МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЖЕРЕБЦОВ ОЛЕКСАНДР МАКСИМОВИЧ

Допускається до захисту:
завідувач кафедри інформаційних технологій, к.т.н., доцент
_____Т.В. Нескородєва

«___» _____20__р.

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ ПРОДАЖУ МУЗИЧНИХ ІНСТРУМЕНТІВ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Нескородєва Т.В., к.т.н., доцент,
завідувач кафедри ІТ
_____,
(назва кафедри)

Оцінка:____/ ____/ _____
(бали за шкалою СКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2021

АНОТАЦІЯ

Жеребцов О.М. Розробка мобільного додатку для продажу музичних інструментів. Спеціальність 122 «Комп’ютерні науки», освітня програма «Сучасні інформаційні технології та програмування», Донецький національний університет імені Василя Стуса, Вінниця, 2021.

У кваліфікаційній (бакалаврській) роботі досліджена проблема створення сучасних мобільних додатків. Розроблений додаток для продажу музичних інструментів, а саме дошка оголошення. Показано застосування сучасних інструментів з метою забезпечення функціональності, адаптивності і зручності для користувача. Встановлена перспектива та актуальні напрямки розширення функціональності мобільних додатків.

67с., 6 рис., 14 джерел.

Ключові слова: мобільний додаток, дошка оголошень, дизайнерський інструмент Figma, мова програмування Dart, SDK Flutter, Firebase, Android, Android Studio.

ABSTRACT

Zherebtsov O.M. Development of a mobile application for the sale of musical instruments. Specialty 122 "Computer Science", educational program "Modern Information Technologies and Programming", Vasyl’ Stus Donetsk National University, Vinnytsia, 2021.

In the qualification (bachelor's) work the problem of creation of modern mobile applications is investigated. Developed an application for the sale of musical instruments, namely the bulletin board. The use of modern tools to ensure functionality, adaptability and user-friendliness is shown. The perspective and actual directions of expansion of functionality of mobile applications are established.

68p., 6 figures., 14 ref.

Keywords: mobile application, bulletin board, design tool Figma, programming language Dart, SDK Flutter, Firebase, Android, Android Studio.

ЗМІСТ

ВСТУП

Розробка мобільних додатків стає актуальнішою не те що з кожним роком, а і з кожним місяцем. Десятки, навіть сотні нових мобільних додатків виходять на онлайн-майданчиках кожен день. В умовах такої високої конкуренції складно пропанувати щось нове, але креативні розробки у співпраці з хорошим маркетингом можуть принести великі прибутки своїм творцям, і яскраве підтвердження цьому - гучна поява TikTok.

Через пандемію COVID-19, на тлі карантину рекордно зросло використання мобільних додатків. Місяці в ізоляції і перехід бізнесу та навчання в онлайн сильно вплинули на наш спосіб життя та наші звички. Ми частіше купуємо в інтернеті, замовляємо їжу онлайн, більше проводимо часу у смартфонах, витрачаємо більше часу на ігри. Це призвело до зростання попиту на різні мобільні додатки. Щомісячний час, що витрачається на мобільні додатки, зріс на 40 % у 2020 році, порівняно з 2019 роком. Така тенденція зберігається і у 2021 році. Відбулося зростання завантажень додатків для мобільних ігор (43%), для бізнесу (105%) і доставки їжі (73%). 2020-й також був роком відеоконференцій: кількість завантажень Zoom зросла на 2000% порівняно з 2019 роком. Додатки перетворилися в найбільшу споживчу систему на планеті: за прогнозами, у 2021 року загальносвітова виручка від додатків досягне $6,3 трлн. [1].

Отже, в умовах сьогодення, нагальною стає проблема розробки майданчиків для онлайн торгівлі та розміщення оголошень.

Розробка мобільного додатку «G-shop» покликана допомогти вирішити цю проблему, створивши спеціалізоване середовище для розміщення онлайн оголошень про купівлю-продаж музичних інструментів та супутніх товарів і послуг.

В Україні великий успіх має olx.ua із щоденною аудиторією більше двох мільйонів та п’ятнадцятьма мільйонами активних оголошень, також швидко розвивається Prom.ua та низка інших платформ.

Існують додатки, що являють собою дошки оголошень без конкретної тематики. Також існують і спеціалізовані дошки оголошень, адміністрацією яких

вітається додавання тільки тематичних оголошень по одній або кількох галузях промисловості, видах товарів чи послуг.

Електронна дошка оголошень - це додаток, на якому розміщуються оголошення. Спочатку це поняття відносилося виключно до BBS (bulletin board system - електронна дошка оголошень). Однак, у міру поширення Інтернету, з'явилося безліч додатків, цілком аналогічних звичайним побутовим дошкам оголошень або ж рекламним газетам. Вони успадкували назву дошок оголошень (проте абревіатура BBS щодо подібних російсько- та україномовних ресурсів вживається рідко). Їх вміст являє собою набір оголошень комерційного та / або некомерційного характеру і розміщується як на платній, так і на безкоштовній основі, в залежності від конкретного додатку. Багато рекламних компанії, що мають паперові видання і працюють в сфері теле- і радіореклами, створюють і підтримують також власні електронні дошки оголошень.

Слід зауважити, що електронні дошки оголошень мають свої переваги та недоліки:

Переваги

- Економія витрат коштів на розміщення оголошень.
- Менше часу витрачається на пошук потрібного оголошення.
- Економія витрат коштів на придбання товару через оголошення.
- Менше часу витрачається на пошук людини, що придбає товар.

Недоліки

- Ризик не отримати оплату за товар.
- Ризик придбати товар поганої якості.
- Ризик натрапити на оголошення автор якого зі злочинними намірами.

З вищесказаного можна зробити висновок про те, що проблема розробки мобільного додатку є важливим питанням, вирішенню якого і присвячується дана робота.

Актуальність теми дослідження. Найпопулярнішою тенденцією it-ринку сьогодні є мобільні додатки. Сучасний мобільний додаток має декілька вирішальних переваг: швидкість, та надзвичайна зручність для користувача.

Створення такого додатку завдання не з простих і має багато варіацій реалізації, оскільки кількість технологій стрімко зростає. Тому ця тема, сьогодні, є найактуальнішою.

Мета. Розробка мобільного додатку «G-shop» з можливістю перегляду та створення оголошень. Користувач мобільного додатку може створювати, редагувати та видаляти свої оголошення або переглядати оголошення інших користувачів. Додаток повинен бути максимально приємним і зрозумілим для використання.

Завдання дослідження.

- Вивчити та з'ясувати як працють Figma, Dart, Flutter SDK, Git, GitHub, Android Studio, Android Emulator, Firebase, Firebase Authentication, Firebase Firestore, Firebase Storage.
- З'ясувати як розроблюються мобільні додатки для ОС Android версії 7.0, застосовуючи SDK Flutter.

Об’єкт дослідження. Методи проектування та технології створення мобільних додатків для ОС Android.

Предмет дослідження. Популярні інструменти, бібліотеки та фреймворки для створення мобільного додатку для ОС Android.

Досягнення поставленої мети потребує вирішення таких задач дослідження**:**

- Розглянути поняття мобільного додатку.
- Обрати інструменти для створення мобільного додатку.
- Розробити дизайн та прототип мобільного додатку.
- Створити мобільний додаток для ОС Android , що буде задовольняти мету роботи.

Практична значущість роботи полягає в тому, що її рекомендації можуть бути використані при вивчені мови програмування Dart, Flutter SDK, а розроблений мобільний додаток знайти конкретну практичну реалізацію в своїй галузі.

Робота складається з 3 розділів, 9 пунктів, 6 рисунків та списку літератури, що містить 14 джерел, загальний обсяг роботи 67 сторінки.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

1.1. Поняття мобільного додатку

Мобільний додаток (застосунок) – це програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях. Багато мобільних додатків встановлені на самому пристрої або можуть бути завантажені на нього з онлайн магазинів мобільних додатків, таких як App Store, Google Play, Windows Phone Store та інших, безкоштовно або за плату.

Спочатку мобільні додатки використовувалися для швидкої перевірки електронної пошти, але їх високий попит призвів до розширення їх призначень і в інших галузях, таких як ігри для мобільних телефонів, GPS, спілкування, перегляд відео та користування інтернетом.

Загалом додатки переслідують, як мінімум, дві явні цілі: не дати власнику нудьгувати та полегшити побут - іншими словами, некорисні та корисні мобільні додатки.

Необхідність мобільного додатку - недоречно навіть обговорювати, тому що поширення доступних мобільних пристроїв, зручні мультисенсорні екрани, вільний, часом безкоштовний, вихід в інтернет, легкий обмін інформацією - разом, неймовірно збільшило аудиторію потенційних клієнтів будь-якого бізнесу.

1.2. Сучасні мобільні додатки

Велика кількість мобільних пристроїв продаються з уже встановленим набором мобільних додатків, таких як веб браузер, поштовий клієнт, календар (наприклад, Google Calendar), застосунок для придбання та прослуховування музики та інші. Деякі, попередньо встановлені додатки, можуть бути вилучені з мобільного пристрою користувачем, за допомогою звичайного процесу видалення, звільняючи більше місця для зберігання інших (бажаних) додатків.

Додатки, що відразу не встановлені на мобільний пристрій, доступні для завантаження та встановлення через платформи їх розповсюдження. Такі платформи називаються магазинами додатків. Вони почали з'являтися у 2008

році, та зазвичай, управляються компанією власником мобільних операційних систем: Apple App Store, Google Play, Windows Phone Store і Black Berry App World. Проте існують свого роду незалежні магазини додатків, такі як Cydia, Get Jar або F-Droid. Значна кількість мобільних додатків є безкоштовною для встановлення та користування, проте існують і платні. Всі додатки зазвичай завантажуються з платформи одразу на цільовий пристрій, але іноді, вони можуть бути завантаженими на ноутбуки чи комп'ютери. Як правило, 20..30% вартості платних додатків надходить до компанії-дистрибутора (наприклад, iTunes), а решта - до виробника. Тому той самий додаток може мати різну вартість, в залежності від мобільної платформи.

Переваги мобільних додатків:

- Повна взаємодія з користувачами. На відміну від мобільних версій сайтів, у випадку з мобільним додатком є можливість відправляти Push-повідомлення. Наприклад, якщо ви опублікували щось новеньке на сайті, користувачі дізнаються про це тільки коли туди зайдуть. З мобільним додатком просто відправляєте усім активним користувачам, повідомлення, обмеження по символам, як в SMS-компаніях не має;
- Локальні оповіщення працюють приблизно однаково, лишень встановлюються на самому пристрої. Якщо користувач виконав якусь дію в додатку, яке очікує від нього реакції через певний час, додаток нагадає йому про це;
- Так само з’являються більш широкі можливості зворотного зв’язку - користувачі можуть залишити повідомлення, свої відгуки в магазинах додатків, персонально через додаток, соцмережах - і вони автоматично транслюватимуться на усі ресурси;
- Якісніший інтерфейс. Всі елементи управління: кнопки, текстові поля, посилання повинні бути зручними для натискання пальця на мобільному девайсі, а не для курсора миші. Екрани пристроїв бувають різними і за розміром, і за пікселями. Екран телефону може мати в роздільній здатності практично, як ноутбук, але при цьому бути невеликим у фізичних розмірах. Отож-бо , невідомо

як виглядатиме сайт на тому чи іншому пристрої, чи зручно і , навіть, чи реально ним взагалі користуватися. Мобільні операційні системи для таких ситуацій мають свою спеціально навчену логіку, щоб відрізняти пристрій із яким працюють, боротися з ефектами потрапляння пальців в маленькі кнопочки. Але працює ця логіка по різному на різних платформах і пристроях;

- Якість інтерфейсу позначається і в навігації. Кожна мобільна операційна система має свою логіку переходу між робочими екранами в додатках. В Android це кнопка Back (Назад), в IOS – провід пальцем від лівого краю екрана. Користувач кожної операційної системи звик до тієї ж поведінки в кожному додатку. Навігація ж на кожному сайті зроблена по-своєму, і зайшовши на черговий з них потрібно кожного разу шукати очима кнопки «Ок», «Назад», «Скасування» т.д., яких може зовсім й не бути;
- Високий рівень персоналізації. Мобільні телефони знають про свого власника дуже багато, і використання цієї інформації для збільшення рівня сервісу є однією з ключових причин бурхливого зростання мобільних додатків. У мобільного додатку завжди є можливість запам’ятовувати всі дані користувача і змінювати інтерфейс залежності від його потреб. Якщо користувач ввів якісь дані (наприклад, свою домашню адресу в додатку доставки), то йому не потрібно буде вводити цю адресу наново. Навіть на різних пристроях при включеній синхронізації користувач буде бачити додатки з заповненими персональними даними. Або IOS запам’ятовує основні часові інтервали поведінки власника пристрою, і у цій черговості піктограми мобільних додатків на екрані;
- Цей же пункт відноситься і до пристрою – в залежності від типу програми, дозволів, підтягуються дані з камери, акселерометра, компаса, барометра і т.п.;
- Локалізація - якщо додаток, наприклад, має відобразити список ресторанів, то він це скоріше за все зробить враховуючи відстань від користувача, знаючи його поточне місце розташування. Логіка роботи мобільного сайту завжди простіша, вона може не враховувати безліч даних, які можуть надавати мобільні пристрої;

•Робота в офлайні. Інтернет є усюди, але не завжди. Навіть при наявності мобільного інтернет-з'єднання його якість в середньому гірша ніж якість домашніх й офісних інтернет-ліній. Якщо поставлена ціль, щоб користувачі не втрачали зв'язок з продуктом відразу, як тільки обірвався інтернет-зв'язок, розробка мобільного додатку – єдине можливе рішення;

•Розробка компанією мобільного додатку – це також ознака престижу. Ознака того, що компанія дбає про своїх клієнтів, даючи їм свободу вибору своїх ресурсів та вищий рівень комфорту, особливо ж якщо потрібно побудувати тривалий, якісний і продуктивніший зв'язок з клієнтами. Наприклад, додаток – де клієнта обслуговує особистий консультант. Плюс, власний мобільний додаток – додатковий рекламний канал, або ж навпаки причина переконувати у його перевагах відсутністю настирливої реклами.

Недоліки мобільних додатків:

•Якщо розмістити QR-коди або якорі на сайт і на додаток, то переходів на сайт завжди буде більше ніж установок мобільного додатку. Частина користувачів після переходу за посиланням на додаток в магазин, все-таки його не встановлює;

•Кросплатформеність -мобільний додаток не може бути доступний на всіх пристроях на відміну від сайту який доступний на усіх пристроях;

•Миттєві оновлення - для оновлення програми в магазинах завжди потрібен певний час на їх перевірку на відміну від сайту. Оновлений сайт доступний користувачам практично відразу;

•Вартість розробки й підтримки мобільного додатку, як правило, завжди вища ніж відповідні цифри щодо сайтів;

•Мобільний додаток просувати, однозначно, дорожче аніж сайт. Ціна одного користувача безпосередньо залежить від порогу його входження, чим він вищий - тим вища ціна його залучення. Для забезпечення заходу користувача на сайт, нехай встановлюється певна вартість кліку по рекламному блоці, після якого відразу отримується відвідувач. У випадку з мобільним додатком треба зробити те ж саме, тільки, не факт що після переходу в магазин додатків,

користувач його встановить. Як наслідок - ціна установки завжди значно вища ціни кліка.

Додатки також можуть бути встановлені власноруч користувачем, як наприклад, запуск пакету програм для Android на пристроях з операційною системою Android.

Спочатку мобільні додатки пропонувалися як інструменти для контролю та оперування загальних потоків інформації, включаючи електронну пошту, календар, контакти, фондовий ринок та інформацію про погоду. Тим не менше, попит та наявність інструментів для розробників зумовили швидке поширення додатків і для інших категорій електронних пристроїв, які функціонують за допомогою настільних прикладних програм. Як і у випадку з іншим програмним забезпеченням, вибухова кількість та різноманіття мобільних додатків призвела до виникнення великої кількості пізнавальних ресурсів про них - з відгуками, рекомендаціями та оглядами, а саме: блоги, журнали та спеціальні служби виявлення додатків в Інтернеті.

Користування мобільними додатками серед користувачів мобільних пристроїв стає все більш і більш популярнішим. Згідно даних дослідження компанії App Annie за 2017 рік кількість завантажень додатків зросла на 60%, споживчі витрати зросли більш ніж удвічі, а час, витрачений на додаток кожним користувачем складає близько 43 дні на рік. Сумарна кількість завантажень додатків у 2019 році зросла до 115 млрд: 31 млрд - в App Store і 84 млрд - в Google Play. Дослідники виявили, що використання мобільних додатків чітко корелюється з їх наповненням та залежить від часу доби і локації розміщення користувача.

Висновок до розділу 1.

В цьому розділі було розглянуто загальне поняття мобільних додатків та сучасних мобільних додатків. Визначено необхідність використання мобільних додатків в сучасних реаліях, висвітлено їх переваги та недоліки. А також перспективу розвитку ринку мобільних додатків.

РОЗДІЛ 2
ПОСТАНОВКА ЗАДАЧІ ТА ІНСТРУМЕНТИ

2.1. Постановка задачі.

Розробити макет та прототип мобільного додатку.

Розробити мобільний додаток для ОС Android версії 7.0, застосовуючи SDK - Flutter.

Етапи створення мобільних додатків:

- визначення аудиторії;
- вибір мобільних платформ;
- розробка концепції;
- формування технічного завдання;
- дизайн інтерфейсу;
- розробка;
- тестування;
- відлагодження;
- реєстрація та реліз у магазинах;
- взаємодія з користувачами;
- періодичні оновлення;

Завданням данної роботи є розробка мобільного додатку для породажу музичних інструментів «G-shop». Отже, мобільний додаток буде являти собою онлайн дошку оголошень, що надає користувачеві можливість переглядати оголошення і створювати їх.

Електронна дошка оголошень функціонально подібна звичайній: це додаток, де кожен бажаючий може вивісити своє оголошення, а всі відвідувачі додатку - прочитати його. Електронна дошка оголошень, як правило, поділена на кілька тематичних розділів, відповідно до змісту оголошень.

Більшість дошок оголошень - безкоштовні. Для розміщення свого оголошення користувачеві потрібно лише ввести в спеціальній формі його тему, своє ім'я / псевдонім або назву організації, а також координати: адреса

електронної пошти, поштова адреса, телефон, URL свого сайту і т. П. (Набір даних залежить від конкретного ресурсу). Як правило, відображаються тільки імена авторів і теми оголошень, а для перегляду повного тексту оголошення користувач повинен клацнути по посиланню, що веде до нього. У деяких дошках оголошення можуть подавати тільки зареєстровані користувачі, в деяких - всі. Зараз в інтернеті існує тисячі і навіть десятки тисяч дошок оголошень. Зазвичай кожна з них присвячується якомусь одного виду оголошень. Існують національні дошки оголошень, призначених для жителів конкретної місцевості.

Електронні дошки оголошень бувають двох видів: модеровані (ті, у яких є так званий модератор - людина, яка контролює роботу цієї дошки) і немодеровані - працюють автоматично.

Більшість дошок оголошень в інтернеті припускають розміщення оголошень про продаж і купівлю товарів і послуг. Але є і такі сервіси, які пропонують розміщувати оголошення про передачу різних речей в дар, а також про пошук нових господарів для домашніх тварин, які передаються в добрі руки безкоштовно. Безкоштовні оголошення - це найкращий спосіб заявити про себе, свої товари і послуги в мережі Інтернет. На сторінках дошок оголошень представлені тисячі безкоштовних оголошень про нерухомість, авто, послуги, пошук роботи, знайомства, купівлю та продаж обладнання, комп'ютерів, засобів зв'язку,тощо. Дошки оголошень дозволяють розмістити оголошення з фотографією і без додаткової реєстрації. Ви можете розмістити не тільки безкоштовне оголошення, але і оголошення на комерційній основі - для максимальної ефективності.

Таким чином, основний функціонал для користувачів:

- Реєстрація користувачів у додатку (пошта та пароль).
- Перегляд користувачем усіх оголошень на головній сторінці.
- Перегляд своїх оголошень.
- Перегляд збережених оголошень.
- Користувач може здійснювати навігацію додатком за допомогою нижньої навігаційної панелі.

- Користувач може переглядати сторінки з оголошеннями на якій будуть відображені картинки, опис та ціна.
- Анімація контенту, коли буде здійснений перехід з головної сторінки на сторінку оголошення.
- При натисканні на кнопку «показати контакт», користувач переходить на сторінку автора оголошення, де є можливість перегляду інформації про користувача, та є можливість зателефонувати йому (звичайний телефонний дзвінок).
- Користувач може створювати, редагувати та видаляти свої оголошення.
- Користувач може редагувати або видалити свій профіль.
- Користувач має можливість змінити кольорову тему додатку.
- Валідація текстових полів.
- Перевірка підключення до інтернету.
- Адаптивність під планшет.
- Адаптивність під горизонтальне положення девайса.
- Пошук у всіх розділах.

2.2. Огляд дизайнерського інструменту Figma.

Існує багато діджітал-інструментів, які доступні дизайнерам, кожен – з власними можливостями та результатом. Їх розвиток триває кожен день, даючи можливість продемонструвати функціональність інтерфейсу, інтерактивність, анімацію та загальне юзабіліті. Тож наразі пошук платформи для себе стає дуже захоплюючим.

Figma - векторний онлайн-сервіс розробки інтерфейсів та прототипування з можливістю організації спільної роботи, що розробляється однойменною компанією. Працює у двох форматах: у браузері та як клієнтський додаток на десктопі користувача. Зберігає онлайн-версії файлів, з якими працював користувач.[2]

Даний редактор підходить як для створення простих прототипів і дизайн-систем, так і складних проектів (мобільні додатки, веб-сайти). У 2018 році

платформа стала одним із тих інструментів для розробників і дизайнерів, що найбільш швидко розвиваються.

З переваг - автоматичне налаштування дизайну до будь-якого екрану, адаптивні дизайни, система «layoutgrids» для впорядкованих елементів, відступи яких зберігаються автоматично, без додаткових маніпуляцій з композицією.

Єдиний недолік цього інструменту це неможливість працювати без інтернет-з'єднання, але в 2020 році, це не є проблемою.

2.3. Огляд мови програмування Dart.

Сполучною ланкою між розробником і додатком є мова програмування. Існує багато мов на яких пишуться мобільні додатки. Сьогодні неможливо працювати в сфері розробки і знати тільки одну мову програмування. Щороку з'являються нові мови ,всі вони створюються для досягнення максимальної ефективності певних задач. А основні затребувані розширюються і прогресують**.**

Dart - мова програмування, створена Google. Dart позиціонується в якості заміни / альтернативи JavaScript. Один з розробників мови Марк Міллер (Mark S. Miller) написав, що JavaScript «має фундаментальні вади» («Javascript has fundamental flaws ...»), які неможливо виправити. Тому і був створений Dart.[3]

Перша загальнодоступна інформація про цю мову програмування з'явилася 12 вересня 2011 року на конференції розробників Goto. 10 жовтня 2011 року була проведена офіційна презентація мови GoogleDart.

Завдання, поставлені перед розробниками мови:

- Створити структуровану і в той же час гнучку мову для веб-програмування.
- Зробити мову схожою на існуючі для спрощення навчання.
- Висока продуктивність одержуваних програм як в браузерах, так і в інших середовищах, починаючи від смартфонів і закінчуючи серверами.
- Спочатку було запропоновано два способи виконання Dart-програм: з використанням віртуальної машини (в підтримуючих мову браузерах) або з проміжної трансляцією в javascript (універсальний).

15 листопада 2013 року Google випустили першу стабільну версію своєї мови програмування - Dart SDK 1.0.

4 липня 2014 року ECMA схвалили першу редакцію стандарту мови, стандарт отримав назву ECMA-408.

2.4. Інструменти для розробки мобільного додатку.

Flutter - SDK з відкритим вихідним кодом для створення мобільних додатків від компанії Google. Він використовується для розробки додатків під Android та iOS, WEB-сайтів, через деякий час з'явиться можливість написання Desktop-додатків, а також це поки єдиний спосіб розробки додатків під Google Fuchsia. [4]

Перша версія Flutter носила назву «Sky» і працювала тільки під Android. Вона була представлена в 2015 році на саміті розробників Dart із заявленою можливістю рендеринга 120 фреймів в секунду. 4 грудня 2018 року під час FlutterLive було оголошено про випуск першої стабільної версії 1.0.

Архітектура:

- платформа Dart
- рушій Flutter
- бібліотека Foundation
- набори віджетів

Додатки Flutter пишуться на мові Dart.

В Android, а також під Windows, macOS і Linux за допомогою Flutter Desktop Embedding, Flutter працює у віртуальній машині Dart з JIT-компілятором. Через обмеження на динамічне виконання коду в AppStore, під iOS Flutter використовує AOT-компіляцію.

Одне з головних переваг платформи Dart - «гаряче перезавантаження», коли зміна вихідного коду застосовується відразу в працюючому додатку без необхідності його перезапуску.

Движок Flutter написаний переважно на C++, він підтримує низькорівневий рендеринг за допомогою графічної бібліотеки GoogleSkia. А також має можливість взаємодіяти з платформозалежними SDK під Android та iOS.

Бібліотека Foundation написана на мові Dart, містить основні класи та методи для створення додатків Flutter і взаємодії з движком Flutter.

Дизайн користувальницького інтерфейсу додатків Flutter зазвичай включає в себе використання і / або створення різних віджетів. Віджет Flutter є незмінний опис будь-якої частини призначеного для користувача інтерфейсу. Всі графічні об'єкти, включаючи текст, форми і анімацію, створюються за допомогою віджетів. Комбінуванням простих віджетів створюються комплексні віджети.

Однак створювати додатки Flutter можна і без віджетів, безпосередньо викликаючи методи бібліотеки Foundation для роботи з канвою.

Фреймворк Flutter складається з двох наборів віджетів, які відповідають конкретним описам дизайну: MaterialDesign від Google і Cupertino для імітації дизайну додатків в AppleiOS.

Stacked - архітектура, розроблена і підтримується спільнотою Filled Stacks. Ця архітектура спочатку була версією MVVM. З тих пір команда розробників додатків Filled Stacks створила 6 виробничих програм з різними вимогами. Цей досвід разом з незліченними запитами на поліпшення і загальні функції - ось що послужило поштовхом до створення цього пакета архітектури. Він спрямований на забезпечення загальних функцій, що спрощують розробку додатків, а також принципів коду, які можна використовувати під час розробки, щоб ваш код залишався підтримуваним [5, 6].

Архітектура дуже проста. Вона складається з 3 основних частин, все інше залежить від стилю реалізації. Реалізація:

View - це користувальницький інтерфейс для користувача. Поодинокі віджети також кваліфікуються як views (для узгодженості термінології) подання, в даному випадку, не є "page", це лише представлення інтерфейсу.

ViewModel - керує станом подання, бізнес-логікою та будь-якою іншою логікою, як це вимагається від взаємодії користувача. Він робить це, користуючись services.

Services - обгортка єдиного набору функцій. Це зазвичай використовується для використання таких речей, як показ діалогового вікна, залиття функціональних можливостей бази даних, інтеграція API тощо.

Принципи, яких слід дотримуватися під час розробки:

- View ніколи не повинні використовувати services.
- View в собі не повинні містити логіки (переважно). Якщо логіка походить лише з елементів інтерфейсу, тоді ми виконуємо найменшу кількість необхідної логіки, а решту передаємо з ViewModel.
- View повинні відображати стан тільки у своєму ViewModel.
- ViewModels для віджетів, що представляють перегляди сторінок, прив'язані лише до одного View.
- ViewModels можуть бути використані повторно, якщо користувальницький інтерфейс вимагає однакових функціональних можливостей.
- ViewModels не повинні знати про інші ViewModels.

Stacked надає класи та функціональні можливості, щоб спростити реалізацію базової архітектури, для чого і була створена ця бібліотека.

Stacked дозволяє використовувати:

- BaseViewModel
- ReactiveViewModel
- StreamViewModel
- FutureViewModel
- MultipleStreamViewModel
- MultipleFutureViewModel
- IndexTrackingViewModel

GetIt - це простий службовий локатор для проектів Dart та Flutter з деякими додатковими функціями. Його можна використовувати замість InheritedWidget або Provider для доступу до об'єктів, наприклад з інтерфейсу користувача.[6]

У міру зростання додатка, в якийсь момент потрібно буде застосувати логіку додатка до класів, які відокремлені від ваших віджетів. Захист віджетів від прямих залежностей робить код краще організованим та простішим для тестування та обслуговування. Але тепер потрібен спосіб отримати доступ до цих об’єктів з коду інтерфейсу. Раніше єдиним способом зробити це було використання InheritedWidgets. Існує спосіб використовувати їх, обернувши їх у StatefulWidget але код стає досить громіздким і має проблеми з постійною роботою. Також не вистачає можливості легко переключити реалізацію на mocked версію без зміни інтерфейсу користувача і ще той факт, що потрібен BuildContext для доступу до об’єктів, робив його недоступним із шару Business.

Доступ до об’єкта з будь-якого місця програми може бути здійснений іншими способами, але:

- Якщо використовувати синглтон, то не можна легко переключити реалізацію на фіктивну версію в тестах
- iOC-контейнери для dependency Injections пропонують подібну функціональність, але з дуже великими витратами часу під час запуску та меншу читабельність, оскільки можна знати, звідки береться введений об’єкт. Більшість бібліотек iOC покладаються на відображення, і їх не можна перенести на Flutter.

Оскільки Dart підтримує глобальні (або euphemistic ambient) змінні, часто призначають свій екземпляр GetIt глобальній змінній, щоб максимально спростити доступ до нього. Хоча підхід із глобальною змінною працює добре, він має свої обмеження, якщо потрібно використовувати GetIt у декількох пакетах. Тому GetIt сам по собі є одностороннім і за замовчуванням доступ до екземпляра GetIt - це виклик.

Завдяки цьому будь-який виклик екземпляра в будь-якому пакеті проекту отримає той самий екземпляр GetIt. Рекомендують просто призначити екземпляр глобальній змінній у проекті, оскільки це зручніше і ніяк не зашкодить (також це дозволяє вказати локатору служб власне ім'я).

Перш ніж можна буде отримати доступ до своїх об'єктів, доведеться зареєструвати їх у GetIt, як правило, безпосередньо у коді запуску.

Як варіант GetIt - це викличний клас, залежно від імені.

GetIt пропонує різні способи реєстрації об'єктів, які впливають на таймлайн цих об'єктів.

Потрібно передати фекторі функцію, яка повертає новий екземпляр реалізації T. Кожного разу, коли викликається get<T>(), повертається новий екземпляр.

Потрібно передати екземпляр T або похідний клас T, який завжди отримується під час виклику, щоб отримати <T> ().

Оскільки створення цього екземпляра може зайняти багато часу під час запуску програми, ви можете перенести створення в той час, коли об’єкт вперше запитується за допомогою: voidregisterLazySingleton<T>(FactoryFunc<T>func).

Потрібно передати фекторіфункцію, яка повертає екземпляр реалізації T. Тільки перший раз, коли викликається get<T>(), ця фекторіфункція буде викликана для створення нового екземпляра. Після цього отримується той самий екземпляр.

Перезапис реєстрацій. Якщо буде спроба зареєструвати тип більше одного разу, то нічого не вийде у режимі дебаг, оскільки зазвичай це не потрібно, і, можливо, помилка. Якщо дійсно потрібно переписати реєстрацію, то можна, встановити властивість: allowReassignment == true.

Тестування, чи вже зареєстровано синглтон. Є можливість перевірити, чи зареєстровано певний тип або екземпляру GetIt за допомогою: boolisRegistered<T>({Objectinstance, StringinstanceName}).

Якщо потрібно, також можна скасувати реєстрацію зареєстрованих синглонів і фекторіі передати додаткову функцію для очищення. У деяких випадках може не знадобитися скасування реєстрації LazySingleton, а замість цього скинути його екземпляр, щоб він отримав нещодавно створений при наступному доступі до нього.

Переваги GetIt:

- Надзвичайно швидкий
- Легко вчиться та легко користуватися

- Не захаращує дерево вашого інтерфейсу спеціальними віджетами для доступу до ваших даних, як це робить постачальник або Redux.

GetIt - це не state management. Це локатор для об’єктів, тому потрібен інший спосіб повідомляти інтерфейс про зміни, такі як Streams або Value Notifiers.

Injectable - це зручний генератор коду для GetIt. Подібні технології: Angular DI, Guice DI та inject.dart.[8]

Використання:

- Встановлення
- Налаштування
- Реєстрація фекторі
- Реєстрація сінглтонів
- Реєстрація асинхронних інджекшин
- Передача параметрів фекторі
- Прив’язка абстрактних класів до реалізацій
- Реєстрація в різних середовищах
- Використання названих фекторі-функцій статичного створення
- Реєстрація сторонніх типів
- Автоматична реєстрація

Все, що потрібно зробити - це анотувати свої інджекшн класи @injectable і почати генерацію. Інджекшн генерує необхідні функції реєстру. Якщо потрібно зробити екземпляр асинхронним, знадобиться статичний метод ініціалізації, оскільки конструктори не можуть бути асинхронними.

Dio - потужний клієнт Http для Dart, який підтримує інтерсептори, глобальну конфігурацію, FormData, скасування запиту, завантаження файлів, час очікування тощо [9].

Виконання HTTP-запитів в мобільному додатку - одна з поширених завдань. Завдяки HTTP-запитами додаток може зв'язуватися з серверною частиною і вибирати дані.

Фреймворк Flutter пропонує пакет http, який відмінно працює, коли нам потрібно робити базові речі. Коли потрібно зробити щось більш складне, потрібно щось більше. І це можна зробити за допомогою Dio. Dio - це бібліотека HTTP-з'єднань, яка має додаткові функції, такі як перехоплювачі, які будуть корисні в багатьох задачах (додавання аутентифікації токена для кожного запиту, ведення журналу запитів). Dio API досить простий, бібліотека надійна і використовується у багатьох проектах на Flutter.

Stacked Themes - ця бібліотека являє собою набір віджетів та класів, які допомагають керувати кількома темами або функцією темної / світлової теми в додатку Flutter. Цей пакет містить кілька базових класів, що полегшують управління темами при створенні вашого додатка з управління темами. Stacked Themes надає вам основну функціональність заміни даних ThemeData, з наданих вашому додатку, до яких можна отримати доступ за допомогою Theme.of(context). На додаток до цього, він також надає вам допоміжні функції, за допомогою яких ви можете змінити будь який колір [10].

Бібліотека використовує Theme Manager для управління всіма функціоналами теми. Для початку потрібно викликати функцію ініціалізації Theme Manager перед запуском програми. Потрібно змінити основну функцію на фьючер функцію, і зробити її асинхронною щоб можна було чекати ініціалізацію виклику статичної функції на Theme Manager. Ми починаємо з обгортання нашого Material App або будь-якого іншого віджета Theme Builder. Функція конструктора повертає контекст, regular Theme, dark Theme та theme Mode. Це відновить вашу програму та передасть нову тему через контекст, який ви зможете використовувати у своєму додатку.

Theme Builder має властивість під назвою themes, де можена надати список Theme Data. Зазвичай це лише 2 або 3, світла, темна, нейтральна, але ми зберігаємо їх у списку, щоб мати їх скільки завгодно. Щоб підтримувати «чистоту», повернеться список тем із функції, створеної в іншому файлі (theme_setup). Це досить проста функція, але вона може почати виглядати досить великою, враховуючи, скільки властивостей теми будете використовуватися у

темах. Саме тому рекомендують все це зберігати в окремому файлі. Спосіб обміну темами відбувається шляхом встановлення індексу теми, яку ми хочемо використовувати.

Firebase - американська компанія, постачальник хмарних послуг, заснована в 2011 році Ендрю Лі і Джеймсом Темпліном, і поглинена в 2014 році корпорацією Google.[11]

Пройшла два раунди інвестицій: в травні 2012 року отримала $ 1,4 млн від Flybridge Capital Partners, Greylock Partners, NEA, в червні 2013 року залучила $ 5,6 мільйона від Union Square Ventures і Flybridge Capital Partners.

Основний сервіс - хмарна СУБД класу NoSQL, що дозволяє розробникам додатків зберігати і синхронізувати дані між декількома клієнтами. Підтримані особливості інтеграції з додатками під операційні системи Android і iOS, реалізовано API для додатків на JavaScript, Java, Objective-C і Node.js, також можливо працювати безпосередньо з базою даних в стилі REST з ряду JavaScript-фреймворків, включаючи AngularJS, ReactJS, Vue.js, Ember.js і Backbone.js. Передбачено API для шифрування даних.

Серед інших послуг, що надавалися компанією - запущений 13 травня 2014 року хостинг для зберігання статичних файлів (таких як CSS, HTML, JavaScript), що забезпечує доставку через CDN і сервіс аутентифікації клієнта з використанням коду тільки на стороні клієнта з підтримкою входу через Facebook, GitHub, Twitter і Google (FirebaseSimpleLogin).

Firebase надає такі функції, як аналітика, бази даних, обмін повідомленнями та звіти про аварійне, тому можна швидко працювати і зосередитися на своїх користувачів.

Firebase побудований на інфраструктурі Google і масштабується автоматично навіть для великих програм.

Продукти Firebase добре працюють індивідуально і разом, вони чудово обмінюються даними.

Firebase зберігає та синхронізує дані, додані у глобальному масштабі, швидко і безпечно доставляє ресурси додатків та має просту і безпечну аутентифікацію користувачів.

GitHub - найбільший веб-сервіс для хостингу IT-проектів і їх спільної розробки. Веб-сервіс заснований на системі контролю версій Git і розроблений на Rubyon Rails і Erlang компанією GitHub, Inc (раніше Logical Awesome). Сервіс безкоштовний для проектів з відкритим вихідним кодом і (з 2019 року) невеликих приватних проектів, надаючи їм всі можливості (включаючи SSL), а для великих корпоративних проектів пропонуються різні платні тарифні плани. Творці сайту називають GitHub «соціальною мережею для розробників». Крім розміщення коду, учасники можуть спілкуватися, коментувати правки один одного, а також стежити за новинами знайомих. За допомогою широких можливостей Git програмісти можуть об'єднувати свої репозиторії - GitHub пропонує зручний інтерфейс для цього і може відображати внесок кожного учасника у вигляді дерева. Для проектів є особисті сторінки, невеликі Вікі і система стеження за вадами. Прямо на сайті можна переглянути файли проектів з підсвічуванням синтаксису для більшості мов програмування. Можна створювати приватні репозиторії, які будуть видні тільки вам і вибраним вами людям. Раніше можливість створювати приватні репозиторії була платною. Є можливість прямого додавання нових файлів в свій репозиторій через веб-інтерфейс сервісу. Код проектів можна не тільки скопіювати через Git, а й скачати у вигляді звичайних архівів з сайту. Крім Git, сервіс підтримує отримання і редагування коду через SVN і Mercurial. На сайті є pastebin-сервіс gist.github.com для швидкої публікації фрагментів коду.[12, 13]

Android Studio - інтегроване середовище розробки (IDE) для роботи з платформою Android, анонсована 16 травня 2013 року на конференції Google I/O.[14]

Дана IDE перебувала у вільному доступі починаючи з версії 0.1, опублікованій в травні 2013, а потім перейшла в стадію бета-тестування, починаючи з версії 0.8, яка була випущена в червні 2014 року. Перша стабільна

версія 1.0 була випущена в грудні 2014 року, тоді ж припинилася підтримка плагіна Android Development Tools (ADT) для Eclipse.

Android Studio, заснована на програмному забезпеченні IntelliJ IDEA від компанії JetBrains, - офіційний засіб розробки Android додатків. Дане середовище розробки доступне для Windows, macOS і Linux. 17 травня 2017, на щорічній конференції Google I/O, Google анонсував підтримку мови Kotlin, що використовувалась в AndroidStudio, як офіційної мови програмування для платформи Android на додаток до Java і C ++.

Нові функції з'являються з кожною новою версією AndroidStudio. На даний момент доступні наступні функції:

- Розширений редактор макетів: WYSIWYG, здатність працювати з UI компонентами за допомогою Drag-and-Drop, функція попереднього перегляду макета на декількох конфігураціях екрану.
- Збірка додатків, заснована на Gradle.
- Різні види збірок і генерація кількох .apk файлів
- Рефакторинг коду
- Статичний аналізатор коду (Lint), що дозволяє знаходити проблеми продуктивності, несумісності версій і інше.
- Вбудований ProGuard і утиліта для підписування додатків.
- Шаблони основних макетів і компонентів Android.
- Підтримка розробки додатків для Android Wear і Android TV.
- Вбудована підтримка Google Cloud Platform, яка включає в себе інтеграцію з сервісами Google Cloud Messaging і AppEngine.
- Android Studio 2.1 підтримує Android N Preview SDK, а це значить, що розробники зможуть почати роботу зі створення програми для нової програмної платформи.
- Нова версія AndroidStudio 2.1 здатна працювати з оновленим компілятором Jack, а також отримала покращену підтримку Java 8 і вдосконалену функцію InstantRun.

- Починаючи з Platform-tools 23.1.0 для Linux виключно 64-розрядна.
- В AndroidStudio 3.0 по стандарту включені інструменти мови Kotlin засновані на JetBrains IDE.

Висновок до розділу 2

Вірний вибір потрібних інструментів, особливо якщо проект створюється в команді розробників, є ключовим моментом, який значно впливає на створення та якість майбутнього проекту.

РОЗДІЛ 3
РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ«G-shop»

3.1. Розробка макета і прототипу мобільного додатку.

UI (User Interface Design) або призначений для користувача інтерфейс - покликаний допомогти організувати взаємодію користувача з пристроєм або програмою. Це те, з чим користувач стикається при погляді на пристрій або екран: розташування графічних елементів інтерфейсу, кнопок і навігації. Від того, наскільки призначений для користувача інтерфейс зручний, залежить бажання людини скористатися сайтом або додатком. Лояльність користувачів багато в чому залежить від зручності і привабливості інтерфейсу. Яскравий дизайн привертає людей і змушує їх зайти в додаток, програму або на сайт. Тому важливо щоб UI дизайн був інтуїтивно зрозумілий. Таким чином, UI-дизайн направлений на формування необхідного емоційного сприйняття продукту.

Етапи розробки UI дизайну:

- Постановка завдання: постановка цілей, для яких буде використовуватися інтерфейс; аналіз потреби користувачів; вибір платформи, на якій необхідно реалізувати проект.
- Розробка ескізів інтерфейсу.
- Створення прототипу, для оцінки зручності інтерфейсу.
- Аналіз отриманих даних і підбір оптимальних варіантів.

При створення UI дизайну потрібно дотримуватися певних правил:

- Дизайн елементів додатків повинен бути максимально простим і зрозумілим.
- Користувач не повинен здійснювати складних дій або довгих ланцюжків переходів.
- Бажано використовувати звичні елементи управління і зрозумілі візуальні образи.
- Логічно пов'язані елементи необхідно об'єднувати в меню і форми.

- Найважливіші елементи призначеного для користувача інтерфейсу повинні йти першими.
- Інтерфейс повинен бути оформлений в єдиному стилі.

Розробляючи UI дизайн мобільного додатку необхідно враховувати ефект «естетика-юзабіліті» (aesthetic-usability effect). Перебуваючи під його впливом людині здається, що інтерфейс додатків, який виглядає краще, є більш зручним. Тому дизайн користувальницького інтерфейсу повинен гармонійно поєднувати красу і зручність.

Залежно від мети проекту та призначення інтерфейсу дизайнер підбирає кольорову схему і оформлення. Необхідно враховувати психологію кольору і те, як вона впливає на людей. При розробці UI інтерфейсу колір можна використовувати, щоб підкреслити якість програми. Червоний асоціюється з силою і пристрастю, зелений - з природою і достатком, синій - зі спокоєм, фіолетовий - з творчістю і мудрістю. Необхідно ретельно опрацьовувати деталі проекту ще на ранніх етапах і слідувати головним принципам розробки мобільного інтерфейсу. Серед останніх трендів дизайну - розробка додатків, які працювали б і на Android і на iOS. Однак UI дизайн краще розробляти окремо для кожного мобільного додатка. Так можна уникнути помилок і неправильного функціонування.

Перед UI інтерфейсом стоять такі завдання : забезпечувати взаємодію користувача з додатком, приносити емоційне задоволення користувачеві в процесі роботи, виглядати привабливо.

UI інтерфейс включає в себе такі елементи:

- інформаційну архітектуру
- проектування взаємодії
- графічний дизайн.

Складові UI:

Візуальний дизайн, або VD важливий для якісного відображення кольорів, шрифтів та інших елементів сайту, оскільки саме це створює естетику сайту. Хороший користувальницький інтерфейс може існувати тільки в сукупності

змісту сторінки і її функціоналу. Крім того, візуальний дизайн направлений на те, щоб привернути користувача і вибудувати його довіру до продукту або послуги.

•Дизайн система - проектування призначеного для користувача інтерфейсу через системи окремих елементів інтерфейсу. Вони потрібні для розробки багаторазових шаблонів.

•Прототип являє собою діючий приклад розробленого призначеного для користувача інтерфейсу. Імітування роботи дає уявлення про відгуки елементів інтерфейсу до їх безпосередньої розробки.

•Зміст. Гаджети привчили користувачів працювати в режимі багатозадачності, тому при проектуванні користувальницького інтерфейсу важливо швидко і коректно продемонструвати основний текст і функціонал додатка, а все вторинне перенести на задній план або прибрати.

•Зворотня комунікація - це використання різного оформлення або анімації відгуку програми на дію користувача. Людині це дає розуміння, що все під контролем і функціонує коректно.

•Робота за допомогою однієї руки. Хороший дизайн інтерфейсу повинен бути адаптований під управління однією рукою. Найлегше керувати програмою, коли необхідні елементи навігації розташовані в нижньому лівому кутку. Також зручний дизайн інтерфейсу передбачає наявність великих кнопок, відступів і яскравого оформлення важливих елементів для легкого взаємодії з ними.

•Швидке введення. Форми для заповнення повинні: вимагати від користувача мінімального кількості інформації; пропонувати варіанти відповідей. Наприклад, при необхідності дізнатися вік користувача краще використовувати список, що випадає, а, дізнаватися адресу через синхронізацію з геолокацією. Загальна рекомендація - мінімальний ручне введення.

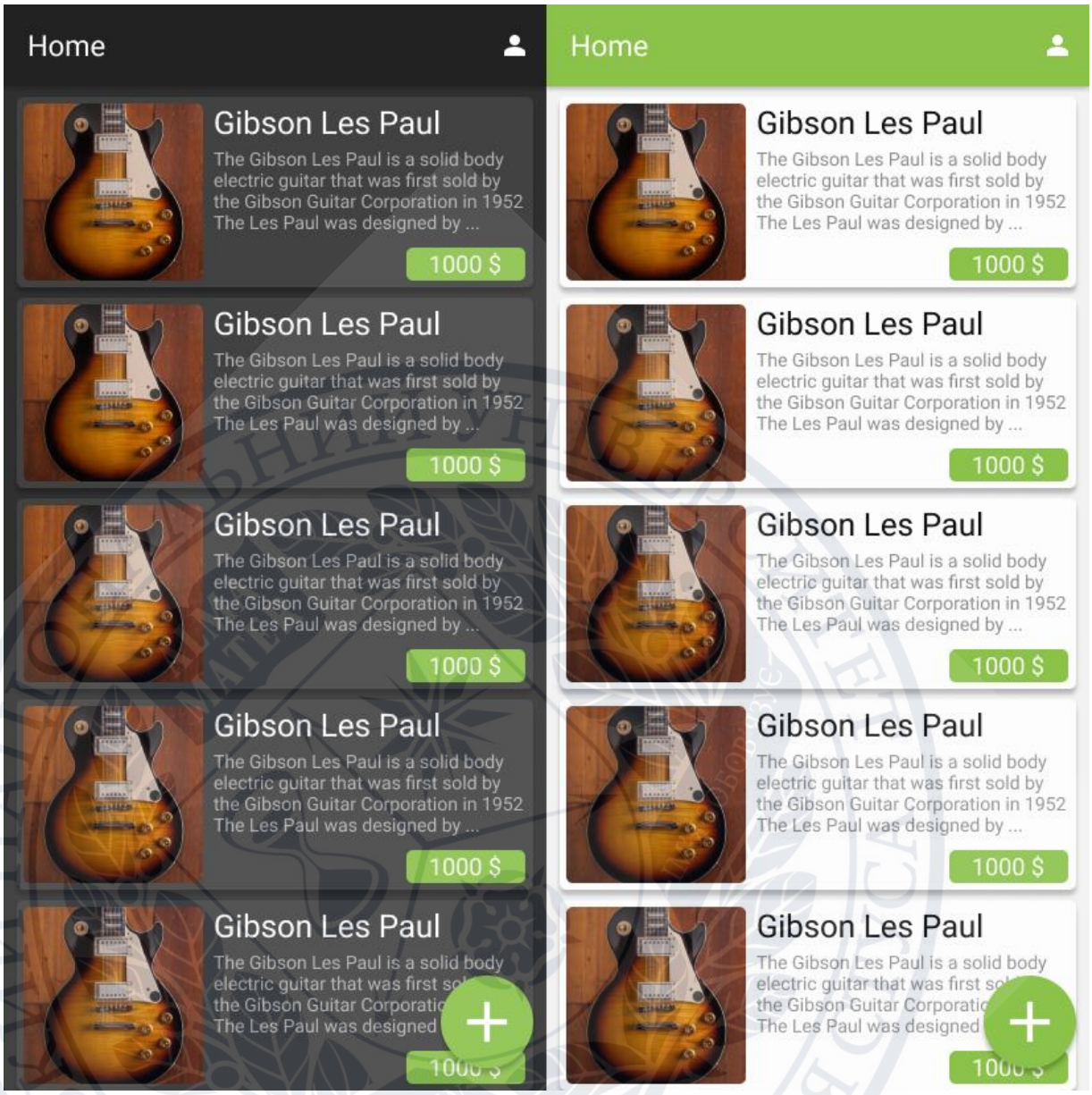

Рисунок 3.1 - Скріншот головної сторінки додатку «G-shop»
(темна та світла теми)

Для створення макета і прототипу майбутнього мобільного додатку була обрана програма Figma. Дизайн мобільного додатку має бути не яскравим з помірними кольорами. Дизайн має візуально привертати увагу, та не бути схожим на інші мобільні додатки, в ньому має бути щось своє, унікальне. Сьогодні актуальним є «плаский дизайн». Особливості такої графіки - у чарівній простоті двомірної площини.

Дизайн повинен підкреслювати інформацію, а не затьмарювати її. Яка цінність в крутій графіці, якщо за нею не видно вашого контенту? Буває таке, що сайт вражає графікою і спецефектами, але що він пропонує, де знаходиться

потрібний розділ і куди взагалі натиснути мишкою, щоб закрити величезний банер - незрозуміло. І ось користувач стає безпорадним, клацає по екрану, навмання наводячи курсор на різні елементи сайту, в надії потрапити, куди треба.

Рисунок 3.2 - Скріншот сторінки оголошення у додатку «G-shop» (темна та світла теми)

І це не перебільшення, це дійсно так. Красивих додатків - сотні, а чи є сенс в цій магічній красі? Тому в створенні сайтів все, як у житті - краса повинна бути розумною. 80% всієї інформації з зовнішнього світу ми сприймаємо за допомогою зору. При вигляді будь-якого об'єкта, оцінці його зовнішнього вигляду, форми і кольору ми створюємо перше враження про нього.

Людина може за частки секунди зрозуміти, що хоче купити товар тільки тому що йому подобається його колір, їх гарне поєднання, вибрана палітра. Дизайн проекту - перше, що кидається в очі людині, а його колір - перше, що оцінюється. Моментально, підсвідомо користувач приймає рішення чи залишить він додаток або видалить і більше не встановить його, чи подобається йому колір сайту, викликають вони у нього приємні і правильні асоціації або ж, навпаки, відштовхують і формують негативні емоції.

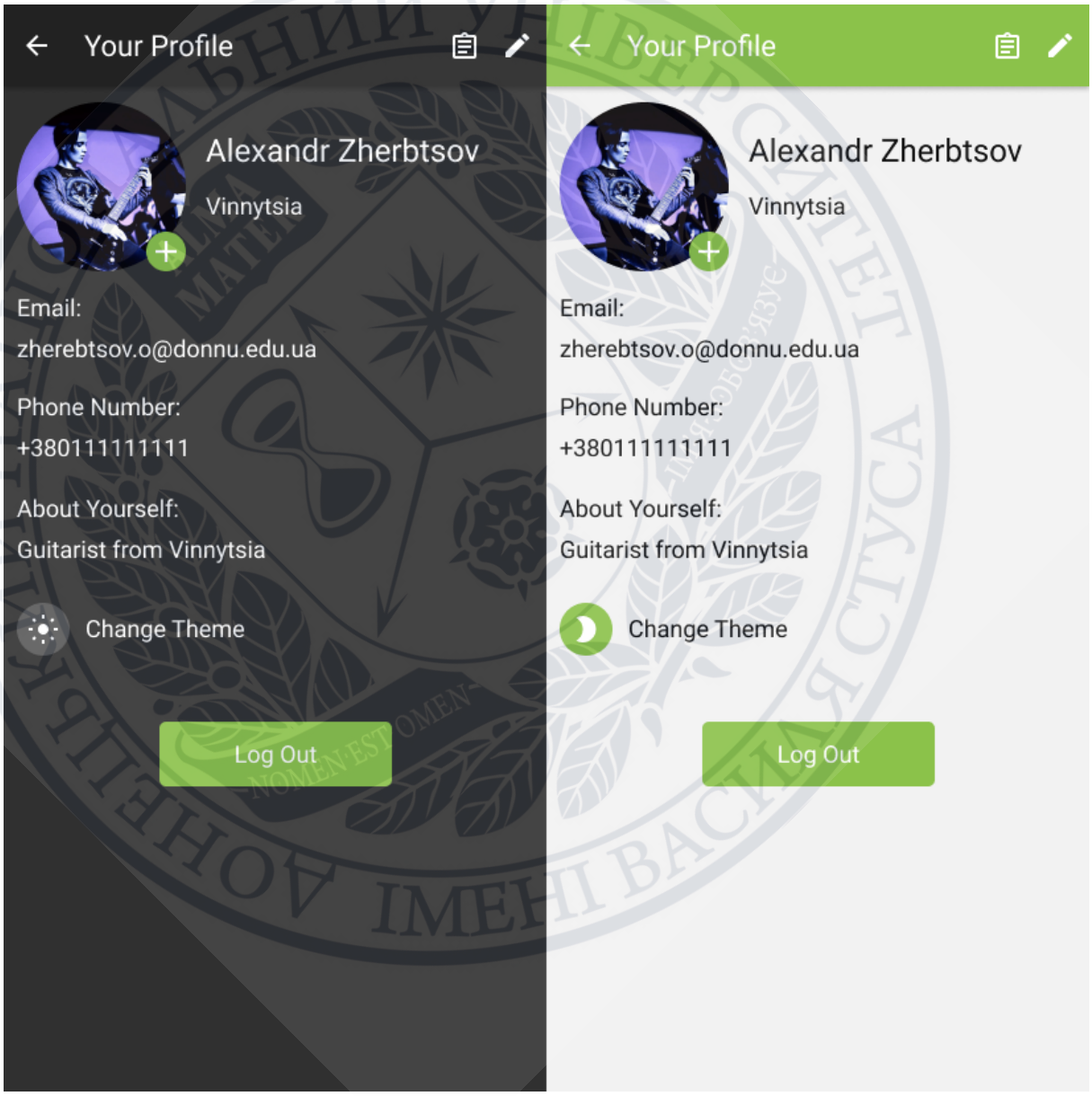

Рисунок 3.3 - Скріншот сторінки вашого профілю у додатку «G-shop» (темна та світла теми)

Фотографії та іконки миттєво розкажуть про що додпток і допоможуть з позиціонуванням: що ви пропонуєте і для кого. Є кілька критеріїв, за якими

можна визначити підходить фото/іконка або ні: неприродне яскраве світло, награні емоції, моделі, які дивляться в камеру. Намагайтеся не використовувати такі фото в додатку - вони сприймаються як візуальне сміття і, в кращому випадку, не здатні звернути на себе увагу цільової аудиторії. Об'єкт на ілюстрації, насправді, може бути будь-яким - головне, щоб він допомагав підкреслити головну суть тексту, додати потрібну атмосферу або трохи іронії. Якщо постаратися, можна знайти зображення-метафору, яке «зіграє» разом з заголовком.

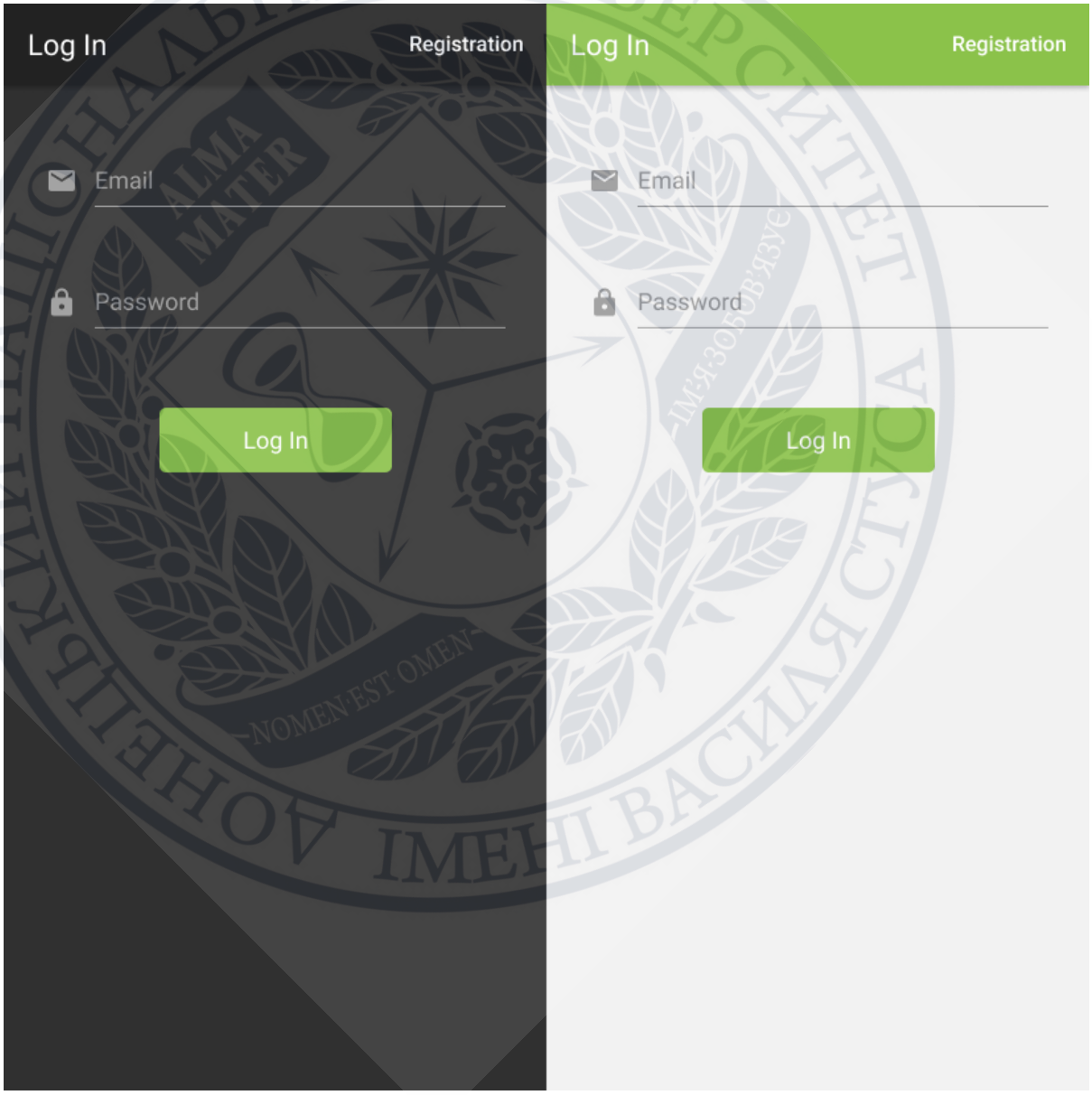

Рисунок 3.4 - Скріншот сторінки логіну у додатку «G-shop»
(темна та світла теми)

Можна надихайтися прикладом інших і «красти» як художник. Немає нічого поганого в тому, щоб орієнтуватися на лідерів в своїй індустрії і слідувати тим, хто робить якісно і красиво. Не потрібно запозичувати все цілком - досить взяти за основу головну ідею ілюстрації і додати свої штрихи.

3.2. Створення архітектури та UI проекту.

Добре відпрацьована архітектура потрібна всім додаткам, і складним, і шаблонним. З її допомогою заощаджується час, зусилля і гроші. Архітектура мобільних додатків - сукупність рішень, як організувати програму. У неї входять: структурні елементи і інтерфейси, зв'язок між обраними елементами, загальний стиль програми.

Гарна архітектура означає вигоду: простота і ефективність. Програму з такою архітектурою легше змінювати, тестувати і налагоджувати. Як зрозуміти, чи гарна архітектура у додатка?

Гнучкість: вибране рішення легко змінювати. Можна змінити один елемент, і це не стане фатальним для інших складових.

Масштабованість: час на розробку і доповнення зменшується. Гарна архітектура дозволяє направити розробку в декілька паралельних потоків.

Тестованість: додаток легко тестується, а значить, зменшується кількість помилок і збільшується його надійність.

Повторне використання: елементи і структуру можна використовувати в інших проектах.

Зрозумілість: код повинен бути зрозумілий як можна більшій кількості людей. Над додатком працює багато людей. Гарна архітектура дозволяє новачкам швидко розібратися в проекті.

Проектування мобільних або веб-додатків може проходити трьома способами, в залежності від завдань проекту:

- Монолітний
- Модульний
- Сервіс-орієнтований

Монолітний - це найдавніший підхід, в ньому немає складних систем. На сервері зберігається необхідна логіка, а в базі - вся потрібна інформація для сервера. Такі програми дуже прості і вимагають порівняно мало часу на розробку. Але є суттєві мінуси, через які цей підхід сьогодні майже не застосовується. У довгостроковій перспективі додатки обов'язково змінюються, тому що повинні відповідати новим платформам, гаджетам і операційним системам. В ході життя будь-якої програми змінюються команди розробників. Невеликий функціонал доповнюється новими ідеями. Тому монолітність, нехай і дешева на старті, не завжди виправдовує себе.

Модульна архітектура - додаток поділяється на модулі, кожен з яких відповідає за одну функцію. Модулі не залежать один від одного, і при збої одного елемента інші продовжують працювати. Це спрощує налагодження програми. Приклад подібної архітектури - PHP-фреймворки, платформи, на основі яких розробляються додатки. В цьому випадку вартість проекту трохи вище в порівнянні з монолітним додатком. Проте модулі дають можливість створювати досить складні додатки.

Сервіс-орієнтований підхід - продовження модульного. З ускладненням додатків деякі модулі виносяться на окремі апаратні частини і сервіси. Модулі тут іноді тримають власні бази даних і розташовуються на окремих пристроях. У цьому є свої плюси і мінуси. Сервіси можуть писатися на різних мовах, і їх взаємодія налаштовується через інтерфейс між елементами архітектури. Приклад подібних модулів - сервіси для електронної пошти, смс-повідомлень. Істотний мінус: тут потрібно дуже ретельно продумувати функції різних сервісів і їх взаємодію, щоб всі ланки ланцюжка працювали без помилок.

Така архітектура вимагає серйозних вкладень на старті, зате при грамотному підході знижуються витрати на наступних етапах розробки. Сервіс-орієнтована архітектура підходить для великих компаній.

Архітектура залежить від обраного типу програми.

Мобільний native-додаток - це програма для iOS, Android і інших платформ. Native означає, що програма створена для однієї платформи. Плюс - ефективність

завдяки відповідності всім вимогам обраної категорії пристроїв. Мінус - додаток погано працює на інших платформах.

Мобільний веб-додаток - сайт, оптимізований для роботи на мобільному пристрої. Плюс - працює на всіх платформах. Мінус - вимагає постійного підключення до Інтернету, тому що розташований на окремому сервері в мережі.

Гібридний додаток - поєднує в собі елементи перших двох типів. Проектування андроїд програм та програм для iOS останнім часом часто обирає за цей тип.

Основа архітектури мобільного додатка - єдиний інтерфейс, через який взаємодіють всі частини програми. Ядро використовує різні файли, які можна розділити на базові та конфігураційні.

Базові перебувають у додатку, яке публікується в магазині: компоненти для відображення сторінок; модулі для синхронізації, імпорту та експорту потрібної інформації; веб-сервіси; доступ до потрібних плагинів.

Конфігураційні включають в себе маніфест і налаштування розділів. Ці файли завантажуються при установці на пристрій. З їх допомогою програма налаштовується так, щоб працювати на конкретному пристрої найкращим чином.

Для дотатку «G-shop», була обрана архітектура Model-View-ViewModel (тобто MVVM) - це шаблон архітектури клієнтських додатків, який був запропонований Джоном Госсманом (John Gossman) як альтернатива шаблонами MVC і MVP при використанні технології зв'язування даних (Data Binding). Його концепція полягає в відділенні логіки представлення даних від бізнес-логіки шляхом винесення її в окремий клас для більш чіткого розмежування. Щоб реалізувати цю архітектуру у Flutter-додатку використовується бібліотека Stacked.[10]

Що ж означає кожна з трьох частин в назві:

- Model - це логіка, яка пов'язана з даними додатка. Іншими словами - це POJO, класи роботи з API, базою даних та ін.

- View - власне, це і є layout екрану, в якому розташовуються всі необхідні віджети для відображення інформації.

- ViewModel - об'єкт, в якому описується логіка поведінки View в залежності від результату роботи Model. Можна назвати його моделлю поведінки View. Це може бути як форматування тексту, так і логіка управління видимістю компонентів або відображення станів, таких як завантаження, помилка, порожні екрани тощо. Також в ній описується поведінка, яка була ініційована користувачем (введення тексту, натискання на кнопку, свайп і т.п.)

Ця архітектура дає нам:

- Гнучкість розробки. Цей підхід підвищує зручність роботи в команді, тому що поки один член команди працює над компонуванням і стилізацією екрану - інший, в цей час, описує логіку отримання даних і їх обробки;

- Тестування. Така структура спрощує написання тестів і процес створення mock-об'єктів. Також, в більшості випадків відпадає потреба в автоматизованому UI-тестуванні, тому що можна обернути unit-тестами сам ViewModel;

- Розмежування логіки. За рахунок більшого розмежування код стає більш гнучким і простим у підтримці, не кажучи про його читабельність. Кожен модуль відповідає за свою конкретну функцію і тільки.

Недоліки обраної архітектури:

- Для невеликих проектів цей підхід може бути виправданим.

- Якщо логіка прив'язки даних занадто складна - налагоджувати додаток буде трохи важче.

Виконання програми Flutter починається з функції main. Для створення графічного інтерфейсу в цій функції викликається інша вбудована функція - runApp (Widgetapp). Вона прикріплює певний віджет до екрану. Тобто по суті це, що ми бачимо під час запуску програми на екрані пристрою. Віджет , що прикріплюється передається в функцію runApp () в якості параметра. Коли користувач дивиться на свій пристрій або, точніше, на додаток, запущений на пристрої, то користувач бачить тільки екран. Насправді, все, що користувач

бачить - це пікселі, які разом складають 2-мірне зображення, і коли ви торкаєтеся екрану пальцем, пристрій розпізнає тільки положення вашого пальця на склі.

Вся краса мобільного додатка (з візуальної точки зору) в більшості випадків полягає в оновленні цього зображення на основі наступних взаємодій:

- з екраном пристрою (наприклад, палець на склі)
- з мережею (наприклад, зв'язок з сервером)
- з переходами (наприклад, hero анімація)
- з іншими зовнішніми датчиками

Візуалізація зображення на екрані забезпечується апаратним забезпеченням (дисплеєм), яке регулярно (зазвичай 60 раз в секунду) оновлює дисплей. Ця називається "частотою оновлення" і виражається в Гц (Герцах). Дисплей отримує інформацію для відображення від GPU (Graphics Processing Unit), що представляє собою спеціалізовану електронну схему, оптимізовану і призначену для швидкого формування зображення з деяких даних (полігонів і текстур). Кількість разів в секунду, яке графічний процесор може генерувати "зображення" (= буфер кадрів) для відображення і відправки його на апаратне забезпечення, називається кадровою частотою (прим: framerate). Це вимірюється за допомогою блоку кадрів в секунду (наприклад, 60 кадрів в секунду або 60 fps).

Одна з головних цілей програми Flutter - це створити 2-мірне зображення і дати можливість взаємодіяти з ним. Також у Flutter, майже все обумовлено необхідністю оновлення екрану швидко і в потрібний момент.

Коли пишуть додаток Flutter, використовуючи Dart, то розробник залишається на рівні Flutter Framework. Flutter Framework взаємодіє з Flutter Engine через шар абстракції, званий Window. Цей рівень абстракції надає ряд API для непрямої взаємодії з пристроєм. Також через цей рівень абстракції Flutter Engine повідомляє Flutter Framework, коли:

- подія, що представляє інтерес, відбувається на рівні пристрою (зміна орієнтації, зміна налаштувань, проблема з пам'яттю, стан роботи програми...)
- якась подія відбувається на рівні скла (жест)
- канал платформи відправляє деякі дані

•коли Flutter Engine готовий до рендерингу нового кадру

Коли ви розробляєте за допомогою Flutter, ви визначаєте структуру свого екрану (екранів), використовуючи віджети, які разом утворюють ієрархічну структуру. Ця структура виглядає, як дерево де перший віджет є його коренем. Також віджет сам по собі може бути агрегацією інших віджетів.

Формулювання «дерево віджетів» (Widget tree) існує тільки для полегшення розуміння, оскільки програмісти використовують віджети, але у Flutter немає дерева віджетів, правильніше буде сказати «дерево елементів» (tree of elements). Кожному віджету відповідає один елемент. Елементи пов'язані один з одним і утворюють дерево. Отже елемент є посиланням на щось в дереві.

Підсумки:

•У Flutter немає дерева віджетів, але є дерево елементів;

•Елементи створюються віджетами;

•Елемент посилається на віджет, який його створив;

•Елементи пов'язані разом з батьківськими співвідношеннями;

•У елемента може бути child або children;

•Елементи також можуть вказувати на Render Object.

•Елементи визначають, як відображаються частини блоків пов'язані один з одним

Усі зміни що вносяться в проект «G-shop» комітяться та пушаться в репозиторій на GitHub. Система контролю версій досить складна . Якщо один розробник працює над своїм проектом, то навіть не обов'язково реєструватися на сервері, можна просто встановити Git і працювати з системою контролю версій локально. Для локальної роботи над проектом не обов'язково навіть буде підключення до інтернету.

Репозиторій який створюється локально - це фактично такий же репозиторій як на сервері, тільки без можливості працювати над проектом розподіленої команди розробників. Система контролю версій - це, по суті, реалізація права програміста на помилку в своєму коді. «Людині властиво помилятися», так ось,

помиляються навіть кращі програмісти і щоб повернутися до свого коду та виправити помилку, додати функціональність й існує така чудова система, як система контролю версій. Крім того використання системи контролю версій - це більш ефективний менеджмент

3.3. Створення логіки додатку та взаємодія з Firebase.

У додатку G-shop використовується Firebase. Firebase - це платформа розробки мобільних та веб-додатків.[11]

Firebase Analytics - безкоштовне рішення для оцінки додатків, яке дає змогу ознайомитись із використанням додатків та залученням користувачів.

Firebase Auth - це служба, яка може аутентифікувати користувачів, використовуючи лише код на стороні клієнта. Він підтримує соціальні логін-провайдери Facebook, GitHub, Twitter і Google (і Google Play Games). Крім того, вона включає в себе систему управління користувачами, за допомогою якої розробники можуть увімкнути автентифікацію користувача за допомогою входу з електронної пошти та пароля, що зберігаються в Firebase.

Firebase надає в режимі реального часу базу даних та бекенд як службу. Ця служба надає розробникам додатків API, який дозволяє синхронізувати дані додатків між клієнтами та зберігати їх у хмарі Firebase. Компанія також надає клієнтські бібліотеки, які дозволяють інтеграцію із додатками Android, iOS, JavaScript / Node.js, Java, Objective-C, Swift. База даних також доступна через REST API та прив'язки до декількох сценаріїв JavaScript, таких як AngularJS, React, Ember.js та Backbone.js. REST API використовує протокол подій із сервером, який є інтерфейсом для створення HTTP-з'єднань для отримання push-повідомлень від сервера. Розробники, які використовують Realtime Database, можуть захищати свої дані за допомогою правил безпеки, що застосовуються на сервері. Cloud Firestore, яка є наступною генерацією Firebase Realtime Database, була випущена у бета-версії.

Firebase Storage забезпечує надійне завантаження та вивантаження файлів для додатків Firebase незалежно від якості мережі. Розробник може використовувати його для зберігання зображень, аудіо-, відео- чи іншого вмісту,

створеного користувачами. Зберігання Firebase підтримується Google Cloud Storage.

Firebase Hosting - це статичний та динамічний веб-хостинг, який було запущено 13 травня 2014 року. Він підтримує хостинг статичних файлів, таких як CSS, HTML, JavaScript та інші файли, а також динамічну підтримку Node.js через Cloud Functions. Служба передає файли через мережу доставки контенту (CDN) за допомогою протоколу HTTPS та шифрування SSL. Firebase підтримує Fastly, CDN, щоб забезпечити підтримку CDN Firebase Hosting. Компанія стверджує, що хостинг Firebase виріс із запитів клієнтів; розробники використовували Firebase для своєї бази даних в режимі реального часу, але вони потребували місця для розміщення їхнього вмісту.

ML Kit - це мобільна система машинного навчання для розробників, яка була запущена в режимі бета-тестування 8 травня 2018 року під час Google I/O 2018. ML Kit API містить різноманітні інструменти, серед яких розпізнавання тексту, розпізнавання облич, сканування баркодів, створення опису для зображень та розпізнавання наземних об'єктів. Наразі вона доступна для iOS та Android розробників. Також можливий імпорт власних моделей Tensor Flow. API можна використовати у пристрої або у хмарі.

Firepad - це редактор для спільної роботи у режимі реального часу із відкритим кодом. Випущений під ліцензією MIT, Firepad використовується декількома редакторами, включаючи редактор Atlassian Stash Realtime Editor та Koding.

Firechat - це програма чату з відкритим кодом у режимі реального часу. Він випущений під ліцензією MIT.

GeoFire - це бібліотека з відкритим кодом, яка використовує Firebase Realtime Database, що дозволяє розробникам додатків зберігати та запитувати набір ключів на основі географічного розташування.

Переваги Firebase:

1. Firebase Authentication покликана спростити створення безпечних систем аутентифікації, покращуючи при цьому вхід в систему і процес адаптації для

кінцевих користувачів. Він забезпечує наскрізне рішення для ідентифікації, підтримує облікові записи електронної пошти та паролів, аутентифікацію по телефону, а також вхід в Google, Twitter, Facebook, GitHub і багато іншого.

2. Firebase UI надає рішення для аутентифікації з відкритим вихідним кодом, яке обробляє потоки призначеного для користувача інтерфейсу для входу користувачів. Компонент Firebase UI Auth реалізує передові методи аутентифікації на мобільних пристроях і веб-сайтах, що може підвищити конверсію входу і реєстрації.
3. Firebase Security, створена тією ж командою, яка розробила GoogleSign-in, Smart Lock і Chrome Password Manager, використовує внутрішній досвід Google в управлінні однією з найбільших баз даних облікових записів в світі.
4. Визначення своєї власної системи аутентифікації може зайняти місяці, і для її обслуговування в майбутньому буде потрібна команда інженерів. Налаштуйте всю систему аутентифікації вашої програми, використовуючи менше 10 рядків коду, навіть для обробки складних випадків, таких як злиття облікових записів.
5. Firebase допомагає розробляти високоякісні додатки, розширювати призначення для користувача бази и заробляти більше грошей. Кожна функція працює незалежно, а разом вони працюють ще краще.
6. Дає можливість розвивати свій бекенд без управління серверами.
7. Дає можливість легко вирішувати поширені проблеми розробки додатків.
8. Легке масштабування для підтримки мільйонів користувачів.
9. Прискорена розробка додатків за допомогою повністю керованої серверної інфраструктури.
10. Можливість виконувати випуск з упевненістю і відстежувати продуктивність і стабільність.
11. Можливість підвищити взаємодію з користувачами за допомогою великої аналітики, A / B-тестування і кампаній з обміну повідомленнями.

12. Використовуючи Firebase Extensions, надаються параметри конфігурації для розширення, які є унікальними. Також можна переглянути включені API, створені ресурси і доступ до розширення.
13. Розширення мають відкритий вихідний код і побудовані на вже знайомих продуктах Firebase і Google Cloud. Розгортання та налаштування розширення виконуються в консолі Firebase або в інтерфейсі командного рядка Firebase. Після розгортання вони не вимагають обслуговування.
14. Firebase допомагає розробляти високоякісні додатки, розширювати призначену для користувача базу і заробляти більше грошей. Кожна функція працює незалежно, а разом вони працюють ще краще.

Можливості Firebase:

1. Створення безсерверних додатків, зберігаючи і синхронізуючи дані JSON між користувачами в режимі, близькому до реального часу, онлайн або офлайн, з надійною користувальницької безпекою.
2. Встановлення прапорців функцій під час прототипування і розробки, щоб була можливість динамічно контролювати і оптимізувати взаємодію з користувачем у виробничому середовищі.
3. Можливість додати в додаток потужні функції машинного навчання за допомогою готових API-інтерфейсів і підтримки розгортання користувальницьких моделей.
4. Можливість отримати інфраструктуру для надійної безкоштовної відправки та отримання push-повідомлень між сервером і пристроями на різних платформах.
5. Можливість створювати безпечні веб-сайти, які швидко завантажуються з підтримкою глобальної CDN, без будь-яких турбот.
6. Легке зберігання та обслуговування призначеного для користувача контенту в міру того, як додаток зростає від прототипу до готового до виробництва.
7. Змінює розмір зображень, завантажених в хмарне сховище, до зазначеного розміру і, при необхідності, зберігає або видаляє вихідне зображення.

8. Управляє доступом до платного контенту, синхронізуючи підписки з Firebase Authentication.
9. Створює і відправляє фірмові рахунки-фактури клієнтам за допомогою платіжної платформи Stripe.
10. Відправляє в реальному часі інкрементні поновлення з вказаною колекції Cloud Firestore в Big Query.
11. Створює і відправляє документ електронною поштою на основі вмісту документа, записаного в зазначену колекцію Cloud Firestore.
12. Додає нових користувачів з Firebase Authentication в зазначену аудиторію Mail chimp.
13. Видаляє дані з ключем для userId з Cloud Firestore, RealtimeDatabase і / або Cloud Storage, коли користувач видаляє свій обліковий запис.
14. Перекладає рядки, записані в колекцію Cloud Firestore, на кілька мов (використовує Cloud Translation API).
15. Записує лічильники подій в масштабі для забезпечення високошвидкісного запису в Cloud Firestore.
16. Скорочує URL-адреси, записані в зазначену колекцію Cloud Firestore (використовує Bitly).
17. Обмежує кількість вузлів зазначеною максимальною кількістю в зазначеному шляху до бази даних реального часу.

Більшості додатків необхідно знати особистість користувача. Знання особистості користувача дозволяє додатку безпечно зберігати призначені для користувача дані в хмарі і забезпечувати однакову персоналізовану роботу на всіх пристроях користувача.

Firebase Authentication надає серверні служби, прості у використанні SDK і готові бібліотеки призначеного для користувача інтерфейсу для аутентифікації користувачів у додатку. Він підтримує аутентифікацію з використанням паролів, номерів телефонів, соцмереж та поштових сервісів, таких як Google, Facebook і Twitter, і інших.

Firebase Authentication тісно інтегрується з іншими сервісами Firebase і використовує галузеві стандарти, такі як OAuth 2.0 і OpenIDConnect, тому його можна легко інтегрувати з призначеним для користувача сервером.

Можна виконувати вхід користувачів в свій додаток Firebase або за допомогою Firebase UI в якості повного рішення для аутентифікації, або за допомогою Firebase Authentication SDK для ручної інтеграції одного або декількох методів входу в програму.

При розробці проекту Android з використанням Firebase можна виявити концепції, які незнайомі або специфічні для Firebase.

Плагін Firebase Assistant для AndroidStudio - це плагін AndroidStudio, який реєструє додаток Android в проекті Firebase і додає необхідні файли конфігурації Firebase, плагіни і залежності в проект Android. Потрібно дотримуватися інструкцій на сторінці початку роботи Android, щоб використовувати Firebase Assistant. Потрібно переконайтися, що використовуються останні версії як Android Studio, так і Firebase Assistant («Файл»> «Перевірити наявність оновлень»).

Коли вибирано певні продукти Firebase для додавання в свій додаток, Firebase Assistant автоматично оголошує необхідні залежності в файлі app / build.gradle. Однак, щоб використовувати функції Firebase, які виходять за рамки поточних можливостей Firebase Assistant, можна вручну змінити ці залежності:

1. Якщо потрібно використовувати Firebase AndroidBoM, треба поновити залежності у файлі Gradle модуля (рівня додатки) (зазвичай app / build.gradle), щоб імпортувати платформу app / build.gradle. Також необхідно видалити версії з кожного рядка залежно бібліотеки Firebase.
2. Якщо потрібно використовувати бібліотеку розширень Kotlin, треба змінити рядок залежностей, який доданий в файл Gradle модуля (рівня додатки) (зазвичай app / build.gradle), щоб замість цього використовувати версію бібліотеки Firebasektx.

Сервіси Google - плагін і файл конфігурації. В рамках додавання Firebase в свій Android проекту, потрібно додати google-services плагін і google-services.json конфігураційний файл проекту.

Якщо потрібно додати Firebase в свій проект Android через консоль Firebase, Management REST API або Firebase CLI, потрібно вручну додати плагін і файл конфігурації в проект. Однак, якщо використовується Firebase Assistant, ці завдання автоматично виконуються під час встановлення.

Firebase AndroidBoM (Bill of Materials) (Специфікація) дозволяє керувати всіма версіями вашої бібліотеки Firebase, вказавши тільки одну версію - версію BoM.

Коли використовується Firebase BoM в додатку, BoM автоматично витягує окремі версії бібліотеки, зіставлені з версією BoM. Всі версії окремих бібліотек будуть сумісні. Коли виконується оновлення версію BoM в своєму додатку, всі бібліотеки Firebase, які використовуютья в додатку, оновляться до версій, зіставлених з цією версією BoM.

Щоб дізнатися, які версії бібліотеки Firebase зіставлені з конкретною версією BoM, потрібно ознайомитися з примітками до випуску для цієї версії BoM. Щоб порівняти версії бібліотеки, зіставлені з однією версією BoM, з іншою версією BoM, потрібно використовувати віджет порівняння.

Використання Firebase Android BoM для оголошення залежностей в файлі Gradle модуля (рівня додатку) (зазвичай app / build.gradle), при використанні BoM не вказуються окремі версії бібліотеки в рядках залежностей: в залежностях потрібно імпортувати BoM для платформи Firebase.

Бібліотеки розширень Firebase Kotlin (KTX) - це невеликі доповнення до базових SDK Firebase, які дозволяють писати красивий і ідіоматичний код Kotlin.

Щоб використовувати бібліотеку KTX в своєму додатку, потрібно змінити залежність, включивши суфікс -ktx. Кожна бібліотека KTX автоматично залежить від базової бібліотеки, тому немає необхідності включати обидві залежності в додаток.

Кожна бібліотека KTX надає різні синтаксичні розширення базової бібліотеки. Наприклад, бібліотека Analytics KTX спрощує реєстрацію подій.

Всі продукти Firebase пропонують бібліотеку KTX, за винятком Firebase ML і індексації додатків.

Деякі елементи екосистеми Google, Firebase і Android мають схожі угоди про імена.

Плагін Google Services Gradle (com.google.gms.google-services), який запускається під час збирання, щоб переконатися, що додаток має правильну конфігурацію для доступу до Firebase і API Google.

Незважаючи на свою назву, цей плагін не має відношення до сервісів Google Play і не впливає на можливості вашого додатку під час виконання.

Цей плагін також обробляє файл google-services.json який потрібно додати в додаток при налаштуванні Firebase.

Сервіси Google Play - це невидима фонова служба, яка працює на пристрої Android і надає кілька поширених API Google (наприклад, GoogleMaps і Google Sign In) для додатків на пристрої.

Завдяки централізації цих загальних API-інтерфейсів в єдину службу він зменшує розмір інших додатків і дозволяє пристрою отримувати автоматичні оновлення безпеки та поліпшення функцій без оновлення ОС.

Google Play магазин – це магазин для завантаження програм, фільмів, книг і багато чого іншого на пристрій Android.

Розробник керує поширенням, випусками тощо, свого додатку через консоль Google Play. Якщо на пристрої є Google Play Store, на ньому також працюють сервіси Google Play.

Більшість SDK Firebase для Android розроблені як бібліотеки з відкритим вихідним кодом в загальнодоступному репозиторії Firebase GitHub. Firebase активно працює над тим, щоб найближчим часом перемістити бібліотеки Firebase, розроблені в приватному порядку, в загальнодоступний GitHub.

Firebase підтримує колекцію прикладів швидкого запуску для більшості API Firebase на Android. Ці короткі керівництва можна знайти в загальнодоступному репозиторії швидкого запуску Firebase на GitHub.

Можна відкрити кожну інструкцію, як проект Android Studio, а потім запустити їх на мобільному пристрої або віртуальному пристрої (AVD). Або використовувати ці короткі керівництва як приклад коду для використання Firebase SDK.

Деякі SDK Firebase для Android залежать від сервісів GooglePlay, що означає, що вони будуть працювати тільки на пристроях і емуляторах з встановленими сервісами GooglePlay. Ці SDK Firebase взаємодіють з фоновою службою сервісів GooglePlay на пристрої, щоб надати додатку безпечний, актуальний і легкий API. На деяких пристроях Android, таких як пристрої Amazon Kindle Fire або пристрої, що продаються в деяких регіонах, не встановлені сервіси GooglePlay.

SDK Firebase можна розділити на три категорії:

1. Для цих SDK потрібні сервіси Google Play, в іншому випадку вони не працюють.
2. Рекомендуються сервіси Play - ці SDK вимагають, щоб сервіси GooglePlay мали повну функціональність, але вони як і раніше пропонують більшу частину функцій навіть без сервісів GooglePlay.
3. Сервіси Play не потрібні - ці SDKS не вимагають, щоб сервіси GooglePlay мали повну функціональність.

API REST управління Firebase дозволяє програмно знаходити і керувати проектами Firebase, включаючи ресурси Firebase та додаткові Firebase.

Загальний робочий процес додавання ресурсів та додатків Firebase у існуючому проекті Google Cloud, який на даний момент не використовує службу Firebase:

Основні кроки:

1. Додати сервіси Firebase в проект.
2. Додати додатки Firebase в проект Firebase.

3. Зв'язати проект Firebase з аккаунтом Google Analytics.
4. Завершити налаштування місцеположення проекту за замовчуванням.

Перш ніж виконувати будь-які дії, потрібно переконатися, що ввімкнені API.

Спочатку необхідно включити Management API для вашого проекту Google Cloud і згенерувати токен доступу.

Необхідно Увімкнути Management REST API для свого Google Cloud.

Якщо це не зроблено, необхідно включити Firebase Management API для використання з проектом Google Cloud:

1. Відкрити сторінку Firebase Management API в консолі Google API.
2. У відповідь на запит обрати проект Google Cloud.
3. Натиснути увімкнути на сторінці Firebase Management API.
4. Натиснути увімкнути на сторінці Firebase Management API.
5. Потім використовувати Firebase Admin SDK, щоб отримати токен доступу з облікових даних службового облікового запису.

Можна знайти проекти Google Cloud, які доступні для публікації сервісів Firebase.

Тіло відповіді від запиту availableProjects.list містить список об'єктів Project Info. Якщо список проектів занадто довгий, тіло відповіді також містить nextPageToken, який можна використовувати в якості параметра запиту для отримання наступної сторінки проектів.

Також є можливість використовувати будь-яке значення project вказане у відповіді з availableProjects.list щоб додати служби Firebase або додатки в свій проект.

Проекти Google Cloud можуть скористатися послугами, пропонованими Firebase. Також можна додати служби Firebase в існуючий проект Google Cloud в консолі Firebase.

Результатом виклику projects.addFirebase є Operation. Перш ніж буде можливість викликати інші кінцеві точки, пов'язані з Firebase, для проекту, операція повинна бути успішною.

Щоб перевірити, чи успішна операція, можна викликати operations.get для операції до тих пір, поки значення done стане true а його response буде FirebaseProject тип FirebaseProject. У разі збою операції встановлюється errorgoogle.rpc.Status.

Оскільки donetrue і тип response - FirebaseProject, проект Google Cloud тепер має служби Firebase. Відповідь також містить іншу корисну інформацію по вашому недавно створеному FirebaseProject, наприклад, про його projectNumber і його resources замовчуванням. Operation автоматично видаляється після завершення.

Firebase Project може використовувати безліч різних додатків, включаючи iOS, Android і веб-додатки. Потрібно звернути увагу, що також можна додати додатки Firebase в існуючий проект Firebase в консолі Firebase.

Також є можливість програмно зв'язати існуючий обліковий запис Google Analytics з існуючим Firebase Project. Потрібно звернути увагу, що також можна пов'язати існуючий проект Firebase з Google Analytics на вкладці «Інтеграції» в налаштуваннях проекту.

Для виклику projects.addGoogleAnalytics потрібно analytics_resource, який може бути або analyticsAccountId або analyticsPropertyId:

1. Треба вказати існуючий analyticsAccountId щоб підготувати новий ресурс GoogleAnalytics у зазначеній облікового запису і зв'язати нову властивість з вашим проектом Firebase.
2. Також треба вказати існуючий analyticsPropertyId щоб зв'язати ресурс GoogleAnalytics з вашим проектом Firebase.

Є можливість знайти analyticsAccountId і будь-який існуючий analyticsPropertyId на веб-сайті GoogleAnalytics.

Коли виконується запит projects.addGoogleAnalytics:

1. Перша перевірка визначає, чи відповідають існуючі потоки даних у властивості Google Analytics будь-яким існуючим програмам Firebase в Firebase Project (на основі packageName або bundleId пов'язаних з потоком

даних). Потім, у разі необхідності, зв'язуються потоки даних і додатки. Ця автоматична прив'язка застосовується тільки до додатків Android і iOS.

2. Якщо відповідні потоки даних для ваших додатків Firebase не знайдені, нові потоки даних надаються у властивості Google Analytics для кожного з ваших додатків Firebase. Новий потік даних завжди готується для веб-додатку, навіть якщо він раніше був пов'язаний з потоком даних в вашому ресурсі Analytics.

Потрібно виконати запит projects.addGoogleAnalytics.

У тілі запиту project.addGoogleAnalytics вказуємо обліковий запис Google Analytics analytics Account Id. Цей виклик надасть нову властивість Google Analytics і зв'яже нову властивість з FirebaseProject.

Результатом виклику projects.addGoogleAnalytics є Operation. Перш ніж буде можливість викликати інші кінцеві точки, пов'язані з Firebase, для проекту, операція повинна бути успішною.

Щоб перевірити, чи успішна операція, можна викликати operations.get для операції, поки значення done стане true a response матиме тип analyticsDetails. У разі збою операції встановлюється errorgoogle.rpc.Status.

Оскільки done має значення true і тип response - analyticsDetails, FirebaseProject тепер пов'язаний з вказаною обліковим записом GoogleAnalytics. Operation автоматично видаляється після завершення.

Завершення налаштування місцеположення проекту за замовчуванням необов'язкове.

Якщо проект Firebase буде використовувати Cloud Firestore, Cloud Storage або додаток AppEngine, можна програмно вказати місце розташування ресурсу Google Cloud Platform (GCP) за замовчуванням для проекту. Також можна ввести місце в консолі Firebase.

Перед налаштуванням цього місця розташування треба ознайомитися з розділом Вибір місць розташування для вашого проекту, щоб дізнатися, яке місце розташування найкраще підходить для проекту. Також потрібно викликати projects.availableLocations щоб отримати список допустимих місць розташування

для проекту, тому що, якщо проект є частиною організації Google Cloud, то політика організації може обмежувати допустимі розташування для проекту.

Виклик цього методу defaultLocation.finalize створює додаток AppEngine з defaultLocation.finalize хмарного сховища за замовчуванням, розташований в locationId який вказується в тілі запиту.

Розташування ресурсу GCP за замовчуванням могло бути вже зазначено, якщо в Project вже є додаток AppEngine або цей метод defaultLocation.finalize був раніше викликаний.

Потрібно projects.defaultLocation.finalize де locationId: місце, де зберігаються дані для сервісів GCP, що вимагають налаштування місцеположення, наприклад Cloud Firestore або Cloud Storage.

Результатом виклику projects.defaultLocation.finalize є Operation. Перш ніж буде можливість викликати інші кінцеві точки, пов'язані з Firebase, для вашого проекту, операція повинна бути успішною.

Щоб перевірити, чи успішна операція, можна викликати operations.get для операції, поки значення done стане true а його response буде типу google.protobuf.Empty. Якщо операція завершилася невдало, повідомлення про error тілі відповіді матиме тип google.rpc.Status. Operation автоматично видаляється після завершення.

Ключові можливості Cloud Firestore:

1. Гнучкість. Модель даних Cloud Firestore підтримує гнучкі ієрархічні структури даних. Зберігання даних в документах, організованих в колекції. Документи можуть містити складні вкладені об'єкти на додаток до вкладених колекцій.
2. Виразні запити. У Cloud Firestore можна використовувати запити для отримання окремих конкретних документів або для отримання всіх документів в колекції, які відповідають параметрам запиту. Запити можуть включати кілька пов'язаних фільтрів і комбінувати фільтрацію і сортування. Вони також індексуються за замовчуванням, тому продуктивність запиту пропорційна розміру набору результатів, а не набору даних.

3. Оновлення в реальному часі. Як і база даних в реальному часі, Cloud Firestore використовує синхронізацію даних для поновлення даних на будь-якому підключеному пристрої. Однак він також призначений для ефективного виконання простих одноразових запитів на вибірку.
4. Офлайн-підтримка. Cloud Firestore кеширує дані, які активно використовує додаток, тому додаток може записувати, читати, прослуховувати і запитувати дані, навіть якщо пристрій знаходиться в автономному режимі. Коли пристрій повертається в мережу, Cloud Firestore синхронізує будь-які локальні зміни з Cloud Firestore.
5. Створено для масштабування. Cloud Firestore пропонує найкраще з потужної інфраструктури Google Cloud: автоматичну реплікацію даних в декількох регіонах, надійні гарантії узгодженості, атомарні пакетні операції і підтримку реальних транзакцій. Google розробили Cloud Firestore для обробки самих складних робочих навантажень баз даних з найбільших додатків світу.

Cloud Firestore - це розміщена в хмарі база даних NoSQL, до якої iOS, Android і веб-додатки можуть отримати доступ безпосередньо через власні SDK. Cloud Firestore також доступний у власних пакетах SDK для Node.js, Java, Python, Unity, C ++ і Go на додаток до REST і RPC API.

Слідуючи моделі даних NoSQL Cloud Firestore, дані зберігаються в документах, які містять поля, зіставлені зі значеннями. Ці документи зберігаються в колекціях, які представляють собою контейнери для документів, які можна використовувати для організації даних і побудови запитів. Документи підтримують безліч різних типів даних, від простих рядків і чисел до складних вкладених об'єктів. Також можна створювати вкладені колекції в документах і будувати ієрархічні структури даних, які масштабуються в міру зростання бази даних. Модель даних Cloud Firestore підтримує будь-яку структуру даних, яка найкраще підходить для програми.

Крім того, запити в Cloud Firestore є виразними, ефективними і гнучкими. Можна створювати неглибокі запити для отримання даних на рівні документа без необхідності отримувати всю колекцію або будь-які вкладені колекції.

Можна додати сортування, фільтрацію і обмеження до своїх запитів або курсором, щоб розбивати результати на сторінці. Щоб дані у запитах додатка залишалися актуальними, не витягуючи всю базу даних при кожному оновленні, треба додати Listeners в реальному часі. Додавання Listeners в реальному часі в додаток дає можливість за допомогою моментального знімка даних щоразу повідомляти, коли дані змінюються, з мометальнимі змінами у клієнтських програмах, що прослуховують ці зміни, отримуючи тільки нові зміни.

Обов'язково треба захистити доступ до даних в Cloud Firestore за допомогою аутентифікації Firebase і правил безпеки Cloud Firestore для Android, iOS і JavaScript або управління ідентифікацією та доступом (IAM) для мов на стороні сервера.

Шлях реалізації:

1. Потрібно інтегрувати SDK Cloud Firestore, та підключити клієнтів через Gradle, Cocoa Pods або сценарій include.
2. Потрібно захистити дані. Треба використовувати правила безпеки Cloud Firestore або управління ідентифікацією та доступом (IAM), щоб захистити дані для мобільних / веб-додатків і розробки серверів відповідно.
3. Додати дані. Потрібно створити документи і колекції в своїй базі даних.
4. Отримати дані. Створюйте запити або використовуйте Listeners в реальному часі для отримання даних з бази даних.

Після створення бази даних в Cloud Firestore треба створити базу даних Cloud Firestore, якщо це не було зроблено, далі треба створити проект Firebase: в консолі Firebase треба натиснути «Додати проект», а потім дотримуючись вказівок на екрані створити проект Firebase або додати служби Firebase в існуючий проект GCP. Далі потрібно перейти в розділ Cloud Firestore в консолі Firebase. Там буде запропоновано вибрати існуючий проект Firebase. Потрібно дотримуватися робочого процесу створення бази даних.

Тестовий режим.

Підходить для початку роботи з мобільними і веб-клієнтськими бібліотеками, але дозволяє будь-кому читати і перезаписувати дані. Після тестування треба захистити дані.

Щоб почати роботу з інтернетом, SDK для iOS або Android, треба вибрати тестовий режим.

Заблокований режим.

Забороняє всі операції читання і запису з мобільних і веб-клієнтів. Аутентифіковані сервери додатків (C #, Go, Java, Node.js, PHP, Python або Ruby) як і раніше можуть звертатися до бази даних.

Щоб почати роботу з клієнтською бібліотекою сервера C #, Go, Java, Node.js, PHP, Python або Ruby, треба вибрати заблокований режим.

Далі необхідно вибрати місце для бази даних. Цей параметр розташування є місцем розташування ресурсу Google Cloud Platform (GCP) за замовчуванням для вашого проекту. Треба звернути увагу, що це місце розташування буде використовуватися для сервісів GCP в проекті, для яких потрібно налаштування місцеположення, зокрема, для сегмента Cloud Storage за замовчуванням і програми AppEngine (що необхідно, якщо використовується Cloud Scheduler).

Якщо не можна ввести місце, значить, у проекті вже є місце розташування ресурсу GCP за замовчуванням. Він був встановлений або під час створення проекту, або при налаштуванні іншої служби, для якої потрібно налаштування місцеположення.

Після того, як було встановлено місце розташування ресурсу GCP за замовчуванням для проекту, його неможливо змінити.

Cloud Firestore і AppEngine: не можна використовувати Cloud Firestore і CloudDatastore в одному проекті, що може вплинути на функції, які залежать AppEngine. Можна спробувати використовувати Cloud Firestore з іншим проектом.

Коли буде ввімкнено Cloud Firestore, він також увімкне API в Cloud API Manager.

Для мобільних розробників, перш ніж говорити про те, як додаток записує і читає з Cloud Firestore, потрібно обрати набір інструментів, які можна використовувати для прототипування і тестування функціональності Cloud Firestore: Firebase Local Emulator Suite. Якщо використовуються різні моделі даних, потрібно оптимізувати правила безпеки або потрібно буде попрацювати над пошуком найбільш економічного способу взаємодії з серверної частиною, можливість працювати локально без розгортання живих сервісів може бути відмінною ідеєю.

Емулятор Cloud Firestore є частиною Local Emulator Suite, який дозволяє додатку взаємодіяти з вмістом і конфігурацією емульованої бази даних, а також з додатковими ресурсами емульованого проекту (функціями, іншими базами даних і правилами безпеки). Потрібно зверніть увагу, що Local Emulator Suite ще не підтримує емульованого хмарного сховища.

Використання емулятора Cloud Firestore включає всього кілька кроків:

1. Додавання рядка коду в тестову конфігурацію вашого додатку для підключення до емулятора.
2. З кореня локального каталогу проекту треба запустити firebase emulators: start.
3. Здійснення викликів з коду прототипу вашого додатку, як зазвичай, за допомогою SDK платформи Cloud Firestore.

Firebase пропонує два хмарних рішення для баз даних, доступних клієнтові, які підтримують синхронізацію даних в реальному часі:

1. Cloud Firestore - новітня база даних Firebase для розробки мобільних додатків. Він заснований на успіху Realtime Database з новою, більш інтуїтивно зрозумілою моделлю даних. Cloud Firestore також пропонує більш насичені і швидкі запити і масштабується далі, ніж база даних в реальному часі.
2. Realtime Database - це вихідна база даних Firebase. Це ефективне рішення з малою затримкою для мобільних додатків, яким потрібна синхронізація станів клієнтів в реальному часі.

Обидва рішення пропонують:

1. Клієнтські SDK без серверів для розгортання та обслуговування

2. Оновлення в реальному часі
3. Рівень безкоштовного користування, або оплатою за те, що використовуєте.

Realtime Database, і Cloud Firestore є базами даних NoSQL.

База даних в реальному часі:

1. Зберігає дані як одне велике дерево JSON.
2. Прості дані дуже легко зберігати.
3. Складні ієрархічні дані складніше організувати в масштабі.

Cloud Firestore:

1. Зберігає дані як колекції документів.
2. Прості дані легко зберігати в документах, які дуже схожі на JSON.
3. Складні ієрархічні дані легше організувати в масштабі, використовуючи вкладені колекції в документах.
4. Потрібна менша денормалізація і вирівнювання даних.

База даних в реальному часі

Всі дані бази даних Firebase Realtime зберігаються у вигляді об'єктів JSON. Можна думати про базу даних як про JSON-дерево, яке розміщено в хмарині. На відміну від бази даних SQL, тут немає таблиць або записів. Коли додаються дані в дерево JSON, воно стає вузлом в існуючій структурі JSON зі зв'язаним ключем. Можна надати власні ключі, такі як ідентифікатори користувачів або семантичні імена, або вони можуть бути надані за допомогою push.

Хоча база даних використовує дерево JSON, дані, що зберігаються в базі даних, можуть бути представлені як певні власні типи, відповідні доступним типам JSON, щоб допомогти написати більш зручний для супроводу код.

Оскільки база даних Firebase Realtime дозволяє розміщувати дані на глибині до 32 рівнів, може виникнути спокуса подумати, що це повинна бути структура за замовчуванням. Однак, коли витягуються дані з місця в базі даних, також отримуються всі її дочірні вузли. Крім того, коли надається комусь доступ для читання або запису в вузлі бази даних, також надається їм доступ до всіх даних в цьому вузлі. Тому на практиці краще зберегти структуру даних як можна більш плоскою.

Якщо замість цього дані розділені на окремі шляхи, так звані денормалізації, вони можуть бути ефективно завантажені окремими викликами в міру необхідності.

При створенні додатків часто краще завантажити підмножину списку. Це особливо часто зустрічається, якщо список містить тисячі записів. Коли цей зв'язок статичний і односпрямований, можна просто вкласти дочірні об'єкти під батьківські.

Іноді цей взаємозв'язок більш динамічний, або може знадобитися денормалізація цих даних. У багатьох випадках можна денормалізувати дані, використовуючи запит для отримання підмножини даних.

Але навіть цього може бути недостатньо. Розглянемо, наприклад, двосторонні відносини між користувачами і групами. Користувачі можуть належати до групи, а групи складають список користувачів. Коли приходить час вирішити, до яких груп належить користувач, все ускладнюється.

Що потрібно, так це елегантний спосіб скласти список груп, до яких належить користувач, і отримати дані тільки для цих груп.

Потрібно дублювати деякі дані, зберігаючи відносини як під записом, так і під групою. Це необхідна надмірність для двосторонніх відносин. Це дозволяє вам швидко і ефективно отримувати членство в Ada, навіть якщо список користувачів або груп обчислюється мільйонами або коли правила безпеки бази даних реального часу запобігають доступ до деяких записів.

Цей підхід, що інвертує дані шляхом перерахування ідентифікаторів в якості ключів і установки значення true, робить перевірку ключа такий же простий, як читання / users / $uid / groups / $group_id і перевірка, чи є він null. Індекс працює швидше і набагато ефективніше, ніж запит або сканування даних.

Модель данних Cloud Firestore

Cloud Firestore - це документно-орієнтована база даних NoSQL. На відміну від бази даних SQL, тут немає таблиць або рядків. Замість цього зберігаються дані в документах, які організовані в колекції.

Кожен документ містить набір пар ключ-значення. Cloud Firestore оптимізований для зберігання великих колекцій невеликих документів.

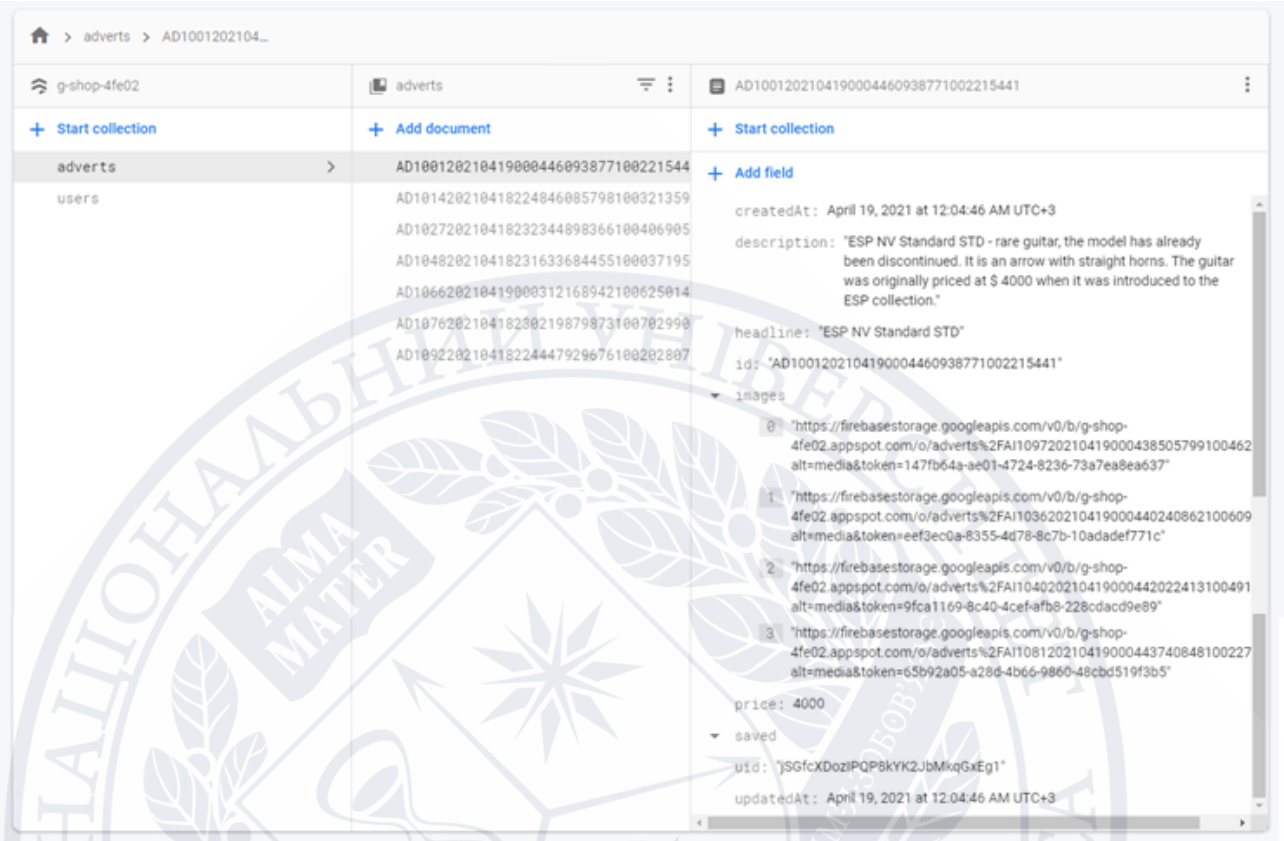

Рисунок 3.5 - Як виглядають колекції та документи у додатку «G-shop» (Firestore)

Всі документи повинні зберігатися в колекціях. Документи можуть містити вкладені колекції і вкладені об'єкти, обидва з яких можуть включати примітивні поля, такі як рядки, або складні об'єкти, такі як списки.

Колекції та документи створюються в Cloud Firestore неявно. Просто призначте дані документу в колекції. Якщо колекція або документ не існує, Cloud Firestore створює їх.

У Cloud Firestore одиницею зберігання є документ. Документ - це полегшеий запис, що містить поля, які зіставляються зі значеннями. Кожен документ ідентифікується ім'ям.

Cloud Firestore підтримує різні типи даних для значень: логічні, числові, строкові, географічні, виконавчі BLOB-об'єкти і тимчасові мітки. Також можна

використовувати масиви або вкладені об'єкти, звані картами, для структурування даних в документі.

Складні вкладені об'єкти в документі називаються картами.

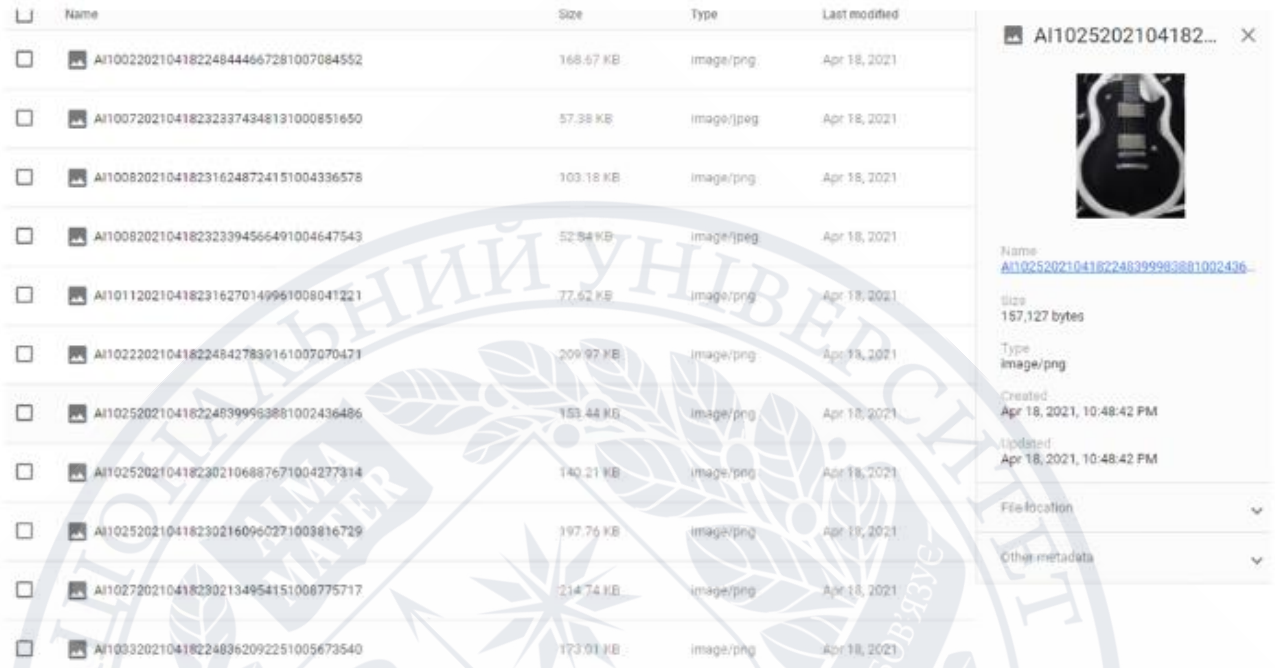

Рисунок 3.6 - Як виглядають завантажені зображення додатку «G-shop» (Cloud)

Після створення документу можна помітити, що документи дуже схожі на JSON. Фактично, вони в основному такі. Є деякі відмінності (наприклад, документи підтримують додаткові типи даних і обмежені за розміром до 1 МБ), але в цілому ви можете розглядати документи як полегшені записи JSON.

Документи живуть в колекціях, які є просто контейнерами для документів. Наприклад, у вас може бути колекція users містить ваших різних користувачів, кожен з яких представлений деяким документом.

Cloud Firestore не має схеми, тому є повна свобода вибору полів, які створюються в кожному документі, і типів даних, які зберігаються в цих полях. Всі документи в одній колекції можуть містити різні поля або зберігати в цих полях різні типи даних. Однак рекомендується використовувати одні і ті ж поля і типи даних в декількох документах, щоб спростити запит документів.

Колекція містить документи і нічого більше. Колекція не може безпосередньо містити необроблені поля зі значеннями і не може містити інші колекції.

Імена документів в колекції унікальні. Можна надати свої власні ключі, наприклад ідентифікатори користувачів, або дозволити Cloud Firestore автоматично створювати для вас випадкові ідентифікатори.

Розробнику не потрібно «створювати» або «видаляти» колекції. Після створення першого документа в колекції колекція існує. Якщо видалити всі документи в колекції, вона більше не існує.

Кожен документ в Cloud Firestore однозначно ідентифікується по його місцю розташування в базі даних. Щоб послатися на це місце в вашому коді, можна створити посилання на нього.

Посилання - це легкий об'єкт, який просто вказує на місце у вашій базі даних. Можна створити посилання незалежно від того, чи існують там дані чи ні, і створення посилання не виконує ніяких мережевих операцій.

Також є можливість створювати посилання на колекції.

Посилання на колекції і посилання на документи - це два різних типи посилань, які дозволяють виконувати різні операції. Наприклад, можна використовувати посилання на колекцію для запиту документів в колекції, і можна використовувати посилання на документ для читання або запису окремого документа.

Для зручності також можна створювати посилання, вказавши шлях до документа або колекції у вигляді рядка з компонентами шляху, розділеними косою рисою (/), наприклад, щоб створити посилання на документ.

Кращий спосіб зберігати дані в колекції - це використовувати вкладені колекції. Підколекція - це колекція, пов'язана з певним документом.

Можна виконувати запити до вкладених колекцій з одним і тим же ідентифікатором колекції, використовуючи запити групи колекцій.

Документи у вкладених колекціях також можуть містити вкладені колекції, що дозволяє додатково вкладати дані. Можна вкладати дані на глибину до 100 рівнів.

При видалення документа не буде видалено його вкладених колекцій. Коли видаляється документ, в якому є вкладені колекції, ці вкладені колекції не видаляються. Наприклад, документ може перебувати в coll / doc / subcoll / subdoc навіть якщо документ coll/doc більше не існує. Якщо потрібно видалити документи з вкладених колекцій при видаленні батьківського документа, потрібно зробити це вручну.

Щоб видалити всю колекцію або вкладену колекцію в Cloud Firestore, треба отримати всі документи в колекції або вкладеної колекції і видалити їх. Якщо в базі даних великі колекції, то їх можна видаляти меншими партіями, щоб уникнути помилок через брак пам'яті. Повторюйте процес, поки не видалите всю колекцію або вкладену колекцію.

Видалення колекції вимагає узгодження необмеженої кількості індивідуальних запитів на видалення. Якщо вам потрібно видалити цілі колекції, робіть це тільки з середовища довіреного сервера. Хоча можна видалити колекцію з мобільного / веб-клієнта, але це негативно позначається на безпеці і продуктивності.

Також можна використовувати інтерфейс командного рядка Firebase для видалення документів і колекцій або можна видаляти документи і колекції зі сторінки Cloud Firestore в консолі. При видаленні документа з консолі видаляються всі вкладені дані в цьому документі, включаючи всі вкладені колекції.

Cloud Firestore підтримує SDK для Android, IOS і Web. У поєднанні з правилами безпеки Cloud Firestore і FirebaseAuth мобільні і веб-SDK підтримують бессерверной архітектури додатків, в яких клієнти підключаються безпосередньо до бази даних Cloud Firestore. У бессерверній архітектурі не потрібно підтримувати проміжний сервер між клієнтами і базою даних Cloud Firestore.

Висновок до розділу 3

Мобільний додаток «G-shop» був розроблений за допомогою Flutter SDK, та найзручніших і найтехнологічніших інструментів, які доступні на сьогодні в pub.dev та Firebase. В основі цього мобільного додатку використовується багато сучасних рішень певних задач, мобільний додаток зрозумілий як розробнику так і простому користувачеві. Мобільний додаток побудовано таким чином, що його легко модифікувати та змінювати і це відповідає сучасним тенденціям.

ВИСНОВКИ

На початку роботи було поставлено мету: розробити спеціалізований мобільний додаток (електронну дошку оголошень) для купівлі-продажу електронних інструментів та супутніх товарів та послуг.

Актуальність цього полягає в тому, що мобільні додатки це найпопулярніша сучасна тенденція it-ринку, особливо в умовах пандемії. Їх переваги : швидкість та зручність у користуванні, адаптивність до різних гаджетів користувачів, можливість роботи online та offline, реалізація в різних варіаціях, завдяки стрімкому розвитку технологій.

Для того щоб розпочати розробку мобільного додатку «G-shop» потрібно було проаналізувати необхідність та актуальність розробки електронної дошки оголошень, зрозуміти що являють собою мобільні додатки, у чому їх відмінність від web-сайтів та які є популярні тенденції по створенню мобільних додатків на сьогодні.

Було досліджено, вивчено і обрано багато інструментів для розробки сучасного мобільного додатку, що надали змогу реалізувати всі необхідні функції. Було вирішено розробити мобільний додаток для ОС Android версії 7.0, застосовуючи SDK – Flutter, як один з найперспективніших. Мова програмування Dart. Це мова програмування , створена Google, яка позиціонується , як альтернатива JavaScript.

Для створення макета і прототипу майбутнього мобільного додатку була обрана програма Figma. Розробляючи UX/UI дизайн мобільного додатку враховано ефект «естетика-юзабіліті» та підібрано кольори і графічні компоненти таким чином, щоб вони привертали увагу , а дизайн у цілому був інтуїтивно зрозумілий користувачу.

Для дотатку «G-shop», була обрана архітектура Model-View-ViewModel. Щоб реалізувати цю архітектуру у Flutter-додатку застосовується бібліотека Stacked. У додатку G-shop використовується Firebase, що надає в режимі реального часу базу даних та бекенд.

Реалізовано основний функціонал мобільного додатку, а саме, реєстрація користувача, можливість користувача мобільного додатку створювати, редагувати та видаляти свої оголошення або переглядати оголошення інших користувачів та отримувати необхідну інформацію та інші.

Викладений у цій роботі огляд інструментів для створення мобільних додатків, розробки UX/UI дизайну та огляд мови програмування, а також рекомендації та висновки можуть бути використані при вивчені мови програмування Dart, Flutter SDK, а розроблений прототип мобільного додатку може бути практично реалізований на ринку мобільних додатків.

Отже, під час виконання поставлених задач, була проробленa об'ємна робота створення сучасної дошки оголошень у вигляді мобільного додатку написаного на Flutter SDK для ОС Android. Розроблено прототип та дизайн мобільного додатку для породажу музичних інструментів «G-shop». Реалізовані усі заплановані функції мобільного додатку.

СПИСОК ЛІТЕРАТУРИ

1. Mobile apps usage reached an all-time high amidst stay-at-home measures due to COVID-19 pandemic. URL: https://www.appannie.com/en/insights/market-data/mobile-app-usage-surged-40-during-covid-19-pandemic/
2. Олександр Окунєв, «посібник по Figma» (2019)
3. Dart language. URL: https://dart.dev/
4. Flutter SDK. URL: https://flutter.dev/
5. Stacked architecture. URL: https://www.filledstacks.com/
6. MVVM in flutter. URL: https://pub.dev/packages/stacked
7. Service locator. URL: https://pub.dev/packages/get_it
8. Injectable. URL: https://pub.dev/packages/injectable
9. Http client for Dart: URL: https://pub.dev/packages/dio
10. Stacked themes. URL: https://pub.dev/packages/stacked_themes
11. Firebase. URL: https://firebase.google.com/
12. Бен Штраубі, Скотт Чакон «Pro Git» 2-ге видання (2014)
13. GitHub Docs. URL: https://docs.github.com/en
14. Android Studio. URL: https://developer.android.com/studio

_____
Прізвище, ім’я, по-батькові

_____
Факультет

_____
Шифр і назва спеціальності

_____
Освітня програма

ДЕКЛАРАЦІЯ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (бакалаврська) робота на тему:

«_____
_____»

є написаною мною особисто.

Одночасно заявляю, що ця робота:

- не передавалась іншим особам і подається до захисту вперше;
- не порушує авторських та суміжних вправ, закріплених статтями 21-25 Закону України «Про авторське право та суміжні права»;
- не отримувались іншими особами, а також дані та інформація не отримувались у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна (бакалаврська) робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».

_____ _____
дата підпис здобувача