УДК 004.42:004.934

С.О. Лисюк
Д.І. Катєльніков

РОЗРОБКА KOTLIN-ЗАСТОСУНКУ ДЛЯ ГОЛОСОВОЇ АВТОМАТИЗАЦІЇ ДІЙ КОРИСТУВАЧА НА ANDROID

Вінницький національний технічний університет

Анотація

У статті поглиблено досліджуються технічні аспекти розробки мобільного застосунку для автоматизації повсякденних дій користувача на платформі Android з використанням елементів голосового керування. Робота зосереджена на описі масштабованої архітектури застосунку, побудованої за шаблоном MVVM. Представлено гнучку реляційну модель зберігання багатоетапних сценаріїв за допомогою локальної бази даних SQLite та бібліотеки Room. Детально розглядаються механізми забезпечення фонового виконання команд засобами операційної системи Android та Kotlin Coroutines, а також алгоритми обробки голосових команд із використанням системного сервісу Android SpeechRecognizer, що базується на технології Google Speech-to-Text. Додатково продемонстровано реалізацію інтуїтивного графічного конструктора, який дозволяє користувачам без навичок програмування поєднувати системні дії та дії в межах застосунку. Зроблено висновки щодо ефективності обраних програмних рішень для забезпечення стабільності, швидкодії та автономності роботи мобільного помічника.

Ключові слова: автоматизація дій; голосове керування; мобільні застосунки.

Abstract

The article deeply explores the technical aspects of developing a mobile application to automate everyday user actions on the Android platform using voice control elements. The work focuses on describing a scalable application architecture built using the MVVM pattern. A flexible relational model for storing multi-stage scenarios using a local SQLite database and the Room library is presented. The mechanisms for ensuring background execution of commands using the Android operating system and Kotlin Coroutines are considered in detail, as well as voice command processing algorithms using the Android SpeechRecognizer system service, which is based on Google Speech-to-Text technology. Additionally, the implementation of an intuitive graphical designer is demonstrated, which allows users without programming skills to combine system actions and actions within the application. Conclusions are drawn regarding the effectiveness of the selected software solutions to ensure stability, speed, and autonomy of the mobile assistant.

Keywords: Android; MVVM; Room; action automation; voice control; Kotlin; Android SpeechRecognizer; mobile applications.

Гнучка модель сценаріїв та масштабована архітектура застосунку

В основу розробленого мобільного застосунку покладено модульний підхід, який забезпечує високу масштабованість системи та можливість гнучкого налаштування користувацьких команд. У додатку запропоновано підхід, що полягає у відході від жорстко закодованих функцій на користь динамічного формування сценаріїв самим користувачем.

Гнучка модель сценарію реалізована через концепцію розбиття будь-якої складної задачі на набір базових, ізольованих кроків. Замість монолітної команди, сценарій у застосунку являє собою впорядкований ланцюжок операцій. Наприклад, сценарій «Підготовка до відпочинку» може складатися з таких послідовних кроків:

- Увімкнення Youtube;
- Пауза на 2 секунди для забезпечення запуску додатка;
- Запуск дії повороту екрана;
- Зміна гучності на 80%.

Така модель є гнучкою, оскільки дозволяє користувачу конструювати безліч унікальних комбінацій з обмеженого базового набору доступних дій, змінювати їх порядок та налаштовувати індивідуальні параметри такі як тривалість пауз, зміна додатка, якого потрібно відкрити, тощо. Без модифікації вихідного коду програми.

Масштабована архітектура застосунку концептуально розділена на два ключові функціональні блоки: «Бібліотеку дій» (Action Library) та «Двигун виконання» (Execution Engine).

- Action Library містить набір незалежних класів-обробників, кожен з яких відповідає за виконання однієї конкретної системної функції. Наприклад зміна налаштувань екрана, запуск стороннього пакета чи виклик системного API.
- Execution Engine виконує роль координатора. Він зчитує збережений ланцюжок кроків з бази даних і послідовно ініціює їх виконання, обробляючи можливі помилки та забезпечуючи переходи між кроками.

Масштабованість такого рішення полягає в тому, що якщо в майбутньому буде зроблеено додавання нових дій, то розробнику достатньо буде додати тільки новий модуль до Action Library. При цьому двигун системи Execution Engine не буде змінено, оскільки він працює з діями через єдиний абстрактний інтерфейс. Візуальне представлення базових компонентів системи та їхньої взаємодії наведено на рисунку 1.

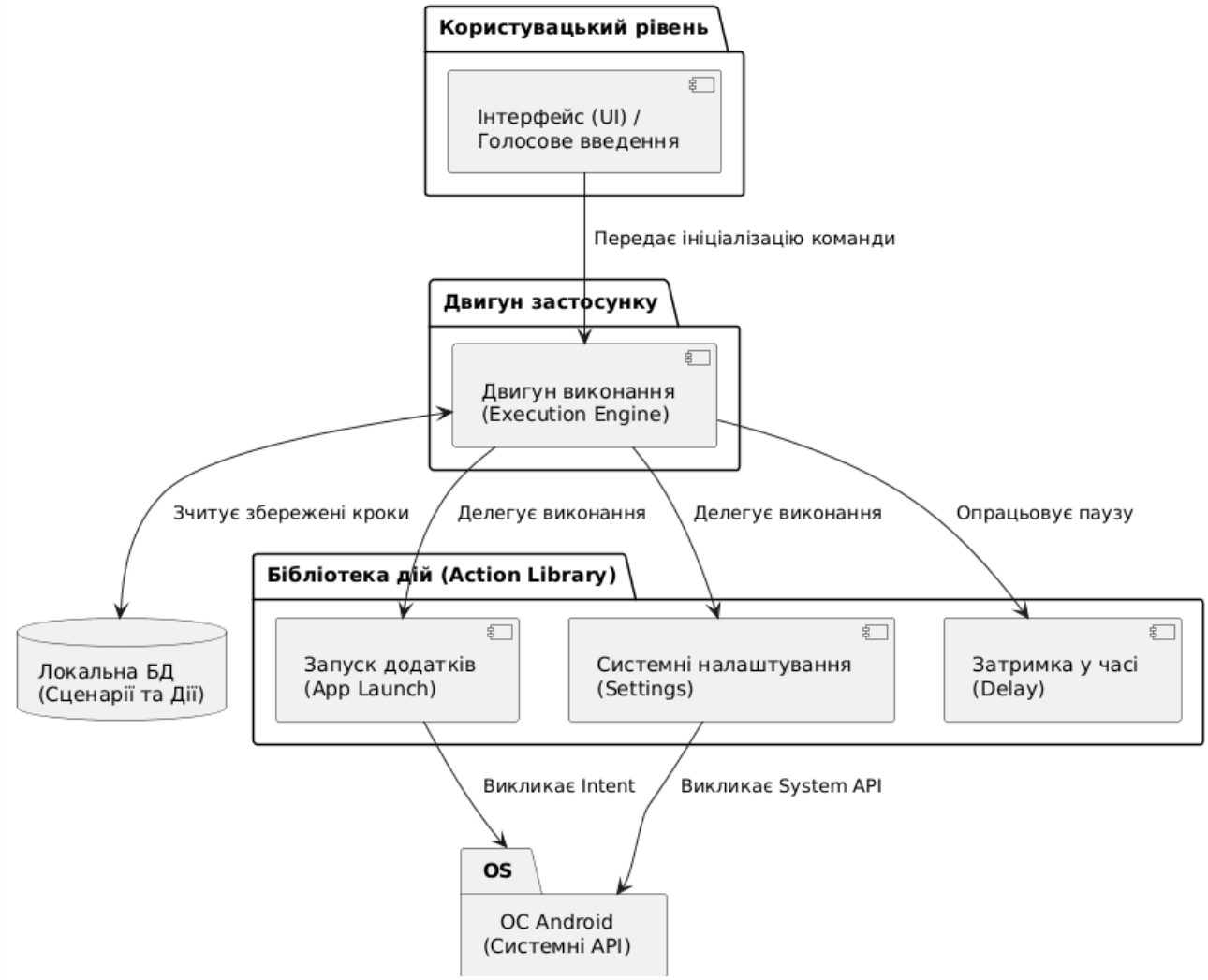

Рисунок 1 – Узагальнена структурна схема архітектури застосунка

Обґрунтування вибору архітектурного шаблону MVVM

Для побудови надійного та легко підтримуваного Android-застосунку критично важливим є вибір правильного архітектурного патерну. У процесі проектування було проведено порівняльний аналіз поширених шаблонів MVC (Model-View-Controller), MVP (Model-View-Presenter) та MVVM (Model-View-ViewModel).

Вибір саме MVVM зумовлений специфікою мобільної розробки та недоліками альтернативних підходів:

- MVC[1] (Model-View-Controller): У класичному MVC контролер часто тісно зв'язується з життєвим циклом Activity або Fragment. Це призводить до надмірного розростання коду в одному класі та значно ускладнює модульне тестування.

- MVP[2] (Model-View-Presenter): Цей шаблон частково вирішує проблеми MVC, повністю ізолюючи логіку в компоненті Presenter. Проте Presenter зберігає пряме посилання на View. В умовах ОС Android це часто призводить до витоків пам'яті при зміні конфігурації. Або викликає непередбачені збої NullPointerException, якщо фоновий процес намагається оновити вже знищений системою інтерфейс.
- MVVM[3] (Model-View-ViewModel): Архітектура MVVM повністю усуває проблему прямої залежності між бізнес-логікою та інтерфейсом. Для забезпечення гнучкості та масштабованості застосунку було обрано архітектуру MVVM, що чітко розділяє логіку, роботу з даними та інтерфейс. Компонент ViewModel не містить жодних посилань на View. Натомість він надає потоки даних, за якими інтерфейс пасивно «спостерігає».

Для застосунку голосової автоматизації вибір MVVM є найбільш оптимальним з огляду на такі фактори:

1. Стійкість до зміни життєвого циклу: Клас ViewModel автоматично зберігає свій стан під час змін конфігурації. Це критично важливо, оскільки процес розпізнавання голосу або виконання тривалого багатокрокового сценарію не повинен переривatися.
2. Асинхронність операцій: Робота з локальною базою даних та системним сервісом розпізнавання мовлення відбувається у фонових потоках. MVVM дозволяє безпечно передавати результати цих асинхронних операцій в UI.
3. Чітке розділення відповідальності: Шар Model відповідає за отримання та збереження сценаріїв, ViewModel – за логіку підготовки команд до виконання, а View – виключно за відображення списку команд та реєстрацію взаємодій користувача.

Схему взаємодії компонентів архітектури MVVM для розробленого застосунку наведено на рисунку 2.

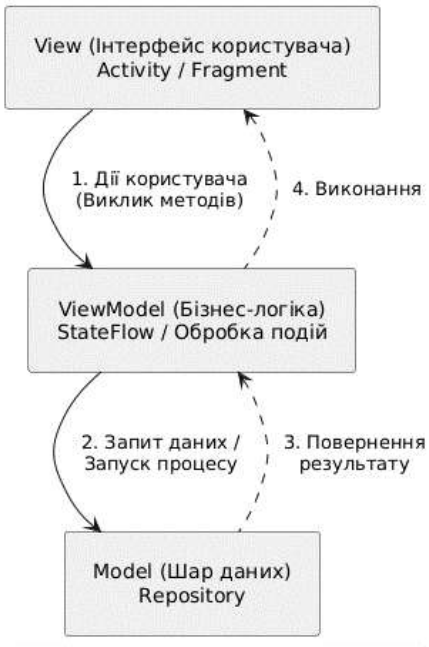

Рисунок 2 – Взаємодія компонентів за шаблоном MVVM

Організація локального зберігання даних за допомогою Room та побудова ER-моделі

Сценарії користувача, які складаються з одного чи кількох послідовних кроків, зберігаються на основі локальної бази даних SQLite через бібліотеку Room. Технологія Room виступає як надійний рівень абстракції над класичною базою даних SQLite. Вона забезпечує автоматичну перевірку SQL-запитів на етапі компіляції програми, суттєво зменшує кількість шаблонного коду та нативно підтримує асинхронні потоки даних, що робить її оптимальним рішенням для сучасних мобільних застосунків.

Реляційна структура бази даних побудована за правилом «один-до-багатьох» і складається з двох основних таблиць. Перша таблиця «commands» зберігає основну інформацію про команду. Вона містить унікальний ідентифікатор з автогенерацією, назву команди, фразу активації та тип команди,

що визначає, чи є вона одиничною системною дією, чи послідовністю дій. Друга таблиця «action_steps» містить деталі кожного кроку всередині сценарію. Кожен запис у цій таблиці посилається на основну команду через зовнішній ключ. Також таблиця включає унікальний ідентифікатор кроку, порядковий номер дії в списку, тип дії, універсальне текстове поле для збереження пакета програми або тривалості паузи, та координати для виконання імітації натискань на екрані.

Збереження даних реалізовано за принципом атомної транзакції через компонент CommandRepository. Процес запису відбувається у кілька етапів. Спочатку дані з об'єктів Kotlin конвертуються у сутності бази даних CommandEntity та ActionStepEntity. Далі застосовується операції Update + Insert. Завдяки стратегії OnConflictStrategy.REPLACE, система автоматично оновлює рядок, якщо ідентифікатор вже існує, або створює новий запис. Перед збереженням оновленого сценарію застарілі кроки з таблиці action_steps видаляються, що дозволяє уникнути дублювання та накладання дій. Крім того, використання компонента Flow на рівні об'єктів доступу до даних дозволяє автоматично та реактивно оновлювати графічний інтерфейс застосунку відразу після зміни даних у базі. Структуру бази даних представлено у вигляді ER-моделі, що зображена на рисунку 3

Рисунок 3 – ER-модель бази даних застосунку «VoiceCut»

Технічна реалізація фонового виконання сценаріїв засобами Android та Kotlin

Забезпечення стабільного виконання сценаріїв у фоновому режимі, коли основний інтерфейс застосунку не є активним або пристрій заблоковано, реалізовано за допомогою поєднання системних механізмів Android та мовних можливостей Kotlin. Ключовим інструментом для цього обрано службу переднього плану. На відміну від звичайних фонових процесів, така служба має вищий пріоритет у системі, що запобігає її примусовому завершенню операційною системою з метою оптимізації енергоспоживання. Це гарантує, що процес розпізнавання голосу та наступне виконання кроків автоматизації не будуть перервані.

Безпосередня логіка обробки сценаріїв покладена на компонент WorkflowExecutor, який працює асинхронно завдяки механізму корутин «Kotlin Coroutines». Використання корутин дозволяє ефективно керувати кроками сценарію, що потребують часових затримок. Наприклад дії типу DELAY. Замість блокування потоку виконання, що призводило б до надмірного споживання ресурсів, застосовується неблокуюче призупинення, яке дозволяє системі гнучко розподіляти ресурси процесора.

Така комбінація технологій забезпечує високу автономність застосунку. Після активації голосової команди через системний API, WorkflowExecutor у межах фонового сервісу послідовно ініціює кожну дію сценарію. Це дозволяє реалізувати безперервний цикл автоматизації, від розпізнавання мовлення до виконання складних жестів на екрані, зберігаючи при цьому стабільність роботи всієї операційної системи.

Проектування модуля обробки голосових команд на базі Android SpeechRecognizer

Для реалізації майбутнього голосового керування у застосунку обрано системний компонент Android SpeechRecognizer[4], який працює на основі технологій Google Speech-to-Text. Вибір цього інструменту зумовлений його інтеграцією в операційну систему, що дозволяє використовувати готові алгоритми обробки мовлення без значного навантаження на ресурси мобільного пристрою.

Згідно з архітектурою застосунку, процес взаємодії з голосом базується на використанні системного запиту «ACTION_RECOGNIZE_SPEECH». Ця команда ініціює запуск стандартного інтерфейсу розпізнавання, який перетворює аудіосигнал з мікрофона у текстовий формат. Запланована логіка обробки передбачає отримання списку розпізнаних варіантів фраз, з яких програма обирає найбільш точний. Отриманий текст автоматично приводиться до нижнього регістру та порівнюється із ключовою фразою «voiceTrigger», що зберігається у базі даних сценаріїв. При виявленні ідентичності знайдена

команда передається до виконавчого ядра WorkflowExecutor для автоматичного запуску відповідної дії.

Реалізація та перспективні напрями розвитку графічного інтерфейсу

Важливою складовою розробленого застосунку є графічний інтерфейс, спроектований за принципом мінімалізму та інтуїтивності. Основна мета конструктора – надати користувачеві можливість створювати складні сценарії автоматизації без необхідності володіння навичками програмування або вивчення складних системних налаштувань.

Екран керування командами, який зображено на рисунку 4, відображає список уже створених команд у вигляді карток, де вказано назву команди та кількість інтегрованих у неї дій. Кожна картка містить елементи швидкого запуску команди та переходу до його редагування.

Рисунок 4 – Екран перегляду створених команд

Процес створення нової команди, який зображено на рисунку 5, реалізовано через покрокове додавання блоків. Користувач може вказати назву сценарію та обрати типи дій із представленого переліку.

Рисунок 5 – Інтерфейс конструктора команди

Висновки

Поточна реалізація інтерфейсу поєднує системні можливості Android та внутрішню логіку застосунку в єдиному візуальному просторі, забезпечуючи чіткий поділ між етапом налаштування та збереження команди. Оскільки представлена версія є робочим прототипом, графічна інтерфейс додатку VoiceCut перебуває на етапі активного тестування та подальшого візуального доопрацювання. Перспективним напрямом розвитку є оновлення стилістики інтерфейсу, що включає перегляд палітри кольорів та зміну графічного вигляду блоків для кращої читабельності. Подальша розробка також передбачає вдосконалення самого конструктора сценаріїв. Оптимізацію розташування елементів керування та розширення переліку доступних дій, що дозволить користувачеві гнучко налаштовувати команди під власні потреби. Такий підхід забезпечить поступову еволюцію інтерфейсу від функціонального макета до повноцінного, візуально завершеного продукту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Модель–вид–контролер [Електронний ресурс] : матеріал з Вікіпедії – вільної енциклопедії. – Електронні дані. – Режим доступу: https://surl.li/aynufe (дата звернення: 04.03.2026). – Назва з екрана.
2. Model–View–Presenter [Електронний ресурс] : матеріал з Вікіпедії – вільної енциклопедії. – Електронні дані. – Режим доступу: https://uk.wikipedia.org/wiki/Model-View-Presenter (дата звернення: 04.03.2026). – Назва з екрана.
3. Android Architecture Design Patterns / Real Tutorials Team, N. S. Kumar, M. Morey [et al.]. – Alexandria : Razeware LLC, 2021. – 434 p.
4. SpeechRecognizer [Electronic resource] / Android Developers. – Electronic data. – 2026. – Mode of access: https://developer.android.com/reference/android/speech/SpeechRecognizer (date of access: 04.03.2026). – Title from the screen.

Лисюк Святослав Олегович - студент групи 2ПІ-22б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, e-mail: m1natoofficial24@gmail.com.

Катєльніков Денис Іванович. – к.т.н., доцент, доцент кафедри ПЗ, Вінницький національний технічний університет, м. Вінниця, e-mail: katielnikov@vntu.edu.ua.

Lysiuk Sviatoslav Olehovych - student of the 2PI-22b group, Faculty of Information Technology and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: m1natoofficial24@gmail.com.

Katielnikov Denis I. – Ph.D., Associate Professor, Department of Software Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: katielnikov@vntu.edu.ua.