Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Веб-застосунок на Flutter з підтримкою персоналізованих нагадувань

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-4
Андрій БЕРНАДСЬКИЙ
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Олег ЖУРИЛО
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Бернадському Андрію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Веб-застосунок на Flutter з підтримкою персоналізованих нагадувань

затверджена наказом по університету від "26" травня 2025 р. № 424Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи 1. Завдання на кваліфікаційну роботу
2. Перелік вимог до проектованого застосунку
3. Література та документація по темі
4. Методичні вказівки з виконання кваліфікаційної роботи

4. Перелік питань, що потрібно опрацювати у роботі 1. Постановка задачі
2. Огляд наявних рішень
3. Порівняння існуючих технологій розробки
4. Вибір технології розробки
5. Реалізація програмного застосунку;
6. Тестування застосунку
7. Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій Слайд-презентація – 15 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання	26.05.2025	
2	Аналіз існуючих методів вирішення задачі	01.06.2025	
3	Реалізація поставленої задачі	03.06.2025	
4	Аналіз отриманих результатів	09.06.2025	
5	Оформлення пояснювальної записки	11.06.2025	

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Олег ЖУРИЛО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 82 с., 34 рис., 1 табл., 2 дод., 8 джерел.

ВЕБ-ЗАСТОСУНОК, МОБІЛЬНИЙ ДОДАТОК, РОЗРОБКА, ANDROID, FLUTTER, НАГАДУВАННЯ.

Метою кваліфікаційної роботи є розробка веб-застосунку та мобільного додатку для створення персоналізованих нагадувань, надання користувачеві можливість легкого створення, а також редагування нагадувань, які будуть спрацьовувати у заданий час і допомагатимуть не втрачати контроль над справами.

У ході виконання кваліфікаційної роботи було розроблено веб-застосунок з нагадувань для користувача та мобільний застосунок для пристроїв з операційною системою Android з використання фреймворку Flutter, що дозволив реалізувати архітектуру, інтерфейс користувача та інтеграцію з хмарними сервісами. Flutter дозволив зручно реалізувати багато елементів інтерфейсу користувача – таких як кнопки, анімовані переходи, діалоги, інтерактивні форми. Застосунок підтримує особисті нагадування, які користувач може додати, редагувати чи видалити. Було реалізовано щоденні повторювані сповіщення, можливість вмикання та вимикання нагадувань, а також підтримка Firebase аби зберігати дані користувачів.

ABSTRACT

Bachelor’s thesis: 82 pages, 34 figures, 1 tables, 2 appendices, 8 sources.

WEB APPLICATION, MOBILE APP, DEVELOPMENT, ANDROID, FLUTTER, REMINDERS.

The major goal of this thesis is develop web application with reminders for the user and a mobile application for creating personalized reminders, providing users with the ability to easily create and edit notifications that will trigger at a specified time and help them stay on top of their tasks,

During the development process, an application was created for devices running the Android operating system using the Flutter framework. This allowed for the implementation of the app's architecture, user interface, and integration with cloud services. Flutter made it convenient to implement various user interface elements -such as buttons, animated transitions, dialogs, and interactive forms.

The application supports personal reminders, which users can add, edit, or delete. Daily recurring notifications were implemented, along with the ability to enable or disable reminders. Firebase integration was used to securely store user data and manage authentication.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ОС – операційна система

ПЗ – програмне забезпечення

Фреймворк – (англ. Framework) – структура або платформа, що забезпечує розробку програмного забезпечення

AI – штучний інтелект (англ., Artificial Intelligence)

API – інтерфейс програмного застосунку (англ., Applecation Programming Interface)

NoSQL – клас нереляційних баз даних (англ., Not Only SQL)

SDK – набір для розробки програмного забезпечення (англ., Software Development Kit)

SMS – сервіс коротких повідомлень (англ., Short Message Service)

UI – інтерфейс користувача (англ., User Interface)

ВСТУП

В сучасні дні складно уявити своє життя без мобільних пристроїв, в наші дні вони вже перестали бути лише засобом зв’язку. Наразі смартфон – це справжній особистий помічник, який тримає нас на постійному зв’язку, допомагає орієнтуватись в навколишньому середовищі, зберігає важливу інформацію, надає доступ до соціальних мереж, новин, фінансових послуг та багато іншого. Завдяки швидкому розвитку мобільних технологій більшість повсякденних справ тепер можна виконати за допомогою телефону, навіть не виходячи з дому.

Однак водночас життя значно прискорилося. Ми легко не помічаємо або забуваємо речі, тому що часто кудись поспішаємо, одночасно виконуємо багато завдань, організовуємо зустрічі, працюємо, навчаємось і взаємодіємо з іншими людьми, і в цій нескінченній метушні дуже легко щось пропустити чи забути. Ми часто помічаємо, що через брак часу чи неуважність забуваємо про якісь важливі події, наприклад: чи то прийом ліків, день народження друга, чи взагалі звичайнісіньку побутову дрібницю. Саме в такі моменти особливо помітно, наскільки важливо вміти організовувати свій час, та мати під рукою надійний інструмент, який допоможе у цьому. За моїми спостереженнями найкращим та діючим способом уникнути забутих справ – це використовувати нагадування.

Зараз ми живемо в епоху, де практично кожен має телефон поруч з собою, тому логічно створити такий мобільний та веб-застосунок, який дозволить швидко та зручно створити нагадування про будь-яку подію. Цей застосунок повинен бути простим у використанні, інтуїтивно зрозумілим, аби задовільнити потреби користувача у повсякденному плануванні.

Тому було обрано темою завдання саме розробку мобільного та веб-застосунку для створення персоналізованих нагадувань. Основна ідея полягає в тому, щоб надати користувачеві можливість легко створювати, а також

редагувати нагадування, які будуть спрацьовувати у заданий час і допомагатимуть не втрачати контроль над своїми справами. Він буде корисним для кожного з нас.

1АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1Актуальність мобільних пристроїв

Ідеї щодо створення мобільного пристрою виникли у середині 20-го століття, такий апарат був величезним, вагою біля 40 кілограм і встановлювався лише в автомобілі. Революціонером в системі комунікації можна вважати Мартіна Купера, працівника компанії Моторола, саме він працював над тим, щоб створити компактний телефон, пізніше перший дзвінок з такого телефону був здійснений навесні 1973 року [1]. У 1980-1990-х роках мобільні телефони стали компактнішими та доступнішими. Подальший розвиток у 1990-х роках був пов’язаний з переходом на цифрові стандарти, що дозволило покращити якість зв’язку, додали нові функції, такі як SMS та передачу даних [2].

Восени 2008 року на широку публіку був представлений телефон, що започаткував еру Android, він став відомий як T1-Mobile G1, він мав мобільний доступ до Інтернету, камеру на 3.2 мегапікселі, а також Android Market [3] (рисунок 1.1).

Рисунок 1.1 – Перший Android смартфон

Опираючись на статистику, в кінці 2020 року 46,45% населення світу вже володіли смартфонами [4]. Станом на 2021 рік користувачі смартфонів використовують приблизно 6,4 мільярда підписок на смартфони, а до 2026 року ця цифра, як очікується, зросте до 7,5 мільярдів [5].

1.2 Огляд наявних рішень

Переглядаючи Play Маркет було помічено велику кількість різноманітних мобільних аплікацій, що дозволяють створювати нагадування. Рішення є як для Android так і для iOS, кожен з них має свій інтерфейс, певний функціонал, а також підхід до організації справ користувача. Використовуючи такі застосунки на повсякденній основі, я помічав, що багато з них мають певні недоліки, деякі не є суттєвими, а деякі прямо таки роздратовують.

Наприклад є застосунок BZ Reminder, який я особисто використовував на старому смартфоні. Цей застосунок простий у використанні (рисунок 1.2), але має одну суттєву проблему, яку я помітив після того, як змінив модель телефону, у нього немає зберігання у хмарі.

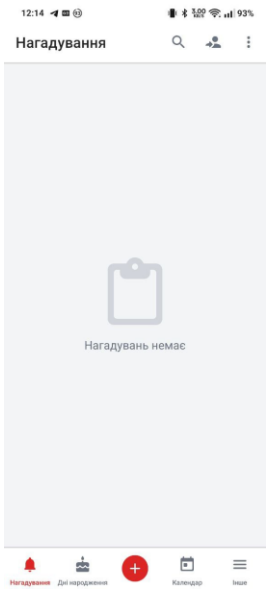


Рисунок 1.2 – Головний екран BZ Reminder

Тобто, якщо видалити застосунок чи просто змінити телефон – всі нагадування будуть втрачені, тому таке рішення не є чудовим, адже виходить, що користувач не має жодної страховки щодо створених нагадувань.

Тепер ще до прикладу візьмено застосунок Alarm and pill

reminder(нагадування про прийом ліків) (рисунок 1.3). В ньому великий функціонал, інтерфейс складний, а особливо дратує, що постійно «вискакує» реклама, іноді навіть перед тим, як має з’явитися нагадування. Також застосунок має постійну рекламу та баги: нагадування можуть не приходити чи приходити з суттєвим запізненням.

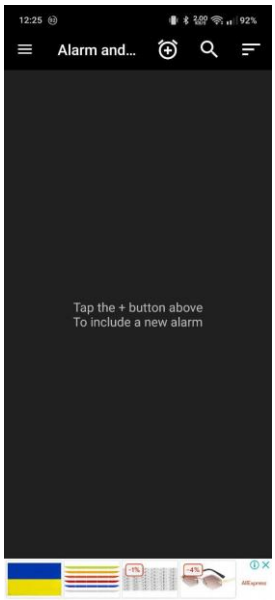


Рисунок 1.3 – Головний екран Alarm and pill reminder

Також є багато додатків, що перенавантажують інтерфейс – забагато кнопок, не потрібних налаштувань чи взагалі не зрозумілий інтерфейс, де важко додати просте нагадування. Усі ці приклади показують, що хоч застосунків багато, але не всі з них є зручними та несуть користь користувачеві.

З веб-застосунків слід виділити один з найвідоміших вебсервісів – Google Calendar, який дозволяє створювати події, зустрічі, завдання та нагадування. Перевагами такого сервісу є хмарна синхронізація, підтримка сповіщень через браузер та можливість спільного доступу до календарів.

Недоліками є надмінна складність інтерфейсу для користувача, якому потрібне лише просте нагадування.

1.3 Постановка задачі

Як зазначалось раніше, метою є створення одночасно простого та інтуїтивно зрозумілого веб-застосунку для нотифікації, а також розробити мобільний застосунок, головною задачею якого буде допомога в організації власного часу за допомогою системи нагадувань. Ідея досить проста, якщо надати людині можливість у будь-який момент створити нагадування про якусь важливу для нього подію, справу чи просто дрібницю, то вона навряд чи її пропустить.

Увагу слід приділити й збереженню введеної інформації. Потрібно зробити так, щоб коли застосунок було видалено (випадково чи через зміну смартфону), після повторного встановлення усі дані були збережені. Для цього повинно бути реалізовано збереження нагадувань у хмарному сховищі. Крім того, потрібно реалізувати можливість створення не тільки одноразових нагадувань, а й тих, що можуть повторюватись. Це потрібно у тих випадках, коли треба нагадувати про регулярні події, такі як-от тренування, полив квітів і так далі. Такий підхід спростить процес планування і зробить програмний продукт універсальним.

В основному потрібно приділити увагу зручності. Інтерфейс користувача повинен бути максимально зрозумілим, щоб навіть дитина, чи людина похилого віку, яка не тримала в руках смартфон, розібралась, як додати нове нагадування, редагувати його чи видалити. Все повинно бути чітко й зрозуміло. Ще однією вимогою при створенні є стабільність роботи. Програма повинна надійно працювати незалежно від того, як давно вона була встановлена, чи яка кількість нагадувань уже збережена. Користувач повинен бути впевненим, що в аплікації не виникне якийсь збій. Також в застосунку не повинно бути нав’язливої реклами, яка відверто дратує.

У підсумку, застосунок повинен вирішити ті проблеми, з якими стикаються юзери у вже наявних подібних рішеннях, а також стати легким та зручним інструментом для планування.

2 АНАЛІЗ ТА ВИБІР ТЕХНОЛОГІЙ І МЕТОДІВ РОЗРОБКИ

2.1 Вибір мови програмування

Dart – це клієнтоорієнтована мова для розробки швидких додатків на будь-якій платформі, її мета – запропонувати найпродуктивнішу мову програмування для багатоплатформної розробки в поєднанні з гнучкою платформою виконання для фреймворків додатків [6].

Мова Dart є сучасною об’єктно-орієнтованою мовою, яку створили в Google з акцентом на простоту, зручність і продуктивність. ЇЇ розробили, щоб задовольнити потребу в єдиній мові для кросплатформної розробки, щоб той самий код міг працювати на різних пристроях і операційних системах. Dart поєднує в собі чіткий синтаксис, подібний до С-подібних мов, із сильними можливостями набору коду, використовуючи класи, абстракції, міксини та дженериків.

Dart було вперше представлено на конференції GOTO в Данії в 2011 році відомими інженерами Ларсом Баком і Каспером Лундом (рисунок 2.1). Мова пройшла довгий шлях з моменту свого офіційного випуску 1.0 у 2013 році – від суперечливих початків (критикованих за спробу замінити JavaScript у браузерах) до того, як вона стала основною технологією для створення сучасних мобільних, веб-програм і десктопних аплікацій.

Рисунок 2.1 – Конференція GOTO в 2011 році

Ключовий поворотний момент стався в 2015 році, коли команда Dart вирішила не інтегрувати свою власну віртуальну машину в Chrome і натомість зосередитися на компіляції Dart у JavaScript. Це рішення робить мову сумісною з усіма популярними браузерами без їх модифікації. У 2018 році був випущений Dart 2.0, який містив оновлену систему типів і новий підхід до структури коду.

Особливо важливим етапом став випуск Dart 2.6, який представив інструмент «dart2native». З його впровадженням стала можливість створювати автономні виконувані файли для десктопних платформ (Windows, macOS, Linux), таке рішення призвело до розширення сфери використання Dart за межами мобільної розробки. Тепер Dart не обмежується лише мобільними програмами через Flutter – його також можна використовувати для написання програм на стороні сервера та розробки консольних інструментів.

У Dart є кілька способів виконання коду:

- для веб-розробки застосовується Dart, який компілюється в JavaScript (через компілятор dart2js), що дозволяє запускати вашу програму в будь-якому сучасному браузері без потреби у віртуальній машині Dart;

- в автономному режимі Dart працює через власну віртуальну машину, яка є частиною SDK, що дозволяє створювати повноцінні консольні програми або сервери;

- у режимі Ahead-of-Time (AOT) Dart негайно компілюється до рідного машинного коду, покращуючи продуктивність програми;

- за допомогою Dart 2.6 можна створювати окремі файли без встановлення Dart SDK на стороні користувача.

Ще один важливий аспект – стандартизація мови. Ecma International створила технічний комітет TC52 для розробки та підтримки офіційної специфікації Dart. Це забезпечує стабільність і відкритість мови для розробників з усього світу (рисунок 2.2).

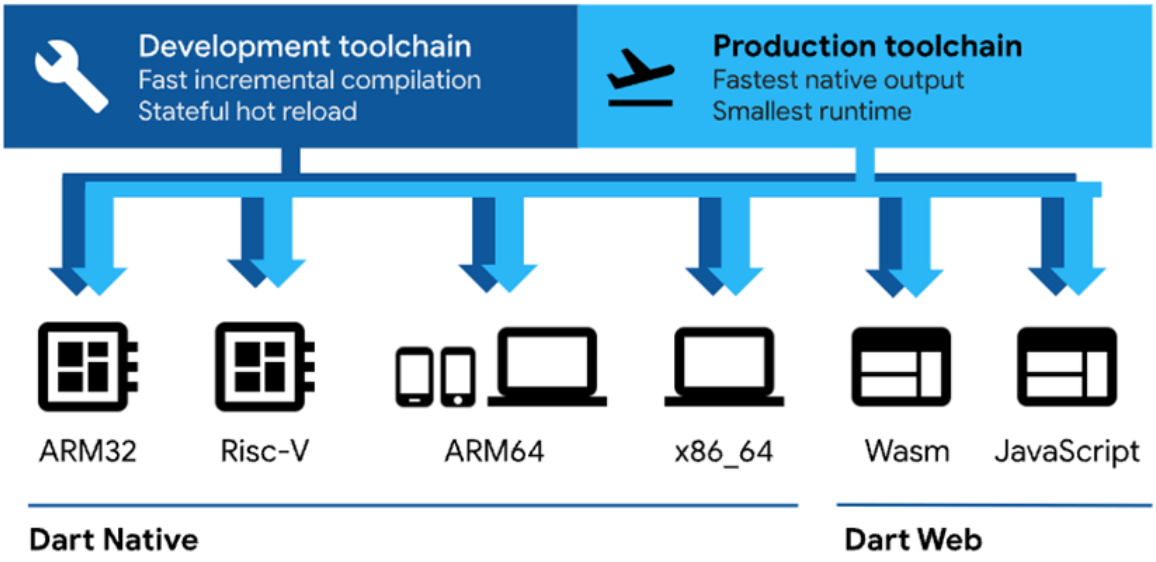


Рисунок 2.2 – Екосистема компіляції dart

2.2 Вибір фреймворку

2.2.1 Що таке Flutter

Кросплатформний фреймворк Flutter з’явився у 2017 році. Відтоді він став одним із найпопулярніших інструментів для розробки мобільних застосунків.

Flutter – це фреймворк із відкритим вихідним кодом. Він дає змогу створювати мобільні застосунки під Android та IOS, веб-застосунки, а також настільні застосунки під Windows, macOS та Linux. Розробкою та покращенням фреймворку займається компанія Google.

Цікавий факт, Google немає офіційної версії походження назви фреймворку. Деякі вважають, що вона відсилає до пісні гурту The Beatles, а можливо, що один з розробників просто запропонував цю назву, і вона всім сподобалась.

У перекладі з англійської to flutter означає «пурхати». В якомусь сенсі це відображає основну ідею фреймворку – швидка і плавна робота програмного забезпечення завдяки анімованим користувацьким інтерфейсам

2.2.2 Розвиток Flutter

У 2015 Google веде секретний проект. На той момент кросплатформні фреймворки вже були, проте React Native не підходив для проектів із важкою і складною графікою та анімацією. QT Mobile не користувався популярністю, а Ionic і PhoneGap використовували для візуалізації додатків веб-технології і зовсів не використовували нативні компоненти. Такий підхід знижував продуктивність.

Google прагнула знайти спосіб розробляти красиві додатки, які б без проблем працювали і на iOS, і на Android. Компанія без зволікань перейшла від теорії до практики і запустила секретний проект під кодовою назвою «Sky».

Команда проєкту опублікувала «Sky SDK» і заявила завдання – домогтися високої швидкості та рівня інтеграції з web. Розробники планували, що «Sky» зможе видавати картинку з частотою 120 кадрів на секунду. Тоді дисплеї смартфонів ще не могли працювати з такою швидкістю – максимум із 60 кадрами на секунду.

Розробники показали додаток, який малював кадр за 1,2 мілісекунди. Іншим додаткам для цього було потрібно 8 мілісекунд.

У 2017 році вийшла альфа-версія Flutter, поки все ще під ім'ям «Sky». Вона працювала тільки на Android і була натхненна рушієм «Skia». Особливість «Skia» в тому, що він абстрагується від графічних API, специфічних для платформи, і дає змогу малювати складні елементи інтерфейсу і будь-які 2D-сцени зі швидкістю 60 кадрів на секунду. А ще у 2017 вийшов перший комерційний продукт на Flutter – додаток до бродвейського мюзиклу «Гамільтон». За словами команди, фреймворк дав змогу розробити додаток за три місяці.

У застосунку можна було переглядати фото і відео, використовувати фільтри в стилі «Гамільтона», брати участь у розіграші квитків, купувати товари бренду і співати в караоке пісні з мюзиклу.

У 2018 році вийшла перша стабільна версія фреймворка – Flutter 1.0. Однією з ключових характеристик релізу стала швидкість завантаження і роботи програми. Розробники збільшили її завдяки впровадженню графічного рушія «Skia 2D».

Ще у Flutter 1.0 з'явилось відоме «гаряче перезавантаження». Ця функція автоматично оновлює додаток у браузері в разі збереження змін у вихідному коді. Вона дає змогу розробникам швидко побачити результати змін. Самостійно перезавантажувати сторінку не потрібно.

Відкрите програмне забезпечення дає змогу використовувати, модифікувати та поширювати код в інших програмах або додатках. Розробникам простіше розуміти, як працює код, коли він у них перед очима.

У 2019 команда додала підтримку веб-технологій у бета-версії. З'явилися стабільні API для інтеграції з Java, Kotlin, Objective-C і Swift. Функція «Add-to-App» дала змогу інтегрувати Flutter у нативні iOS і Android-додатки.

У 2021 році команда Google розширила можливості Flutter як мультиплатформового фреймворку. Версія Flutter 2.8 підтримувала шість платформ: iOS, Android, web, macOS, Windows і Linux.

У релізі з'явилася можливість монетизувати Flutter-додатки за допомогою Google «Mobile SDK». З'явилися п'ять рекламних форматів: банери, повноекранна реклама, відео, нативна реклама і реклама під час запуску програми. Також Flutter-додатки з цього моменту можна було інтегрувати з «AdMob» і «Ad Manager».

В релізі 3.19 з'явилася бета-версія Google «AI Dart SDK». Вона дає змогу вбудовувати у Flutter-додатки функції на основі моделей генеративного ШІ сімейства Gemini. Gemini API можна використовувати для детального управління анімацією віджетів.

Flutter у 2025 році отримав значне оновлення – версію Flutter 3.32, яка була офіційно представлена на конференції Google I/O у травні 2025 року. Це оновлення принесло низку нових можливостей і покращень для розробників

мобільних, веб та десктопних застосунків. Гаряче перезавантаження для вебу стало доступним у вигляді експериментальної функції, що дозволяє швидко оновлювати веб-інтерфейс без повного перезапуску сторінки.

Було додано підтримку Cupertino-компонентів із формою squircle, що точніше відтворює стиль iOS. З'явився новий інструмент Flutter Property Editor, який дозволяє змінювати властивості віджетів у реальному часі прямо в DevTools.

Покращено продуктивність Dart-аналітизатора, підвищено рівень доступності інтерфейсу, оновлено компоненти Material та покращено стабільність застосунків для Android та iOS. Рендеринг-рушій Impeller став доступний для пристроїв з Android 10 і новіших версій, забезпечуючи швидше та плавніше відображення інтерфейсу.

2.2.3 Архітектура Flutter

Google задумали Flutter як гнучку та модульну систему, яку можна розширити під різні потреби [7]. Уся архітектура Flutter складається з трьох основних частин (рисунок 2.3):

- Embedder;
- Engine;
- Framework.

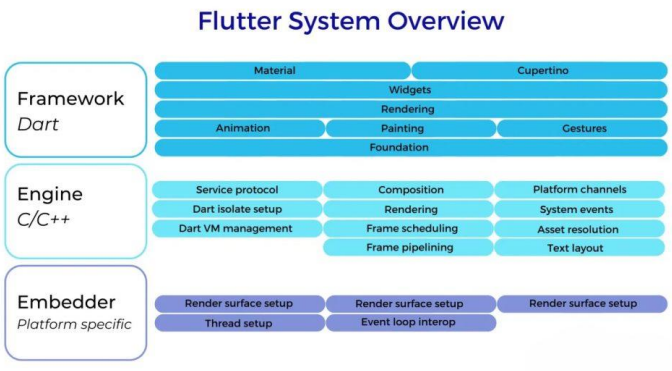


Рисунок 2.3 – Архітектурна модель Flutter

Рівень Embedder є базовим, він забезпечує взаємодію Flutter-застосунку з конкретною операційною системою. Простими словами, він згортає код в рідну оболонку. Для Android використовуємо Java або Kotlin, для iOS використовується Swift чи Objective-C, а для Windows чи Linux – C++. Embedder також відповідає за обробку вводу, event loop (цикл подій) та доступність.

Engine можна назвати серцем Flutter, там відбувається вся складна робота: графіка, введення/виведення, рендеринг, підключення плагінів, робота зі шрифтами, а також компіляція Dart-коду. Движок був написаний на C++ і використовує бібліотеку «dart:ui», вона дозволяє зв'язати dart-код із низькорівневою реалізацією. Для рендерингу графіки використовується рушій «Skia», який було описано в історії розвитку.

Framework складається з набору бібліотек, які дозволяють будувати UI, додавати анімації, працювати з навігацією, жестами, мережами тощо. Плюси та мінуси Flutter показано на рисунку 2.4.

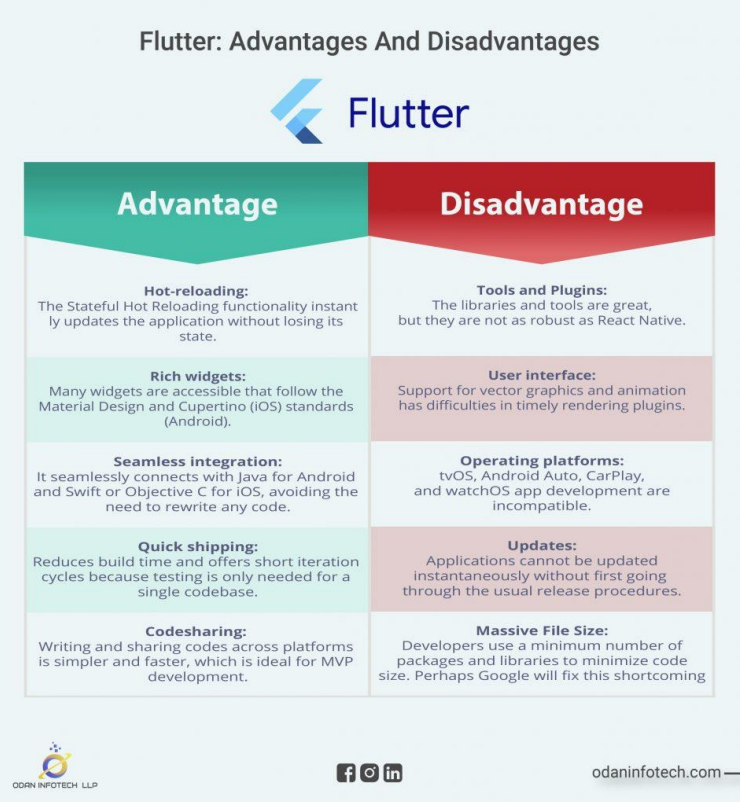


Рисунок 2.4 – Переваги та недоліки Flutter

2.2.4 Статистика використання Flutter

StackOverflow щорічно публікує звіти про технології, які розробники планують використовувати надалі. У 2020 році Flutter отримав 68.8% голосів, яку розробники хочуть і далі використовувати, в той час React Native набрав 57.09% (рисунок 2.5).

GitHub показує статистику активних учасників у проектах з відкритим кодом. За цією статистикою Flutter також попереду. Тринадцять тисяч контриб'юторів, у той час React Native має біля дев'яти тисяч (рисунок 2.5). У 2024 ця статистика практично не змінилася.

Подивимось на ситуацію з точки зору глобального інтересу. Google Trends показує, як часто користувачі шукають певний термін у Google. За рік запит Flutter отримав 86 балів, в той час як React Native – 58. Крім того, через Google Trends можна порівняти фреймворки з іншими рішеннями та відстежувати зміну інтересу до них в динаміці.

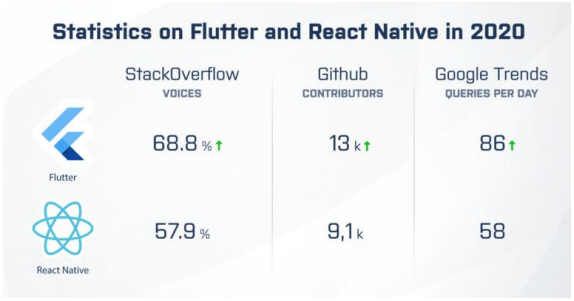


Рисунок 2.5 – Статистика за 2020 рік

2.3 Використання хмарного сховища Firestore

2.3.1 Переваги Firestore

Cloud Firestore – це рішення для роботи з базами даних NoSQL від Google Firebase, призначене для масштабованої та універсальної платформи для створення сучасних веб-застосунків і мобільних додатків. Firestore

забезпечує синхронізацію, зберігання та вилучення даних у режимі реального часу, пропонуючи при цьому потужні функціональні можливості, включно з автономною підтримкою, ієрархічною організацією даних і повним набором можливостей запитів.

Firestore пропонує можливість синхронізації у реальному часі, що дає змогу легко доставляти дані на кілька пристроїв і платформ, гарантуючи, що відповідні оновлення та зміни миттєво відображаються для всіх користувачів.

Firestore забезпечує вбудовану автономну підтримку як веб, так і мобільних застосунків, що дає змогу створювати додатки, які безперебійно працюють, навіть якщо користувачі не підключені до Інтернету. Firestore кешує дані локально на пристроях і синхронізує оновлення із сервером після відновлення з'єднання.

Firestore пропонує багатий API запитів, що дає змогу створювати складні запити, які з легкістю фільтрують, сортують і маніпулюють даними. Firestore також підтримує нумерацію сторінок на основі курсору, що дає змогу додаткам ефективно завантажувати та відображати великі набори даних.

Firestore використовує ієрархічну модель даних, яка організовує дані в колекціях і документах, включно зі складними та вкладеними структурами даних. Такий підхід спрощує структурування даних і управління ними, забезпечуючи при цьому високу гнучкість і масштабованість.

2.3.2 Модель даних Firestore

Модель даних Firestore заснована на концепції колекцій і документів, які забезпечують ієрархічну організацію та управління даними (рисунок 2.6).

Це дозволяє ефективно структурувати інформацію та швидко виконувати запити до потрібних рівнів даних.

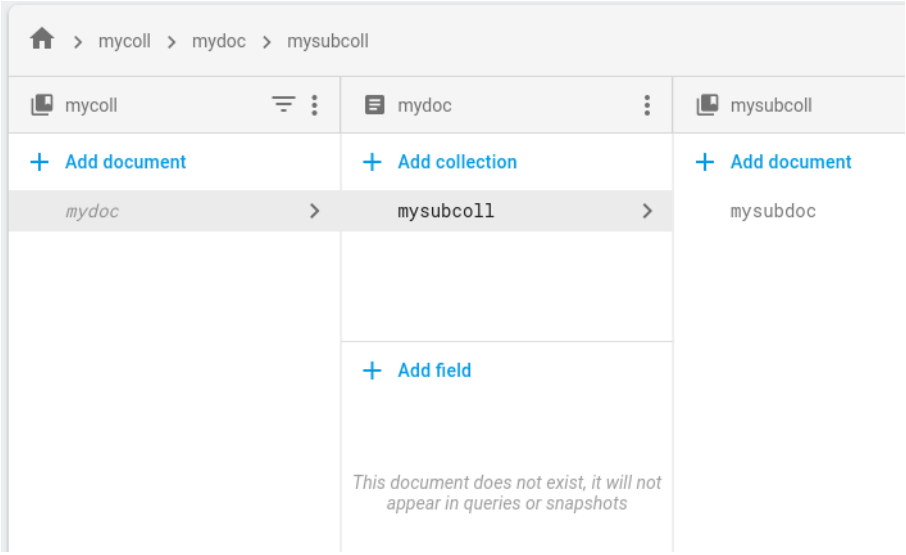


Рисунок 2.6 - Модель даних Firestore

У Firestore колекції – це контейнери, в яких зберігаються документи. Колекції допомагають організувати дані таким чином, щоб їх можна було легко запитувати та керувати ними, колекції також можуть містити підколекції, що дає змогу здійснювати подальший підрозділ логічних груп пов'язаних даних.

Документи - це окремі записи в Firestore, що містять фактичні значення даних, як правило, документи складаються з пар ключ-значення, відомих як поля, кожне з яких має ім'я і відповідне значення. Firestore підтримує різні типи даних для полів, включно з рядками, числами, логічними значеннями, масивами, картами та багато іншого, документи в Firestore можна розглядати як контейнери, які можуть містити як дані, так і підколекції, така вкладена структура забезпечує більшу гнучкість під час проектування та управління складними структурами даних у додатках [8].

2.4 Огляд архітектури управління станом на Flutter

Усе, що використовується у Flutter, складається з віджетів, вони можуть бути видимими, невидимими, містити дочірні віджети і взаємодіяти між собою. Кожен із них може бути як віджетом без стану («Stateless

Widget»), так і віджетом, у якого є стан («Stateful Widget»). Основна відмінність - можливість повторно відтворювати віджети під час виконання програми. «Stateless Widget» буде відтворюватися тільки один раз і є незмінним. «Stateful Widget» може відтворюватися безліч разів залежно від зміни внутрішнього стану віджета (рисунок 2.7).

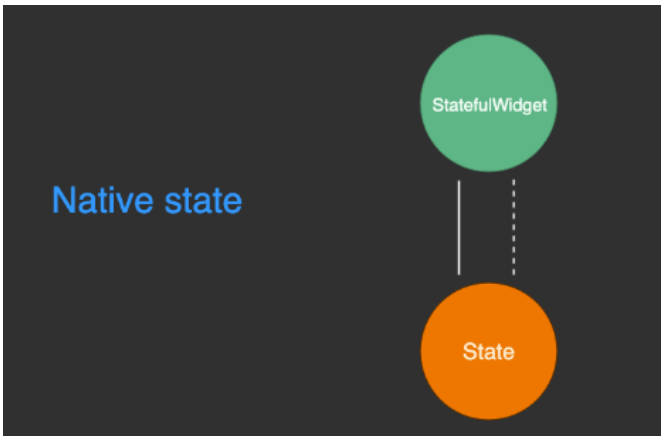


Рисунок 2.7 – Native State

Для створення «Stateful Widget» потрібно створити 2 класи. Перший клас має успадковуватися від «Stateful Widget», який своєю чергою успадковується від «Widget» і є незмінним.

Зразок цього класу не створюється під час кожного відмалювання і використовується для зберігання переданих параметрів та ініціалізації стану. Другий – клас стану, який має доступ до «Stateful Widget» через внутрішню властивість і займається безпосередньо відтворенням стану, реагуючи на його зміну.

Метод «setState()» є основним механізмом оновлення інтерфейсу в Native State, коли викликається «setState()», Flutter заново виконує метод «build()» для поточного віджета.

Проте це не означає, що оновлюється увесь інтерфейс, оновлюється лише той, у якому виклик відбувся. Також потрібно пам’ятати, що Flutter працює за принципом одностороннього потоку даних, тобто дані змінюють інтерфейс, але не навпаки.

2.5 Редактор коду Android Studio

Android Studio є інтегрованим середовищем розробки, яке було створене компанією Google для створення застосунків на платформі Android (рисунок 2.8). Базується на середовищі Intellij IDEA від JetBrains та є безкоштовним. Це середовище підтримується на основних операційних системах: Windows, macOS, Linux, і надає розробникам усі необхідні інструменти для написання, налагодження, тестування та розгортання мобільних застосунків.

Ключова особливість Android Studio в наявності інтелектуального редактора коду с підсвіткою синтаксису, автодоповненням і рефакторингом, який відчутно пришвидшує процес написання застосунків. В середовищі є інтеграція з Android SDK, підтримка симуляції роботи через віртуальні емулятори, а також гнучке керування залежностями через систему збірки Gradle.

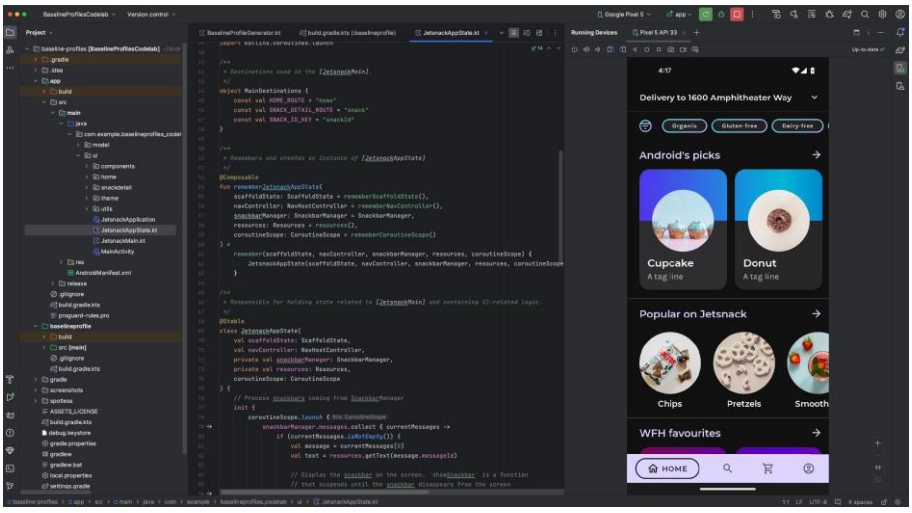


Рисунок 2.8 – Редактор коду Android Studio

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Архітектура та структура веб-проекту

Веб-застосунок побудований за принципами багаторівневої архітектури та був організований у вигляді компонентної структури за допомогою фреймворку Flutter Web.

Архітектура такого застосунку відповідає концепту поділу відповідальностей, де кожен клас чи елемент інтерфейсу відповідає за чітко визначену функцію.

Основна логіка починається з ініціалізації застосунку через віджет «ReminderApp», він встановлює глобальну тему та стартову сторінку. Сторінка «MainPage» реалізована як станова структура, яка зберігає поточний стан обраної вкладки та список запланованих нагадувань.

Застосунок поділено на дві основні візуальні сторінки: створення нагадування та перегляд уже створених. На сторінці створення нагадування користувач вводить текст повідомлення, вибирає час за допомогою таймпікера, після чого інформація зберігається в локальному списку нагадувань і одночасно створюється «Timer», який через заданий інтервал викликає браузерне сповіщення з текстом повідомлення. Для цього використовується можливість взаємодії з «Web Notification API» через бібліотеку «dart:htn1», про яку було описано у 2 розділі. У разі, якщо час обрано раніше поточного моменту, сповіщення автоматично переноситься на наступний день.

Сторінка перегляду запланованих нагадувань реалізована у вигляді списку, кожен елемент якого представлений карткою з інформацією про повідомлення та час його виконання. Додатково передбачено можливість видалення нагадування з цього списку, що відображається миттєво у візуальному інтерфейсі.

Всі картки мають сучасне оформлення з темною стилістикою, закругленими кутами, неоновими елементами, контрастними кнопками та іконками. Уся логіка нагадування інкапсулюється у класі Reninder, який є моделлю даних і містить поля для зберігання дати та часу та тексту повідомлення. Інтерфейс оформлений з використанням темної теми, у якій переважають кольори темного сірого та неонової бірюзи. Такий вибір кольорової палітри підкреслює сучасність та стильність дизайну. Всі елементи розташовані з достатнім відступом і врахуванням адаптації до розміру екрана браузера.

3.2 Архітектура та структура мобільного проекту

Архітектура створеного застосунку була побудована згідно з сучасними принципами модульності, повторного використання коду та логічного розділення відповідальностей. Фреймворк Flutter складає основу цього проекту, саме він забезпечує гнучкість розробки. Архітектуру проекту умовно можна поділити на кілька ключових етапів: UI, бізнес-логіка, сервісний та модельний. Кожен з них буде детально описано нижче.

3.2.1 Архітектурний патерн застосунку

В проекті застосовано популярну архітектурну схему, яка використовується в розробці програмного забезпечення. Така схема схожа на MVC, але з деякими ключовими відмінностями.

MVVM – це шаблон архітектури ПЗ, який розділяє програму на три компоненти: Model, View та ViewModel. Model представляє дані та логіку програми, View забезпечує відображення даних для юзера, а ViewModel відповідальний за надання даних введення користувача та обробки. Саме ViewModel і є ключовою відмінністю від шаблону MVC. У MVC контролер обробляє введення користувача та оновлює модель та перегляд відповідно, а

в MVVM ViewModel обробляє введення користувача та оновлює модель, яка оновлює перегляд.

Серед переваг можна виділити те, що цей патерн розділяє додаток на три чіткі компоненти, кожна з них має свої власні обов'язки. Таке розділення полегшує підтримку а також зміну коду. Також відокремлення проблем та модульності дозволяє повторно використовувати код у декількох додатках чи компонентах.

MVVM забезпечує масштабовану архітектуру, яка може бути розширена для підтримки нових функцій та функціональності. Тому застосування цього патерну є повністю виправданим в межах даного застосунку (рисунок 3.1).

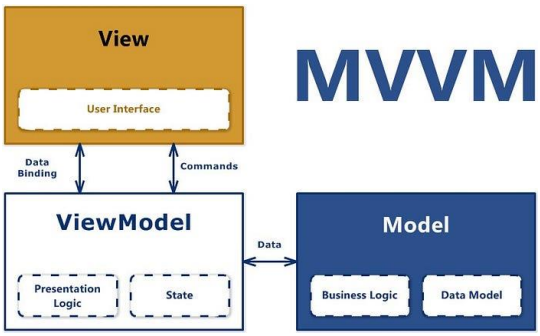


Рисунок 3.1 – Патерн MVVM

3.2.2 UI рівень застосунку

Відповідний рівень відповідає за представлення зовнішньої інформації, тобто сам інтерфейс користувача. Усі візуальні компоненти були реалізовані у вигляді віджетів Flutter, вони виконують роль основної одиниці побудови інтерфейсу в межах фреймворку. Інтерфейс побудований з урахуванням сучасних стандартів, в додатку присутні плавні анімації, інтуїтивна навігація та адаптивність до різних екранів.

В розробленій аплікації, головним елементом є екран HomeScreen, саме він відображає список активних нагадувань, дозволяє взаємодіяти з ними, а також переглядати профіль чи події сьогоднішнього дня (рисунок 3.2).

3.2.3 Бізнес-логіка застосунку

В даній аплікації бізнес-логіка реалізована безпосередньо у «StatefulWidget» компонентах через методи класів «State». Це дозволило локалізувати логіку в межах певного екрану. Наприклад, логіка відображення, обробки та оновлення нагадувань реалізована в середині HomeScreen. А логіка роботи зі сповіщеннями була окремо реалізована у класі NotificationLogic, який є універсальним сервісом для показу сповіщень, планування повторів та скасування повідомлень.

Рисунок 3.2 – Екран HomeScreen

3.2.4 Робота з Firebase

Firebase застосовується для хмарного бекенду і складається з таких компонентів: Firebase Authentication, Cloud Firestore, Firebase Core. Authentication використовується для реєстрації, входу чи виходу користувача, перевірки електронної пошти, а також входу через Google акаунт. Firestore є масштабованою базою даних, яка дозволяє зберігати структури документів і колекцій. Інформація користувача зберігається у вигляді вкладених колекцій. Core забезпечує базову ініціалізацію сервісів.

Структура даних передбачає наявність колекції users, в ній зберігається унікальний документ кожного користувача, його UID. Всередині кожного документа зберігається вкладена колекція reminder, яка містить усі нагадування користувача (рисунок 3.3).

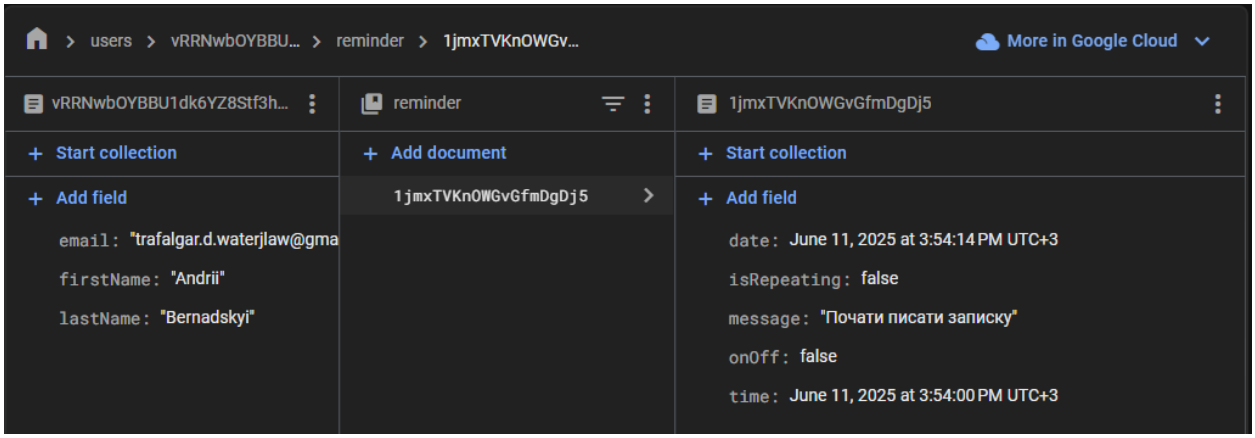


Рисунок 3.3 – Структура бекенду

3.2.5 Модельний рівень мобільного застосунку

Для того аби зберігати та передавати структуровані дані у застосунку було створено клас-модель – ReminderModel (лістинг 3.1). У ньому визначено основні параметри нагадування: час, повідомлення, повторюваність, активність, а також дату виконання, що дозволяє централізовано зберігати та оновлювати дані у зручному форматі. Модельний рівень також відповідальний за логіку перетворення даних між UI та Firestore, завдяки цьому, дані, що зберігаються у базі, відображаються у зручному для користувача вигляді.

Timestamp відповідає за точний момент часу, коли має бути активоване нагадування. Використовується для збереження значення у форматі сумісному з Firestore.

OnOff вказує на активність нагадування, якщо true, то нагадування буде спрацьовувати у заданий час, якщо false – нагадування вимкнено.

Messege містить текст повідомлення, яке буде надіслано користувачеві

у разі спрацювання.

Date зберігає дату спрацювання нагадування, зручному для локальної обробки

IsRepeating визначає, чи є нагадування повторюваним.

Лістинг 3.1 – Модель ReminderModel

```
class ReminderModel {
  Timestamp? timestamp;
  bool? onOff;
  String? message;
  DateTime? date;
  bool? isRepeating;
  ReminderModel({
    this.timestamp,
    this.onOff,
    this.message,
    this.date,
    this.isRepeating = false,
  });
```

3.3 Структура мобільного застосунку

Flutter надав можливість створити мобільний застосунок з використанням стандартної структури android-проекту, вона включає в себе набір конфігурацій та сервісних файлів. Така структура дає змогу регулювати збірку, визначати версії SDK, під'єднувати залежності, плагіни, а також реалізовувати права доступу до системних ресурсів (рисунок 3.4). Основними конфігураційними файлами, які формують структуру проекту є:

- settings.gradle;
- build.gradle;
- AndroidManifest.xml;
- local.properties;
- gradle.properties.

Такий файл як setting.gradle має роль має роль конфігуратора всього проекту, він визначає, які модулі варто підключити до збірки, також задає параметри керування плагінами використовуючи блок pluginManagement:

- динамічно зчитується шлях до SDK з файлу local.properties;

- включається набір утиліт, які забезпечують інтеграцію Flutter у проект – flutter_tools/gradle;

- підключаються необхідні плагіни: flutter-plugin-loader, який відповідальний за ініціалізацію фреймворка, com.android.application є головним android плагіном, com.google.gms.google-services підключає конфігурацію Firebase.

Модульний файл build.gradle конфігурує android додаток, включаючи налаштування SDK, залежностей і так далі. Головним підключеним плагіном до аплікації є Google Services, який потрібний для правильної роботи Firebase, адже він зчитує google.services.json і автоматизовано додає залежності, які потрібні в проект. Секція android у build.gradle визначає параметри застосунку, а саме: контролює сумісність, ідентичність та функціональність під час процесу компіляції та запуску. В цій є характеристики версії Android SDK, параметри сумісності та інформацію про версію самого застосунку. З контексту стає зрозуміло, що ці налаштування впливають на те, з яких саме пристроїв може бути використаним цей застосунок.

Серцем Android налаштувань можна назвати AndroidManifest, в ньому описуються дозволи, метадані та активності. В лістингу 3.2 описані дозволи, які вкрай необхідні для правильної роботи застосунку, для правильної роботи з точними таймерами, повноекранними сповіщеннями, а також автоматичним запуском після перезавантаження системи.

Лістинг 3.2 – Дозволи user-permission

```
<uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission
android:name="android.permission.SCHEDULE_EXACT_ALARM"
    tools:ignore="ProtectedPermissions" />
<uses-permission
android:name="android.permission.USE_EXACT_ALARM"
    tools:ignore="ExactAlarm" />
<uses-permission
```

```
android:name="android.permission.USE_FULL_SCREEN_INTENT" />
```

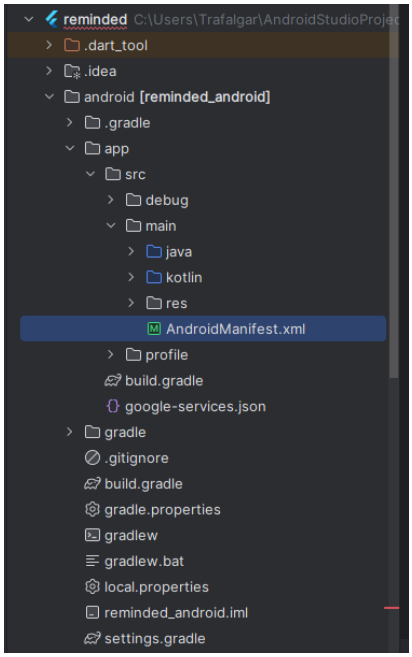


Рисунок 3.4 – Основна структура Android проекту

Після опису структури, що є основою усіх Android застосунків, розглянемо опис структури, завдяки якій було відтворено застосунок в цілому.

В каталозі assets знаходяться усі ресурси, які є візуальною частиною застосунку, але не є кодом. В ній знаходиться папка icons, де розташовуються іконки, що були використані в різних частинах інтерфейсу користувача. В загальному асети використовуються як декоративні чи функціональні елементи, підключається в pubspec.yaml.

Каталог lib основною директорією з вихідним кодом додатку, де відбувається реалізація всього функціоналу. В свою чергу цей каталог поділено на інші підкаталоги задля розділення логіки.

Підкаталог utils в якому є файл app_colors.dart несе задачу у створенні візуального стилю та дизайну у всьому застосунку. В будь-яких великих чи малих застосунках важливо дотримуватись правил сталої кольорової палітри, задля того, щоб забезпечити зручність та позбутись візуального шуму. В цьому файлі були зібрані усі кольори, які застосовані в інтерфейсі. Це було

зроблено для того, аби кожного разу не прописувати значення кольору вручну для кожного віджета, а посилатися до вже оголошених констант, щоб централізовано змінювати стиль застосунку. Серед основних оголошень є:

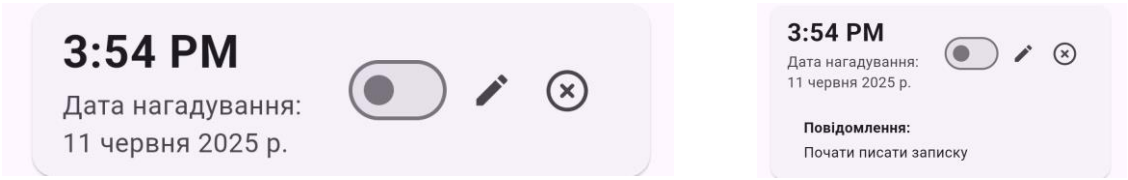
- primaryColor1, primaryColor2, які є основною градієнтною парою для кнопок, фонів та заголовнів;

- secondaryColor1, secondaryColor2 виконують роль допоміжніх кольорів, для створення додатного візуального акценту;

- grayColor, midGrayColor, LightGrayColor застосований для тексту підказок, іконок у неактивному стані і так далі.

У файлі присутні getter-методи primary та secondary для того, щоб повернути списки з градієнтами, щоб зручно створювати візуальні елементи з переходом кольору.

Також присутній файл custom_animation.dart, оскільки в сучасних мобільних застосунках важливо зробити інтерфейс плавним та привабливим для юзера. Цей файл містить кастомізований віджет аби зробити анімацію плавною та контрольованою. Він використовується для відображення розкриття блоків з плавною та контрольованою анімацією. Компонент використаний для відображення детальної інформації про нагадування. У складеному стані юзер бачить заголовок (рисунок 3.5 а), а при розгорнутому – додаткову інформацію (рисунок 3.5 б).



а) б)

Рисунок 3.5 – а) Згорнутий стан б) Розгорнутий стан на одній сторінці

Підкаталог screens має основні екрани застосунку, а саме компоненти, які користувач бачить та з якими він взаємодіє. Вони є повноцінними частинами інтерфейсу, кожна з яких відповідає за певну функціональність.

Коли користувач вирішить скачати застосунок, відкривши його, перше, що він побачить – буде login_screen. Він є екраном авторизації, що дозволяє юзеру увійти в систему. На цьому етапі відбувається введення електронної пошти та пароля, а також можливість входу через Google. Тут відбувається перевірка на правильність введених даних, а також наявні допоміжні функції, такі як, скидання паролю.

Якщо користувач не має аккаунту, він може перейти до signup_screen, тобто екрану реєстрації. Тут відбувається створення нового облікового запису з валідацією введених даних.

Виконавши вхід до застосунку, користувач потрапляє до центрального та найголовнішого екрану аплікації. На робочому столі юзеру доступний перелік створених нагадувань, де можна ними керувати, а також створювати нові та редагувати їх. Також додатковим функціоналом є можливість відкрити бічне меню. Ця панель показує святкові дати на поточний день (рисунок 3.6).

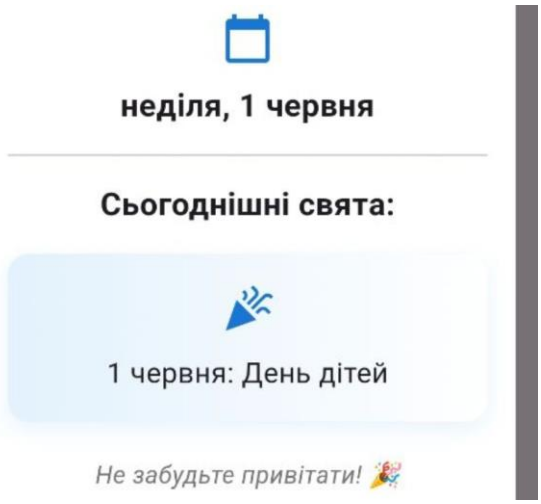


Рисунок 3.6 – Панель поточних свят

Присутній файл profile_screen, який дозволяє редагувати та переглядати персональні дані. Було реалізовано перегляд імені, прізвища та електронної пошти. Юзер має змогу змінити пароль, через надісланий на пошту лист.

У піддиректорії widgets зберігаються інтерфейсні компоненти, що застосовуються в різних частинах додатку. Такі екрани можна назвати частинами більших екранів, винесених задля покращення повторного використання та підтримки коду.

Файл add_reminder виконує функція в керуванні модальним вікном, яке з'являється, коли користувач створює нове нагадування. З точки зору візуального представлення, воно реалізовано як діалогове вікно, яке дозволяє юзеру обирати тип нагадування.

Схожий на попередній файл, edit_reminder створений для редагування наявних нагадувань. Він дозволяє змінити час та дату нагадування, а також його вміст.

Ізольованим діалоговим елементом є delete_reminder, призначений для видалення нагадувань. Реалізовано у вигляді спливаючого вікна, яке містить кнопки для підтвердження дії чи скасування.

Віджет switcher реалізовує перемикач для активації чи деактивації нагадування. При його використанні можна змінювати стан конкретного нагадування, не видаляючи його.

Розділ services містить сервіси, які реалізують фонову логіку роботи програми. Це не частина візуального інтерфейсу, вони взаємодіють з системними службами та зовнішніми API, а також локальним механізмом пристрою.

Один з найважливіших файлів є notification_logic, він відповідає за показ локальних сповіщень, забезпечує ініціалізацію сервісу сповіщень, формування одноразових чи щоденних нагадувань, їх оновлення та видалення. Цей механізм реалізовано, щоб користувач отримував повідомлення навіть коли застосунок закрито чи працює у фоновому режимі.

Файл daily_fact_service взаємодіє із зовнішнім API, з якого і отримується святкові дати на поточний день. Також в файлі виконується переклад назв свят на українську мову. Отримані дані використовуються задля створення панелі, що відкривається з головного екрану (рисунок 3.6).

Файл reminder_model описує структуру об’єкта нагадування. Використовується як шаблон для збереження та обробки даних про сповіщення. Він дозволяє формалізувати дані для зручної співпраці під час збереження до бази даних (рисунок 3.7).

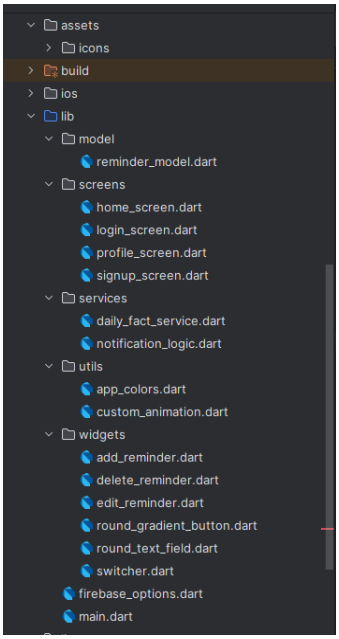


Рисунок 3.7 – Структура застосунку

3.4 Реалізація автентифікації користувача

Розробка почалась з налаштування проекту через Firebase Console. В ній було створено новий проект, після чого активовано модуль Authentication, після було підключено Cloud Firestore для зберігання інформації про користувача та його нагадувань, та Google-In – для входу через акаунт Google (рисунок 3.8).

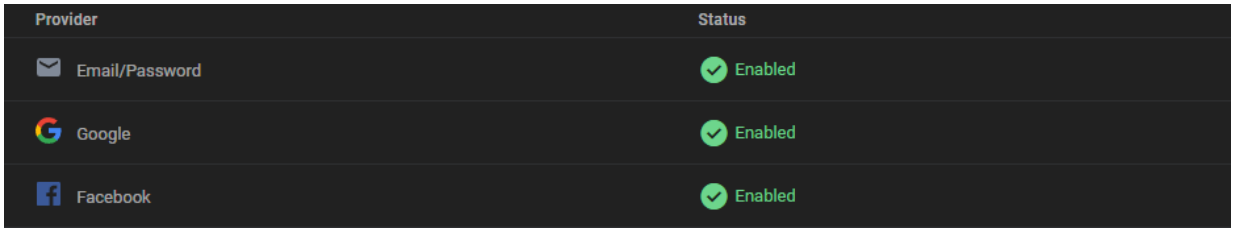


Рисунок 3.8 – Підключені провайдери

3.4.1 Вхід через Google

Щоб підключити Firebase до проекту було застосовано інструмент flutterfire configure, який згенерував файл firebase_options.dart, що містить унікальні параметри, наприклад, ключ доступу до API Firebase та унікальний ідентифікатор проекту.

Для підключення усіх необхідних плагінів в pubspec.yaml було виконано команду flutter pub get, де було отримано пакети (таблиця 3.1). Для того, щоб вхід через Google не викликала помилку: «Google sign in failed com.google.android.gms.common.api.ApiException: 10», обов'язково потрібно згенерувати SHA-1 ключ.

Цей криптографічний ключ використовується для валідації застосунку, в іншому випадку вхід працювати не буде. Отриманий ключ було додано до консолі Firebase до вкладки Project Settings → General → Your apps → SHA certificate fingeprints

Таблиця 3.1 – Отримані пакети

Пакет	Призначення
Firebase_core	Ініціалізація Firebase у Flutter
Firebase_auth	Основний функціонал автентифікації
Google_sign_in	Вхід через акаунт Google
Cloud_firestore	Зберігання додаткових даних
Fluttertoast	Відображення повідомлень

Вхід за допомогою Google був реалізований через приватну функцію «_signInWithGoogle()», яка виконує процес входу користувача через акаунт Google з використанням Firebase Authentication. Спочатку функції виконується метод «await GoogleSignIn().signOut()», через це процес входу розпочинається з нуля, не використовуючи минулі сесії, це зроблено для того, аби користувач міг увійти під декількома почтовими скринями. Без

цього методу, якщо користувач увійде один раз через Google, то увійти під іншим профілем він вже не зможу, адже буде збережено минулу сесію.

Після цього відбувається ініціалізація входу за допомогою «GoogleSignIn().signIn()», на цьому етапі відкривається діалогове вікно, де користувач обирає потрібний акаунт, якщо користувач передумав чи закрив вікно, то функція просто завершиться, не виконуючи ніяких дій. Якщо вибір облікового запису завершився успіхом, то система отримує об’єкт «GoogleSignInAccount», де міститься базова інформація про користувача.

Подальші дії це запит на отримання даних авторизації – «accessToken» та «idToken», вони підтверджують вхід в систему, а також слугують для ідентифікації у Firebase. На основі токенів створюється об’єкт credential, він є ключевим для автентифікації у Firebase. Через метод «signInWithCredential», такий «credential» передається на сервер Firebase, де відбувається підтвердження автентичності користувача. Після успішної операції система повертає об’єкт «userCredential», з якого отримується користувач.

Далі було реалізовано логіку збереження та перевірки користувача у базі даних Firestore. З об’єкта «User» витягується унікальний ідентифікатор «uid» та «email». Далі у Firestore відкривається документ у колекції «users», під цим «uid». Після система перевіряє, чи існує такий документ, в даному випадку, чи входив такий користувач раніше до системи. Якщо запис не було створено, тобто користувач входить вперше, у базі буде створено новий документ з його електронною поштою. Поля «firstName» та «lastName» залишаються порожніми, в подальшому їх можна доповнити через екран налаштування профілю.

Завершивши всі операції та перевірки відбувається перехід до головного екрану застосунку. Виконується «Navigator.pushReplacement», тобто вхід здійснюється без можливості повернутись назад, що є логічним після успішного входу.

Якщо виникає помилка на будь-якому етапі, помилка перехопиться через «catch» і користувач отримає завдяки «SnackBar» відповідне

повідомлення. Така відповідність дозволяє уникнути аварійного завершення роботи застосунку, а також забезпечує зворотній зв’язок (рисунок 3.9).

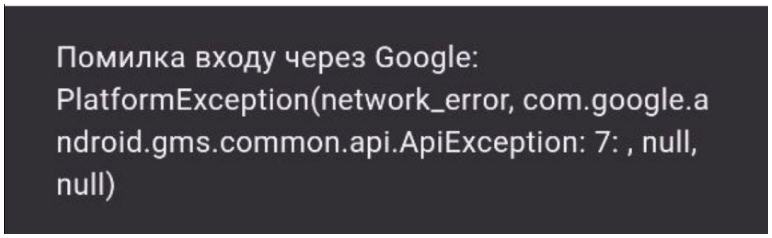


Рисунок 3.9 – Помилка входу через відсутність мережі

3.4.2 Вхід через електронну пошту та пароль

Функцію входу реалізовано в асинхронному вигляді. Коли користувач вводить облікові дані, в даному випадку пошту та пароль, і натискає кнопку «Увійти», функція викликається для перевірки правильності введеної інформації.

Спроба входу виконується методом «signInWithEmailAndPassword», він надсилає на сервер запит з переданими даними, якщо вони коректні, то повертає об’єкт «UserCredential», який в собі містить деталі про автентифікованого користувача.

З отриманого об’єкта вилучається конкретний користувач, який містить повну інформацію про авторизованого клієнта, його «email» та «uid», стан підтвердження пошти. Після цієї операції відбувається перевірка на підтвердження електронної пошти, така перевірка є важливою, адже запобігає створенню облікових записів, які не існують чи чужими поштовими адресами.

Якщо система визначила, що пошта не підтверджена, то автоматично надсилається лист-підтвердження на електронну скриню через метод «sendEmailVerification», а також в «SnackBar» приходить повідомлення, який просить користувача підтвердити пошту, і функція повертає «null», тобто користувач не допущений до системи.

Якщо пошта була підтверджена, з’являється повідомлення про успішний вхід і після виконується переадресація до головного вікна програми.

У разі будь яких помилок спрацьовує блок обробки винятків «catch», де користувач отримає повідомлення про помилку і програма не завершить роботу в аварійному режимі (рисунок 3.10).

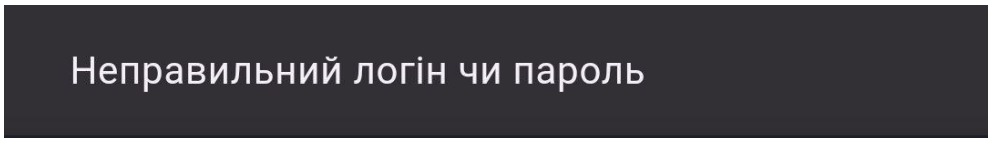


Рисунок 3.10 – Помилка входу, неправильні вхідні дані

3.5 Реєстрація облікового запису

Для реєстрації облікового запису користувач повинен заповнити поля з особистою інформацією: ім’я, прізвище, електронна пошта та пароль. Кожне поле має валідацію, якщо порожнє – з’являється відповідне повідомлення про помилку, для пароля є перевірка на мінімальну довжину. Також обов’язковим етапом є підтвердження політики конфіденційності. Після заповнення форми та натискання кнопки «Зареєструватись» відбувається перевірка на коректне введення даних і встановлення галочки згоди. Якщо перевірка пройдена, тоді розпочинається процес створення облікового запису. Метод «createUserWithEmailAndPasswor» надсилає введену електрону адресу та пароль на сервер Firebase Authentication. Після створення викликається метод «sendEmailVerification», який надсилає на вказану адресу лист-підтвердження. Окрім створення облікового запису в Firebase Authentication, ще створюється окремий запис у Firestore. Відкривається колекція users, в ній додається документ з унікальним «uid», де зберігається пошта, ім’я та прізвище. Після операцій користувач буде автоматично перенаправлений на сторінку входу в обліковий запис, де після підтвердження пошти зможе увійти.

3.6 Реалізація логіки нотифікації

Важливою реалізацією є система локальних сповіщень, вони дозволяють сповіщати користувача навіть в тих моментах, коли застосунок перебуває у фоновому режимі, або повністю закритий. Для цієї реалізації було застосовано пакет «flutter_local_notifications», що дав функціонал для створення, планування та керування локальними сповіщеннями на пристрої.

Логіка сповіщень була винесена в окремий клас «NotificationLogic», це дозволило керувати всіма процесами пов'язаними з нотифікацією і зробити проект модульним. Клас включає методи, які дозволяють:

- ініціалізувати систему сповіщень під час запуску застосунку;
- запланувати одноразове нагадування;
- запланувати щоденне нагадування;
- скасувати сповіщення;
- обробити натискання на сповіщення та виконати відповідну дію.

Під час запуску аплікації викликається ініціалізація з Android налаштуванням, де задається іконка налаштування, взята з асетів проекту. Також ініціалізуються часові зони для коректної роботи з часом.

Метод «showNotification» планує сповіщення на конкретний день і час. Параметрами є заголовок, повідомлення, дата і час, а також ідентифікатор, який дозволяє видалити чи редагувати це повідомлення. Метод «showDailyNotification» реалізує логіку регулярних та щоденних нагадувань на певний час.

3.7 Реалізація створення, редагування та керування нагадуваннями

Функція для створення нового нагадування реалізована через віртуальний компонент, він відкривається у формі модального «bottom sheet», основним призначенням є забезпечення зручного інтерфейсу без переходу на окрему сторінку. При натисканні кнопки «+» відкривається

вибір типу нагадування: одноразове чи щоденне. Кожен тип має свою іконку, заголовок а також опис, після вибору типу відкривається форма, яка дозволяє налаштувати параметри нагадування. Аби заповнити форму для одноразового нагадування, потрібно:

- обрати дату ;
- вказати час;
- ввести повідомлення, текст який з’явиться у вигляді сповіщення;
- натиснути кнопку «додати» для збереження нагадування.

Така форма реалізована завдяки кастомному дизайну та кольорового оформлення та відповідає загальному стилю застосунку.

Після того, як кнопка «додати» була натиснута, створиться об’єкт на основі моделі «ReminderModel». Вона отримає дату, час, повідомлення, тип та статус активності. Такий об’єкт буде сконвертовано до формату «Map<String, dynamic>» і буде збережено у підколекції reminder документа користувача у Firestore.

При натисканні кнопки редагування, відкриється така ж сама форма у вигляді вікна, але вже із заповненими раніше даними. Юзер має змогу змінити параметри: відтермінувати дату чи переписати текст. Після збереження інформація у Firestore оновлюэться через метод update, без створення нового документа, що дозволяє зберегти ідентифікатор нагадування та пов’язані з ним дані.

Функція видалення реалізована як у більшості застосунках через спливаюче діалогове вікно підтвердження. Після підтвердження видаляється документ із Firestore та скасовується локальне сповіщення через виклик методу з класу «NotificationLogic»

На головній сторінці користувач може побачити список своїх нагадувань, кожке з який має перемикач, який в свою чергу відповыдає за увімкнення та вимкнення нагадувань. Функціональність реалізована у вигляді окремого віджета, який отримує значення «UID» користувача, ідентифікатор та час. Зміна стану перемикача одразу оновлює базу даних.

3.8 Отримання свят через зовнішнє API

В проекті було реалізовано блок, який показує користувачу свята на поточну дату, це збільшує привабливість додатку, а також додає щоденний динамічний вміст. Задача функції зробити інтерфейс цікавішим та інформативним, для покращення взаємодії з користувачем. Для його реалізації було використано зовнішнє API – сервіс Calendarific, він дозволив отримати інформацію про міжнародні та національні свята (рисунок 3.11).

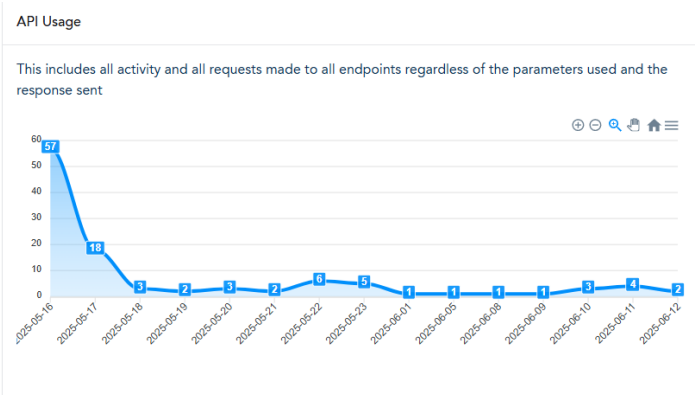


Рисунок 3.11 – Запити виконані API

Функція «getHolidaysForToday» надсилає запит до онлайн-API для отримання переліку свят на конкретну дату. Ця функція є асинхронною. На початку функція визначає поточну дату і формує її у формат уууу-MM-dd, який потрібний для API-запиту. Формується URL-адреса до API- Calendarific, у яку встановлюються динамічні параметри: рік, місяць, день.

Коли сервер повертає статус-код, який відмінний від 200, то функція генерує повідомлення про помилку. Якщо запит був успішним, то отримана відповідь декодується з формату JSON. Потім з тіла витягується список свят, що стосуються поточної дати. Якщо така відсутня, то одразу повертається тестове повідомлення «На сьогодні свят немає» (рисунок 3.12). Слід зазначити, що API- Calendarific повертає назви свят англійською мовою. Для того аби забезпечити повну локалізацію було реалізовано автоматичний переклад назв свят на українську мову. Було використано пакет translator,

який надає доступ до перекладача Google Translate. Якщо в процесі сталася помилка, тоді функція перехоплює виняток у блоці «catch». У такому випадку буде виведено на екран текстове повідомлення: «Сталася помилка при отриманні свят.»

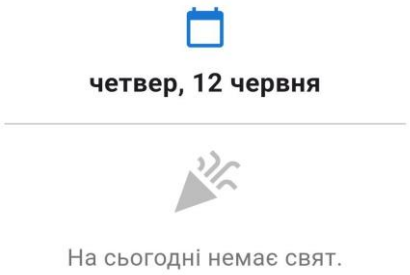


Рисунок 3.12 – Успішне виконання запиту

3.9 Екран користувацького профілю

Для зручності було реалізовано екран профілю користувача, який надає доступ до даних облікового запису. Такий розділ відповідає за виведення та редагування введеної раніше особистої інформації про користувача. Після відкриття екрану застосунок завантажує дані поточного користувача:

- викликається метод «_loadUserData()» під час ініціалізації;
- через об’єкт FirebaseAuth визначається «UID» поточного користувача;
- виконується запит до колекції users у Firestore за цим «UID».

Якщо користувач натиснув кнопку редагування профілю, то відкриється модальне вікно, яке містить форму з текстовими полями: нове ім’я, нове прізвище. Після натискання кнопки «Зберегти» дані будуть збережені у базі Firestore, локальні змінні оновлюються також, щоб інформація одразу змінилась на екрані без перезавантаження, а після виводиться сповіщення про вдале редагування.

В наявності функція скидання пароля, вона реалізована через вбудований механізм Firebase. При натисканні на відповідну кнопку,

викликається метод «sendPasswordResetEmail(email: email!). Цей метод надсилає лист на вказану адресу з інструкцією щодо скидання пароля (рисунок 3.13).

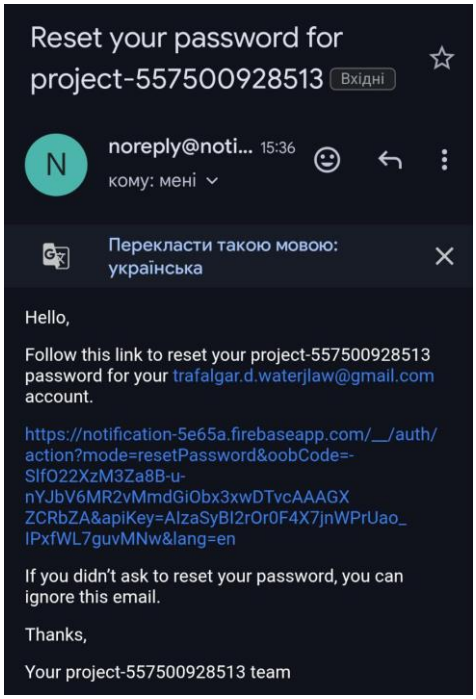


Рисунок 3.13 – Лист на зміну пароля

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Користування веб-застосунком

Після відкриття веб-застосунку користувач бачить головну сторінку з двома розділами: створення нагадування та перегляд запланованих сповіщень (рисунок 4.1). За замовчуванням відкрито вкладку створення нагадування. У верхній частині екрана розміщений заголовок з назвою функціонального блоку, а нижче – поле для введення тексту нагадування. Користувач може вписати будь-яке повідомлення, яке він хоче отримати у майбутньому, наприклад, «Випити воду» або «Підготувати звіт».

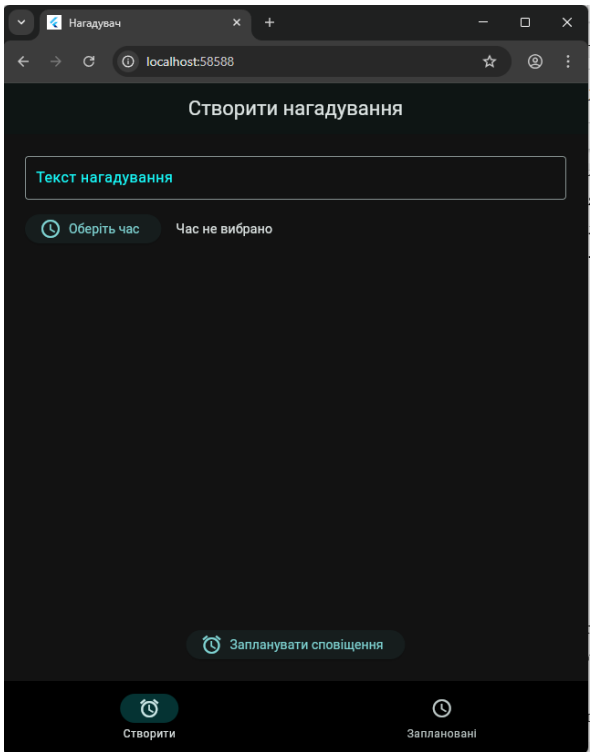


Рисунок 4.1 – Головне вікно аплікації

Під текстовим полем розташована кнопка вибору часу, при натисканні на яку відкривається стандартне вікно вибору години та хвилин. Після того, як час обрано, він відображається поруч із кнопкою у текстовому вигляді,

щоб користувач міг переконатися у правильності введення. Якщо користувач не вибере час, система не дозволить створити нагадування, попередивши його про помилку.

Коли всі дані заповнені, користувач натискає кнопку «Запланувати сповіщення», після чого на екрані з'являється повідомлення про успішне створення нагадування. В цей момент система автоматично запускає таймер, який чекає настання обраного часу. Як тільки цей час настає, у браузері з'являється спливаюче повідомлення – нотифікація з текстом, який користувач задав раніше (рисунок 4.2). Це повідомлення відображається у вигляді системного сповіщення у правому нижньому кутку екрана (на Windows) або у зоні сповіщень браузера (на інших операційних системах). Для коректної роботи цієї функції браузер повинен мати дозволи на показ сповіщень, які система запитує автоматично при першому використанні.

Рисунок 4.2 – Заплановане сповіщення

Щоб переглянути список усіх створених нагадувань, користувач може перейти до другої вкладки, натиснувши на іконку годинника внизу екрана. У цій вкладці відображається список запланованих нагадувань у вигляді карток, кожна з яких містить текст повідомлення та вказаний час (рисунок 4.3). Якщо користувач більше не потребує якогось нагадування, він може легко його видалити, натиснувши на іконку кошика праворуч від відповідного рядка. Всі дії виконуються у реальному часі, інтерфейс одразу оновлюється після кожної зміни – додавання або видалення нагадування. У разі успішного видалення система також показує коротке підтвердження дії у вигляді спливаючого повідомлення.

Завдяки простоті інтерфейсу та мінімалістичному дизайну користувач легко орієнтується в застосунку навіть при першому використанні. Таким чином, процес взаємодії з веб-застосунком зводиться до трьох основних кроків: введення повідомлення, вибір часу та підтвердження створення. Усі елементи розміщені логічно та послідовно, що дозволяє зосередитися лише на головному – нагадуваннях.

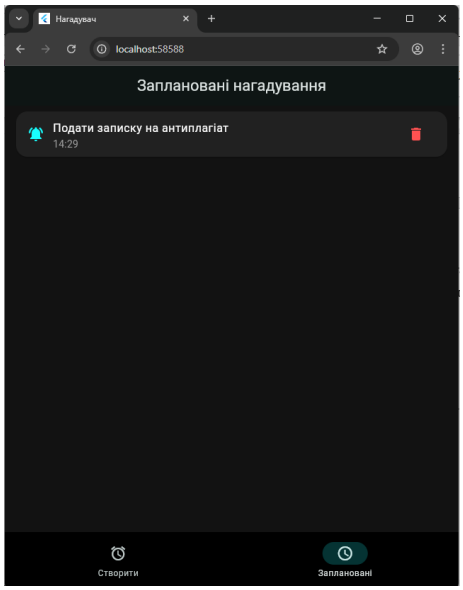


Рисунок 4.3 – Вікно запланованих сповіщень

4.2 Користування мобільним застосунком

4.2.1 Реєстрація з використанням email та пароля

Для того аби увійти до сервісу, потрібно пройти етап реєстрації. Для цього потрібно заповнити поля у формі (рисунок 4.4):

- ім’я;
- прізвище;
- електронна пошта;
- пароль.

Підтвердити згоду з політикою конфіденційності, встановивши галочку.

Після заповлення форми, потрібно натиснути кнопку «Зареєструватись» та підтвердити пошту.

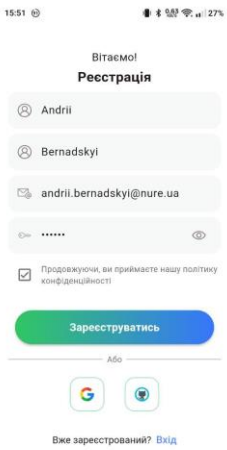


Рисунок 4.4 – Сторінка реєстрації

4.2.2 Вхід до застосунку

Для того, щоб почати використовувати застосунок, потрібно ввести дані, які були використані для реєстрації, після чого натиснути кнопку «Вхід» (рисунок 4.5). Також є можливість увійти через Google, натиснувши на відповідну іконку, після чого вибрати обліковий запис.

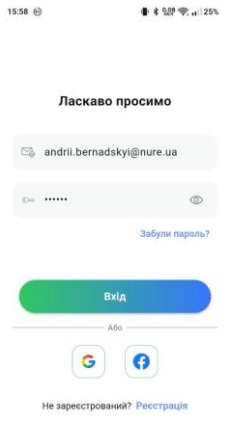


Рисунок 4.5 – Сторінка логінізації

4.2.3 Навігація головною сторінкою

Щоб створити нагадування в нижній правій частині головного екрану розташована кнопка з іконкою «+», при натисканні на неї, відкриється діалогове вікно з варіантами типу нагадування (рисунок 4.6).

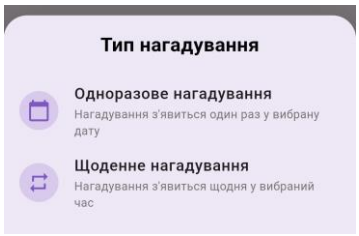


Рисунок 4.6 – Вибір типу нагадування

Ви маєте змогу обрати між одноразовим та щоденним нагадуванням. Після чого відкриється форма, в якій можна вказати дату, час та текст повідомлення, яке з’явиться у вигляді сповіщення (рисунок 4.7).

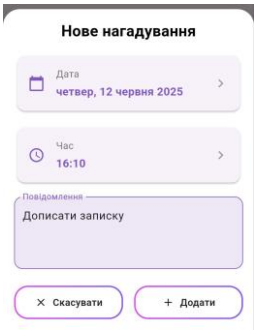


Рисунок 4.7 – Форма створення нагадування

Створене нагадування можемо бачити на головному екрані (рисунок 4.8), поруч з ним відображається: текст нагадування (якщо натиснути на створене нагадування воно розгорнеться), дата та час (або позначка щодня для щоденного нагадування), кнопка редагування (при натисканні якої ви повернетесь до заповнення форми на рисунку 4.7) чи видалення, а також перемикач, щоб тимчасово вимкнути чи увімкнути сповіщення без його видалення.

Рисунок 4.8 – Головний екран застосунку

Коли настане встановлений час, сповіщення прийде, навіть якщо застосунок був згорнутий чи неактивний, таке сповіщення має вигляд push-повідомлення системи Android, у верхній частині буде відображена іконка нагадування (рисунок 4.9). Сповіщення буде містити назву, а також текст. При натисканні на нього автоматично перейдемо до застосунку.

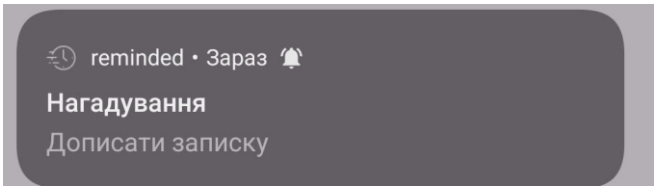


Рисунок 4.9 – Заплановане сповіщення

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було реалізовано повноцінний веб-застосунок та мобільний застосунок з нагадування, який дозволяє користувачам створювати, зберігати та керувати персональними нагадуваннями з можливістю сповіщення у заданий час. Основна увага приділялася зручності інтерфейсу, коректній роботі з базою даних та стабільності роботи системи нотифікацій.

Завдяки інтеграції з Firebase Authentication було реалізовано безпечну систему автентифікації користувача із підтримкою входу за допомогою електронної пошти та пароля, а також через Google-акаунт. Це забезпечує зручний доступ до персоналізованого функціоналу, а також гарантує захищене зберігання даних. Для збереження нагадувань та даних користувача було використано Cloud Firestore - сучасну хмарну базу даних, що дозволяє працювати з інформацією в реальному часі. Завдяки правильно структурованій моделі зберігання було забезпечено чітке розділення нагадувань між різними користувачами та гнучке керування кожним записом.

Особливу роль у застосунку відіграє система локальних сповіщень, реалізована за допомогою пакету «flutter_local_notifications». Вона дозволяє нагадуванням з'являтись навіть у фоновому режимі, що є критично важливим для функціональності такого типу програм. Також було реалізовано підтримку повторюваних щоденних сповіщень.

Крім основного функціоналу, було додано інформаційний блок із відображенням свят на поточну дату, що реалізовано через зовнішнє API та автоматичний переклад назв українською мовою. Цей компонент підвищує інформативність та робить застосунок більш інтерактивним.

Розроблений інтерфейс є інтуїтивно зрозумілим, з адаптивним дизайном, побудованим на принципах сучасного мобільного UX. Структура

проєкту чітко організована: логіка розділена між екранами, моделями, утилітами та сервісами, що забезпечує легке масштабування та підтримку коду в майбутньому.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Cooper, M. Cutting the Cord: The Cell Phone Has Transformed Humanity [Текст] / M. Cooper. – New York: RosettaBooks, 2021. – 242 с.

2. Trestian, R. (ред.). Mobile and Wireless Communications with Practical Use-Case Scenarios [Текст] / під ред. R. Trestian. – Boca Raton: CRC Press, 2021. – 346 с.

3. Burnette, E. Hello, Android: Introducing Google's Mobile Development Platform [Текст] / E. Burnette. – Greenwich: Pragmatic Bookshelf, 2009. – 252 с.

4. Panhale, M. Beginning Hybrid Mobile Application Development [Текст] / M. Panhale. – Berkeley (США): Apress, 2015. – 241 с.

5. Sheppard, D. Beginning Progressive Web App Development [Текст] / D. Sheppard. – Berkeley (США): Apress, 2017. – 281 с.

6. Windmill, E., Rischpater, R. Flutter in Action [Текст] / E. Windmill, R. Rischpater. – Shelter Island, NY: Manning Publications, 2020. – 368 с.

7. Dart. Official Website [Електронний ресурс] / Dart Language Team. – Режим доступу: https://dart.dev/overview (дата звернення: 18.05.2025).

8. Cloud Firestore Data Model Documentation [Електронний ресурс] / Firebase. – Режим доступу: https://firebase.google.com/docs/firestore/data-model (дата звернення: 18.05.2025).