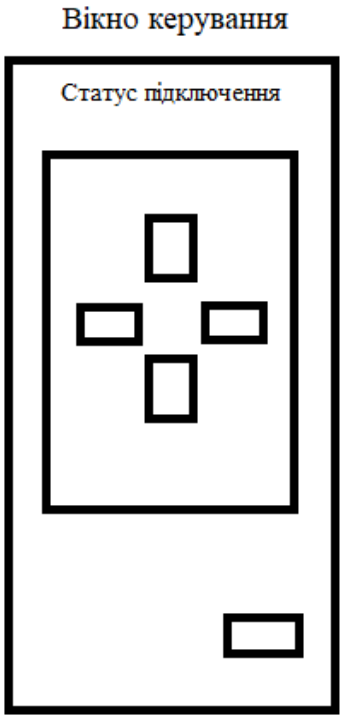


Рисунок 3.5 – Схема інтерфейсу додатку

Створивши приблизну схему інтерфейсу додатку, була почата робота над самим інтерфейсом з елементами керування.

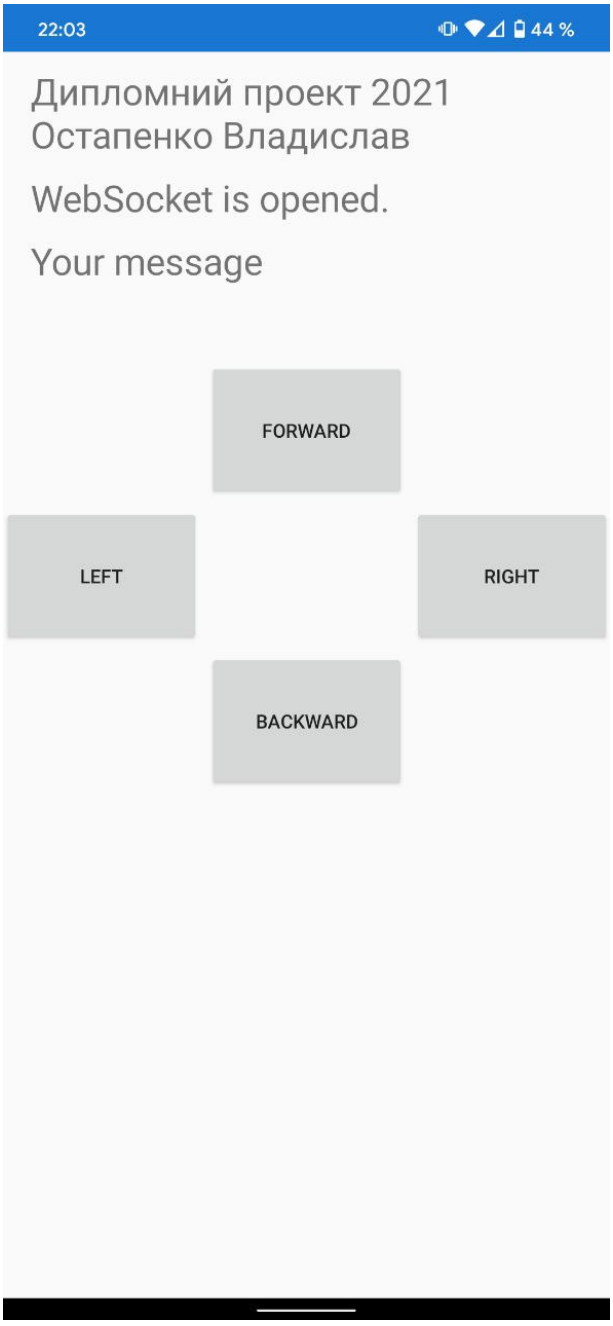


Рисунок 3.6 – Головне меню додатку

Було створено вікно з елементами керування роботом-гідом. Воно має статус строку у якій пишеться статус підключення та чотири кнопки керування:

— Forward – для руху уперед;

— Backward – для руху назад;

— Left – для повороту вліво;

— Right – для повороту вправо;

```
<StackLayout Margin="20,10,20,50" Spacing="10">
    <Label x:Name="Label1" FontSize="27" Text="Дипломний проект 2021 Остапенко Владислав"></Label>
    <Label x:Name="Label3" FontSize="27" Text="WebSocket is opened."></Label>
    <Label x:Name="Label2" FontSize="27" Text="Your message"></Label>
</StackLayout>
```

Рисунок 3.7 – Код створення текстового поля

Для створення інтерфейсу на мові XAML, потрібно прописати параметри розмітки у файлі з форматом .xaml, спочатку було створено батьківський StackLayout, взагалі StackLayout визначає розміщення елементів. Він організовую позиціювання дочірніх елементів у вигляді стеку, або горизонтально, або вертикально.

На рис. 3.8 зображено два StackLayout, перший це батьківський, другий це дочірній.

```
<StackLayout>
    <StackLayout>

    </StackLayout>

</StackLayout>
```

Рисунок 3.8 – StackLayout

На рис 3.7 та рис 3.9 зображено код та текстовий елемент який він створює в додатку, в який можна самому записати текст, або змінювати його через кнопку.

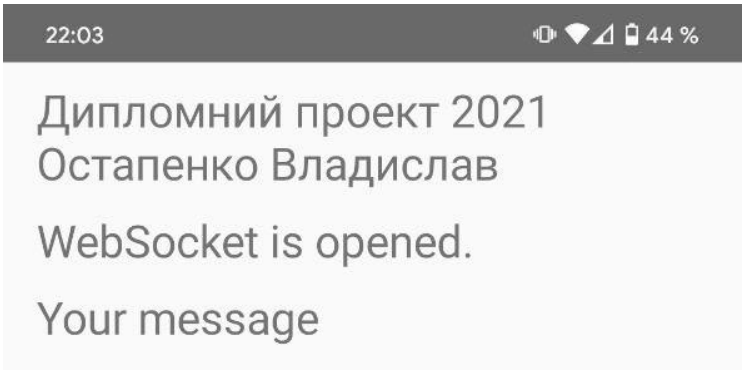


Рисунок 3.9 – Вигляд текстового поля у додатку

Далі був написан Grid, з англійського сітка. Grid створює сітку елементів, можна визначити кількість стовпців додаючи <ColumnDefinition/> у <Grid.ColumnDefinitions>, також можна визначити кількість рядків вписуючи <RowDefinitions/> у <Grid.ColumnDefinitions>, це можна побачити на рис 3.10.

```
<Grid x:Name="ControlGrid" VerticalOptions="Center">
    <Grid.RowDefinitions>
        <RowDefinition Height="auto" />
        <RowDefinition Height="auto" />
        <RowDefinition Height="auto" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
```

Рисунок 3.10 – Код сітки

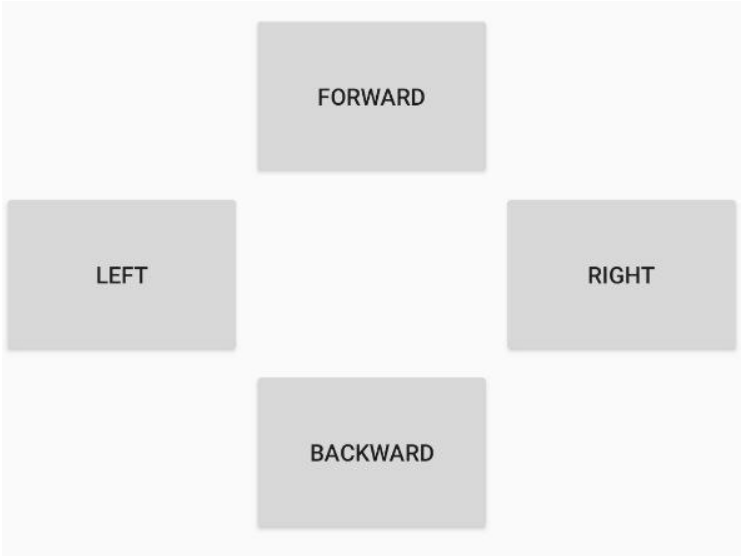


Рисунок 3.11 – Елементи керування

На рис 3.11 можна побачити сам вигляд елементів керування. Як було описано вище, це було побудовано за допомогою сітки Grid.

Самі кнопки описуються так як зображено на рис 3.12. Кнопку можна змінювати використовуючи різні параметри, такі як:

Text – визначає який текст буде зображено на кнопці;

— VerticalOptions та HorizontalOptions – визначає положення елементу в просторі;
— HeightRequest та WidthRequest – робить запит на висоту, або ширину об’єкту, якщо це можливо зробити;
— Grid.Row та Grid.Column – визначає місце у сітці Grid;
— Clicked – при натисканні на кнопку, буде виконуватися визначена функція.

```
<Button Text="Left"
        VerticalOptions="CenterAndExpand"
        HorizontalOptions="Center"
        WidthRequest="150"
        HeightRequest="100"
        Grid.Row="1"
        Grid.Column="0"
        Clicked="btnLeft"/>
```

Рисунок 3.12 – Код елементу Button

У даному випадку btnLeft виконує функцію призначення message_server літери «l», як зображено на рис. 3.13.

```
private void btnLeft(object sender, EventArgs e)
{
    message_server = "l";
    Label2.Text = mess;
    ClientLaunchAsync();
}
```

Рисунок 3.13 Функція btnLeft

Створений інтерфейс має легкий дизайн, великі кнопки, які використовуються як елементи керування. Має інформаційні поля, у яких виводяться різні дані. Масштабується під різні розміри екранів, там може бути використаним на різних платформах.

# ВИСНОВКИ

Під час виконання даного дипломного проекту було розглянуто сучасні аналоги крос-платформного фреймворку Xamarin, їх можливості, преваги та недоліки. Були описані такі фреймворки: React Native, Flutter, Ionic, Phone Gap. Після їх аналізу було зроблено висновок що Xamarin являється одним з найкращих крос-платформних фреймворків для написання додатків на різні платформи. Також Xamarin був обрано через його можливість підключення до операційної системи Windows, використовуючи одноплатний комп'ютер Panda.

Була описана історія створення фреймворку Xamarin, мова програмування яка використовується для крос-платформного створення додатків. Був описан проект Mono, на якому працює Xamarin.Android та Xamarin.iOS. Показано основні стани додатків. Були описані технології та методи розробки додатків на крос-платформному фреймворку Xamarin. Також були описані різні стани activity, та їх життєві цикли при роботі додатку на Android. Описана технологія Xamarin.Forms.

Загалом у дипломному проекті:

- проведено аналіз сучасних технологій для написання крос-платформних додатків;
- проведено аналіз інтегрованого середовища Visual Studio;
- проведено розробку інтерфейсу з елементами керування для роботу-гіду;
- описані технології які використовувались;
- проведено тестування додатку.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Cross-Platform App Frameworks — [Електронний ресурс] — Режим доступу: https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2019/

2. Xamarin – Wikipedia — [Електронний ресурс] — Режим доступу: https://en.wikipedia.org/wiki/Xamarin

3. Activity Lifestyle – Xamarin — [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/en-us/xamarin/android/app-fundamentals/activity-lifecycle/

4. Building Cross-Platform iOS/Android Apps with Xamarin, Visual Studio, and C# - Part 1 by Jim Wilson — [Електронний ресурс] — Режим доступу: https://app.pluralsight.com/library/courses/cross-platform-ios-android-visualstudio-csharp/table-of-contents/.

5. Xamarin.Android Application Fundamentals — [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/en-us/xamarin/android/app-fundamentals/

6. Справочник по элементам управления — [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/ru-ru/xamarin/xamarin-forms/user-interface/controls/

7. Xamarin.Forms Device Class – Xamarin — [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/platform/device

8. Introduction to Portable Class Library — [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/pcl?tabs=windows

9. Use Shared Project to Share code – Xamarin — [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/shared-projects?tabs=windows

10. Общие сведения о Visual Studio — [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2019

11. WebSocket – Wikipedia — [Електронний ресурс] —
Режим доступу: https://en.wikipedia.org/wiki/WebSocket
12. USB – Wikipedia — [Електронний ресурс] —
Режим доступу: https://en.wikipedia.org/wiki/USB
13. Android как хост USB | android | programming — [Електронний ресурс] —
Режим доступу: http://microsin.net/programming/android/usb-host.html
14. What is an API Endpoint? — [Електронний ресурс] —
Режим доступу: https://smartbear.com/learn/performance-monitoring/api-endpoints/
15. How do WebSocket works? — [Електронний ресурс] —
Режим доступу: https://sookocheff.com/post/networking/how-do-websockets-work/