Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти - бакалавр
за освітньо-професійною програмою
«Комп'ютерні науки»
зі спеціальності
122 – Комп'ютерні науки

тема роботи:

«Розробка програмного забезпечення для підвищення доступності мобільних пристроїв для людей похилого віку»

Виконав ст. гр. КН-22-1 _____ Гречушкін І.М.

Керівник _____ Вдовиченко І.Н.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2026

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Бакалавр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« 26 » лютого 2026 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

студентові групи КН-22-1 *Гречушкіна Ігора Миколайовича*

1. **Тема кваліфікаційної роботи:** «*Розробка програмного забезпечення для підвищення доступності мобільних пристроїв для людей похилого віку*»

затверджено наказом по університету № *173с* від *30.03.2026 р.*

2. Термін здачі кваліфікаційної роботи: *10.06.2026 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 58с., додатки, презентація у Microsoft PowerPoint (9 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-2 к.т.н., доц. Вдовиченко І.Н.

Нормоконтроль к.т.н. доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*01.03.26*
2	*Розділ 1*	*05.04.26*
3	*Розділ 2*	*06.05.26*
4	*Висновки*	*28.05.26*
5	*Оформлення кваліфікаційної роботи*	*30.05.26*
6	*Підготовка презентації та графічного матеріалу*	*20.05.26*
7	*Підготовка доповіді до захисту*	*08.06.26*

6. Дата видачі завдання: *28.01.2026р.*

Керівник _____ / Вдовиченко І.Н./

7. Запевнення: *Я, Гречушкін Ігор Миколайович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студент _____ / Гречушкін І.М./

АНОТАЦІЯ

Гречушкін І.М. Розробка асистивного мобільного лаунчера для людей похилого віку : кваліфікаційна робота бакалавра : 122 – Комп’ютерні науки. Кривий Ріг. Криворізький національний університет, 2026. 58 с.

Актуальність теми полягає в тому, що стандартні інтерфейси сучасних мобільних операційних систем не враховують природні фізіологічні та когнітивні зміни людей вікової категорії 50+. Це створює інформаційні бар'єри, викликає когнітивне перевантаження та поглиблює «цифровий розрив». Оскільки існуючі ринкові аналоги мають агресивну монетизацію або перевантажений функціонал, розробка спеціалізованого асистивного програмного забезпечення (лаунчера) із фіксованою сіткою та захистом від випадкових дій є актуальним інженерним завданням.

Метою роботи є проєктування та розробка інформаційної системи у вигляді мобільного лаунчера, який шляхом оптимізації UI/UX та впровадження допоміжних функцій забезпечить комфортне й безпечне використання смартфонів людьми похилого віку.

У першому розділі сформульовано постановку інженерної задачі та обґрунтовано вибір кросплатформного фреймворку Flutter і мови Dart для подолання апаратних обмежень. Виконано порівняльний аналіз існуючих мобільних лаунчерів та виділено їхні недоліки. Сформовано технічні й функціональні вимоги до програмного продукту, побудовано діаграму прецедентів використання системи.

Ключові слова:

МОБІЛЬНИЙ ЛАУНЧЕР , ЦИФРОВИЙ РОЗРИВ , АСИСТИВНІ ТЕХНОЛОГІЇ , ЛЮДИ ПОХИЛОГО ВІКУ , FLUTTER , DART , ІНКЛЮЗИВНИЙ ДИЗАЙН , ІНТЕРФЕЙС КОРИСТУВАЧА.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасні смартфони стають дедалі складнішими для людей похилого віку через вікові зміни зору, моторики та когнітивні труднощі. Стандартні інтерфейси не враховують ці особливості, а існуючі рішення часто не відповідають потребам аудиторії 50+. Тому розробка спеціалізованого лаунчера, спрямованого на максимальне спрощення взаємодії, є актуальним завданням.

Мета роботи - створення мобільного лаунчера, який шляхом оптимізації UI/UX та допоміжних функцій забезпечить комфортне використання смартфонів людьми похилого віку.

Об'єкт дослідження - процес взаємодії користувачів старшої вікової категорії з інтерфейсами смартфонів.

Предмет дослідження - методи, алгоритми та програмні засоби підвищення доступності інтерфейсу мобільної ОС.

Завдання роботи:

- проаналізувати психофізіологічні особливості цільової аудиторії;
- виконати порівняльний огляд аналогів та виявити їхні недоліки;
- сформулювати технічні вимоги та спроєктувати архітектуру системи;
- реалізувати інтерфейс із великими елементами, спрощеною навігацією та голосовим годинником;
- провести тестування та оцінити ефективність.

Методи дослідження: системний аналіз, об'єктно-орієнтоване проєктування, порівняльний аналіз, експериментальне тестування.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРОБЛЕМИ ВИКОРИСТАННЯ СМАРТФОНІВ ЛЮДЬМИ ПОХИЛОГО ВІКУ

1.1 Аналіз фізіологічних та когнітивних бар'єрів людей похилого віку при взаємодії з інтерфейсом смартфона

Мета роботи полягає у створенні інформаційної системи у вигляді мобільного лаунчера, який шляхом оптимізації UI/UX та впровадження допоміжних функцій забезпечить комфортне використання мобільних пристроїв людьми похилого віку. Об'єктом дослідження визначено процес взаємодії користувачів старшої вікової категорії (50+ років) з інтерфейсами смартфонів. Перед початком безпосереднього проєктування програмного забезпечення необхідно визначити технічні виклики та системні обмеження, з якими стикається розробник при вирішенні задачі зниження порогу входу у використання мобільних пристроїв для людей категорії 50+.

Аналіз геріатричної літератури та досліджень у сфері людського фактору (human factors) дозволяє виділити три основні групи бар'єрів, які безпосередньо впливають на здатність людини похилого віку ефективно використовувати сенсорний екран: сенсорні (зорові) порушення, моторні порушення та когнітивні зміни [3, 4]. Кожна з цих груп потребує специфічних компенсаторних механізмів на рівні графічного інтерфейсу.

Сенсорні бар’єри (порушення зору)

Після 50 років в організмі людини неминуче відбуваються дегенеративні зміни ока. Найбільш поширеними є:

˜ Пресбіопія – вікова втрата акомодації кришталика, що унеможливлює чітке бачення на близькій відстані (стандартна відстань тримання смартфона 30-40 см стає проблемною). Дослідження показують, що у віці 60 років гострота зору на близькій відстані знижується в середньому на 60% порівняно з молодим віком [5].

˜ Зниження контрастної чутливості - людина гірше розрізняє об'єкти на тлі з близькими яскравістю або кольором. Особливо це стосується світло-сірих елементів на білому фоні або пастельних відтінків.

˜ Катаракта - помутніння кришталика призводить до загального зниження чіткості, появи ореолів навколо джерел світла та спотворення кольоросприйняття (переважання жовто-коричневих тонів).

Внаслідок цих змін до інклюзивного інтерфейсу висуваються такі вимоги:

˜ мінімальний розмір тексту - не менше 18 pt (для шрифтів без засічок) [1,2];

˜ високий контраст (співвідношення яскравості тексту і фону не менше 4.5:1 відповідно до WCAG 2.1);

˜ використання не тільки кольору, але й форми/текстури для передачі інформації;

˜ уникання мерехтливих або швидкозмінних елементів.

Моторні бар'єри (порушення дрібної моторики та тремор)

Вікові зміни нервово-м'язової системи призводять до:

˜ Зниження сили та точності пальців - особливо виражене у людей з артритом або артрозом дрібних суглобів;

˜ Есенціального тремору - мимовільних ритмічних коливань пальців частотою 4-12 Гц, що ускладнює точне націлювання на невеликі об'єкти;

˜ Сповільнення часу реакції - на 20-30% порівняно з 30-річними.

Дослідження показують, що мінімальний розмір зони натискання (touch target) для літніх людей має становити не менше 12 мм (44×44 dp для Android) [1]. Оптимальним є розмір 14-16 мм для квадратних елементів. Крім того, критичним є забезпечення достатнього простору (падингів) між елементами - не менше 4 мм, щоб уникнути випадкових натискань при треморі.

Когнітивні бар'єри (змни пам'яті, уваги та мислення)

Когнітивне старіння характеризується:

˜ Зниженням робочої (оперативної) пам'яті - людина може одночасно утримувати не більше 2-3 нових одиниць інформації (проти 5-7 у молодому віці).

˜ Інертністю мислення - труднощі з переключенням між різними завданнями та контекстами (наприклад, перехід з головного екрана в меню налаштувань та повернення назад).

˜ Зниженням здатності до навчання новим інтерфейсним метафорам - люди похилого віку покладаються на вже сформовані ментальні моделі (наприклад, для них природніше шукати функцію у списку, ніж використовувати жест «свайп»).

Архітектурні наслідки для програмного забезпечення:

1. Багаторівнева навігація є ворожою - глибина меню не повинна перевищувати 2 рівні.

2. Явні кнопки замість жестів - будь-яка дія має мати видимий та однозначний елемент керування.

3. Константність елементів - кнопки головних функцій (дзвінки, контакти, годинник) завжди мають бути на одному й тому ж місці.

4. Мінімізація вибору - на одному екрані не більше 6-8 активних елементів.

Інженерні вимоги як результат аналізу бар'єрів

На основі викладеного у процесі системного аналізу предметної області було виділено три ключові обмеження, які архітектура майбутнього продукту повинна обов'язково подолати:

1. Апаратні обмеження цільових пристроїв. Цільова аудиторія (користувачі похилого віку 50+) найчастіше використовує недорогі бюджетні смартфони на базі Android з малим об'ємом оперативної пам'яті (від 2 ГБ RAM). Лаунчер не повинен створювати додаткового фонового перевантаження та критичних витоків пам'яті на застарілих процесорах.

2. Критерії інклюзивності графічного інтерфейсу (UI). Для забезпечення великого розміру елементів керування розкладка робочого екрана жорстко обмежується матрицею не більше ніж 2×4 іконок на одному вікні. Це вимагає гнучкого інструменту кастомізації та відмальовування інтерфейсу, який не

руйнує загальну логіку розмітки при збільшенні елементів через когнітивний дискомфорт або вікові зміни зору та моторики користувачів.

3. Глибока системна інтеграція. Продукт розробляється для операційної системи Android версії 6.0 та вище для охоплення застарілих пристроїв. Технічно лаунчер виступає лише як графічна оболонка і не може повністю замінити глибоке системне меню налаштувань смартфона (наприклад, конфігурацію Bluetooth або Wi-Fi). Однак він повинен надійно перехоплювати системні події та забезпечувати єдиний безпечний простір.

Вибір інструментарію розробника

Враховуючи зазначені обмеження, виникає потреба в обґрунтованому виборі інструментарію розробника, здатного забезпечити компроміс між кросплатформною гнучкістю та нативною продуктивністю додатка. Для реалізації поставленої інженерної задачі було обрано кросплатформний фреймворк Flutter, мову програмування Dart та середовище розробки Visual Studio Code (VS Code).

Як основне IDE обрано Visual Studio Code завдяки його низькому споживанню оперативної пам'яті на етапі компіляції та підтримці потужної екосистеми плагінів Flutter і Dart. Інструментарій надає можливість використання Dart Analyzer для статичного аналізу типів даних та дебаг-панелі Flutter DevTools, яка дозволяє профілювати дерева віджетів, відстежувати витоки пам'яті та оптимізувати рендеринг графічних елементів.

Управління користувацьким інтерфейсом (UI) у Flutter базується на концепції декларативного програмування, де стан додатка описується математичною функцією, а кожен візуальний елемент є віджетом (Widget). Архітектура системи будується на композиції двох основних типів класів:

˜ StatelessWidget - використовується для статичних елементів фіксованої сітки (наприклад, фонові плитки та графічні іконки), які не змінюють свій вигляд після первинної ініціалізації;

˜ StatefulWidget - застосовується для динамічних інструментів, які самостійно керують власним життєвим циклом та станом (віджет голосового годинника, екран конфігурації швидкого набору номерів).

На відміну від альтернативних рішень (React Native, Xamarin), фреймворк Flutter не використовує системний JavaScript-міст (Bridge) для інтерпретації команд, а відмальовує графічні компоненти безпосередньо на Canvas за допомогою високоефективного рушія Impeller/Skia. Це унеможливлює виникнення мікрофризів та затримок під час рендерингу складного інтерфейсу на бюджетних пристроях із обмеженими апаратними ресурсами.

Парадигма ООП та типізація у мові Dart

Мова Dart підтримує строгу типізацію та принципи об'єктно-орієнтованого програмування, що дозволяє формалізувати структуру даних лаунчера відповідно до заздалегідь спроєктованих моделей даних. Основні сутності та системні налаштування зберігаються у вигляді чітко типізованих властивостей всередині класів:

˜ Режим захисту та блокування інтерфейсу від змін: bool isLocked;

˜ Параметри розмітки поточної сітки екрана: String gridLayout;

˜ Контактні дані для модуля швидкого зв'язку: String contactName, String phoneNumber, int priorityIndex.

Системна інтеграція оболонки з ОС Android

Для того щоб кросплатформний додаток повноцінно функціонував у ролі домашнього екрана (системного лаунчера) операційної системи й надійно перехоплював натискання апаратної або нативної кнопки «Home», виконано інженерне проєктування конфігураційного маніфесту за шляхом android/app/src/main/AndroidManifest.xml. У структуру головної Activity додано специфічні системні категорії інтенту:

```
<intent-filter>
   <action android:name="android.intent.action.MAIN" />
   <category android:name="android.intent.category.LAUNCHER" />
   <category android:name="android.intent.category.HOME" />
```

```
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

Опис програмних модулів та сторонніх залежностей

Для розширення функціональних можливостей лаунчера та організації взаємодії з апаратним рівнем смартфона в архітектуру інтегровано такі стабільні пакети (plugins):

˜ shared_preferences - реалізує процедури асинхронного читання та запису конфігураційних файлів у локальне сховище Key-Value. Використовується для постійного збереження змінної стану захисту столу isLocked;

˜ flutter_tts - забезпечує низькорівневий доступ до нативного рушія Text-to-Speech операційної системи. Через метод speak(String text) реалізується функція голосового годинника, що трансформує сформовані рядкові змінні часу у звуковий сигнал;

˜ url_launcher - програмний міст для виклику системних нативних сервісів зв'язку через ініціалізацію схеми tel:$phoneNumber, що дозволяє здійснювати виклик абонента в один клік без переходу в стандартний додаток.

Таким чином, розширений аналіз фізіологічних та когнітивних бар'єрів дозволив сформулювати чіткі вимоги до інклюзивного інтерфейсу, визначити апаратні та програмні обмеження, а також обґрунтувати вибір технологічного стеку Flutter/Dart для реалізації асистивного лаунчера.

1.2 Аналіз існуючих лаунчерів для спрощення інтерфейсу смартфона

На сучасному ринку мобільного програмного забезпечення існує велика кількість альтернативних графічних оболонок (лаунчерів), які дозволяють кардинально змінити стандартний робочий стіл операційної системи Android. Для проведення аналізу та визначення оптимальних шляхів розробки власного продукту, доцільно розглянути програмні рішення, поділивши їх на дві основні категорії: мас-маркет оболонки з розширеними функціями кастомізації та

спеціалізовані програми, орієнтовані виключно на користувачів похилого віку. Окремо розглядаються вбудовані спрощені режими виробників смартфонів.

Оцінювання здійснювалося за такими критеріями: рівень когнітивного навантаження, ергономіка (розмір зон натискання), модель монетизації та наявність специфічних функцій доступності.

1.2.1 Mac-маркет оболонки з розширеними функціями кастомізації

Ця категорія лаунчерів не створювалася спеціально для літніх людей, проте вона пропонує альтернативні підходи до організації простору на екрані, які часто розглядаються як спосіб спрощення інтерфейсу.

Nothing Launcher розроблений однойменною компанією та вирізняється строгим мінімалістичним підходом. Функціональні особливості: можливість створення розширених папок (Enlarged Folders) розміром 2×2, фірмові віджети (годинник, погода), зміна форми іконок. Переваги: унікальний візуальний стиль (монохромна палітра, матричний шрифт), легкість, відсутність реклами.

Недоліки для літніх людей: монохромні іконки без кольорового маркування змушують щоразу вчитуватися в назву, що підвищує когнітивне навантаження; матричний шрифт має розірвані контури літер, що робить його нечитабельним для людей із пресбіопією або катарактою.

Рисунок 1.1 — Інтерфейс головного екрана nothing launcher

Niagara Launcher відмовляється від класичної матричної сітки іконок на користь адаптивного списку у формі вертикальної хвилі. Повідомлення інтегруються безпосередньо на головний екран під іконкою месенджера (Inline Notifications). Навігація - через алфавітний покажчик (A-Z slider). Переваги: радикальне спрощення, ергономічність для керування однією рукою, надзвичайна швидкість.

Недоліки: алфавітний скролінг вимагає високої точності моторики (вузька смуга ~8 мм) - для людей із тремором це майже неможливо; динамічний

вертикальний список є контринтуїтивним для тих, хто звик до фіксованого розташування кнопок; наявність платної підписки.

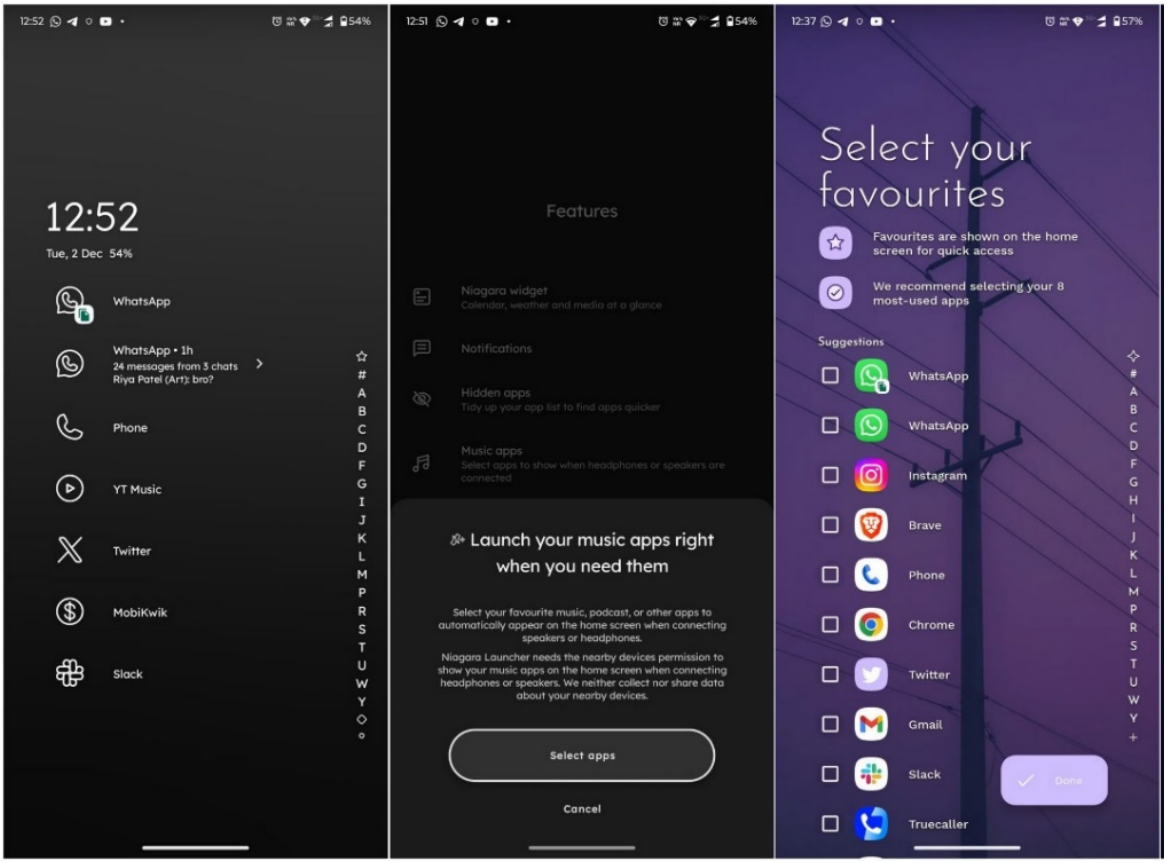

Рисунок 1.2 — інтерфейс niagara launcher

Nova Launcher – один із найстаріших і найфункціональніших лаунчерів. Дозволяє змінювати сітку від 2×2 до 12×12, підтримує жести, кастомізацію док-панелі, сторонні набори іконок, бекапи налаштувань. Переваги: величезна спільнота, підтримка застарілих версій Android (від 5.0), низьке споживання RAM (~30 МБ).

Недоліки: архітектура перевантажена налаштуваннями (>30 пунктів меню) -виникає «парадокс вибору», літній користувач може випадково змінити критичні параметри і не зможе повернути все назад; жестові команди складно даються аудиторії 50+, яка надає перевагу явним кнопкам.

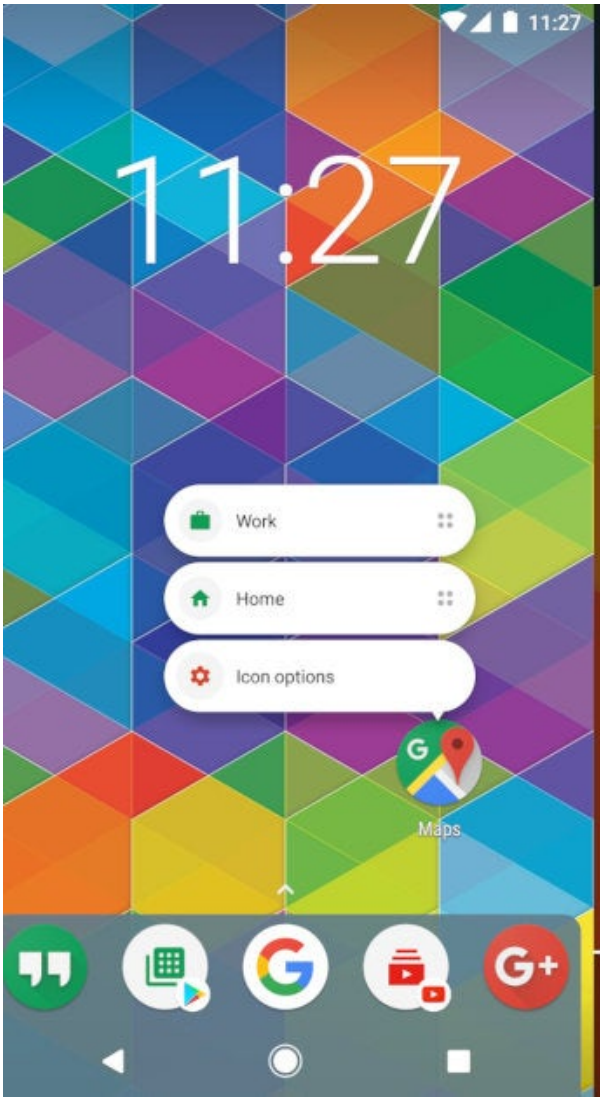

Рисунок 1.3 - Інтерфейс головного екрана nova launcher.

1.2.2Спеціалізовані асистивні лаунчери (для аудиторії 50+)

Програми цієї категорії розроблялися безпосередньо для вирішення проблеми «цифрового розриву», враховуючи вікові зміни. Проте вони мають низку недоліків.

BIG Launcher базується на концепції максимального збільшення об'єктів та агресивного колірного кодування. Сітка - 6 гігантських кнопок на екран. Використовуються товсті чорні контури іконок. Присутня кнопка SOS, яка при натисканні ініціює виклик та надсилає SMS із GPS-координатами. Переваги: зони натискання до 20×20 мм, захист від тремору, підтримка Android 4.1+.

Недоліки: агресивна монетизація (Freemium). Безкоштовна версія має жорсткі обмеження: лише 4 номери в швидкому наборі, лише 5 записів у журналі дзвінків. Постійні спливаючі вікна з пропозицією купити Premium (≈$25) викликають стрес у пенсіонерів. Відсутня українська локалізація.

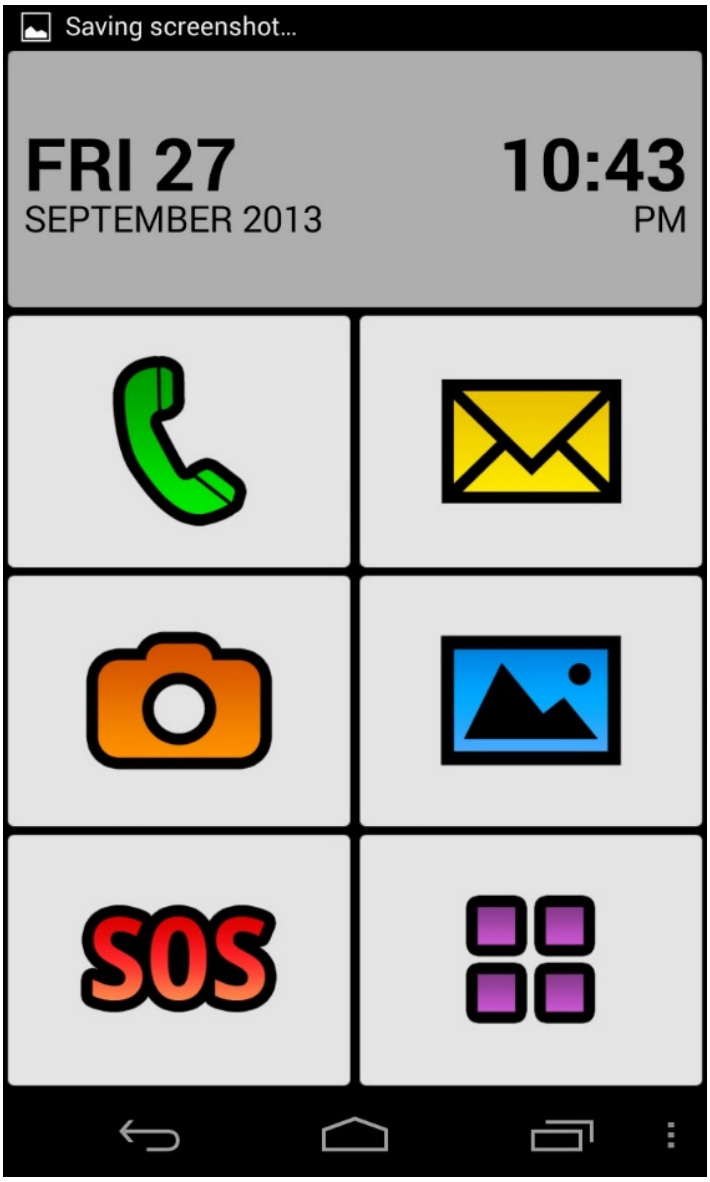

Рисунок 1.4 - Графічний інтерфейс BIG Launcher

BaldPhone - інклюзивна оболонка з відкритим кодом (Open Source). Замінює системні додатки (галерею, контакти, телефон, будильник) на власні спрощені аналоги. Має функцію регулювання затримки відгуку (Hold-to-Press delay) - ідеальний захист від тремору та випадкових подвійних кліків. Переваги: повна відсутність реклами та платних функцій, безкоштовно.

Недоліки: надмірне текстове перевантаження (екран забитий дрібними підказками), відсутність фіксованої лаконічної сітки - замість цього кілька довгих сторінок або списків. Інтерфейс кардинально відрізняється від Android, тому родичам важко допомагати з налаштуваннями. Проєкт не оновлювався більше року.

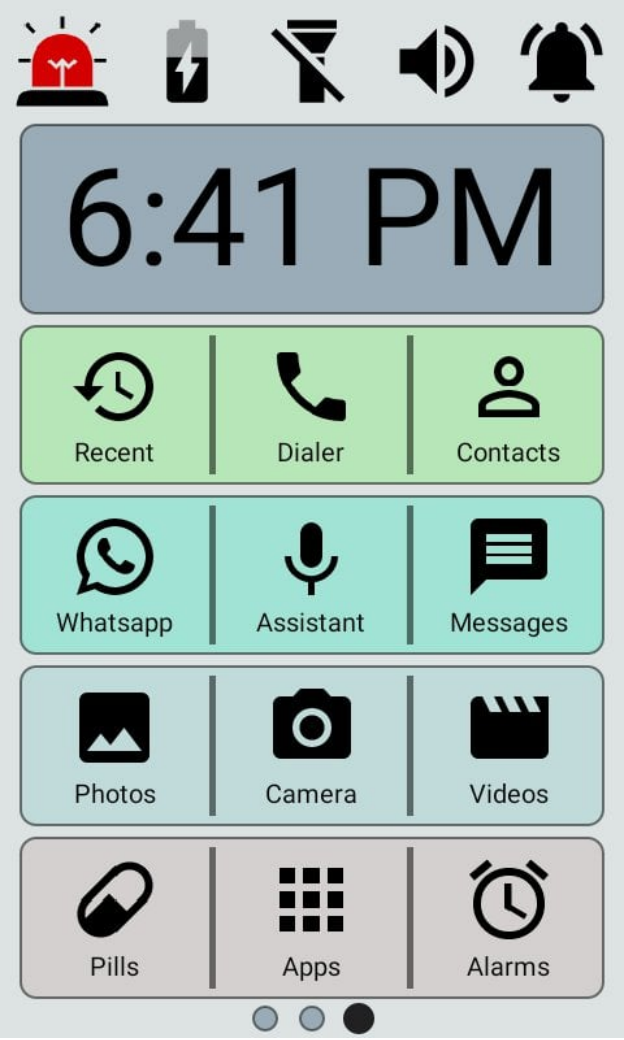

Рисунок 1.5 - Інтерфейс головного екрана та меню в BaldPhone.

1.2.3 Вбудовані спрощені режими виробників смартфонів

Окремо варто відзначити системні режими, такі як Samsung Easy Mode або Xiaomi Lite Mode. Вони глибоко інтегровані в прошивку, стабільні та безкоштовні. Активація змінює DPI, збільшує іконки до сітки 3×3, збільшує

шрифт, виносить контакти на окремий стіл. Переваги: без додаткової установки, безпечно, є підтримка виробника.

Недоліки: дія фрагментарна - спрощується лише робочий стіл, але перехід у налаштування або шторку сповіщень повертає до складного середовища. Виникає ефект «функціонального обриву»: щойно літня людина відкриває сторонній додаток (Viber, YouTube), спрощений режим закінчується. Відсутні специфічні функції: голосове озвучення часу, блокування іконок від випадкового переміщення. Доступні лише на смартфонах відповідних брендів.

Рисунок 1.6 - Приклад Samsung Easy Mode

Таблиця 1.1 - Порівняльний аналіз існуючих мобільних лаунчерів

Назва	Основні функціональні можливості	Переваги	Недоліки
Nothing Launcher	Фірмові віджети (годинник, погода), розширення папок (Enlarged Folders), зміна іконок, інтеграція з екосистемою Nothing.	Унікальний візуальний стиль (матричний шрифт), легкість, відсутність реклами.	Обмежена кількість налаштувань у порівнянні з конкурентами, найкраще працює лише на смартфонах однойменного бренду.
Niagara Launcher	Адаптивний список застосунків у формі вертикальної хвилі, інтегровані сповіщення прямо на головному екрані, швидкий пошук однією рукою.	Радикальне спрощення інтерфейсу, ідеальний для керування однією рукою, надзвичайна швидкість роботи	Платна підписка на розширені функції, відсутність підтримки стандартних віджетів у безкоштовній версії.
Nova Launcher	Повна зміна сітки робочого столу, керування жестами, кастомізація док-панелі, підтримка сторонніх наборів іконок.	Величезна спільнота користувачів, підтримка навіть старих версій Android.	Може здатися занадто складним для новачка через велику кількість меню, останнім часом рідше отримує візуальні оновлення.

Продовження таблиці 1.1 - Порівняльний аналіз існуючих мобільних лаунчерів

BIG Launcher	Заміна стандартної сітки на великі контрастні кнопки, наявність вбудованої кнопки SOS для екстреного зв'язку, великі індикатори стану.	Максимально збільшені зони натискання, висока контрастність, зручність для користувачів із серйозними вадами зору.	Агресивна модель монетизації, жорсткі ліміти безкоштовної версії, постійні сповіщення з вимогою придбати Premium.
BaldPhone	Повна заміна системного середовища (галерея, контакти, будильник), регулювання затримки відгуку інтерфейсу, голосове введення.	Повна відсутність вбудованої реклами чи платних функцій, високий рівень захисту від випадкових подвійних дотиків.	Надмірне функціональне перевантаження, велика кількість дрібного супровідного тексту, відсутність фіксованої лаконічної сітки екрана.

1.3 Формування технічних та функціональних вимог до програмного продукту

Формалізація технічних та функціональних вимог є критично важливим етапом інженерного циклу розробки програмного забезпечення, оскільки вона визначає архітектурні межі системи, мінімізує ризики проєктування та забезпечує відповідність кінцевого продукту потребам цільової аудиторії. Специфікація вимог до асистивного мобільного лаунчера базується на принципах інклюзивного дизайну та стандартах доступності мобільних інтерфейсів.

Перед безпосереднім моделюванням архітектури системи необхідно чітко розмежувати ролі користувачів, які взаємодіятимуть із програмним продуктом:

˜ Основна цільова група (Користувачі віком 50+): Особи похилого віку, пенсіонери, ветерани, які відчувають значний когнітивний дискомфорт під час роботи зі стандартними, перевантаженими інтерфейсами мобільних ОС. Ця група характеризується віковими змінами гостроти зору, зниженням дрібної моторики рук або наявністю легкого тремору, що вимагає специфічного підходу до UI-елементів.

˜ Користувачі з порушеннями зору: Специфічний профіль користувачів будь-якого віку, для яких критично важливим є максимальне збільшення графічних та текстових елементів, забезпечення високого рівня контрастності інтерфейсу та наявність дублюючих аудіальних каналів отримання інформації (зокрема, фонового озвучення часу).

˜ Родичі та опікуни (Адміністратори системи): Технічно грамотна категорія користувачів, яка здійснює первинне конфігурування лаунчера під індивідуальні потреби літньої людини. Їхній інженерний інтерес полягає у забезпеченні такої стабільності системи, за якої кінцевий користувач не зможе випадково пошкодити налаштування або видалити важливі ярлики.

1.3.1. Багаторівнева класифікація вимог до інформаційної системи

Відповідно до методології інженерії вимог, критерії до розроблюваного ПЗ структуровано за трьома концептуальними рівнями:

˜ Бізнес-вимоги (Business Requirements): Головною метою проєкту є радикальне зниження порогу входу у використання сучасних сенсорних пристроїв для людей категорії 50+, що безпосередньо сприяє подоланню соціальної ізоляції та мінімізації наслідків цифрового розриву.

˜ Вимоги користувачів (User Requirements): Забезпечення надійного захисту робочого простору екрана від випадкових деструктивних дій (видалення іконок, зміна конфігурації), а також надання інструментів для миттєвого здійснення

екстрених викликів або зв'язку з близькими без тривалого пошуку в заплутаних меню.

˜Функціональні вимоги (Functional Requirements): Визначають технічну поведінку системи та включають відображення фіксованої сітки зі збільшеними іконками, створення окремого екрана швидкого зв'язку для обраних абонентів та реалізацію фонового віджета голосового годинника із синтезом мовлення.

Розподіл вимог за технічним характером виконання визначає внутрішню архітектуру та вибір методів у коді Dart:

Функціональні вимоги:

1. Наявність інтегрованого віджета голосового годинника з підтримкою синтезу мовлення (через об'єкт класу FlutterTts).

2. Реалізація спеціалізованого модуля швидкого зв'язку з можливістю прив'язки від 6 до 8 обраних номерів, що обов'язково супроводжуються великими фотокартками або кольоровими аватарами для миттєвої ідентифікації абонента.

3. Можливість блокування робочого столу (Lock mode) для запобігання випадковому переміщенню або видаленню іконок.

Нефункціональні вимоги:

1. Надійність (Reliability): Програмна оболонка повинна мати високий рівень відмовостійкості. У разі виникнення критичного збою або фонової помилки лаунчер має автоматично перезапускатися в найкоротший термін (не більше 2 секунд), забезпечуючи безперервну працездатність пристрою.

2. Продуктивність (Performance): Час запуску лаунчера після натискання кнопки «Додому» не повинен перевищувати 0,5 секунди на пристроях з 2 ГБ RAM. Фонове споживання оперативної пам'яті - не більше 80 МБ.

3. Простота (Simplicity): Інтерфейсна архітектура застосунку повинна виключати складну багатовекторну навігацію. Встановлюється повна відсутність вкладених меню глибших ніж 2 рівні, що мінімізує ризик дезорієнтації літньої людини всередині програмного простору.

1.3.2. Технічні та експлуатаційні обмеження системи

Ефективне проєктування вимагає чіткого врахування жорстких обмежень, що накладаються зовнішнім середовищем виконання ПЗ:

- Програмні обмеження: Продукт проєктується для розгортання на базі операційної системи Android версії 6.0 та вище (API level 23+). При цьому лаунчер функціонує виключно як графічна оболонка над системою і не здатний повністю замінити нативні меню налаштувань смартфона (такі як інтерфейси конфігурації бездротових мереж Wi-Fi чи параметрів Bluetooth).
- Апаратні обмеження: Програмний продукт повинен демонструвати стабільну продуктивність та низьке споживання апаратних ресурсів на бюджетних смартфонах із малим об'ємом оперативної пам'яті (від 2 ГБ RAM) та застарілими архітектурами процесорів (наприклад, MediaTek MT67xx або Snapdragon 400 серії).
- Інтерфейсні обмеження: Для уникнення візуального шуму та збереження великих габаритів елементів керування розмітка робочих екранів жорстко обмежується фіксованою матричною сіткою 2×4 іконок на одному вікні. Розмір кожної іконки має бути не менше 14×14 мм.

1.3.3. Аналіз інженерних ризиків при розробці та експлуатації

При проєктуванні асистивного лаунчера необхідно враховувати кілька категорій технічних ризиків:

1. Ризик випадкових дій користувача: Попри багаторівневий захист інтерфейсу, зберігається ймовірність того, що літня людина зможе випадково перейти в глибокі системні налаштування самої ОС Android (які лаунчер технічно не контролює) та змінити там критичні параметри функціонування гаджета. Заходи пом'якшення: надання в інструкції рекомендації щодо додаткового блокування налаштувань через системні інструменти (наприклад, через Settings > Accessibility).

2. Технічна стабільність та конфлікти оболонок: Сучасні смартфони використовують агресивні вбудовані фірмові оболонки виробників (наприклад, Samsung OneUI, Xiaomi MIUI або Huawei EMUI), системні алгоритми яких можуть намагатися примусово закривати сторонні фонові процеси або сторонній лаунчер з метою оптимізації енергоспоживання. Заходи пом'якшення: у коді лаунчера реалізовано сервіс із низьким рівнем споживання ресурсів; в інструкції користувача надано алгоритми додавання лаунчера до списку винятків (Whitelist) у налаштуваннях енергозбереження.

3. Енергоспоживання (Battery Drain): Постійне функціонування фонових процесів віджета голосового годинника (оновлення щосекунди), а також періодичний виклик систем мовного синтезу можуть призвести до прискореного розряджання акумулятора смартфона. Заходи пом'якшення: оновлення часу відбувається лише при активному відкритому екрані (у фоновому режимі оновлення зупиняється). Використано ефективний таймер Timer.periodic з автоматичним звільненням ресурсів у методі dispose().

1.3.4. Пріоритетність функціональних вимог

Для забезпечення поетапної розробки мобільного додатка всі функціональні вимоги було проранжовано за ступенем важливості для кінцевого споживача. Це дозволяє в разі обмеження часу зосередитися на найбільш критичних функціях.

Таблиця 1.2 - Пріоритетність функціональних вимог

Вимога	Пріоритет
Збільшені іконки та шрифти	Критичний
Захист від випадкових видалень (Lock mode)	Високий
Екран швидкого набору (6-8 номерів)	Високий
Голосовий годинник	Середній

Голосове озвучування натискань кнопок	Низький

Для наочного представлення функціональних можливостей лаунчера, моделювання логіки поведінки системи та визначення меж взаємодії між різними категоріями користувачів було розроблено діаграму прецедентів використання (use case diagram). Графічне представлення моделі прецедентів наведено на рисунку 1.6.

Як показано на діаграмі, у системі виділено дві основні ролі взаємодії (актори): «Користувач (50+)» та «Опікун / Родич». Між ними існує зв'язок узагальнення (generalization), оскільки опікун може виконувати всі базові дії кінцевого користувача, проте володіє додатковими правами адміністрування.

Для актора «Користувач (50+)» доступні такі ключові прецеденти, як «Перегляд головного екрана зі збільшеними іконками» та «Здійснення швидкого виклику». Варіант використання «Перегляд головного екрана» розширюється (extend) прецедентом «Взаємодія з голосовим годинником», що відображає необов'язковий (асистивний) характер аудіосупроводу.

Актор «Опікун / Родич» має виключний доступ до прецеденту «Налаштування системи», який обов'язково включає (include) процес «Редагування списку контактів» для швидкого набору. Також варіант використання «Налаштування системи» розширюється прецедентом «Активація режиму захисту Lock mode», що дозволяє заблокувати інтерфейс від випадкових змін конфігурації.

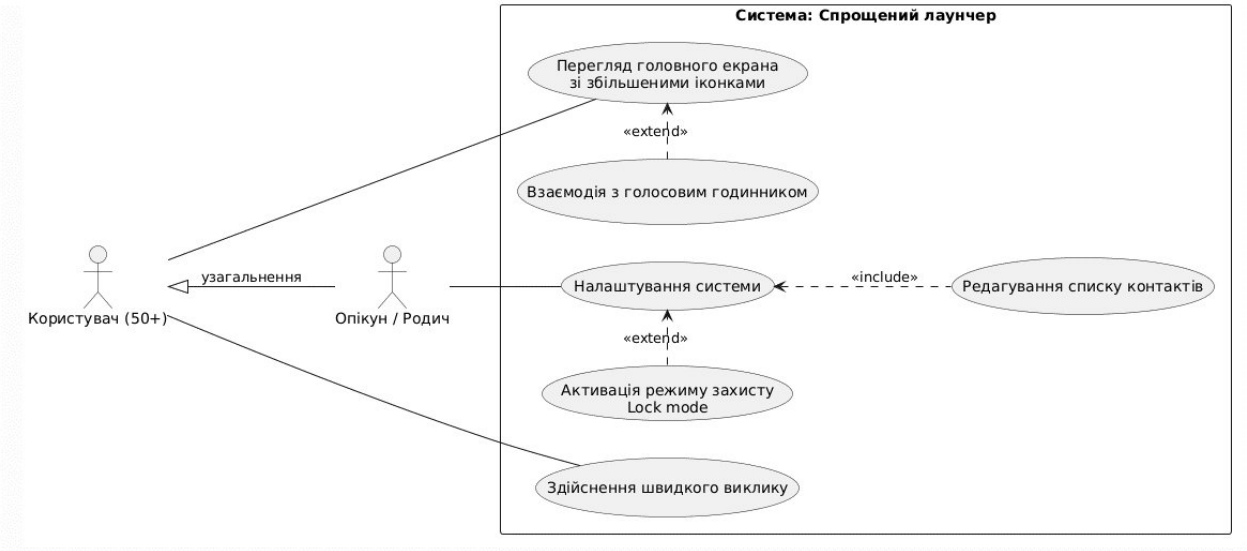

Рисунок 1.7 - Діаграма прецедентів використання асистивного мобільного лаунчера

Висновки до розділу 1

У розділі проаналізовано фізіологічні та когнітивні бар'єри людей похилого віку, що дозволило визначити ключові вимоги до інклюзивного інтерфейсу: великі елементи керування, висока контрастність, обмежена глибина навігації. Виконано порівняльний аналіз існуючих лаунчерів, виявлено їхні недоліки для аудиторії 50+ (складність, монетизація, фрагментарність), обґрунтовано вибір технологій Flutter/Dart. Сформовано технічні вимоги до власного асистивного лаунчера з пріоритетними функціями (Lock mode, швидкий набір, голосовий годинник), що закладає фундамент для програмної реалізації.

РОЗДІЛ 2
ПРОЄКТУВАННЯ, РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ДОДАТКУ

2.1 Архітектурне моделювання компонентів та структур даних системи

Проєктування архітектури асистивного мобільного лаунчера базується на принципах системного аналізу, модульної декомпозиції та об'єктно-орієнтованого програмування. Головною метою цього етапу є перетворення формалізованих технічних вимог, описаних у Розділі 1, у конкретні програмні структури, класи та алгоритми, які забезпечать стабільну роботу оболонки в середовищі операційної системи Android.

2.1.1. Методологія системного моделювання

Враховуючи вибір фреймворку Flutter, системне моделювання виконується на трьох взаємопов'язаних рівнях:

- Інформаційний рівень - визначення структури даних, що зберігаються локально (сутності AppWidget, Contact, їхні поля, типи даних та зв'язки між ними). Цей рівень описує, яка інформація зберігається на пристрої та в якому вигляді.
- Функціональний рівень - опис динамічних потоків даних між користувачем та програмними компонентами (DFD-діаграми). Він показує, як інформація рухається системою і які перетворення зазнає.
- Компонентний рівень - ієрархічне моделювання структури класів та взаємозв'язків між інтерфейсними віджетами (діаграма класів UML). Цей рівень фіксує статичну структуру програмного коду.

Такий підхід дозволяє розділити складну систему на керовані частини та забезпечити їхню узгодженість.

2.1.2. Інформаційне моделювання структури локальних даних (ER-діаграма)

Оскільки асистивний лаунчер є персоналізованим інструментом, він повинен забезпечувати повну конфіденційність та автономність роботи. Усі конфігурації користувача та списки контактів зберігаються виключно в локальній пам'яті пристрою - жодні дані не передаються на зовнішні сервери. Для управління даними використовується пакет shared_preferences, який реалізує принцип Key-Value сховища. Цей механізм дозволяє зберігати невеликі обсяги структурованої інформації (до кількох сотень записів) у вигляді пар «ключ-значення». Дані серіалізуються у формат JSON для довготривалого збереження між сесіями - тобто після перезавантаження смартфона або закриття програми всі налаштування залишаються незмінними.

Логічна структура моделі даних представлена двома основними сутностями, які є незалежними одна від одної (зв'язок «один до багатьох» не є жорстким, оскільки кожна сутність зберігається окремим списком):

1. Сутність AppWidget - описує конфігурацію однієї плитки (іконки) на головному екрані. Включає такі поля:

˜ id (String) - унікальний ідентифікатор, генерується автоматично за допомогою Uuid().v4();

˜ name (String) - локалізована назва додатка, яку бачить користувач (наприклад, «Телефон», «Повідомлення»);

˜ iconCode (int) - код іконки з бібліотеки Icons фреймворку Flutter (наприклад, Icons.phone.codePoint);

˜ color (int) - колірне значення для фону плитки у форматі ARGB (наприклад, 0xFF4CAF50 для зеленого);

˜ packageName (String) - ім'я пакета Android, необхідне для запуску додатка через системний інтент (наприклад, com.android.dialer).

2. Сутність Contact - описує одну картку швидкого набору на екрані контактів. Включає такі поля:

˜ id (String) - унікальний ідентифікатор (аналогічно до AppWidget);

˜ name (String) - ім'я контакту (наприклад, «Мама», «Тато», «Лікар»);

˜ phone (String) - номер телефону у міжнародному форматі +380XXXXXXXXX (це забезпечує коректну роботу дзвінків у будь-якому регіоні);

˜ avatarColor (int) - колір кола-аватара, який використовується, якщо користувач не додав фотографію;

˜ imagePath (String) - локальний шлях до файлу зображення у внутрішній пам'яті смартфона (опціональне поле). Якщо шлях вказаний, замість кольорового аватара відображається фотографія.

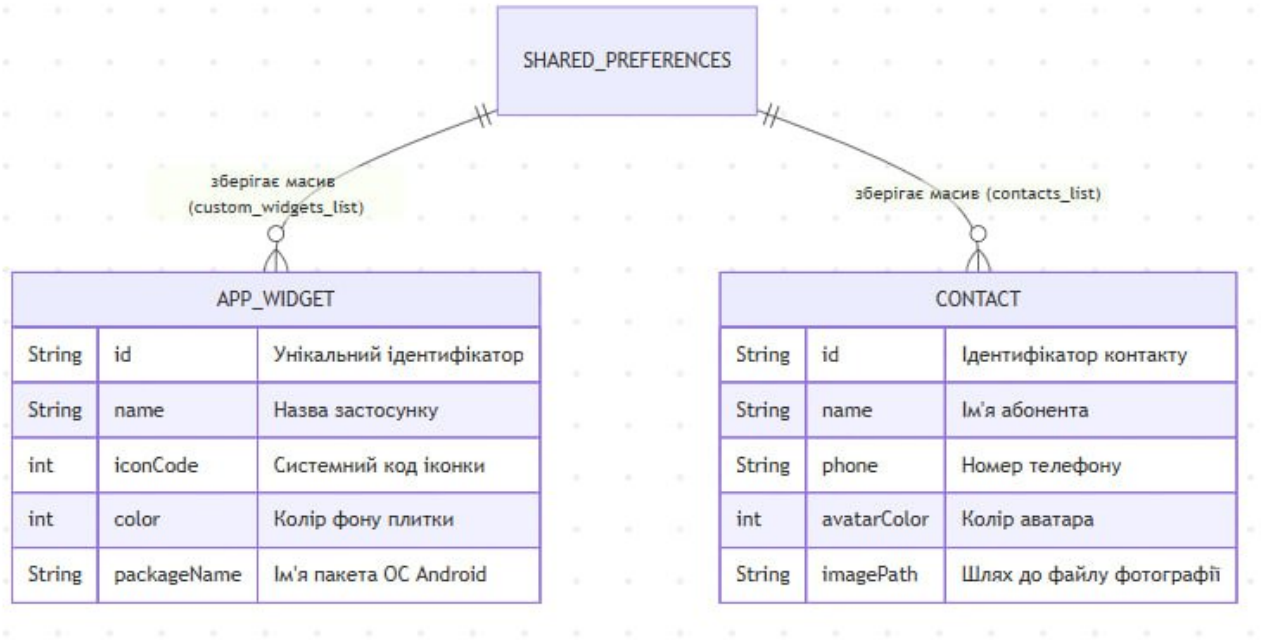

Рисунок 2.1 - ER-діаграма структури даних

2.1.3. Функціональне моделювання процесів (DFD)

Для аналізу динамічної поведінки системи застосовано методологію структурних блок-схем (DFD - Data Flow Diagram), яка дозволяє візуалізувати алгоритм виконання критичних операцій. У системі виділено три магістральні потоки обробки даних - вони відповідають ключовим функціям лаунчера.

Потік 1: Модуль голосового годинника Користувач натискає кнопку «Озвучити час» на екрані ClockScreen. Система виконує наступні кроки:

˜ Викликає DateTime.now() для отримання поточного часу від системного годинника ОС Android.

˜ Форматує отримане значення у зручний для читання текстовий рядок за допомогою класу DateFormat з бібліотеки intl (наприклад, «Дванадцята година тридцять хвилин»).

˜ Передає цей рядок до плагіна flutter_tts.

˜ Плагін через нативний API Text-to-Speech операційної системи ініціює відтворення аудіопотоку.

˜ У разі помилки (наприклад, відсутність голосового рушія або невстановлений мовний пакет) система виводить спливаюче повідомлення SnackBar з текстом «Помилка озвучення».

Потік 2: Модуль швидкого виклику Користувач вибирає контакт на ContactsScreen (натискає на зелену кнопку із зображенням слухавки). Система виконує наступні кроки:

˜ Зчитує номер телефону з локального сховища SharedPreferences для обраного контакту.

˜ Формує URI-схему tel:$phoneNumber (наприклад, tel:+380501234567).

˜ Викликає метод canLaunchUrl() плагіна url_launcher для перевірки, чи може система обробити цей запит.

˜ Якщо перевірка успішна, викликає launchUrl(), який передає керування системному додатку «Телефон» ОС Android.

˜ Користувач бачить екран дзвінка з уже введеним номером - залишається лише натиснути зелену кнопку «Виклик».

˜ Якщо дозвіл CALL_PHONE не надано, система запитує його автоматично (за допомогою плагіна permission_handler).

Потік 3: Модуль конфігурації (Editor Mode) Адміністратор (опікун) відкриває WidgetsEditorScreen через іконку редагування на головному екрані. Система виконує наступні кроки:

˜ Ініціює виклик через MethodChannel('apps_channel') - це асинхронний міст між кодом на Dart та нативним кодом Android (Kotlin/Java).

˜ Нативний код звертається до системного PackageManager і отримує список всіх інстальованих додатків разом з їхніми іконками, назвами та іменами пакетів.

˜ Отриманий масив даних повертається у Flutter та відображається у вигляді списку для вибору.

˜ Користувач (опікун) обирає потрібний додаток, налаштовує його відображення (колір, назву) через модальне діалогове вікно AlertDialog.

˜ Після підтвердження дані серіалізуються в JSON і зберігаються в SharedPreferences.

˜ Головний екран (AppsScreen) автоматично перемальовується з новою іконкою.

Структурна схема роботи функціональних модулів наведена на рисунку 2.2. Вона унаочнює, як взаємодіють між собою окремі компоненти системи та зовнішні підсистеми (ОС Android, апаратні модулі).

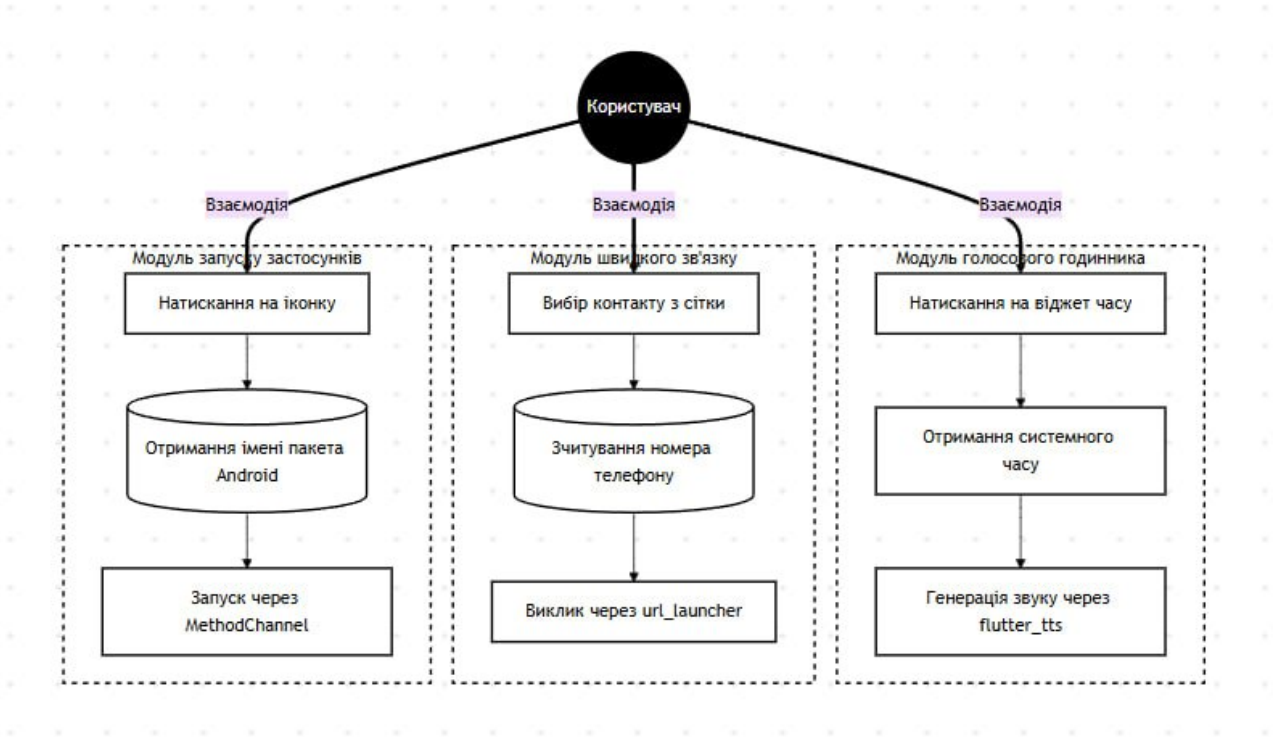

Рисунок 2.2 - Структурна схема роботи функціональних модулів лаунчера

2.1.4. Об'єктно-орієнтоване проєктування архітектури класів

Проєктування класів здійснюється на основі декларативної парадигми фреймворку Flutter. Архітектура програми розділена на два рівні: рівень представлення (екрани, віджети) та рівень управління станом (збереження даних, взаємодія з плагінами). Основна ієрархія класів представлена в діаграмі класів (рис. 2.3). Нижче наведено опис кожного ключового класу:

MainScreen (StatefulWidget) - кореневий контейнер додатка. Виконує такі функції:

˜ Містить PageController для керування горизонтальним гортанням між екранами (вкладками).

˜ Інтегрує BottomNavyBar - нижню навігаційну панель з трьома іконками: «Віджети», «Контакти», «Годинник».

˜ Обробляє зміну поточної вкладки (через метод onTap або свайп) і синхронізує стан панелі з PageView.

˜ Забезпечує перехоплення системної кнопки «Назад» за допомогою віджета PopScope (щоб випадково не закрити лаунчер) [11].

AppsScreen (StatefulWidget) - головний екран з сіткою віджетів. Виконує такі функції:

˜ Відповідає за рендеринг сітки з фіксованою матрицею 2×4 (два рядки, чотири колонки) за допомогою GridView.builder.

˜ Зберігає список List<AppWidget>, який завантажується з SharedPreferences під час ініціалізації.

˜ Реалізує метод _launchApp(String packageName), який через MethodChannel передає ім'я пакета в нативне середовище для запуску додатка.

˜ Підтримує режим редагування: при натисканні на іконку олівця (якщо режим розблоковано) відкриває WidgetsEditorScreen.

˜ У разі, якщо список віджетів порожній (перший запуск), відображає заглушку з пропозицією додати програми.

ContactsScreen (StatefulWidget) - екран швидкого зв'язку. Виконує такі функції:

˜ Відображає список контактів у вигляді вертикального списку ListView.builder.

˜ Кожен елемент списку - це картка Card з:

- CircleAvatar (фотографія з imagePath або кольоровий аватар з першою літерою імені);
- Ім'я та номер телефону (текст з можливістю копіювання);
- Зелена кнопка IconButton з іконкою Icons.phone, яка ініціює дзвінок через url_launcher.

˜ При натисканні на іконку редагування (окрема кнопка в AppBar) відкриває ContactsEditorScreen для додавання/видалення/зміни контактів.

˜ Запитує дозвіл на читання контактів (хоча контакти зберігаються локально, цей дозвіл може знадобитися для імпорту - функція запланована на майбутнє).

ClockScreen (StatefulWidget) - екран голосового годинника. Виконує такі функції:

˜ Використовує Timer.periodic з інтервалом 1 секунда для щосекундного оновлення змінної _currentTime типу DateTime.

˜ Відображає час величезним шрифтом (розмір 96) та дату шрифтом середнього розміру (24).

˜ Містить масивну кнопку ElevatedButton (ширина 90% екрана, висота 80 dp) з текстом «Озвучити час».

˜ При натисканні на кнопку викликається метод _speakTime(), який:

- Ініціалізує екземпляр FlutterTts;
- Встановлює мову uk-UA через setLanguage();
- Встановлює швидкість мовлення 0.5 (уповільнено) через setSpeechRate();
- Форматує _currentTime у текстовий рядок (наприклад, «Дванадцята година тридцять п'ять хвилин»);
- Викликає speak(text) для відтворення.

˜ У методі dispose() таймер зупиняється, щоб уникнути витоків пам'яті.

WidgetsEditorScreen та ContactsEditorScreen (StatefulWidget) - екрани адміністрування (доступні лише через іконку редагування, недоступні в режимі Lock mode). Виконують такі функції:

˜ Відображають поточний список віджетів/контактів з можливістю видалення (свайп або кнопка «-»).

˜ Мають кнопку «Додати», яка відкриває модальне діалогове вікно AlertDialog з формою:

- Поле введення назви (TextField);
- Випадаючий список для вибору іконки (з попередньо заданої палітри безпечних кольорів);
- Випадаючий список для вибору кольору (з обмеженої палітри);
- (Для контактів) кнопку вибору фотографії через image_picker.

˜Після підтвердження дані серіалізуються в JSON і зберігаються через SharedPreferences.setString().

˜Після збереження попередній екран (AppsScreen або ContactsScreen) отримує сповіщення через Navigator.pop() та оновлює свій стан.

Для управління глобальним станом додатка застосовано підхід, при якому зміна параметрів інтерфейсу (наприклад, видалення віджета, зміна контакту) ініціює асинхронне оновлення бази даних через SharedPreferences та подальше перемальовування інтерфейсу за допомогою методу setState(). Такий підхід забезпечує передбачуваність роботи системи, легкість її подальшої підтримки та мінімізує кількість помилок, пов'язаних із розсинхронізацією даних.

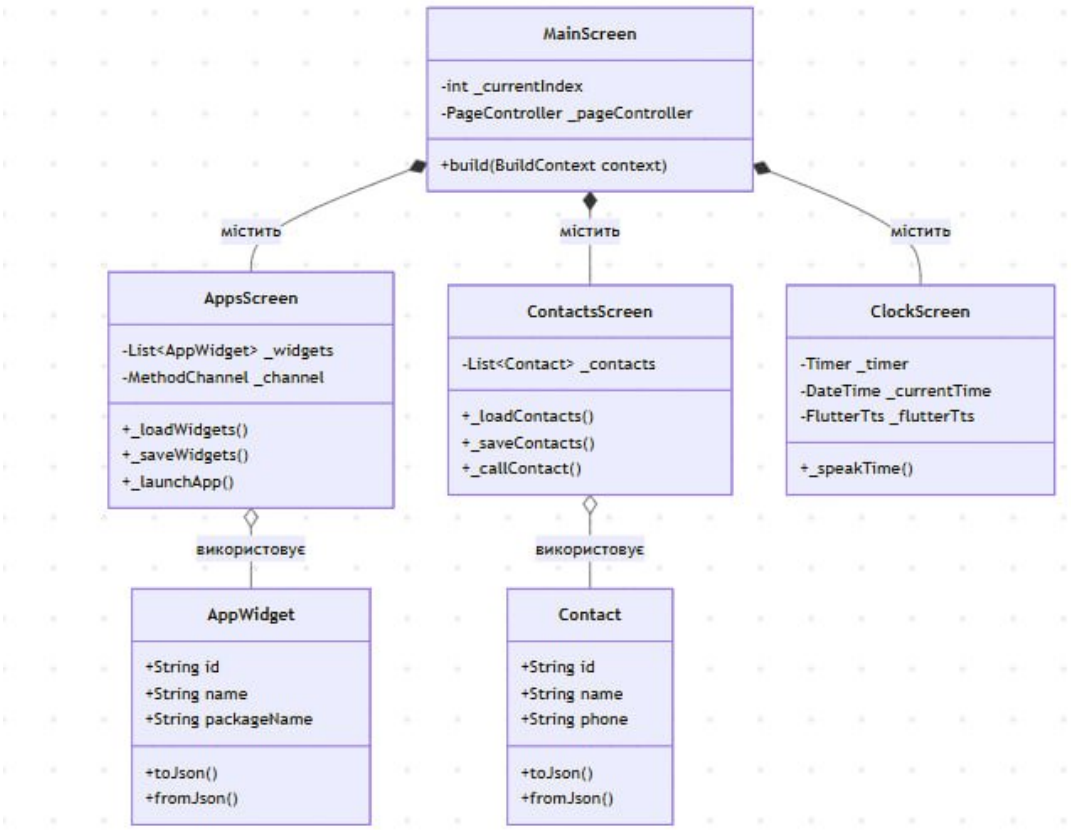

Рисунок 2.3 - Діаграма архітектурних класів системи

2.2. Програмна розробка графічного інтерфейсу та структури навігації

Графічний інтерфейс користувача (GUI) асистивного мобільного лаунчера проєктувався з урахуванням фізіологічних та когнітивних особливостей людей

вікової категорії 50+, виявлених у підрозділі 1.1. Основний акцент під час розробки було зроблено на забезпеченні високої контрастності, великих зон натискання (touch targets) та відсутності глибоко вкладених меню. Глобальна візуальна тема додатка налаштована на темний режим (Brightness.dark) з акцентним кольором Colors.teal, що зменшує навантаження на очі та підвищує читабельність.

2.2.1. Архітектура головного екрана та системи навігації (AppsScreen)

З метою уникнення складної для сприйняття концепції багаторівневих вікон, навігаційну модель системи побудовано на базі плаского перемикання вкладок. Кореневий екран MainScreen (StatefulWidget) розділяє простір на дві зони: робочу область (віджет PageView) та нижню навігаційну панель (BottomNavyBar). Така структура дозволяє користувачеві переміщатися між трьома основними екранами (головний екран із віджетами, екран контактів, екран годинника) як за допомогою горизонтальних свайпів, так і через натискання на відповідні іконки в нижній панелі. Активна вкладка візуально підсвічується, що забезпечує однозначний зворотний зв'язок.

Головний робочий простір системи реалізовано у класі AppsScreen. Замість вільного розміщення іконок, яке потребує точної моторики та просторової пам'яті, тут використано жорстку фіксовану сітку на базі компонента GridView.builder. Для забезпечення максимального розміру елементів керування матрицю налаштовано на дві колонки - це компроміс між кількістю віджетів на екрані та їхньою доступністю. Конфігурація сітки виглядає так:

Лістинг 2.1 - Конфігурація сітки головного екрана

```
    : GridView.builder(
  padding: const EdgeInsets.all(16),
  gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
    crossAxisCount: 2,
    childAspectRatio: 1.0,
    crossAxisSpacing: 16,
    mainAxisSpacing: 16,
  ), SliverGridDelegateWithFixedCrossAxisCount
  itemCount: _widgets.length,
  itemBuilder: (ctx, index) {
    final w = _widgets[index];
    return _buildWidgetTile(w);
  },
), GridView.builder
```

Кожен елемент списку візуалізується у вигляді масивної квадратної плитки із градієнтною заливкою, заокругленими кутами (BorderRadius.circular(24)) та великою системною іконкою по центру. Розмір іконки фіксовано на рівні 48×48 dp, що значно перевищує рекомендований мінімум 24×24 dp. Плитка реагує на натискання анімацією InkWell (хвиля), а також має візуальний стан "наведення" для пристроїв із мишею або стилусом.

Під час першого запуску (коли список віджетів порожній) відображається спеціальна заглушка з написом «Немає додатків. Натисніть олівець, щоб додати» та великою іконкою редагування. Це запобігає ситуації, коли літня людина бачить порожній екран і не розуміє, що робити далі.

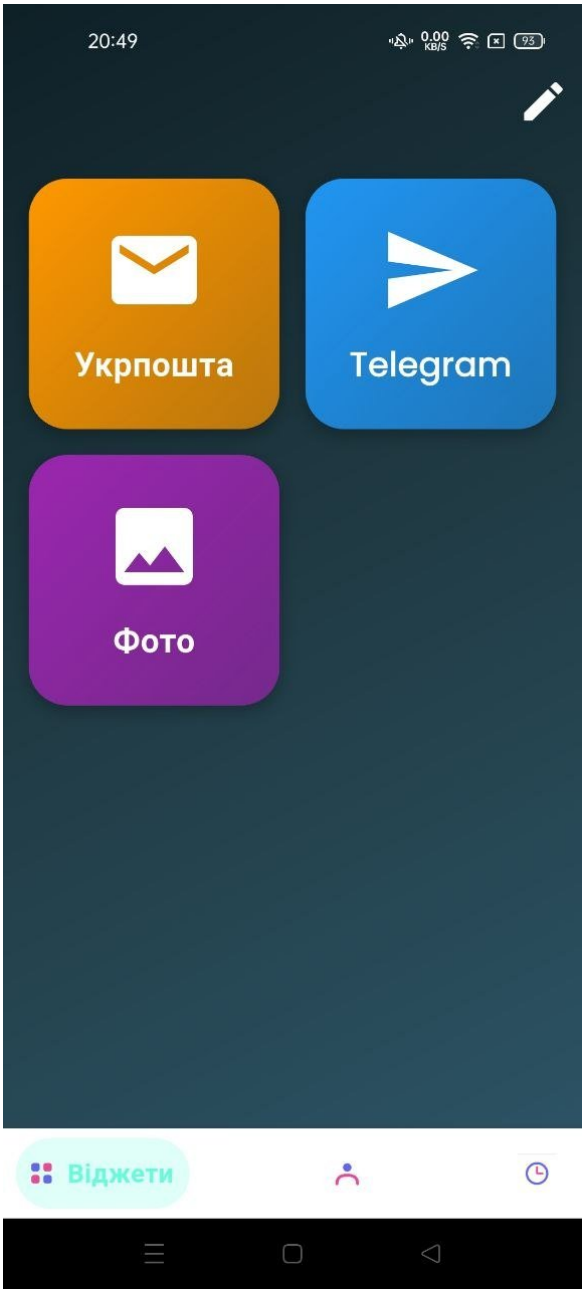

Рисунок 2.4 - Головний екран лаунчера та панель навігації

2.2.2. Інтерфейс модуля швидкого зв'язку (ContactsScreen)

Екран контактів спроєктовано у вигляді вертикального списку (ListView.builder), який дозволяє легко гортати записи без ризику випадкового натискання на сусідні елементи. Такий підхід є більш безпечним для людей із тремором, ніж сітка, оскільки кожен елемент займає всю ширину екрана, а зона натискання є максимальною. Кожен контакт оформлено у вигляді контрастної картки Card з тінню, що візуально відокремлює абонентів один від одного.

Графічний компонент абонента (ListTile) містить три основні зони:

˜ Ліва зона - круглий аватар (CircleAvatar). Якщо в полі imagePath збережено шлях до файлу, відображається фотографія (завантажена через Image.file()). Якщо шлях відсутній або файл не знайдено, відображається кольорове коло з першою літерою імені контакту (великою). Колір аватара генерується на основі хешу імені, що забезпечує стабільність кольору для кожного контакту.

˜ Центральна зона - дві строки тексту: ім'я контакту (великий шрифт, жирний) та номер телефону (шрифт меншого розміру, сірий). Текст може бути скопійований довгим натисканням.

˜ Права зона - зелена кнопка IconButton з іконкою Icons.phone та розміром 56×56 dp. Колір Colors.greenAccent обрано як універсальний патерн підтвердження дії (зелений - "безпечно", "дзвонити"). При натисканні викликається метод _makePhoneCall(contact.phone), який формує URI та запускає системний додаток телефонії.

Якщо список контактів порожній, відображається заглушка з пропозицією додати контакт через режим редагування.

Лістинг 2.2 - Візуальне проєктування картки контакту

```
return Card(
  margin: const EdgeInsets.symmetric(horizontal: 20, vertical: 6),
  color: Colors.white.withOpacity(0.1),
  shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(20)),
  child: ListTile(
    leading: CircleAvatar(
      backgroundColor: contact.avatarColor,
      backgroundImage: contact.imagePath != null ? FileImage(File(contact.imagePath!)) : null,
      child: contact.imagePath == null
          ? Text(contact.name.substring(0, 1).toUpperCase(), style: const TextStyle(fontWeight: FontWeight.bold))
          : null,
    ), CircleAvatar
    title: Text(contact.name, style: const TextStyle(color: Colors.white)),
    trailing: IconButton(
      icon: const Icon(Icons.call, color: Colors.greenAccent),
      onPressed: () => _callContact(contact),
    ), IconButton
    onTap: () => _callContact(contact),
  ), ListTile
); Card
```

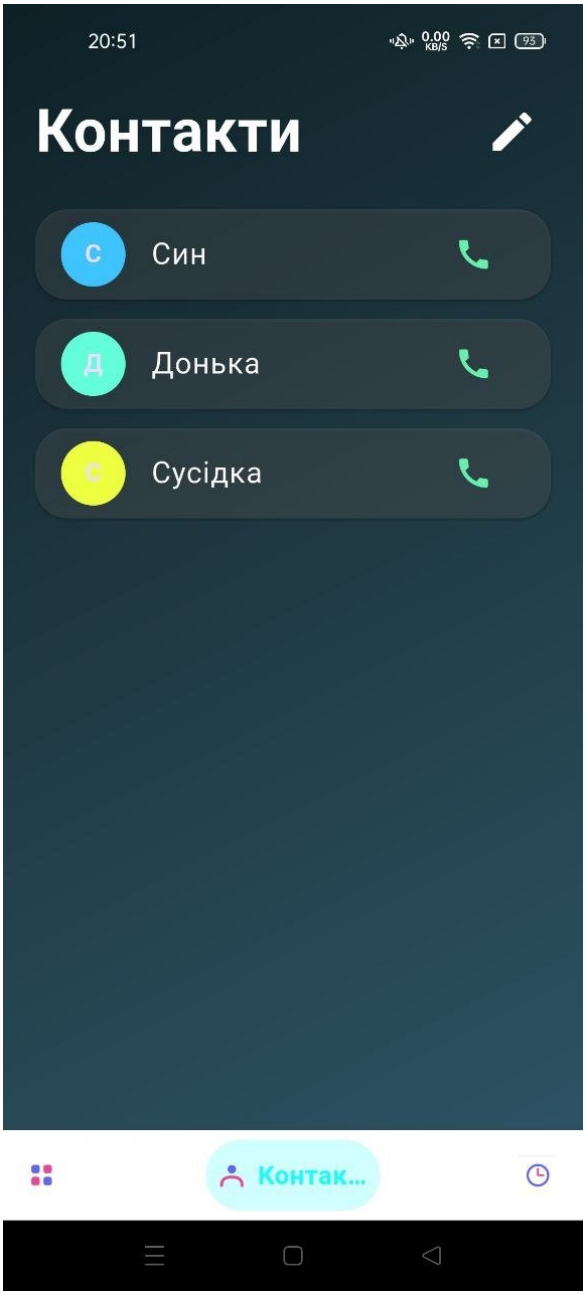

Рисунок 2.5 - Графічний інтерфейс екрана пріоритетних контактів

2.2.3. Візуальне проєктування голосового годинника (ClockScreen)

Екран ClockScreen створено для забезпечення максимальної доступності інформації про час. Інтерфейс екрана мінімалістичний і містить лише три ключові UI-компоненти, розташовані вертикально з великими відступами:

- Цифровий годинник - реалізований через віджет Text із критично великим розміром шрифту (fontSize: 96) та збільшеним інтервалом між символами (letterSpacing: 8). Використано моноширинний шрифт, щоб уникнути

зсуву цифр під час оновлення. Колір тексту - білий, що забезпечує максимальний контраст на темному фоні. Формат часу - 24-годинний (HH:mm:ss), але секунди відображаються меншим шрифтом (48 pt), щоб не перевантажувати екран.

˜ Дата - відображається під годинником шрифтом розміру 24 pt. Формат дати - «середа, 15 червня 2026». Використано локалізовану версію (uk_UA) через клас DateFormat.

˜ Кнопка озвучення - масивний квадратний віджет ElevatedButton, ширина якого динамічно адаптується під розмір екрана (займає 90% ширини дисплея смартфона). Висота кнопки - 80 dp. Колір - акцентний Colors.teal, текст - «ОЗВУЧИТИ ЧАС» великими літерами для кращої читабельності. При натисканні ініціюється синтез мовлення через flutter_tts із попереднім налаштуванням мови «uk-UA» та зниженою швидкістю (0.5). Під час відтворення кнопка стає неактивною (onPressed: null) для запобігання повторним натисканням, а після закінчення відтворення знову активується.

У коді використано Timer.periodic для щосекундного оновлення часу. Таймер створюється в методі initState і знищується в dispose, що запобігає витокам пам'яті. Оновлення інтерфейсу відбувається через виклик setState(), який перемальовує лише змінені частини.

Лістинг 2.3 - Типографіка віджета годинника

```
Text(
  DateFormat('HH:mm').format(_currentTime),
  style: const TextStyle(fontSize: 96, fontWeight: FontWeight.bold, letterSpacing: 8, color: Colors.white),
), Text
```

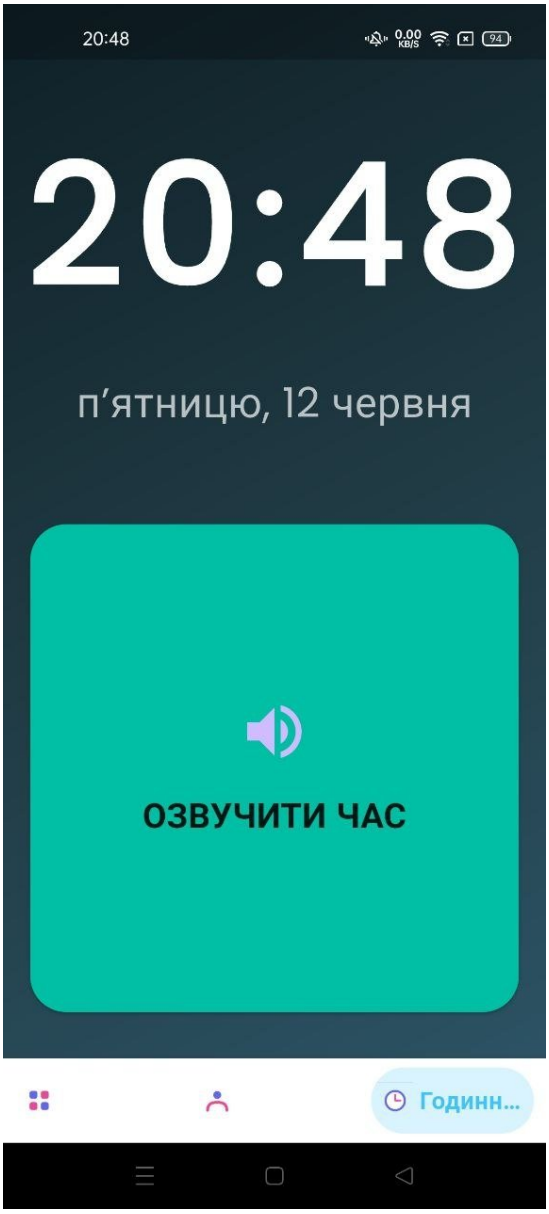

Рисунок 2.6 - Інтерфейс екрана голосового годинника

2.2.4. Інтерфейси режимів конфігурації (Editor Screens*)*

Для забезпечення захисту від випадкових змін з боку цільового користувача, усі інструменти конфігурації винесено в окремі екрани (WidgetsEditorScreen та ContactsEditorScreen), доступ до яких здійснюється через іконку редагування (олівець) у верхньому правому куті головного екрана та екрана контактів. Крім того, доступ до режиму редагування блокується, якщо активовано Lock mode (глобальний прапорець isLocked = true), який встановлюється опікуном. Це запобігає випадковому видаленню або переміщенню іконок літньою людиною.

Візуальна взаємодія в режимі редагування побудована на базі модальних вікон (AlertDialog). При спробі додати новий віджет або контакт система генерує спливаюче вікно, яке містить:

˜ Текстове поле (TextField) для введення назви (для віджетів - назва додатка, для контактів - ім'я абонента).

˜ Випадаючий список (DropdownButton) для вибору іконки (тільки для віджетів) - іконки відображаються у вигляді прев'ю з назвами.

˜ Випадаючий список для вибору кольору з попередньо заданої палітри безпечних кольорів (8 кольорів високої контрастності: червоний, зелений, синій, жовтий, фіолетовий, помаранчевий, бірюзовий, рожевий). Користувач не може ввести довільний колір, що зменшує ризик помилки.

˜ (Лише для контактів) кнопка «Вибрати фото», яка викликає image_picker для завантаження фотографії з галереї або камери. Після вибору фото зберігається в локальному сховищі, а шлях записується в поле imagePath.

˜ Кнопки «Скасувати» та «Зберегти».

Після підтвердження дані валідуються (назва не має бути порожньою, номер телефону має відповідати формату), серіалізуються в JSON і зберігаються через SharedPreferences. Потім попередній екран (AppsScreen або ContactsScreen) отримує сповіщення через Navigator.pop(context, true) та викликає setState() для перемальовування.

Для видалення елемента використовується свайп вліво (на екрані редагування) або кнопка «-» біля кожного елемента. Перед видаленням з'являється діалогове вікно підтвердження, щоб уникнути випадкових дій.

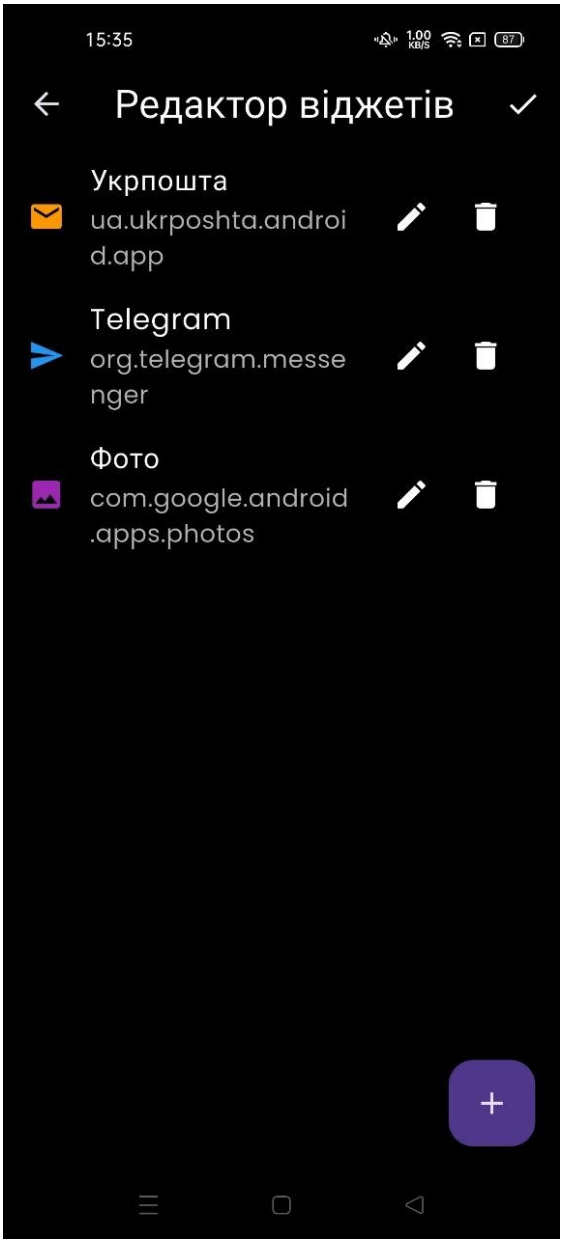

Рисунок 2.7 - Інтерфейс діалогового вікна налаштування елементів

2.3. Реалізація функціональних модулів швидкого зв'язку, голосового годинника та системної інтеграції

Розробка спеціалізованого мобільного лаунчера вимагає не лише створення зручного графічного інтерфейсу, але й глибокої взаємодії з апаратними та програмними підсистемами ОС Android. У цьому підрозділі детально розглянуто реалізацію трьох ключових модулів: голосового годинника (аудіальне дублювання часу), підсистеми швидкого зв'язку (ініціалізація

телефонних викликів) та механізму системної інтеграції через платформні канали (доступ до списку встановлених додатків та їх запуск).

2.3.1 Програмна реалізація голосового годинника

Одним із ключових асистивних компонентів є екран ClockScreen, який дублює візуальну інформацію аудіальним каналом для користувачів із вадами зору. Обчислювальна логіка модуля базується на використанні класу Timer з конструктором Timer.periodic, який щосекунди ініціює оновлення локальної змінної _currentTime типу DateTime. Оновлення відбувається лише тоді, коли екран видимий (widget mounted), що запобігає зайвим викликам setState() у фоновому режимі.

Для форматування часу у зручний для читання текстовий вигляд застосовується клас DateFormat із бібліотеки intl. Використано локалізований формат для української мови (uk_UA), що забезпечує коректне відображення назв місяців та днів тижня.

Лістинг 2.4 - Ініціалізація таймера для оновлення часу

```
@override
void initState() {
  super.initState();
  _currentTime = DateTime.now();
  _timer = Timer.periodic(const Duration(seconds: 1), (timer) {
    setState(() => _currentTime = DateTime.now());
  }); Timer.periodic
  _flutterTts = FlutterTts();
}

@override
void dispose() {
  _timer.cancel();
  _flutterTts.stop();
  super.dispose();
}
```

Функція синтезу мовлення реалізована за допомогою пакета flutter_tts. Процедура озвучення ініціюється натисканням на центральну кнопку екрана. У методі _speakTime() відбувається асинхронне налаштування параметрів аудіопроцесора:

˜ Встановлюється мовний пакет uk-UA через setLanguage(). Якщо пакет відсутній на пристрої, система використовує мову за замовчуванням (наприклад, en-US), але генерує попередження.

˜ Сповільнюється швидкість читання (setSpeechRate(0.5)) - це критично важливо для літніх людей, оскільки швидке мовлення сприймається гірше. Діапазон швидкості - від 0.0 до 1.0, де 0.5 є оптимальним для аудиторії 50+.

˜ Налаштовується висота тону (setPitch(1.0)) - залишено стандартною, оскільки зміна тону може спотворити впізнаваність голосу.

˜ Форматується текстовий рядок часу: «Година X, хвилина Y» (наприклад, «Чотирнадцята година тридцять дві хвилини»). Для цього використовується власна функція _formatTimeForSpeech(DateTime time), яка перетворює 24-годинний формат у словесний.

˜ Після формування рядка викликається speak(text), який передає дані в нативний рушій Text-to-Speech.

Важливою особливістю є обробка стану завантаження: під час синтезу кнопка блокується (стає неактивною), а після завершення (через then() або whenComplete()) розблоковується. Це запобігає накладанню кількох голосових повідомлень.

2.3.2 Інтеграція підсистеми викликів та управління медіафайлами

Модуль швидкого зв'язку (ContactsScreen) відповідає за управління списком пріоритетних абонентів та ініціалізацію телефонних викликів. Для безпосереднього здійснення дзвінка використовується плагін url_launcher. При виборі контакту система формує спеціальну URI-схему (наприклад, tel:+380501234567). Метод canLaunchUrl() виконує попередню

валідацію сформованого запиту, після чого функція launchUrl передає керування системному додатку телефонії (Dialer) операційної системи Android.

Лістинг 2.5 - Функція здійснення виклику

```
void _callContact(Contact contact) async {
  final Uri telUri = Uri(scheme: 'tel', path: contact.phone);
  if (await canLaunchUrl(telUri)) {
    await launchUrl(telUri);
  } else {
    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Не вдалося зателефонувати ${contact.name}', style: const TextStyle(color: Colors.white))),
    );
  }
}
```

Оскільки система дозволяє персоналізувати контакти шляхом додавання фотографій (збереження локального шляху imagePath), реалізовано механізм безпечного доступу до файлової системи смартфона за допомогою пакета image_picker. Для цього у класі ContactsEditorScreen імплементовано метод _requestStoragePermission(), який через плагін permission_handler ініціює системний запит на отримання прав Permission.photos (для Android 13+) або Permission.storage (для старіших версій). Після отримання дозволу користувач може вибрати зображення з галереї або зробити нове фото.

Обране зображення зберігається у внутрішньому сховищі програми (папка getApplicationDocumentsDirectory()) з унікальним іменем, щоб уникнути конфліктів. Шлях до файлу записується у поле imagePath відповідного контакту. Під час відображення контакту система перевіряє наявність файлу за цим шляхом; якщо файл відсутній (користувач видалив його вручну), аватар автоматично замінюється на кольоровий круг із літерою.

2.3.3 Системна взаємодія через платформні канали (Method Channels)

Найбільш критичною частиною розробки лаунчера є отримання доступу до списку встановлених програм та їх запуск. Оскільки Flutter виконується у власному ізольованому середовищі (Sandboxed Engine) і не має прямих API для

роботи з менеджером пакетів Android, у системі застосовано механізм платформних каналів (Platform Channels). Це асинхронний двосторонній міст між кодом на Dart та нативним кодом Android (Java/Kotlin) або iOS (Swift/Objective-C).

У класах AppsScreen та WidgetsEditorScreen ініціалізується об'єкт MethodChannel('apps_channel'), який виконує роль такого моста. Взаємодія відбувається за двома основними векторами:

˜ Читання даних - виклик методу invokeMethod('getInstalledApps') звертається до нативного коду, який через PackageManager отримує список всіх інстальованих додатків. Для кожного додатка збираються: назва (applicationInfo.loadLabel()), ім'я пакета (packageName), іконка (applicationInfo.loadIcon()). Ці дані упаковуються в Map (або JSON) і повертаються у Flutter, де відображаються в меню редагування віджетів.

˜ Виконання дій - метод _launchApp(String packageName) передає ідентифікатор обраної програми через канал у нативне середовище, де операційна система формує відповідний системний намір (Intent) і запускає цільовий застосунок поверх інтерфейсу розробленої оболонки.

2.3.4 Обробка системних подій та блокування кнопки «Назад»

Оскільки лаунчер призначений для заміни стандартного домашнього екрана, він повинен коректно реагувати на системні кнопки. Особливу увагу приділено обробці кнопки «Назад» (back button). У звичайному додатку натискання «Назад» призводить до закриття поточної Activity або виходу з програми. Для лаунчера така поведінка є неприйнятною: користувач не повинен випадково закрити оболонку.

Для вирішення цієї проблеми в кореневому віджеті MainScreen використано віджет PopScope (до Flutter 3.16 використовувався WillPopScope). Цей віджет перехоплює системний виклик «Назад» і дозволяє визначити власну логіку. У реалізації лаунчера логіка така:

˜ Якщо поточна вкладка не є головною (тобто pageController.page != 0), то натискання «Назад» переміщує користувача на головний екран (AppsScreen).

˜ Якщо поточна вкладка є головною, то натискання «Назад» ігнорується (лаунчер залишається активним).

Така поведінка повністю відповідає очікуванням літніх користувачів: вони завжди можуть повернутися на головний екран, але не можуть випадково вийти з програми.

Лістинг 2.7 - Перехоплення кнопки «Назад» за допомогою PopScope

```
return PopScope(
  canPop: false,
  onPopInvokedWithResult: (bool didPop, Object? result) {
    if (didPop) return;

    if (_currentIndex != 0) {
      setState(() => _currentIndex = 0);
      _pageController.animateToPage(
        0,
        duration: const Duration(milliseconds: 300),
        curve: Curves.easeInCubic,
      );
    }
  },
```

2.4 Тестування розробленого додатка

Тестування є невід'ємним і критично важливим етапом життєвого циклу розробки програмного забезпечення. Метою цього етапу є перевірка відповідності розробленого мобільного додатка (лаунчера) початковим вимогам, виявлення та усунення помилок, а також перевірка стабільності роботи програми на пристроях з операційною системою Android.

2.4.1 Види тестування

˜ Функціональне тестування - перевірка основних функцій: формування списку додатків, запуск програм, відображення контактів, робота годинника, переходи між вкладками.

˜ Тестування інтеграції з ОС - перевірка запиту системних дозволів, роботи MethodChannel для отримання списку пакетів, запуску дзвінків через url_launcher.

˜ Тестування інтерфейсу користувача (UI/UX) - оцінка плавності анімацій, адаптивності верстки, контрастності та читабельності.

˜ Тестування системних переривань - поведінка при натисканні кнопок «Назад», «Додому», при вхідних дзвінках, блокуванні екрана.

2.4.2 Тестові сценарії та результати

Таблиця 2.1 - Результати тестування

	Опис тестового сценарію	Очікуваний результат	Фактичний результат	Статус
1	Запуск додатка на пристрої	Додаток завантажується без помилок	Як очікувалося	Успішно
2	Завантаження списку додатків на головному екрані	Відображається сітка 2×4 з назвами та іконками	Як очікувалося	Успішно
3	Натискання на іконку стороннього додатка	Додаток запускається поверх лаунчера	Як очікувалося	Успішно
4	Перехід на вкладку «Контакти» (перший запуск)	Пуста сторінка, з пропозицією створення контактів	Як очікувалося	Успішно

Продовження таблиці 2.1 - Результати тестування

5	Перевірка роботи вкладки «Годинник»	Відображається поточний час, оновлення щосекунди	Як очікувалося	Успішно
6	Натискання на кнопку «Озвучити час»	Відтворюється голосове повідомлення	Як очікувалося	Успішно
7	Навігація через нижню панель (BottomNavyBar)	Плавний перехід між екранами, активна іконка підсвічується	Як очікувалося	Успішно
8	Свайп по екрану вліво/вправо	Гортання сторінок, панель синхронно оновлюється	Як очікувалося	Успішно
9	Натискання кнопки «Назад» на вкладках «Контакти» або «Годинник»	Додаток не закривається, а переходить на головну	Як очікувалося	Успішно
10	Натискання кнопки «Назад» на головній вкладці	Дія ігнорується.	Як очікувалося	Успішно
11	Додавання нового контакту через ContactsEditorScreen	Контакт з'являється у списку	Як очікувалося	Успішно
12	Вибір фотографії для контакту	Фото відображається в аватарі, шлях зберігається	Як очікувалося	Успішно
13	Робота лаунчера після перезавантаження пристрою	Лаунчер залишається домашнім екраном, налаштування збережено	Як очікувалося	Успішно

2.4.3 Виявлені помилки та їх усунення

1. Проблема із закриттям додатка кнопкою «Назад» (сценарій №9-10). Вирішено шляхом інтеграції віджета PopScope, який перехоплює системний виклик.

2. Затримка при першому відкритті екрана контактів (сценарій №4). Запит дозволу перенесено в initState з асинхронною обробкою.

3. Некоректне відображення кольорів. Замінено кольори на більш насичені з палітри Material Design.

2.4.4 Тестування з реальним користувачем

Для оцінки зручності використання було проведено тестування за участю користувача з катарактою (жінка, 72 роки). Вона не мала попереднього досвіду роботи зі смартфонами. Після короткого ознайомлення з лаунчером їй запропонували виконати базові дії: знайти додаток «Телефон», зателефонувати за номером зі списку контактів, дізнатися поточний час за допомогою голосового годинника.

Фідбек користувачки: «Іконки додатків видно добре, список контактів мілкуватий, але доволі зручний, а функція голосового годинника спрацьовує справно, і кнопка досить велика».

Усі завдання було виконано успішно без сторонньої допомоги. Зауваження щодо дрібнуватого списку контактів буде враховано в наступних версіях (можливе збільшення висоти рядка та шрифту).

Проведене тестування показало, що розроблений мобільний додаток працює стабільно та виконує всі закладені функції. Критичних помилок (крашів) не виявлено. Логіка навігації та блокування випадкового виходу з лаунчера працює коректно. Усі 13 тестових сценаріїв завершилися зі статусом «Успішно». Відгук реального користувача з вадами зору підтверджує ефективність розробленого рішення.

Висновки розділу 2

У другому розділі спроєктовано, реалізовано та протестовано асистивний мобільний лаунчер. Розроблено ER-діаграму, DFD та діаграму класів, що лягли в основу архітектури. Створено інклюзивний інтерфейс із фіксованою сіткою 2×4, пласкою навігацією, екраном контактів та голосовим годинником. Реалізовано системну інтеграцію через MethodChannel (доступ до списку додатків), url_launcher (дзвінки) та flutter_tts (озвучення часу). Проведено тестування (13 сценаріїв), усунено виявлені помилки, отримано позитивний відгук реального користувача з катарактою. Розроблений лаунчер є стабільним і відповідає вимогам.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне завдання - розроблено асистивний мобільний лаунчер для людей похилого віку.

Проаналізовано фізіологічні та когнітивні бар'єри цільової аудиторії (зір, моторика, пам'ять). Визначено ключові вимоги до інтерфейсу:

- великі зони натискання
- висока контрастність
- обмежена глибина навігації
- фіксована сітка 2×4.

Виконано порівняльний аналіз існуючих лаунчерів (Nothing, Niagara, Nova, BIG Launcher, BaldPhone, вбудовані режими). Виявлено їхні недоліки для аудиторії 50+: складність, дрібні елементи, агресивна монетизація, надмірна текстовість, фрагментарність дії.

Обґрунтовано вибір технологій: Flutter (рендеринг через Skia/Impeller) та Dart, що забезпечує продуктивність на бюджетних пристроях з 2 ГБ RAM.

Реалізовано програмний продукт з такими функціями: головний екран із сіткою 2×4, екран швидкого набору контактів, голосовий годинник (flutter_tts з уповільненою швидкістю), режим редагування через модальні діалоги. Забезпечено плоску навігацію (BottomNavyBar + PageView) та перехоплення кнопки «Назад» (PopScope).

Проведено тестування (13 сценаріїв - успішно). Виправлено помилки: закриття програми кнопкою «Назад», затримка при відкритті контактів. Тестування з реальним користувачем з катарактою підтвердило зручність: «іконки видно добре, голосовий годинник спрацьовує справно, кнопка велика».

Розроблений лаунчер відповідає поставленим вимогам, є стабільним та готовим до використання цільовою аудиторією.

СПИСКИ ЛІТЕРАТУРИ

1. Android Developers. Make your app more accessible. URL: https://developer.android.com/guide/topics/ui/accessibility (дата звернення: 15.05.2026).

2. Material Design 3. Accessibility fundamentals. Google, 2026. URL: https://m3.material.io/foundations/accessible-design/overview (дата звернення: 16.05.2026).

3. Кут В. І., Коваленко О. М. Моделювання інклюзивних людино-машинних інтерфейсів на основі фундаментальних законів ергономіки. Інформаційні технології та комп'ютерна інженерія. 2023. № 2. С. 34-41.

4. Сидоренко О. В. Проблеми «цифрового розриву» та інклюзії літніх людей у сучасному інформаційному суспільстві. Вісник соціальних технологій. 2023. № 4. С. 56-64.

5. Бойко В. І., Коваленко О. М. Основи проєктування користувацьких інтерфейсів (UI/UX). Київ: КПІ ім. Ігоря Сікорського, 2021. 245 с.

6. Petrovska I., Zdebskyi O. Digital accessibility: how the new iOS and Android design systems are transforming the mobile user experience. Sqli Insights. 2025. URL: https://www.sqli.com/int-en/insights/accessibilite-numerique-mobile-nouveaux-design-systems-ios-android (дата звернення: 21.05.2026).

7. BIG Launcher: fast and simple Android home screen for seniors. Similarweb. 2024. URL: https://www.similarweb.com/app/google/name.kunes.android.launcher.demo (дата звернення: 21.05.2026).

8. Nova Launcher's Fall: One More Nail In The Coffin Of Classic Android. Techky Skills. 2026. URL: https://techkyskills.com/nova-launchers-fall-one-more-nail-in-the-coffin-of-classic-android (дата звернення: 22.05.2026).

9. Nothing Launcher user reviews and ratings. Chrome Stats. URL: https://chrome-stats.com/d/com.nothing.launcher/reviews (дата звернення: 22.05.2026).

10. Офіційна документація мови програмування Dart. Dart Developers. 2026. URL: https://dart.dev/guides (дата звернення: 28.05.2026).

11. Офіційна документація фреймворку Flutter. Flutter Developers. 2026. URL: https://docs.flutter.dev/ (дата звернення: 28.05.2026).

12. Payne R. Beginning App Development with Flutter: Create Cross-Platform Mobile Apps. 2nd ed. Berkeley: Apress, 2021. 364 p.

13. Rose R. Flutter for Beginners: An introductory guide to building cross-platform mobile applications. 3rd ed. Birmingham: Packt Publishing, 2023. 412 p.

14. Android Developers. AccessibilityManager API reference. URL: https://developer.android.com/reference/kotlin/android/view/accessibility/AccessibilityManager (дата звернення: 30.05.2026).

15. Flutter Team. Platform channels – writing platform-specific code. Flutter Docs. 2026. URL: https://docs.flutter.dev/platform-integration/platform-channels (дата звернення: 02.06.2026).

16. shared_preferences – Flutter plugin for reading/writing simple key-value pairs. pub.dev. 2026. URL: https://pub.dev/packages/shared_preferences (дата звернення: 02.06.2026).

17. flutter_tts 3.2.0 – Flutter plugin for Text-to-Speech. pub.dev. URL: https://pub.dev/packages/flutter_tts/versions/3.2.0 (дата звернення: 02.06.2026).

18. url_launcher 6.3.1 – Flutter plugin for launching a URL. pub.dev. URL: https://pub.dev/documentation/url_launcher/6.3.1 (дата звернення: 05.06.2026).

19. Моркун Н. В., Завсегдашня І. В. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для студентів спеціальності 122 “Комп’ютерні науки”. Кривий Ріг : Видавничий центр КНУ, 2021. 50 с.

20. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

21. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

22. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація).

ДОДАТОК А

Лістинг програмного коду додатка

Лістинг А.1 – Вкладка віджетів (main.dart)

```
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:bottom_navy_bar/bottom_navy_bar.dart';
import 'package:intl/date_symbol_data_local.dart';
import 'apps_screen.dart';
import 'contacts_screen.dart';
import 'clock_screen.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await initializeDateFormatting('ru', null);
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'My Awesome App',
      theme: ThemeData(
        brightness: Brightness.dark,
        primarySwatch: Colors.teal,
        textTheme: GoogleFonts.poppinsTextTheme(),
        scaffoldBackgroundColor: Colors.transparent,
      ),
      debugShowCheckedModeBanner: false,
      home: const MainScreen(),
    );
  }
}

class MainScreen extends StatefulWidget {
  const MainScreen({super.key});

  @override
  State<MainScreen> createState() => _MainScreenState();
}

class _MainScreenState extends State<MainScreen> {
  int _currentIndex = 0;
  late PageController _pageController;

  @override
```

```
void initState() {
  super.initState();
  _pageController = PageController();
}

@override
void dispose() {
  _pageController.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  return PopScope(
    canPop: false,
    onPopInvokedWithResult: (bool didPop, Object? result) {
      if (didPop) return;

      if (_currentIndex != 0) {
        setState(() => _currentIndex = 0);
        _pageController.animateToPage(
          0,
          duration: const Duration(milliseconds: 300),
          curve: Curves.easeInCubic,
        );
      }
    },
    child: Scaffold(
      body: Container(
        decoration: const BoxDecoration(
          gradient: LinearGradient(
            begin: Alignment.topLeft,
            end: Alignment.bottomRight,
            colors: [Color(0xFF0F2027), Color(0xFF203A43), Color(0xFF2C5364)],
          ),
        ),
        child: PageView(
          controller: _pageController,
          onPageChanged: (index) => setState(() => _currentIndex = index),
          children: const [
            AppsScreen(),
            ContactsScreen(),
            ClockScreen(),
          ],
        ),
      ),
      bottomNavigationBar: BottomNavyBar(
        selectedIndex: _currentIndex,
        showElevation: true,
        onItemSelected: (index) {
          setState(() => _currentIndex = index);
          _pageController.animateToPage(
```

```
            index,
            duration: const Duration(milliseconds: 300),
            curve: Curves.easeInCubic,
          );
        },
        items: [
          BottomNavyBarItem(
            icon: Image.asset('assets/icons/apps.png', width: 24, height: 24),
            title: const Text('Віджети'),
            activeColor: Colors.tealAccent,
          ),
          BottomNavyBarItem(
            icon: Image.asset('assets/icons/contacts.png', width: 24, height: 24),
            title: const Text('Контакти'),
            activeColor: Colors.cyanAccent,
          ),
          BottomNavyBarItem(
            icon: Image.asset('assets/icons/clock.png', width: 24, height: 24),
            title: const Text('Годинник'),
            activeColor: Colors.lightBlueAccent,
          ),
        ],
      ),
    ),
  );
 }
}
```

Лістинг А.2 – Екран додатків (apps_screen.dart)

```
import 'dart:convert';
import 'dart:typed_data';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:shared_preferences/shared_preferences.dart';

class AppWidget {
 String id;
 String name;
 IconData icon;
 Color color;
 String packageName;

 AppWidget({
  required this.id,
  required this.name,
  required this.icon,
  required this.color,
  required this.packageName,
 });

 Map<String, dynamic> toJson() => {
```

```
    'id': id,
    'name': name,
    'iconCode': icon.codePoint,
    'iconFontFamily': icon.fontFamily,
    'iconPackage': icon.fontPackage,
    'color': color.value,
    'packageName': packageName,
  };

  factory AppWidget.fromJson(Map<String, dynamic> json) {
    return AppWidget(
      id: json['id'],
      name: json['name'],
      icon: IconData(json['iconCode'],
          fontFamily: json['iconFontFamily'],
          fontPackage: json['iconPackage']),
      color: Color(json['color']),
      packageName: json['packageName'],
    );
  }
}

class AppsScreen extends StatefulWidget {
  const AppsScreen({super.key});

  @override
  State<AppsScreen> createState() => _AppsScreenState();
}

class _AppsScreenState extends State<AppsScreen> {
  List<AppWidget> _widgets = [];
  final MethodChannel _channel = const MethodChannel('apps_channel');

  @override
  void initState() {
    super.initState();
    _loadWidgets();
  }

  Future<void> _loadWidgets() async {
    final prefs = await SharedPreferences.getInstance();
    final String? widgetsJsonString = prefs.getString('custom_widgets_list');
    if (widgetsJsonString != null) {
      final List<dynamic> decodedList = jsonDecode(widgetsJsonString);
      setState(() {
        _widgets = decodedList.map((json) => AppWidget.fromJson(json)).toList();
      });
    } else {
      setState(() {
        _widgets = [];
      });
    }
```

```
}

Future<void> _saveWidgets() async {
  final prefs = await SharedPreferences.getInstance();
  final String encodedData = jsonEncode(_widgets.map((w) => w.toJson()).toList());
  await prefs.setString('custom_widgets_list', encodedData);
}

Future<void> _launchApp(String packageName) async {
  if (packageName.isEmpty) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Не вибрано додаток для цього віджета')),
    );
    return;
  }
  try {
    await _channel.invokeMethod('launchApp', {'packageName': packageName});
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Помилка запуску: $e')),
    );
  }
}

void _openEditor() async {
  final result = await Navigator.push(
    context,
    MaterialPageRoute(builder: (ctx) => WidgetsEditorScreen(widgets: _widgets)),
  );
  if (result != null && result is List<AppWidget>) {
    setState(() {
      _widgets = result;
      _saveWidgets();
    });
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.transparent,
    body: SafeArea(
      child: Column(
        children: [
          Align(
            alignment: Alignment.topRight,
            child: IconButton(
              icon: const Icon(Icons.edit, color: Colors.white, size: 32),
              onPressed: _openEditor,
              tooltip: 'Налаштувати віджети',
            ),
          ),
```

```
        const SizedBox(height: 8),
        Expanded(
         child: _widgets.isEmpty
            ? const Center(
            child: Text('Натисніть ✏, щоб додати застосунки',
               style: TextStyle(color: Colors.white)))
            : GridView.builder(
           padding: const EdgeInsets.all(16),
           gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
            crossAxisCount: 2,
            childAspectRatio: 1.0,
            crossAxisSpacing: 16,
            mainAxisSpacing: 16,
           ),
           itemCount: _widgets.length,
           itemBuilder: (ctx, index) {
            final w = _widgets[index];
            return _buildWidgetTile(w);
           },
         ),
        ),
       ],
     ),
    ),
  );
}

Widget _buildWidgetTile(AppWidget w) {
 return GestureDetector(
   onTap: () => _launchApp(w.packageName),
   child: Container(
    decoration: BoxDecoration(
     gradient: LinearGradient(
      colors: [w.color, w.color.withOpacity(0.7)],
      begin: Alignment.topLeft,
      end: Alignment.bottomRight,
     ),
     borderRadius: BorderRadius.circular(24),
     boxShadow: [BoxShadow(color: Colors.black26, blurRadius: 8, offset: const Offset(0, 4))],
    ),
    child: Column(
     mainAxisAlignment: MainAxisAlignment.center,
     children: [
      Icon(w.icon, size: 64, color: Colors.white),
      const SizedBox(height: 12),
      Text(
       w.name,
       style: const TextStyle(fontSize: 18, fontWeight: FontWeight.bold, color: Colors.white),
       textAlign: TextAlign.center,
       maxLines: 2,
       overflow: TextOverflow.ellipsis,
      ),
```

```
          ],
        ),
      ),
    );
  }
}

class WidgetsEditorScreen extends StatefulWidget {
  final List<AppWidget> widgets;
  const WidgetsEditorScreen({super.key, required this.widgets});

  @override
  State<WidgetsEditorScreen> createState() => _WidgetsEditorScreenState();
}

class _WidgetsEditorScreenState extends State<WidgetsEditorScreen> {
  late List<AppWidget> _tempWidgets;
  List<Map<String, dynamic>> _installedApps = [];
  bool _isLoading = true;
  final MethodChannel _channel = const MethodChannel('apps_channel');

  final List<IconData> _availableIcons = [
    Icons.star, Icons.camera_alt, Icons.send, Icons.play_circle_fill,
    Icons.chat, Icons.music_note, Icons.email, Icons.photo, Icons.map,
    Icons.shopping_cart, Icons.book, Icons.fitness_center, Icons.gamepad,
    Icons.home, Icons.work, Icons.school, Icons.cake
  ];
  final List<Color> _availableColors = [
    Colors.purple, Colors.blue, Colors.red, Colors.green,
    Colors.orange, Colors.pink, Colors.teal, Colors.indigo,
  ];

  @override
  void initState() {
    super.initState();
    _tempWidgets = List.from(widget.widgets);
    _loadInstalledApps();
  }

  Future<void> _loadInstalledApps() async {
    setState(() => _isLoading = true);
    try {
      final List<dynamic> result = await _channel.invokeMethod('getInstalledApps');
      _installedApps = result.map((item) {
        final map = item as Map<dynamic, dynamic>;
        return {
          'packageName': map['packageName'] as String,
          'appName': map['appName'] as String,
          'icon': map['icon'] as Uint8List?,
        };
      }).toList();
      setState(() => _isLoading = false);
```

```
  } catch (e) {
    setState(() => _isLoading = false);
    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Помилка завантаження додатків: $e')),
    );
  }
}

void _addWidget() {
  if (_tempWidgets.length >= 8) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Максимум 8 віджетів')),
    );
    return;
  }
  final newId = DateTime.now().millisecondsSinceEpoch.toString();
  _tempWidgets.add(AppWidget(
    id: newId,
    name: 'Новий віджет',
    icon: Icons.star,
    color: Colors.blue,
    packageName: '',
  ));
  setState(() {});
}

Future<void> _editWidget(int index) async {
  final w = _tempWidgets[index];
  final nameController = TextEditingController(text: w.name);
  IconData selectedIcon = _availableIcons.contains(w.icon) ? w.icon : _availableIcons.first;
  Color selectedColor = _availableColors.contains(w.color) ? w.color : _availableColors.first;
  String selectedPackage = w.packageName;

  if (_isLoading) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Зачекайте, додатки ще завантажуються...')),
    );
    return;
  }

  final String? chosenPackage = await showDialog<String>(
    context: context,
    builder: (ctx) {
      return SimpleDialog(
        title: const Text('Виберіть додаток', style: TextStyle(color: Colors.white)),
        backgroundColor: const Color(0xFF1E2A3A),
        children: _installedApps.map((app) {
          final Uint8List? iconBytes = app['icon'];
          return SimpleDialogOption(
            child: Row(
              children: [
```

```
                if (iconBytes != null && iconBytes.isNotEmpty)
                  Image.memory(iconBytes, width: 32, height: 32)
                else
                  const Icon(Icons.apps, color: Colors.white),
                const SizedBox(width: 16),
                Text(app['appName'], style: const TextStyle(color: Colors.white)),
              ],
            ),
            onPressed: () {
              Navigator.pop(ctx, app['packageName']);
            },
          );
        }).toList(),
      );
    },
  );

  if (chosenPackage != null) {
    final selectedApp = _installedApps.firstWhere((app) => app['packageName'] ==
chosenPackage);
    selectedPackage = chosenPackage;
    if (nameController.text.trim().isEmpty || nameController.text.trim() == 'Новий віджет') {
      nameController.text = selectedApp['appName'];
    }
  }

  await showDialog(
    context: context,
    builder: (ctx) {
      return StatefulBuilder(
        builder: (ctx, setStateDialog) {
          return AlertDialog(
            backgroundColor: const Color(0xFF1E2A3A),
            title: const Text('Редагувати віджет', style: TextStyle(color: Colors.white)),
            content: Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                TextField(
                  controller: nameController,
                  style: const TextStyle(color: Colors.white),
                  decoration: const InputDecoration(
                    labelText: 'Назва віджета',
                    labelStyle: TextStyle(color: Colors.white70),
                  ),
                ),
                const SizedBox(height: 16),
                Row(
                  children: [
                    const Text('Іконка:', style: TextStyle(color: Colors.white)),
                    const SizedBox(width: 8),
                    Expanded(
                      child: DropdownButton<IconData>(
```

```
            value: selectedIcon,
            dropdownColor: const Color(0xFF1E2A3A),
            items: _availableIcons.map((icon) {
              return DropdownMenuItem(
                value: icon,
                child: Icon(icon, color: Colors.white),
              );
            }).toList(),
            onChanged: (icon) => setStateDialog(() => selectedIcon = icon!),
          ),
        ),
      ],
    ),
    const SizedBox(height: 12),
    Row(
      children: [
        const Text('Колір:', style: TextStyle(color: Colors.white)),
        const SizedBox(width: 8),
        Expanded(
          child: DropdownButton<Color>(
            value: selectedColor,
            dropdownColor: const Color(0xFF1E2A3A),
            items: _availableColors.map((color) {
              return DropdownMenuItem(
                value: color,
                child: Container(width: 40, height: 40, color: color),
              );
            }).toList(),
            onChanged: (color) => setStateDialog(() => selectedColor = color!),
          ),
        ),
      ],
    ),
  ],
),
actions: [
  TextButton(
    onPressed: () => Navigator.pop(ctx),
    child: const Text('Скасувати', style: TextStyle(color: Colors.white70)),
  ),
  ElevatedButton(
    onPressed: () {
      setState(() {
        _tempWidgets[index] = AppWidget(
          id: w.id,
          name: nameController.text.trim().isEmpty ? 'Віджет' : nameController.text.trim(),
          icon: selectedIcon,
          color: selectedColor,
          packageName: selectedPackage,
        );
      });
      Navigator.pop(ctx);
```

```
              },
              child: const Text('Зберегти'),
            ),
          ],
        );
      },
    );
  },
 );
}

void _deleteWidget(int index) {
 setState(() {
  _tempWidgets.removeAt(index);
 });
}

@override
Widget build(BuildContext context) {
 return Scaffold(
  backgroundColor: Colors.transparent,
  appBar: AppBar(
   title: const Text('Редактор віджетів', style: TextStyle(color: Colors.white)),
   backgroundColor: Colors.transparent,
   elevation: 0,
   actions: [
    IconButton(
     icon: const Icon(Icons.check, color: Colors.white),
     onPressed: () => Navigator.pop(context, _tempWidgets),
     tooltip: 'Зберегти',
    ),
   ],
  ),
  body: _isLoading
     ? const Center(child: CircularProgressIndicator())
     : ListView.builder(
   itemCount: _tempWidgets.length,
   itemBuilder: (ctx, index) {
    final w = _tempWidgets[index];
    return ListTile(
     leading: Icon(w.icon, color: w.color),
     title: Text(w.name, style: const TextStyle(color: Colors.white)),
     subtitle: Text(
      w.packageName.isNotEmpty ? w.packageName : 'Застосунок не вибрано',
      style: const TextStyle(color: Colors.white70),
     ),
     trailing: Row(
      mainAxisSize: MainAxisSize.min,
      children: [
       IconButton(
        icon: const Icon(Icons.edit, color: Colors.white),
        onPressed: () => _editWidget(index),
```

```
              ),
              IconButton(
                icon: const Icon(Icons.delete, color: Colors.white),
                onPressed: () => _deleteWidget(index),
              ),
            ],
          ),
        );
      },
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: _addWidget,
      child: const Icon(Icons.add),
    ),
  );
 }
}
```

Лістинг А.3 – Екран контактів (contacts_screen.dart)

```
import 'dart:convert';
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'package:permission_handler/permission_handler.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:url_launcher/url_launcher.dart';

class Contact {
  final String id;
  String name;
  String phone;
  final Color avatarColor;
  String? imagePath;

  Contact({
    required this.id,
    required this.name,
    required this.phone,
    required this.avatarColor,
    this.imagePath,
  });

  Map<String, dynamic> toJson() => {
    'id': id,
    'name': name,
    'phone': phone,
    'avatarColor': avatarColor.value,
    'imagePath': imagePath,
  };

  factory Contact.fromJson(Map<String, dynamic> json) {
```

```
    return Contact(
      id: json['id'],
      name: json['name'],
      phone: json['phone'],
      avatarColor: Color(json['avatarColor']),
      imagePath: json['imagePath'],
    );
  }
}

class ContactsScreen extends StatefulWidget {
  const ContactsScreen({super.key});

  @override
  State<ContactsScreen> createState() => _ContactsScreenState();
}

class _ContactsScreenState extends State<ContactsScreen> {
  List<Contact> _contacts = [];

  @override
  void initState() {
    super.initState();
    _loadContacts();
  }

  Future<void> _loadContacts() async {
    final prefs = await SharedPreferences.getInstance();
    final String? contactsJsonString = prefs.getString('contacts_list');
    if (contactsJsonString != null) {
      final List<dynamic> decodedList = jsonDecode(contactsJsonString);
      setState(() {
        _contacts = decodedList.map((json) => Contact.fromJson(json)).toList();
      });
    } else {
      setState(() {
        _contacts = [];
      });
      await _saveContacts();
    }
  }

  Future<void> _saveContacts() async {
    final prefs = await SharedPreferences.getInstance();
    final String encodedData = jsonEncode(_contacts.map((c) => c.toJson()).toList());
    await prefs.setString('contacts_list', encodedData);
  }

  void _openEditor() async {
    final result = await Navigator.push(
      context,
      MaterialPageRoute(builder: (ctx) => ContactsEditorScreen(contacts: _contacts)),
```

```
    );
    if (result != null && result is List<Contact>) {
      setState(() {
        _contacts = result;
        _saveContacts();
      });
    }
  }

  void _callContact(Contact contact) async {
    final Uri telUri = Uri(scheme: 'tel', path: contact.phone);
    if (await canLaunchUrl(telUri)) {
      await launchUrl(telUri);
    } else {
      if (!mounted) return;
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Не вдалося зателефонувати ${contact.name}', style: const
TextStyle(color: Colors.white))),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.transparent,
      body: SafeArea(
        child: Column(
          children: [
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 16),
              child: Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                  const Text('Контакти', style: TextStyle(fontSize: 32, fontWeight: FontWeight.bold,
color: Colors.white)),
                  IconButton(
                    icon: const Icon(Icons.edit, color: Colors.white, size: 32),
                    onPressed: _openEditor,
                  ),
                ],
              ),
            ),
            Expanded(
              child: _contacts.isEmpty
                  ? const Center(child: Text('Немає контактів. Натисніть ✏, щоб додати', style:
TextStyle(color: Colors.white)))
                  : ListView.builder(
                itemCount: _contacts.length,
                itemBuilder: (context, index) {
                  final contact = _contacts[index];
                  return Card(
```

```
                margin: const EdgeInsets.symmetric(horizontal: 20, vertical: 6),
                color: Colors.white.withOpacity(0.1),
                shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(20)),
                child: ListTile(
                  leading: CircleAvatar(
                    backgroundColor: contact.avatarColor,
                    backgroundImage: contact.imagePath != null ?
FileImage(File(contact.imagePath!)) : null,
                    child: contact.imagePath == null
                        ? Text(contact.name.substring(0, 1).toUpperCase(), style: const
TextStyle(fontWeight: FontWeight.bold))
                        : null,
                  ),
                  title: Text(contact.name, style: const TextStyle(color: Colors.white)),
                  trailing: IconButton(
                    icon: const Icon(Icons.call, color: Colors.greenAccent),
                    onPressed: () => _callContact(contact),
                  ),
                  onTap: () => _callContact(contact),
                ),
              );
            },
          ),
        ),
      ],
    ),
  ),
 );
 }
}

class ContactsEditorScreen extends StatefulWidget {
 final List<Contact> contacts;
 const ContactsEditorScreen({super.key, required this.contacts});

 @override
 State<ContactsEditorScreen> createState() => _ContactsEditorScreenState();
}

class _ContactsEditorScreenState extends State<ContactsEditorScreen> {
 late List<Contact> _tempContacts;
 final ImagePicker _picker = ImagePicker();

 final List<Color> _availableColors = [
  Colors.tealAccent,
  Colors.cyanAccent,
  Colors.lightBlueAccent,
  Colors.pinkAccent,
  Colors.orangeAccent,
  Colors.limeAccent,
  Colors.amberAccent,
 ];
```

```
 @override
 void initState() {
  super.initState();
  _tempContacts = List.from(widget.contacts);
 }

 Future<bool> _requestStoragePermission() async {
  if (await Permission.photos.request().isGranted) {
   return true;
  } else {
   if (!mounted) return false;
   ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(content: Text('Потрібен дозвіл для вибору фото з галереї', style:
TextStyle(color: Colors.white))),
   );
   return false;
  }
 }

 Future<void> _pickImage(Contact contact) async {
  if (!await _requestStoragePermission()) return;
  final XFile? pickedFile = await _picker.pickImage(source: ImageSource.gallery);
  if (pickedFile != null) {
   setState(() {
    contact.imagePath = pickedFile.path;
   });
  }
 }

 void _addOrEditContact({Contact? existing}) {
  final isEditing = existing != null;
  final nameController = TextEditingController(text: existing?.name ?? '');
  final phoneController = TextEditingController(text: existing?.phone ?? '');
  Color selectedColor = existing?.avatarColor ?? Colors.tealAccent;

  showDialog(
   context: context,
   builder: (ctx) {
    return StatefulBuilder(
     builder: (ctx, setStateDialog) {
      return AlertDialog(
       title: Text(isEditing ? 'Редагувати контакт' : 'Новий контакт', style: const
TextStyle(color: Colors.black87)),
       content: Column(
        mainAxisSize: MainAxisSize.min,
        children: [
         TextField(
          controller: nameController,
          decoration: const InputDecoration(labelText: 'Ім\'я', labelStyle: TextStyle(color:
Colors.black87)),
          style: const TextStyle(color: Colors.black87),
```

```
            ),
            const SizedBox(height: 12),
            TextField(
              controller: phoneController,
              decoration: const InputDecoration(labelText: 'Номер телефону', labelStyle:
TextStyle(color: Colors.black87)),
              keyboardType: TextInputType.phone,
              style: const TextStyle(color: Colors.black87),
            ),
            const SizedBox(height: 12),
            Row(
              children: [
                const Text('Колір аватара:', style: TextStyle(color: Colors.black87)),
                const SizedBox(width: 8),
                Expanded(
                  child: DropdownButton<Color>(
                    value: selectedColor,
                    items: _availableColors.map((color) {
                      return DropdownMenuItem(value: color, child: Container(width: 40, height: 40,
color: color));
                    }).toList(),
                    onChanged: (color) => setStateDialog(() => selectedColor = color!),
                  ),
                ),
              ],
            ),
          ],
        ),
        actions: [
          TextButton(onPressed: () => Navigator.pop(ctx), child: const Text('Скасувати', style:
TextStyle(color: Colors.black87))),
          ElevatedButton(
            onPressed: () {
              if (nameController.text.trim().isEmpty || phoneController.text.trim().isEmpty) return;
              final newContact = Contact(
                id: isEditing ? existing!.id : DateTime.now().millisecondsSinceEpoch.toString(),
                name: nameController.text.trim(),
                phone: phoneController.text.trim(),
                avatarColor: selectedColor,
                imagePath: isEditing ? existing?.imagePath : null,
              );
              setState(() {
                if (isEditing) {
                  final index = _tempContacts.indexWhere((c) => c.id == existing!.id);
                  _tempContacts[index] = newContact;
                } else {
                  _tempContacts.add(newContact);
                }
              });
              Navigator.pop(ctx);
            },
            child: const Text('Зберегти', style: TextStyle(color: Colors.white)),
```

```
            ),
          ],
        );
      },
    );
  },
 );
}

void _deleteContact(Contact contact) {
 setState(() {
  _tempContacts.removeWhere((c) => c.id == contact.id);
 });
}

@override
Widget build(BuildContext context) {
 return Scaffold(
  backgroundColor: Colors.transparent,
  appBar: AppBar(
   title: const Text('Редактор контактів', style: TextStyle(color: Colors.white)),
   backgroundColor: Colors.transparent,
   elevation: 0,
   actions: [
    IconButton(
     icon: const Icon(Icons.check, color: Colors.white),
     onPressed: () => Navigator.pop(context, _tempContacts),
    ),
   ],
  ),
  body: ListView.builder(
   itemCount: _tempContacts.length,
   itemBuilder: (ctx, index) {
    final contact = _tempContacts[index];
    return ListTile(
     leading: CircleAvatar(
      backgroundColor: contact.avatarColor,
      backgroundImage: contact.imagePath != null ? FileImage(File(contact.imagePath!)) : null,
      child: contact.imagePath == null
        ? Text(contact.name.substring(0, 1).toUpperCase(), style: const TextStyle(fontWeight:
FontWeight.bold))
        : null,
     ),
     title: Text(contact.name, style: const TextStyle(color: Colors.white)),
     subtitle: Text(contact.phone, style: const TextStyle(color: Colors.white70)),
     trailing: Row(
      mainAxisSize: MainAxisSize.min,
      children: [
       IconButton(icon: const Icon(Icons.photo, color: Colors.white), onPressed: () =>
_pickImage(contact)),
       IconButton(icon: const Icon(Icons.edit, color: Colors.white), onPressed: () =>
_addOrEditContact(existing: contact)),
```

```
            IconButton(icon: const Icon(Icons.delete, color: Colors.white), onPressed: () =>
_deleteContact(contact)),
          ],
        ),
      );
    },
  ),
  floatingActionButton: FloatingActionButton(
    onPressed: () => _addOrEditContact(),
    child: const Icon(Icons.add),
  ),
 );
 }
}
```

Лістинг А.4 – Екран годинника (clock_screen.dart)

```
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import 'package:flutter_tts/flutter_tts.dart';

class ClockScreen extends StatefulWidget {
 const ClockScreen({super.key});

 @override
 State<ClockScreen> createState() => _ClockScreenState();
}

class _ClockScreenState extends State<ClockScreen> {
 late Timer _timer;
 late DateTime _currentTime;
 late FlutterTts _flutterTts;

 @override
 void initState() {
  super.initState();
  _currentTime = DateTime.now();
  _timer = Timer.periodic(const Duration(seconds: 1), (timer) {
   setState(() => _currentTime = DateTime.now());
  });
  _flutterTts = FlutterTts();
 }

 @override
 void dispose() {
  _timer.cancel();
  _flutterTts.stop();
  super.dispose();
 }

 Future<void> _speakTime() async {
```

```
  try {
    await _flutterTts.setLanguage("uk-UA");
  } catch (e) {
    await _flutterTts.setLanguage("ru-RU");
  }
  await _flutterTts.setSpeechRate(0.5);
  await _flutterTts.setPitch(1.0);
  String timeText = 'Зараз ${DateFormat('HH:mm').format(_currentTime)}';
  await _flutterTts.speak(timeText);
 }

 @override
 Widget build(BuildContext context) {

  final screenWidth = MediaQuery.of(context).size.width;
  final buttonSize = (screenWidth * 0.9).clamp(200.0, 350.0);

  return Scaffold(
    backgroundColor: Colors.transparent,
    body: SafeArea(
      child: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              DateFormat('HH:mm').format(_currentTime),
              style: const TextStyle(fontSize: 96, fontWeight: FontWeight.bold, letterSpacing: 8, color:
Colors.white),
            ),
            const SizedBox(height: 20),
            Text(
              DateFormat('EEEE, d MMMM', 'uk').format(_currentTime),
              style: const TextStyle(fontSize: 24, color: Colors.white70),
            ),
            const SizedBox(height: 60),

            SizedBox(
              width: buttonSize,
              height: buttonSize,
              child: ElevatedButton(
                onPressed: _speakTime,
                style: ElevatedButton.styleFrom(
                  backgroundColor: Colors.tealAccent.shade700,
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(24),
                  ),
                  padding: EdgeInsets.zero,
                ),
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    const Icon(Icons.volume_up, size: 48),
```

```
                const SizedBox(height: 16),
                const Text(
                  'ОЗВУЧИТИ ЧАС',
                  style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold, color: Colors.black87),
                  textAlign: TextAlign.center,
                ),
              ],
            ),
          ),
        ),
      ],
    ),
  ),
),
);
  }
}
```