МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
(повне найменування університету)
ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
(назва факультету)
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
(повна назва кафедри)

Дипломна магістерська робота

на тему

«Розробка мобільних додатків за допомогою технологій Flutter та Dart»

Виконав: студент групи МгІТ-2-20

спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
Студент Гомілко О. О.
(прізвище та ініціали)

Керівник Демківська Т. І.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Київ 2021

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Спеціальність **122 Комп'ютерні науки**

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
_____проф. Щербань В.Ю.
"_____" ______2021 року

З А В Д А Н Н Я

НА ДИПЛОМНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Гомілку Олегу Олеговичу

1. Тема роботи ***Розробка мобільних додатків за допомогою технологій Flutter та Dart***

Науковий керівник роботи Демківська Тетяна Іванівна, к.т.н., доцент

затверджені наказом вищого навчального закладу від " _04_ " ____жовтня_ 2021 року № 286

2. Строк подання студентом роботи 21.12.2021 р.

3. Вихідні дані до роботи *Розробка кафедри комп'ютерних наук.*

4. Зміст дипломної роботи (перелік питань, які потрібно розробити) : РОЗДІЛ 1, Теоретична частина: актуальність мобільних додатків; РОЗДІЛ 2, Алгоритмічне забезпечення: застосування Flutter та Dart для розробки мобільних додатків; РОЗДІЛ 3, Програмне забезпечення: розробка додатку «Мій гаманець»; презентація дипломної магістерської роботи з основними результатами дослідження (в роздрукованому вигляді представлена у додатках).

6. Консультанти розділів дипломної магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	Демківська Т.І., доцент	20.09.21	
Розділ 2	Демківська Т.І., доцент	20.09.21	
Розділ 3	Демківська Т.І., доцент	20.09.21	
Висновки	Демківська Т.І., доцент	20.09.21	

7. Дата видачі завдання 20.09.2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної магістерської роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	25.09.2021	
2	Розділ 1. Теоретична частина	10.10.2021	
3	Розділ 2. Алгоритмічне забезпечення	20.10.2021	
4	Розділ 3. Програмне забезпечення	05.11.2021	
5	Висновки	15.11.2021	
6	Оформлення дипломної магістерської роботи (чистовий варіант)	20.11.2021	
7	Здача дипломної магістерської роботи на кафедру для рецензування (за 14 днів до захисту)	25.11.2021	
8	Перевірка дипломної магістерської роботи на наявність ознак плагіату (за 10 днів до захисту)	01.12.2021	
9	Подання дипломної магістерської роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	05.12.2021	

Студент _____ Гомілко О.О.
(підпис)

Науковий керівник роботи _____ Демківська Т. І.
(підпис)

Керівник відділу магістратури _____ _____
(підпис)

АНОТАЦІЯ

Гомілко О.О. ***Розробка мобільних додатків за допомогою технологій Flutter та Dart.***

Дипломна магістерська робота за спеціальністю 122 - «Комп'ютерні науки та технології» – Київський національний університет технологій та дизайну, Київ, 2021 рік.

Проведено аналіз можливостей технологій Flutter та Dart, їх переваг та недоліків при розробці додатків, а також було проведено порівняльний аналіз з існуючими аналогами. Вивчено та застосовано найголовнішу особливість розробки – кросплатформенність.

Розроблено мобільний кросплатформенний додаток «Мій гаманець», що надає можливість додавати та фіксувати витрати за тиждень, перегляди список покупок, які були здійснені, а також переглядати кошти, що були витрачені. Додаток стане у нагоді, якщо ви хочете краще контролювати свої витрати.

Ключові слова: Flutter, Dart, кросплатформенна розробка, Android, iOS, hot reload, мобільні додатки.

ANNOTATION

Homilko O.O. Mobile app using development Flutter and Dart technologies

Graduate Master's degree in specialty 122 - "Computer Science and Technologies" - Kyiv National University of Technology and Design, Kyiv, 2021.

The result of the research project is learning innovative technologies Flutter and Dart. Analysis of technologies possibilities, advantages and disadvantages during the development process and comparison analysis among existent analogs. Learning the most significant feature of development – cross platform ability.

Applying knowledge while creating a cross platform mobile app «My Wallet» which allows to add and save a list of purchases which were made during the week, review it all and be able to check on the money that were spent on specific products and days. It comes hands if you want to control your daily expences.

Keywords: Flutter, Dart, cross platform development, Android, iOS, hot reload, mobile apps.

ЗМІСТ

ВСТУП

Дана дипломна робота присвячена вивченню технологій Flutter та Dart для створення мобільних додатків.

Акцент зроблено на тому, щоб вивчити нові технології, дослідити переваги та недоліки відносно відомих аналогів.

Метою дипломної роботи є ознайомлення з мовою програмування Dart та фреймворком Flutter, опанування нових навичок та застосування отриманих знань.

Для досягнення цієї мети були поставлені наступні завдання:

1. Ознайомлення з документацією та опанування основ Flutter та Dart
2. Дослідження переваг та недоліків
3. Здобуття практичних навичок
4. Створення перших мобільних додатків

Розділ 1. Теоретична частина. Актуальність мобільних додатків

1.1 Невпинний розвиток технологій

Сьогодні технології невпинно розвиваються. Ми легко можемо пристосуватися до нових умови, автоматизувати звичні рішення, які ще не так давно здавались не можливими, якщо не робити це вручну. Навіть з широким поширення вірусу COVID-19, люди знайшли вихід і почали повністю працювати з дому, причому не тільки програмісти, але і бухгалтери, економісти, юристи та багато інших.

Технології настільки швидко розвиваються, що здивувати людство доволі складно. Як же сподобатись кінцевому користувачеві та привернути увагу до свого товару з ціллю збільшити попит? В еру нескінченних інформаційних війн буває дуже важко знайти необхідну інформацію та вирішення своєї проблеми, адже інформації, товару та пропозицій настільки багато, що заблукати та зробити неправильний вибір дуже легко. Саме цим і користуються прогресивні сучасні маркетологи, тому що переконати «блудного» юзера/покупця найлегше. У 2021 році людям хочеться бути у вирі подій, бути епіцентром кожної нової події або ж бути учасником кожної гучної прем’єри, тому що бути в курсі – це модно та престижно, бо як то кажуть: «Хто володіє інформацією, той володіє світом».

Пропозицій щодо товару сьогодні настільки багато, що ціна та пропозиція хоч і грають не останню роль, але є далеко не вирішальними факторами, коли доходять до фінальної стадії прийняття рішення.

Хочете бути актуальними? – вирізняйтесь, будьте неподібні ні на кого та ні що – і тоді точно привернете увагу, бо оригінальна, нестандартна та естетично приваблива картинка відіграє важливу роль, тому що ми живемо в епоху естетів, у тій чи інакшій формі, всі ми любимо творчість, саме тому будь-яку науку, справу чи то навіть приготування їжі сьогодні перетворюють у мистецтво. Підсумовуючи вище сказане, можна зробити висновок, що людям цікаве те, що милує їх око та

те, що вирізняється серед аналогів, а також те, що їм близьке, зрозуміле, просте та цікаве.

1.2 Ера мобільних застосунків

Сьогодні більшу частину свого життя можна помістити у компактний смартфон, адже величезний спектр потреб може задовольнити каталог додатків, що нам пропонують в Play Market або ж в Apple Store.

Додаток для знайомств онлайн, застосунок, що допомагає вести здоровий образ життя та підтримувати водний баланс, чи навіть банально гра-«стрілялка» - усе це може завжди бути під рукою та стати в нагоді при найменшій потребі.

Саме тому мобільні застосунки набирають такої популярності: це економить час та вирішує питання за лічені секунди роблячи наше життя простішим, цікавішим та інтерактивнішим.

Висновки

Розглянуто важливість та швидкість розвитку технологічного процесу та виявлено, що більшість потреб, як особистих, так і бізнес-рішень, часто задовольняються функціоналом, що надають сучасні мобільні додатки. Зроблено висновок про те, що мобільні додатки лише набиратимуть своєї популярності, а, отже, технології для їх розробки матимуть попит.

Розділ 2. Алгоритмічне забезпечення. Застосування Flutter та Dart для розробки мобільних додатків

Усе вище сказане ми імплементуємо у релевантну для нас тему – розробка мобільних додатків. Саме поєднання технологій Flutter та Dart надає можливість нам створювати прості, інтуїтивно зрозумілі, а головне привабливі застосунки, які ви точно захочете завантажити та використовувати та навіть більше, можливо, ви вже ними користуєтесь, просто не знаєте, що ваші улюблені застосунки, якими ви щодня користуєтесь, написані саме на Flutter.

2.1 Коротко про Flutter

Flutter – це безкоштовний фреймворк з відкритим кодом, що був створений компанією Google та випущений у травні 2017 року як аналог React Native від Facebook. У кількох словах, цей фрейморк дозволяє створювати нативні мобільні застосунки на основі лише однієї бази коду. Це означає, що ви можете використовувати одну мову програмування та одну кодову базу для створення двох різних додатків (для iOS та Android).

Flutter складається з <u>двох важливих частин:</u>

- SDK (Software Development Kit): набір інструментів, які допоможуть вам розробити ваші програми. Сюди входять інструменти для компіляції вашого коду в нативний машинного коду (код для iOS та Android).
- Framework (UI бібліотека на основі віджетів): набір елементів інтерфейсу користувачів (кнопки, поля введення тексту, повзунки тощо), які ви можете персоналізувати для власних потреб.

Застосунки, що створюються на основі Flutter використовуються мову програмування Dart. Мова була створена компанією Google у жовтні 2011 року.

Dart – це об’єктно-орієнтована, строго типізована мова програмування.

Якщо говорити про синтаксис, то це така собі суміш Java, Javascript та C#, тож якщо у вас був досвід програмування хоча б на одній з зазначених мов, то труднощів виникнути у вас не повинно.

Dart зосереджується на розробці інтерфейсу, і ви можете використовувати його для створення мобільних застосунків та веб-додатків.

2.2 Архітектура Flutter

Основна ідея побудови UI використовуючи Flutter – це побудова інтерфейсу за допомогою написання коду. Ви завжди будуєте дерево з віджетів у вашому застосунку. У вас не буде drag-and-drop інтерфейсу для додавання кнопок чи тексту на екран, який бачить юзер, натомість ви будете писати лише код.

Рис.1. Особливості роботи Flutter

Flutter також охоплює різні платформи (Android та iOS), тобто надає можливість створювати одночасно додаток для як для власників смартфонів на базі Android та і для власників айфонів при цьому пишучи лише один код. Працюючи над розробкою лише одного додатку ви маєте можливість створювати різні графічні інтерфейси в певних частинах застосунку, якщо в цьому є потреба.

2.3 Віджети

Віджетом називається абсолютно кожний елемент у Flutter.
Не важливо, чи це текст, кнопка, іконка або ж навіть поле для введення тексту – усі ці елементи є віджетами. Розглянемо приклад:

Рис.2. Віджети

Верхня панель – це віджет, що містить у собі інші менші віджети (дерево віджетів). Назви, поля для вводу, поле для прикріплення документів, кнопка для відправлення – усе це віджети. Абсолютно весь додаток буде побудований з віджетів, навіть уся сторінка є віджетом, та і весь додаток «загорнутий» у віджет. Що ж таке віджет? Це шматок коду, що виконує певну інструкцію, щоби відобразити потрібний елемент на екрані користувача.

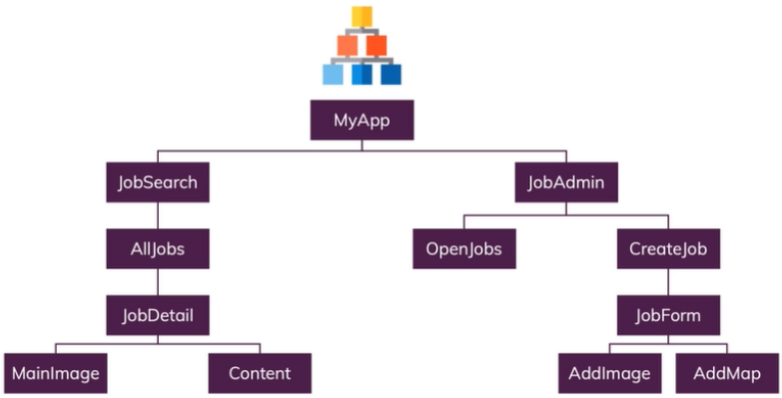


Рис.2. Дерево з віджетів

Використовуючи віджети, ми будуємо так зване дерево із віджетів, де коренем дерева є наш додаток, а його сини – це відповідно інші віджети, які будуть відповідати за те, щоб відобразити необхідну нам сторінку.

2.4 Компіляція Flutter/Dart у нативний додаток

Код, написаний на Dart, використовує Flutter фреймворк – це набір віджетів (вбудованих у Flutter, а також ваші власні), що необхідно скомпілювати для додатків на Android та iOS. Flutter компілює Dart код у нативний код для кожної із цих платформ за допомогою Flutter SDK.

Як результат ви отримаєте додаток для кожної з платформ на основі вашого коду. Flutter не використовує платформені примітиви. Наприклад, вам необхідно додати кнопку. Це не означає, що при компіляції Flutter створює нативний еквівалент кнопки для Android та iOS, натомість Flutter має власний механізм, що дозволяє контролювати на рендерити кожен піксель на екрані, що відображається користувачеві. Це надає Flutter повний контроль над інтерфейсом.

2.5 Порівняння з аналогами

Звичайно, що Flutter не єдиний інструмент, що дозволяє розробляти мобільні додатки використовуючи одну мову програмування, тому ми порівняємо його з уже відомими популярними інструментами.

Таблиця 1 - Порівняння технологій

Технологія	**Flutter**	**React Native**	**Ionic**
Основа	Flutter + Dart	JavaScript/React.js	Javascript
Результат	Скомпільований нативний додаток.	Частково скомпільований нативний додаток (бо є також частини в Javascript, які не компілюються, а переносяться в нативний додаток і працюють як Javascript, а не нативний код).	Нічого не компілюється. Ви отримуєте веб додаток, який «загорнений» в нативний додаток. Тому це не повноцінний нативний додаток, а лише оболонка, всередині якої знаходиться ваш веб

			застосунок. Перевагою такого підходу є те, що ви можете використовуватися звичні вам веб технології, щоб розробляти кросплатформені додатки, однак звідси випливає недолік, тому що це може негативно впливати на роботу додатку.
Особливості компіляції	Ми не компілюємо UI компоненти в Android чи iOS. Flutter самотужки контролює весь дисплей та кожен піксель, що відображається на ньому.	Відбувається компіляція в iOS та Android компоненти, тому скомпільовані частини – це інтерфейс користувача. Суттєвим недоліком цього є те, що через це у вас набагато менше інструментів для налаштувань. Наприклад: якщо ви не можете додати тінь до нативної iOS кнопки, то ви не зможете додати її і до React Native кнопки, тому що згодом вона мала б	Оскільки відсутня компіляція в нативний додаток, то ви можете легко стилізувати все так як і у будь-якому веб застосунку.

		скомпілюватись в iOS кнопку.	
Можливості	Дозволяє розробляти кросплатформені мобільні, веб та десктопні застосунки.	Дозволяє розробляти мобільні додатки (+React Native Web)	Дозволяє розробляти кросплатформені, мобільні, веб та десктопні застосунки.
Розробник	Google	Facebook	Ionic

2.6 Переваги

Швидке написання коду

Для Flutter розробників це означає швидший і динамічніший розвиток мобільних додатків. Ми можемо внести зміни в код і побачити їх відразу в додатку! Це так зване гаряче перезавантаження, яке зазвичай займає лише секунду і допомагає розробникам додавати функції, виправляти помилки та експериментувати швидше.

Гаряче перезавантаження також дуже зручне, коли ми хочемо вдосконалити або експериментувати із виглядом програми та перевіряти ефекти на місці. Іншими словами, з Flutter ваш дизайнер або тестер може працювати разом із розробником в інтерфейсі користувача, вносячи зміни, наприклад, "Поставте 2 пікселі правильно" або "Зробити анімацію швидше" - і побачити їх негайно.

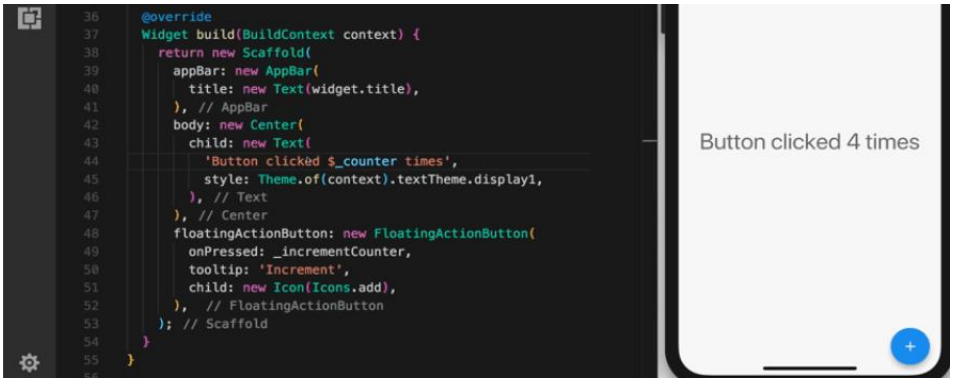


Рис.3. Hot-reload. До змін

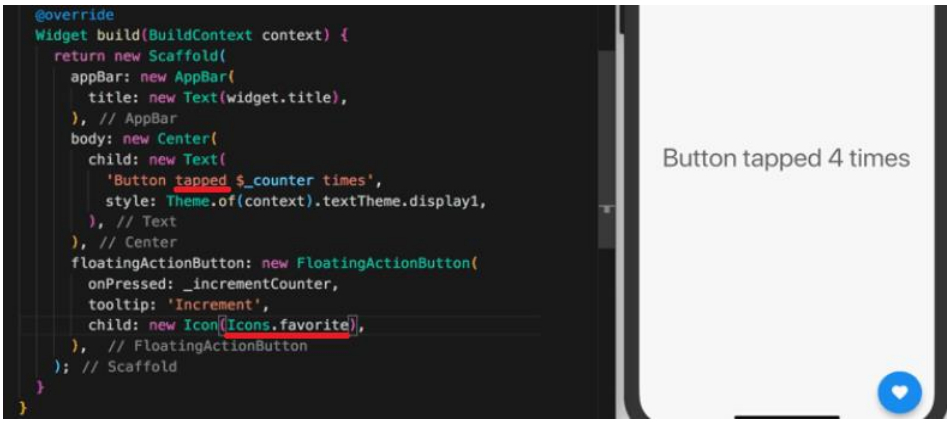


Рис.4. Hot-reload. Відразу відображені зміни без необхідності перезапуску додатку

Один код для двох платформ

Розробники пишуть лише одну кодову базу охоплюючи платформи Android та iOS. Flutter не залежить від платформи, оскільки має власні віджети та дизайни. Це означає, що у вас однаковий додаток на двох платформах. Але що важливо – це те, що якщо ви хочете розмежувати свої додатки, то це можна легко зробити.

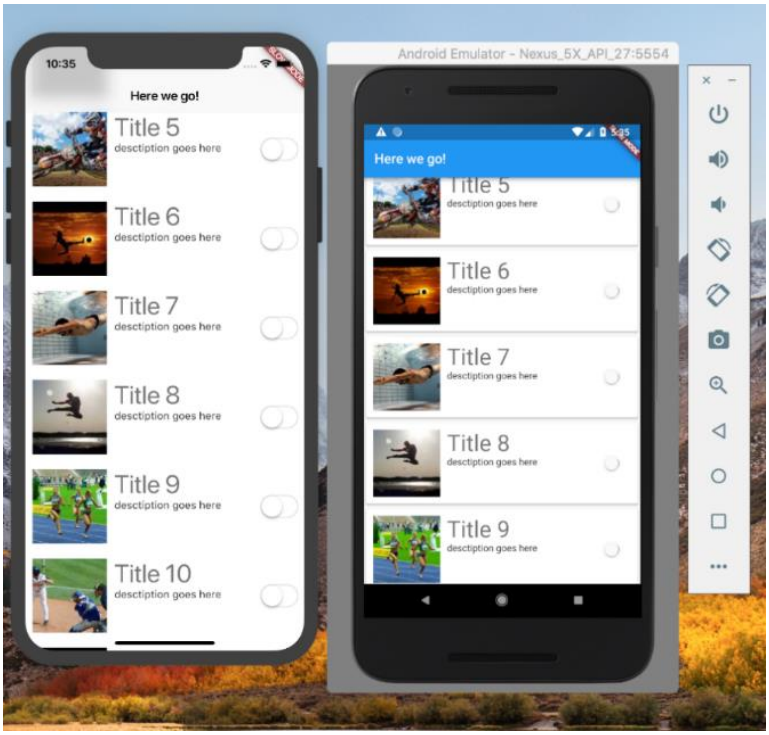


Рис.6.Можливість створювати додаток для різних платформ

Причому, розробляти настільки зручно, що можна вносити певні зміни до інтерфейсу в залежності від платформи у будь-який потрібний момент розробки

додатку не вносячи глобальних змін та не витрачаючи часу на те, щоб писати два різних варіанти коду на кожному кроці.

Менше тестування

Якщо у вас однаковий додаток для двох платформ – це означає менше тестування! Процес забезпечення якості може бути швидшим. Через одну кодову базу розробники пишуть автоматичні тести лише один раз. Більше того, фахівці із забезпечення якості мають менше роботи, тому що у них є лише одна програма для перевірки. Однак, якщо ваші програми мають деякі відмінності, їх потрібно перевірити на обох платформах.

Дизайн, який сподобається користувачам

Красиві та унікальні дизайни для ваших віджетів можна створити, використовуючи Flutter. Ви також можете налаштувати свої віджети відповідно до вашої потреби. Flutter розроблений так, щоб спростити створення власних віджетів або налаштування наявних віджетів. Flutter містить два набори віджетів. Віджети Material Design для імплементації мови дизайну Google та Cupertino Widget для імітації дизайну Apple iOS від Apple.

Інтерфейс підтримується на старих версіях

Ваш новий додаток буде виглядати так само, навіть на старих версіях Android та iOS. Додаткові витрати на підтримку старих пристроїв не потрібні. Flutter працює на Android Jelly Bean або новіших, а також на iOS 8 або новіших.

Висока продуктивність

На ефективність програми впливає безліч факторів, включаючи використання процесора, номер кадру в секунду, середній час відповіді, номер запиту в секунду тощо. Flutter пропонує постійні 60 кадрів в секунду, це швидкість, з якою сучасні екрани відображають гладку і чітку картину.

Розробники намагаються тримати рух на цьому рівні, оскільки будь-яке відставання в цій частоті кадрів може бути визначене людським оком. У порівнянні з React Native та Xamarin, цей фреймворк лідирує з 220-мілісекундним часом запуску та 58 кадрами в секунду.

Доступність та інтернаціоналізація

Унаслідок відстоювання інклюзивності та різноманітності, Google пропонує інтегровані можливості розробити додатки, доступ до яких може отримати широкий спектр користувачів. Зазвичай, коли вам потрібно, щоб ваш додаток працював у різних регіонах та підтримував різні мови, ви хочете підготувати свій код для локалізованого вмісту, який зазвичай створюється пізніше. Цей процес називають інтернаціоналізацією.

Flutter для мобільних розробок пропонує віджети, що базуються на пакеті Dart intl, що робить цей процес більш простим. Зараз він підтримує 24 мови, а також одиниці вимірювання, параметри компонування, валюти та дати.

Відсутність проблем із сумісністю

Усі віджети та їх рендери є частиною програми, а не платформи. Для забезпечення сумісності з пристроями iOS та Android немає необхідності в будь-яких додаткових бібліотеках. Однак є деякі обмеження. Запуск Flutter можливий на 64-розрядних пристроях iOS та всіх пристроях Android вище 4.4 або 4.1. з наданням програмного забезпечення.

Легкість у вивченні

Якщо вивчити Dart просто, то ознайомитись із цим інструментом буде ще простіше. Багато людей з невеликим досвідом розробки можуть розробляти прототипи та програми з самого початку. Крім того, вам не потрібен досвід мобільної розробки.

Більше того, Google відомий своєю структурованою та детальною документацію.

Dart – це просто та ефективно.

Dart - проста і ефективна мова, орієнтована на програмістів Java.

Dart - це сучасна об'єктно-орієнтована мова, яка нагадує Java або C# своїм синтаксисом. Підтримує як сильні, так і слабкі стилі друку, що спрощує підбір для початківців.

XML-файли не потрібні. У розробці на Android робота розділена на макет і код. Макет повинен бути записаний у XML у вигляді Views, на який потім поси-

лається код Java. Dart подбає про це, зберігаючи макет і код в одному місці. Оскільки все у Flutter є віджетом, макет створюється також у Dart.

Open-source

Flutter – це інструмент з відкритим кодом, що означає, що він має незліченну кількість можливостей налаштувати майже все у фреймворку: від віджетів Material та Cupertino до анімації та жестів.

2.7 Недоліки

Нова мова

Незважаючи на те, що Dart - це легка мова для вивчення, вона все ж є мовою для вивчення. Ось чому, оскільки Flutter є не так давно на міжнародній арені, перші кроки можуть бути складними для тих, хто шукає певної онлайн-допомоги та підтримки від громади.

Проблеми з iOS

Оскільки фреймворк Flutter для мобільних розробок створила компанія Google, розробники можуть хвилюватися щодо імплементації на iOS. Оскільки Google має прямий інтерес до швидкого виправлення помилок, створення Android-програм на Flutter є приємним та швидким.

Налаштування IPhone були створені, щоб забезпечити можливості віджетів Cupertino. Але ці та інші функції дизайну були оновлені пізніше та базувалися на функціях iOS 10, хоча iOS 11 вже був випущений на той момент часу. Отже, не ясно, чи будуть оновлення продовжувати випускатися так само швидко, як і версії Android.

Масивний розмір файлу

Розробники роблять усе можливе, щоб зменшити розмір програми. Щоб мінімізувати розмір коду, програмісти зазвичай уникають анімації, стискають зображення та зменшують кількість пакетів та бібліотек.

Фреймворк дуже розчарував розробників після того, як додаток «Hello, World!» зайняв 6,7 МБ. Навіть коли він був знижений до 4,7 Мб, він залишився значно більшим за Kotlin, який становить 550 КБ, і Java, що становить 539 КБ.

Ніяких сторонніх бібліотек

Сторонні пакети та бібліотеки відіграють значну роль в автоматизації розробки програмного забезпечення для розробників і звільняють їх від потреби запрограмовувати все з самого початку. Ці бібліотеки, як правило, з відкритим кодом, попередньо протестовані та легко доступні. Для більшості популярних та старих технологій отримати необхідний пакет легко.

Однак, оскільки Flutter є відносно новим, знайти такі безкоштовні пакети та бібліотеки непросто. Офіційний ресурс безкоштовних пакетів все ще вдосконалюється, а список його інструментів все ще зростає. Отже, вам доведеться почекати, перш ніж вирішити використовувати його для довгострокового розвитку.

2.8 Приклад додатку – “Hello, World!”

Розглянемо приклад стандартного додатка “Hello, World!” з якого, зазвичай, і починають вивчення нової мови програмування, щоб краще познайомитись з особливостями нових технологій.

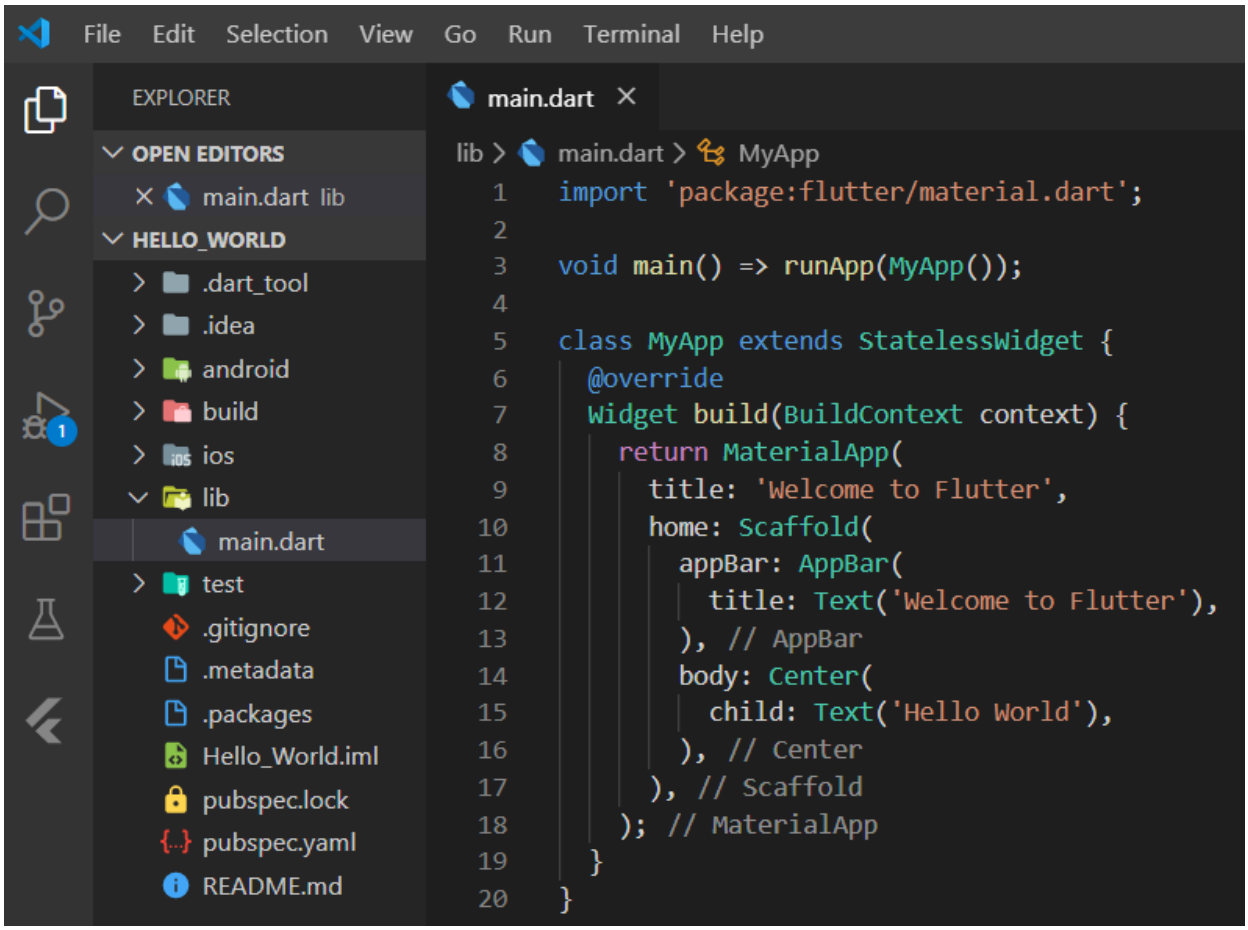


Рис.5. Лістинг до програми "Hello, World!"

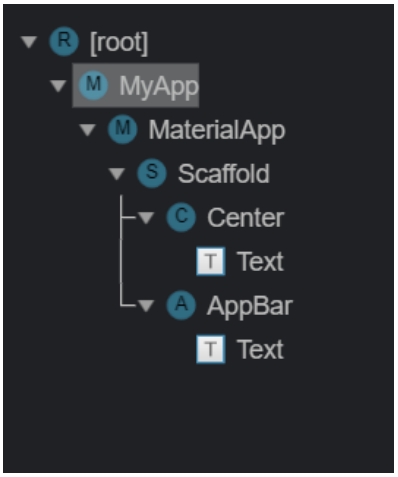


Рис.6. Widget Tree додатку "Hello, World"

Розглянемо дерево віджетів нашого додатку та розберемо наведений вище лістинг.

Перша за все потрібно імпортувати пакет material.dart, адже він і дозволяє нам використовувати усі віджети, що наявні у коді (ми будемо використовувати завжди!)

MyApp – це наш додаток і, одночасно, віджет, адже, як було сказано раніше, у Flutter усе є віджетом, навіть сам додаток.

MaterialApp – віджет, що являє собою загальну «оболонку», яка охоплює усі елементи-віджети на екрані та допомагає їх рендерити.

Scaffold – віджет, що надає нашому застосунку звичайного вигляду; тобто, він створює пусту сторінку, що по дизайну інтуїтивно пояснює, що за платформа, на якій ми запускаємо додаток.

Далі віджет Scaffold має безліч аргументів, ми можемо переглянути всі доступні (шорткат: ctrl + space), у нашому випадку це AppBar та Body, оскільки, ці два параметри – це аргументи одного і того ж самого віджета Scaffold, то вони розташовані на одному рівні у дереві віджетів.

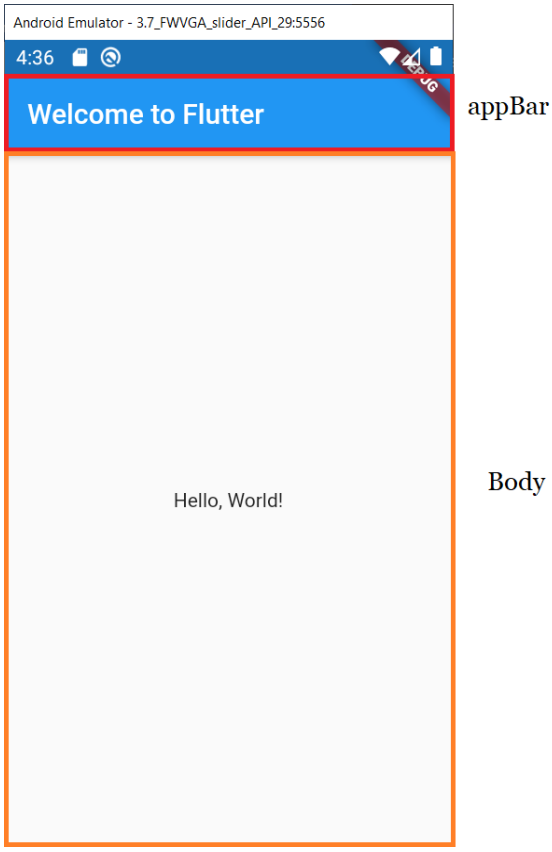


Рис.7. Додаток "Hello, World!"

На скріншоті можна чітко побачити як реалізовано додаток візуально.

По суті, якщо не брати до уваги віджетів, що мають більш функціональне навантаження (MyApp, MaterialApp, Scaffold), то саме візуальні складові, що Flutter відрендерив – це віджети Appbar та Center.

Віджети **AppBar** та **Center** – це віджети, що є вкладеними у віджет Scaffold як його властивості і саме це робить Flutter дуже гнучним та простим у використанні, адже в основі цього проса істина: один віджет може містить в собі інший, який містить у собі ще декілька віджетів, які потім і складають собою сукупність віджетів – дерево віджетів.

Розглянемо детальніше:

AppBar – це віджет, що рендерить верхню панель (яка так і називається: AppBar), ми також використали властивість title цього віджета, яка у свою чергу приймає інший віджет Text.

Center – це віджет, який відповідає за те, щоб одразу розмістити те, що буде всередині нього по центру, оскільки він має нащадка – віджет Text, що приймає звичайний String, то його вміст буде вирівняно по центру (див. рисунок).

Висновки

У цьому розділі ми розглянули основний структурний елемент Flutter – віджет. Поєднання віджетів утворює дерево віджетів, що відповідає структурі додатку. Виконали порівняльну характеристику Flutter з існуючими аналогами, зрозуміли істотні відмінності фреймворку, а, також, розглянули його переваги та недоліки. Продемонстрували отримані навички на найпершому базисному додатку – “Hello, World!”.

Розділ 3. Програмне забезпечення. Розробка додатку «Мій гаманець»

Для більш наглядної демонстрації можливостей Flutter та Dart, було вирішено розробити додаток. Головна мета полягає в тому, щоб написати кросплатформений додаток (використовуючи одну кодову базу, адже це і є одною з ключових переваг Flutter), продемонструвати роботу різних віджетів, їх різновиди (різні платформи), візуальну складову, можливості поєднувати віджети та створювати свої власні. Також, важливим пунктом є те, щоб наш додаток був адаптивним та зручним для користування.

3.1 Ідея розробки додатку

Для демонстрації можливостей Flutter було вирішено створити інтуїтивно не складний, але, одночасно, і корисний додаток, який, як мінімум, зацікавив би користувачів. Саме тому було вирішено розробити додаток «Мій гаманець», що допоможе підрахувати витратити юзера.

3.2 Схема додатку

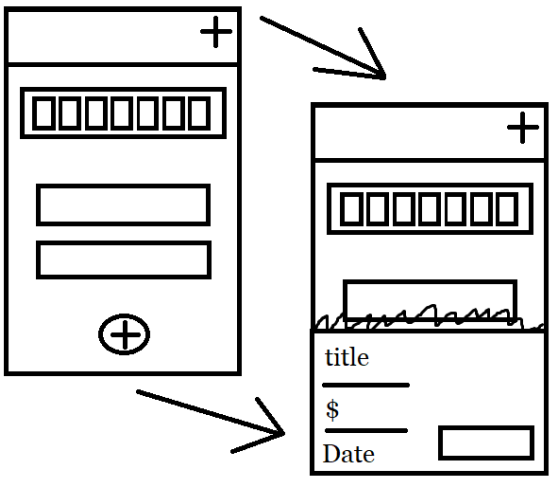


Рис.8. Схема додатку "Мій гаманець"

На рисунку схематично зображено майбутній додаток.

У нас буде бар, що міститиме кожен із останніх семи днів тижня та показуватиме рівень витрат у співвідношенні поточний день до загальних витрат.

Нижче цього бару, буде перелік усіх наших витрат (які ми уже внесли в додаток), у нас буде сума, дата та назва кожної покупки, яку ми внесли.

Також, на верхній панелі та внизу екрану, буде кнопка +, що дозволятиме додавати нову покупку.

При натисканні на неї, у нас буде з’являтися панель, заповнивши яку (усе ті ж назва, сума, дата покупки), можна буде додати покупку до загального списку, де вона і з’явиться, а витрати будуть підраховані та візуально вигляд бару відповідного дня зміниться.

3.3 Реалізований додаток

Нижче наведено скріншоти реалізованого додатку.

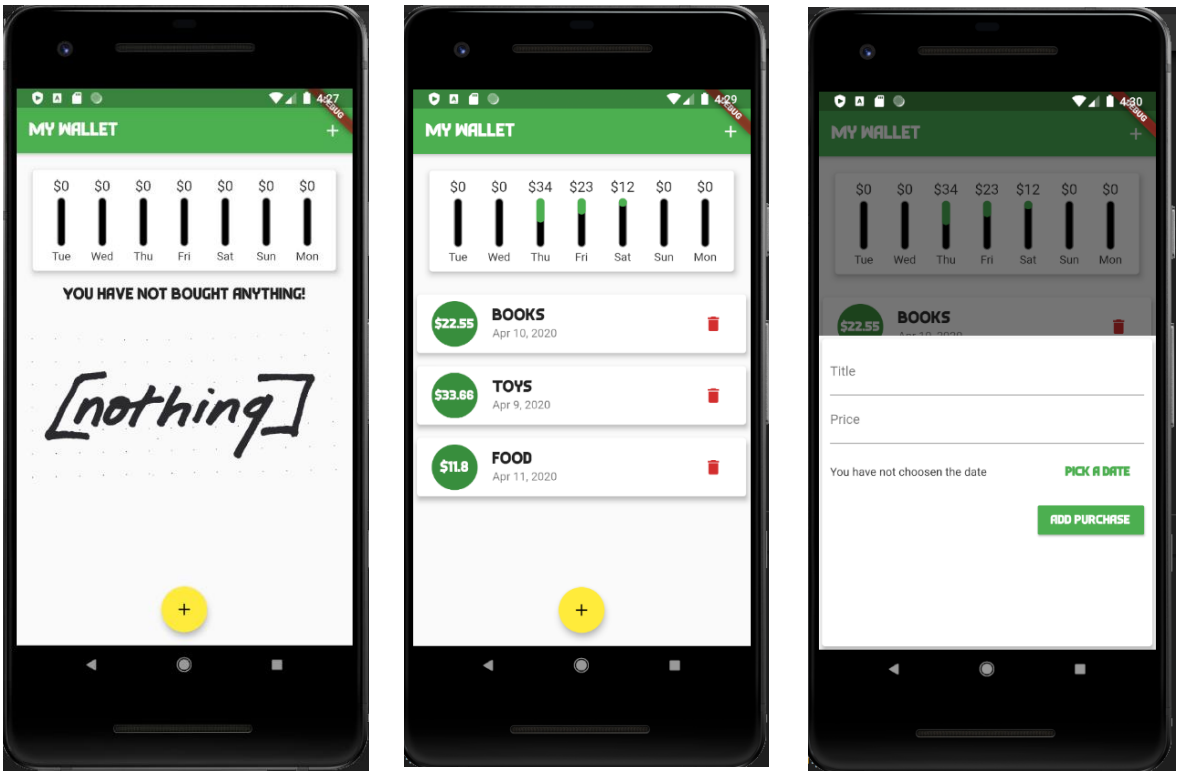


Рис.9. Додаток "Мій гаманець"

На першому рисунку ми можемо побачити додаток у «дефолтному» стані, коли ще не було нічого додано.

На другому рисунку ми можемо бачити роботу нашого «бару», та як він заповняється відповідно до доданих покупок.

На третьому рисунку, ми можемо побачити як ми можемо додавати нову покупку до нашого списку.

Обов’язкові елементи нашого додатку: у нас є бар, на якому є «лічильник», що демонструє витрати за останні сім дні (це обраховується від моменту запуску, алгоритм працює так, що відображає останні сім днів, починаючи від поточного дня тижня, коли було запущено додаток), якщо витрат немає, то всі бари пусті, а список витрат відсутній.

Якщо у нас додані витрати, то ми можемо бачити список витрат нижче блок з нашими барами, ми можемо бачити дату, ціну та назву кожної із покупок, також, у нас є можливість видалити уже існуючу покупку, що була додана.

Кнопка +, що розташована внизу екрану та на верхній панелі в правому кутку відповідає за те, щоб додавати нові покупки до нашого списку. Результат її роботи можна побачити на третьому рисунку, після того, як ви її натиснете, з’явиться панель, де ви повинні ввести назву покупки, ціну, а також вибрати день, коли вона була здійснена. Ця покупка буде додана лише за умови, якщо всі поля є заповненими.

3.4 Принцип побудови віджетів

Щоб краще зрозуміти принцип роботи Flutter та те як він допомагає побудувати наш додаток, розглянемо приклади побудови деяких елементів нашого додатку.

```dart
@override
Widget build(BuildContext context) {
  return Card(
    elevation: 6,
    margin: EdgeInsets.all(20),
    child: Padding(
      padding: EdgeInsets.all(10),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        children: weekPurchase.map((data) {
          return Flexible(
            fit: FlexFit.tight,
            child: IndicatorBar(
              data['day'],
              data['amount'],
              totalSum == 0.0
                  ? 0.0
                  : (data['amount'] as double) / totalSum,
            ), // IndicatorBar
          ); // Flexible
        }).toList(),
      ), // Row
    ), // Padding
  ); // Card
}
}
```

Рис.10. Лістинг віджету "IndicatorsChart"

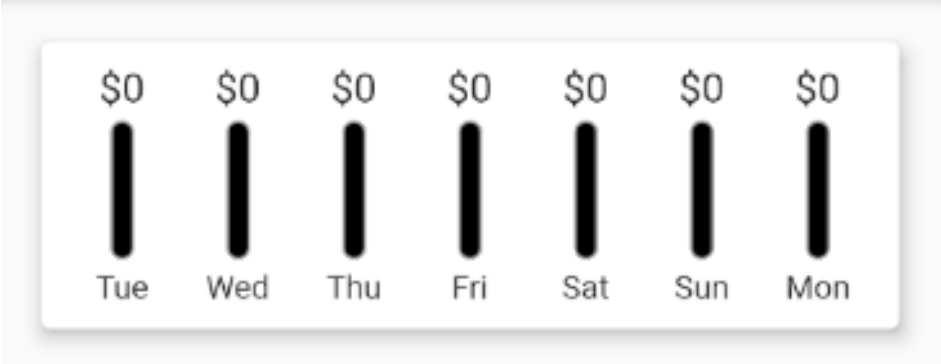


Рис.11. Результат роботи коду

Увесь наш чарт – це віджет Card, розглянемо усі параметри та віджети, що визначені у ньому.

Elevation – це тінь, що визначена у розмірі 6-ти пікселів.

Margin – це наш відступ, що, у нашому випадку, визначений для всіх сторін однаково, у розмірі 20 пікселів, саме тому, наша картка відмежована зі всіх боків.

Нащадком (child) нашого віджету Card є віджет Padding, у нього визначені елементи padding (що віддаляє зі всіх боків від країв нашої картки «бари-показники»). Нащадком (child) нашого віджету Padding є віджет Row (був обраний саме цей віджет, оскільки усі наші «бари» розташовані в ряд, один за одним). Для цього віджету визначені наступні елементи: параметри mainAxisAligment (а саме варіант spaceAround, що дозволяє утворити однакові відстані між кожним

елементом віджету). Нащадком (child) нашого віджету Row є віджет Flexible. Чому був обраний саме такий віджет? Він допомагає візуально естетично красиво розмістити усі його складові за допомогою параметру fit: FlexFit.tight.

Цей віджет буде викликаний 7 разів (адже як ми можемо побачити у наведеному вище лістингу, він викликається для кожного елементу, що повертає метод weekPurchase, який у свою чергу повертає list map, де зберігаються пари: день тижня та витрати, що були зроблені у цей день). Нащадком цього віджета є інший віджет – ChartBar. Це також користувацький віджет, що був визначений нами.

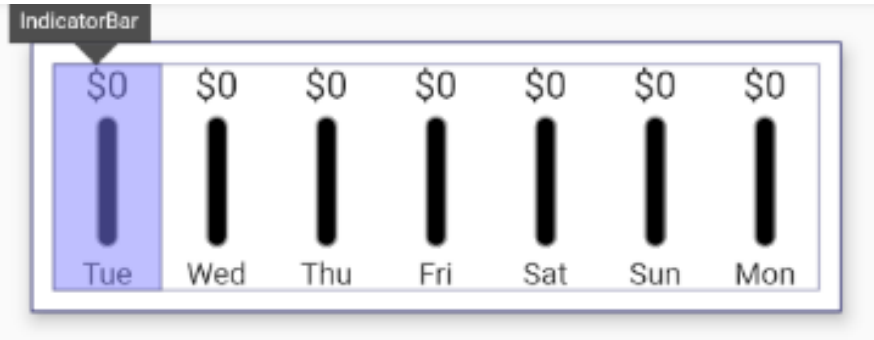


Рис.12. Віджет "IndicatorBar"

Розглянемо складові віджета «IndicatorBar».

```
@override
Widget build(BuildContext context) {
  return LayoutBuilder(
    builder: (ctx, constraints) {
      return Column(
        children: <Widget>[
          Container(
            height: constraints.maxHeight * 0.15,
            child: FittedBox(
              child: Text('\$${totalSum.toStringAsFixed(0)}'),
            ), // FittedBox
          ), // Container
          SizedBox(
            height: constraints.maxHeight * 0.05,
          ), // SizedBox
          Container(
            height: constraints.maxHeight * 0.6,
            width: 10,
            child: Stack(
              children: <Widget>[
                Container(
                  decoration: BoxDecoration(
                    border: Border.all(color: Colors.grey, width: 1.0),
                    color: Color.fromRGBO(220, 220, 220, 1),
                    borderRadius: BorderRadius.circular(10),
                  ), // BoxDecoration
                ), // Container
                FractionallySizedBox(
                  heightFactor: percentageOfTotal,
                  child: Container(
                    decoration: BoxDecoration(
                      color: Theme.of(context).primaryColor,
                      borderRadius: BorderRadius.circular(10),
                    ), // BoxDecoration
                  ), // Container
                ), // FractionallySizedBox
              ], // <Widget>[]
            ), // Stack
          ), // Container
          SizedBox(
            height: constraints.maxHeight * 0.05,
          ), // SizedBox
          Container(
            height: constraints.maxHeight * 0.15,
            child: FittedBox(
              child: Text(text),
            ), // FittedBox
          ), // Container
        ], // <Widget>[]
      ); // Column
    },
  ); // LayoutBuilder
}
```

Рис.13. Лістинг віджету "IndicatorBar "

Перш за все, весь наш віджет – це вбудований віджет **LayoutBuilder**, який зручний у цій ситуації за рахунок того, що він дозволяє задати параметри, які будуть підлаштовувати розмір вмісту цього віджету залежно від розмірів його батьківського елементу (тобто від віджету Flexible), що і задається наступним чином builder: (ctx, constrains); ці параметри задають обмеження про які було сказано раніше для наступного віджета **Column**. Чому ми використовуємо саме цей віджет?

Пригадаємо, зараз ми будуємо бар для кожного із останніх 7-ми днів тижня, рахуючи від поточного. Як ми можемо побачити на скріншоті, наш бар включає в себе: суму, що була витрачена в цей день, індикатор (показник витрат від загальної суми за тиждень), а також назву дня тижня. Оскільки ці всі елементи розташовані на одному рівні, то найзручніше використовувати віджет Column, адже він дозволяє зберегти таке положення елементів.

Віджет **Column** має декількох нащадків (масив) про які було сказано вище. Перший з них – це віджет контейнер, який містить нащадка віджет **FittedBox**, який у свою чергу містить нащадком віджет **Text**, у який ми і передаємо загальну суму витрат за поточний день. Навіщо було обгортати текст стількома віджетами замість того, щоб відразу його відобразити? Все просто, ми хочемо контролювати розміщення на екрані та те, скільки місця займатиме певний віджет, саме тому у віджеті **Container** визначено наступний параметр height: constrains.maxHeight * 0.15, який визначає, що поточний віджет і всі його складові (у нас це наш текст з сумою витрат), будуть займати лише 15% від загального розміру віджета, а відштовхуємося ми від constrains, тому що це саме те обмеження, що визначає розміри залежно від батьківського віджета. Саме це і забезпечить динамічний рендинг нашого додатку (саме в цьому місці – конкретно віджету **ChartBar**) незалежно від розміру девайсу, на якому він використовуватиметься.

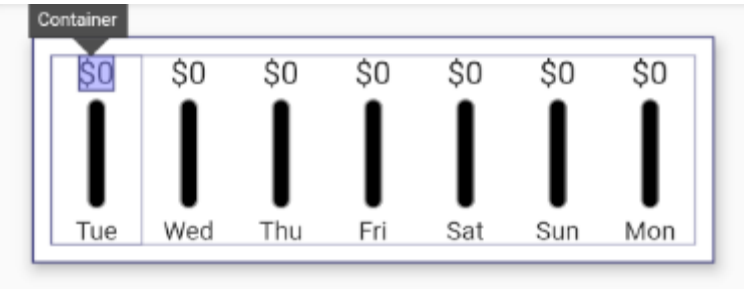


Рис.14. Віджет Container

Наступний нащадок – віджет SizedBox, для якого визначено лише те, що він займатиме 5% від загального розміру всього віджету: height: constrains.maxHeight * 0.05, тобто, це пусте місце, такий собі невидимий віджет, який додано лише для того, щоб гарно розмежувати складові нашого віджету Column.

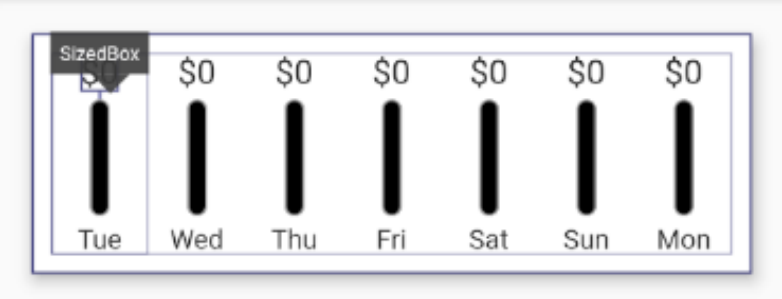


Рис.15. Віджет SizedBox

Наступний нащадок – віджет Container. Ми знову використовуємо метод обгортки, щоб за таким же принципом визначити, що нам бар займатиме 60% нашого віджету Column.

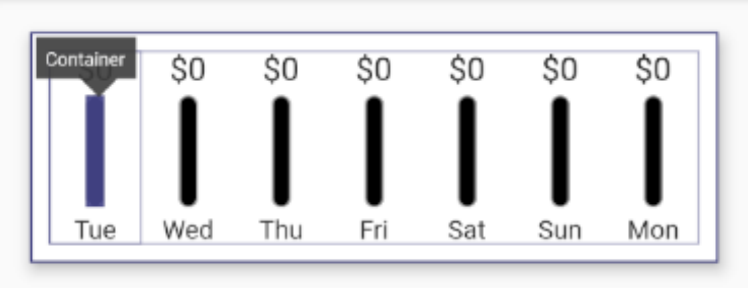


Рис.16. Віджет Container – 1

Нащадком цього віджету є віджет Stack. Чому саме Stack? В основі цього віджету структура даних – стек, де кожен новий елемент накладається поверх іншого, допоки стек не заповниться. Так само і тут, у нас існує два бари, що визначені двома віджета Container та деякими стильовими параметрами, які відомі нам з CSS. У випадку, якщо витрати =! 0, то новий (кольоровий) бар накладається поверх порожнього, щоб відобразити витрати за день.

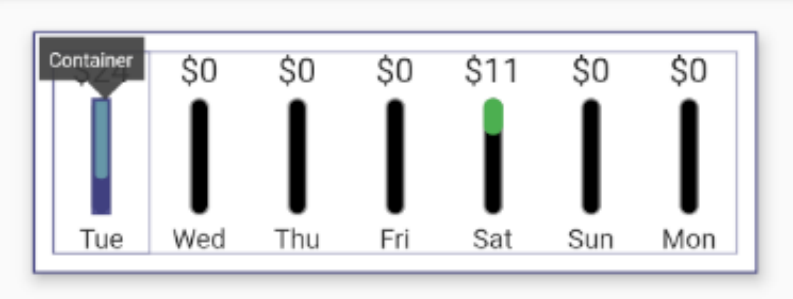


Рис.17. Віджет Container – 2

Наступним нащадком віджета – є ще один SizedBox, який, знову ж таки, для красивого візуального відображення; він так само займає лише 5% від загально площі вільного місця.

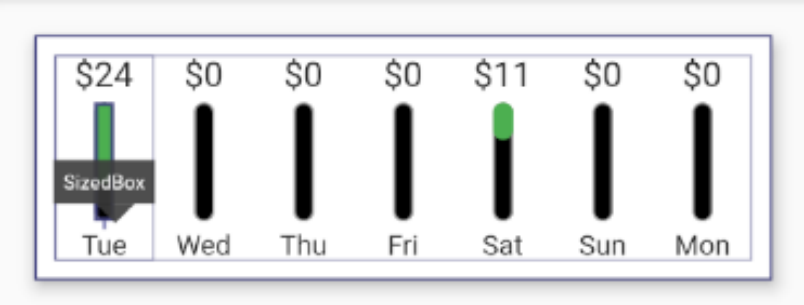


Рис.18. Віджет SizedBox

Останнім нащадком – є знову віджет Container, який визначає, що цей віджет займатиме 15% від загальної площі. А його нащадком є віджет FittedBox, що містить нащадка Text, який і приймає текст – назву дня.

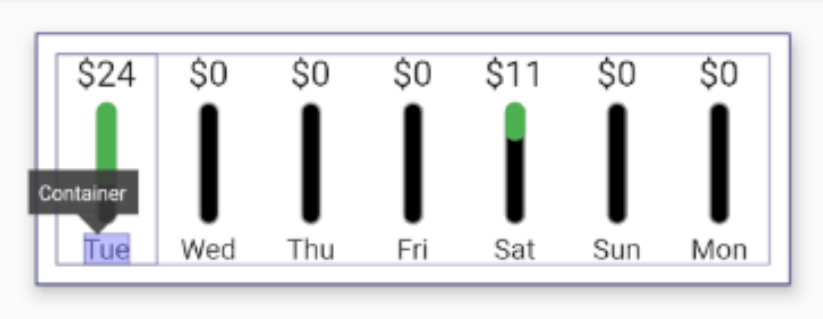


Рис.19. Віджет Container – день тижня

Якщо тепер уважно переглянути, то додавши розміри усіх визначених віджетів, то отримаємо 100% загалом, тобто ми динамічно визначали розмір кожного із віджетів відповідно до розмірів екрану, що надає наш девайс, це і дозволить рендерити наш додаток на девайсі будь-якого розміру та при цьому зберігати привабливу візуальну складову.

Важливо розуміти те, що є два види віджетів – Stateless та Stateful, тобто ті, що мають декілька станів (можуть змінюватися) та статичні. Наприклад, наш віджет Chart – буде Stateless віджетом, оскільки він постійно буде перемальовуватись на нашому екрані. З додаванням кожної нової покупки, наш індикатор змінюватиметься, сума за витрати збільшуватиметься або ж зменшуватиметься, якщо

існуюча покупка була видалена. Щоб сказати Flutter про те, що певний віджет потрібно перемалювати, потрібно використовувати метод SetState, у якому ми і «інформуємо» Flutter про нові зміни.

Розібравшись з принципом побудови віджетів, можна побачити, що будувати додаток у Flutter легко та цікаво, ми використовуємо як вбудовані віджети, так і визначаємо наші власні, які, у свою чергу, використовують так само як вбудовані віджеті, так і ті, які були визначені нами. Це процес можемо повторюватися стільки, скільки нам потрібно. Гарною практикою у Flutter вважається розбиття усіх елементів на більш дрібні віджети з метою забезпечити більшу гнучкість у керуванні додатком, а також, щоб мати можливість краще відслідковувати поведінку роботи нашого застосунку. Також, така практика є гарною через те, що нам легке відслідкувати зміни та рендерити лише певні елементи на екрані заново, при необхідності, замість того, щоб відмальовувати весь додаток з самого початку (дбаймо про оптимізацію).

3.5 Адаптивність

Основною перевагою у використанні Flutter є те, що пишучи лише один код, ми можемо створити додаток як для Android, так і для iOS. Однак, поки що наш додаток написаний лише для Android платформи (у звичному вигляді). Звісно, ми можемо запустити цей додаток і для iOS, але такий вигляд дещо не властивий для iOS додатків.

Саме тому, ми використаємо пакет Cupertino, що містить в собі віджети, які властиві для iOS додатків. Тепер у місцях, де нам потрібно вставлятися наші віджети ми внесемо незначні зміни. За допомогою методу Platform.isIOS ми перевірятимемо на якій платформі запущений наш додаток, якщо це Android – використовуватимемо віджет із пакету Material, якщо це iOS, то з Cupertino.

Розглянемо наш додаток, що буде запущено на iOS платформі:

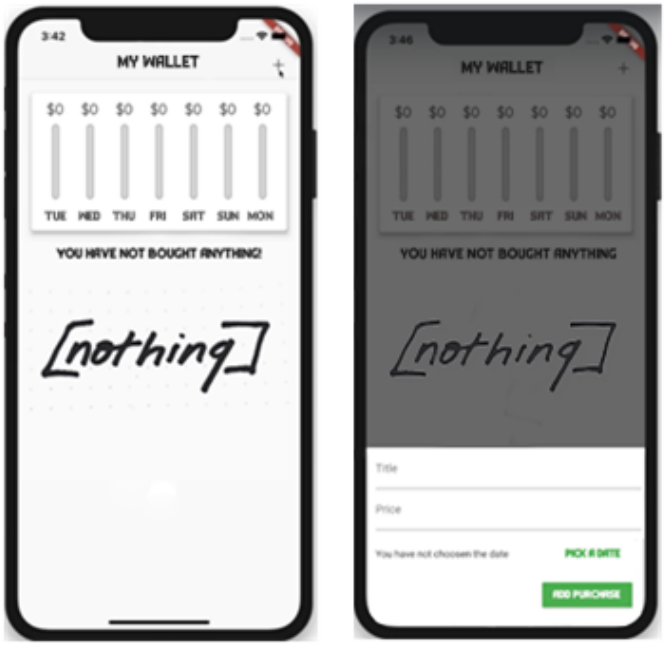


Рис.20. Додаток на iOS платформі

Ми можемо побачити, що тепер додаток має вигляд, що є властивим для iOS платформи. Відсутній верхній бар, відсутня кнопка внизу (що властиво лише для Android), усі кнопки та поля відповідно такі, якими ми звикли їх бачити на екрані айфонів.

3.6 Зручність у використанні

Можна вважати, що наш додаток готовий та працює, однак, є ще одна річ, яку ми трішки покращимо з метою, аби наш додаток можна було використовувати трішки інакше, зручніше та універсальніше.

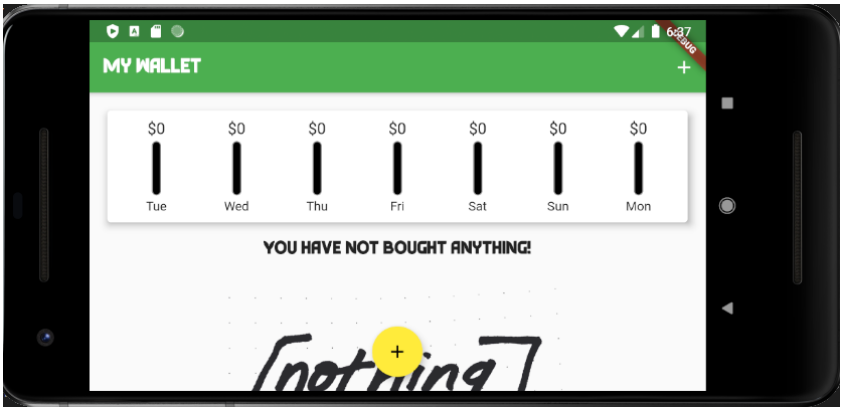


Рис.21. Додаток у landscape режимі

Візуально сприймається непогано, проте сторінка виглядає нагромадженою, поміщається лише дві покупки, а скролинг усіх покупок дуже незручний.

Саме тому, за допомогою вбудованого функціоналу, ми будемо визначати, яка орієнтація у нашого девайсу, і, відповідно до цього, виводити різний контент на екрані.

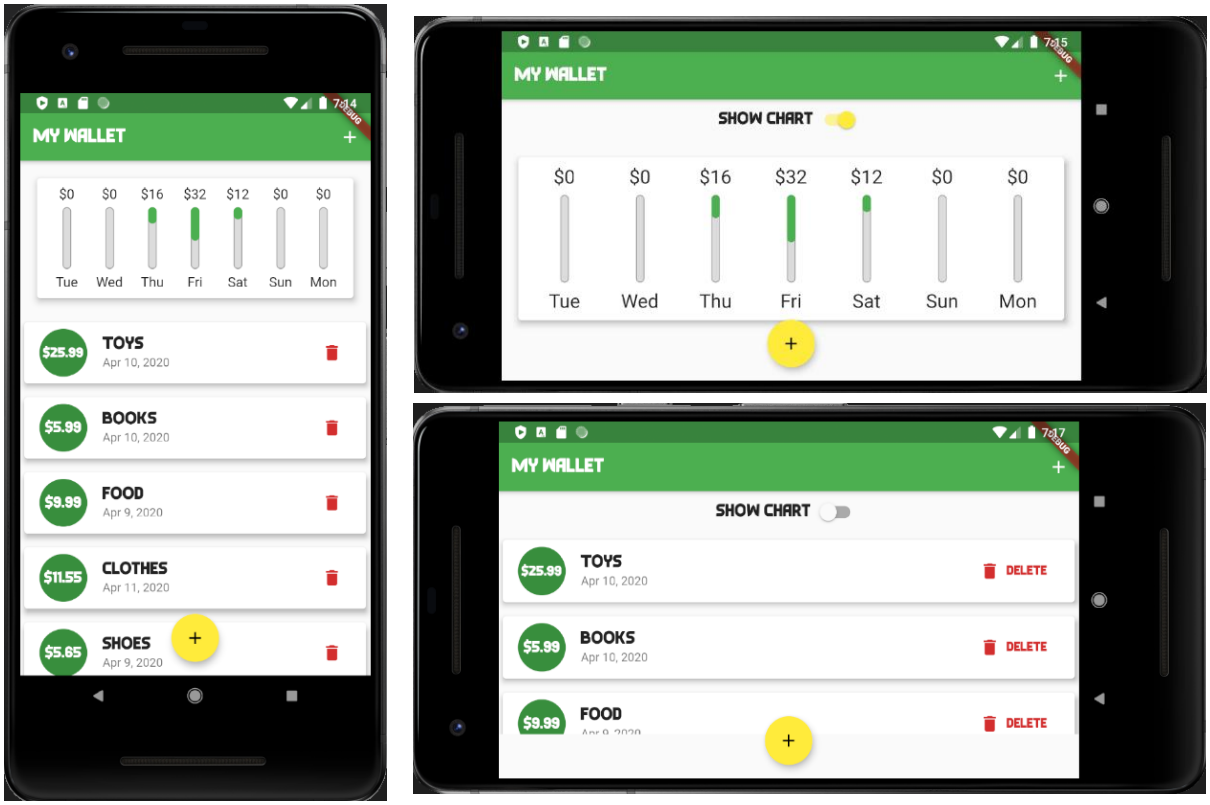


Рис.22. Зміни на девайсі при різному положенні екрану

Була додана перевірка, яка власне визначає положення екрану, якщо девайс у вертикальному положенні, то виводиться на екран і чарт з показниками, і усі покупки. Якщо ж екран у горизонтальному положенні, тоді з'являється важіль, який відображає контент на екрані в залежності від свого положення. Якщо важіль ввімкнено, то ми можемо бачити лише чарт з нашими індикаторами, які займають увесь кран, якщо ж важіль вимкнено, тоді на екрані буде лише перелік із покупок. Звісно, що у будь-який момент часу можна перейти у будь-який із режимів, це було зроблено для більш зручного відображення вмісту додатку при горизонтальному режимі, адже так інформації більше та вона краще відображається на екрані.

3.7 Стилі та форматування

У Flutter дуже легко налагоджувати усі налаштування відносно стилів: колір тексту, кнопок, шрифт, розмір та інше. Увесь процес дуже схожий з CSS, однак замість того, щоб визначати це все в окремому файлі, а також підключати його, це все робиться в main файлі нашого додатку.

```
theme: ThemeData(
    primarySwatch: Colors.green,
    accentColor: Colors.yellow,
    fontFamily: 'TypeSauce',
    textTheme: ThemeData.light().textTheme.copyWith(
        title: TextStyle(
          fontFamily: 'TypeSauce',
          fontSize: 18,
        ), // TextStyle
        button: TextStyle(color: Colors.white),
      ),
    appBarTheme: AppBarTheme(
      textTheme: ThemeData.light().textTheme.copyWith(
            title: TextStyle(
              fontFamily: 'TypeSauce',
              fontSize: 20,
            ), // TextStyle
          ),
    )), // AppBarTheme // ThemeData
```

Рис.23. Лістинг коду, де задаються стилі

Саме в цій частині задаються головні налаштування відносно графічної складової. Це дуже зручно, адже якщо раптом ми захочемо змінити кольорову гаму нашого додатку, то це можна зробити за лічені секунди.

Наприклад, ми хочемо, щоб наш додаток був у червоно-чорних тонах. Замінимо дві стрічки:

Таблиця 2 – Зміни в коді

Було	Стало
primarySwatch: Color.green,	primarySwatch: Color.red,
accentColor: Colors.yellow.	accentColor: Colors.black.

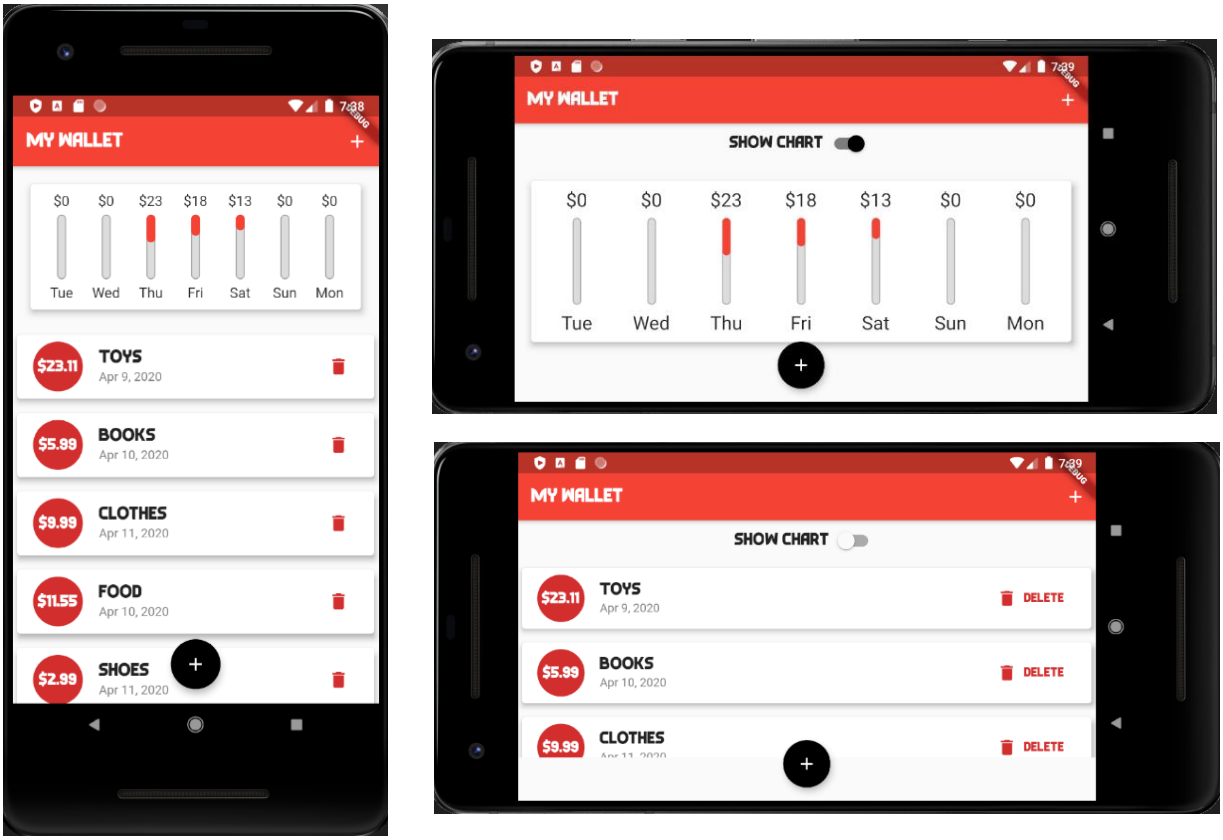


Рис.24. Додаток після певних змін у коді

3.8 Можливості відлагоджування

Як у всіх мовах програмування, так і у Dart є набір інструментів, що дозволяють відстежувати помилки. Особливим інструментом для відлагодження є Dart: DevTools. Запустити його можна у режимі дебаг.

Цей інструмент є доволі цікавим, він надає ряд переваг при розробці додатку.

У цьому режимі ви можете відслідкувати дерево віджетів починаючи від найпершого і до кінця, включно з абсолютно усіма нащадками, це надає краще розуміння принципу роботи віджетів та те, як один віджет використовує інший.

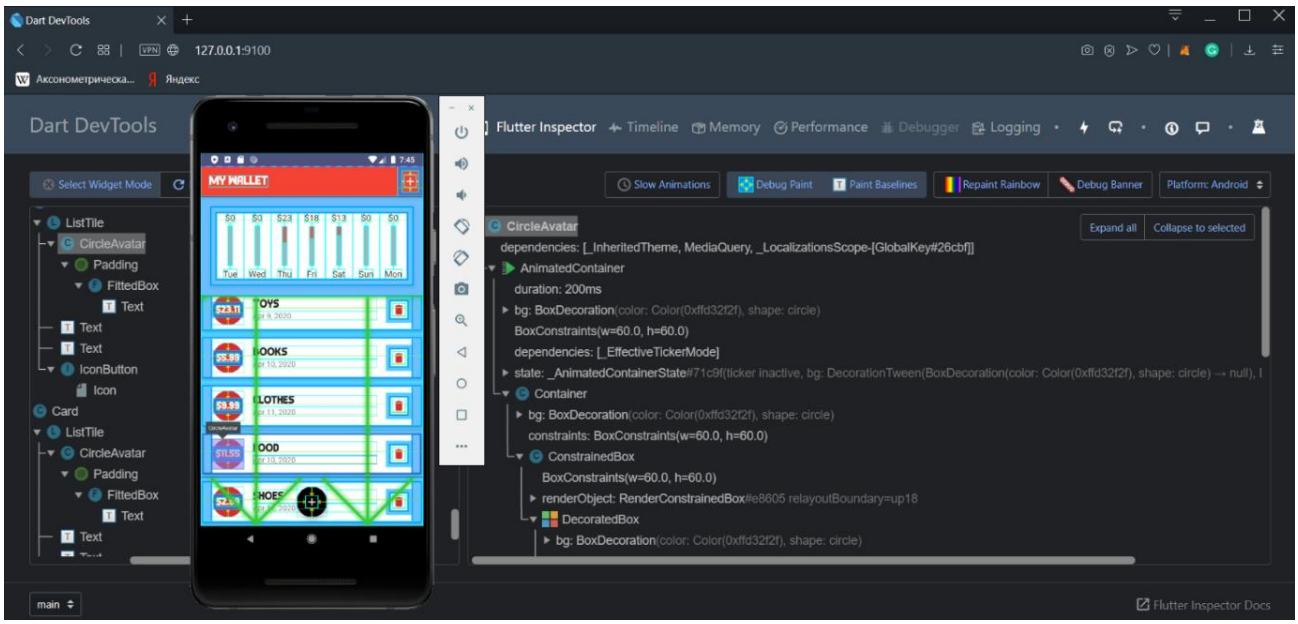


Рис.27. Dart: DevTools

Увімкнувши режим Select Widget Mode, ви, при натиску на будь-який віджет на екрані, потрапляєте на відповідний віджет у дереві віджетів. Зліва у вас буде обраний віджет та його дерево (його батько та нащадки, якщо вони є), а також позиція у загальному дереві, а справа ви зможете побачити усі властивості цього віджету.

Вибравши режим Debug Point та Paint Baselin ви зможете побачити «розмітку» додатку, межі віджетів та взагалі краще зрозуміти як віджети рендеряться на екран.

Також, приємним бонусом є те, що можна забрати набридливу стрічку Debug Banner.

Загалом, цей інструмент є унікальним, від допомагає краще зрозуміти як Flutter використовує віджети, що таке дерево віджетів, як воно будується та як віджети відображаються на екрані.

Також, можна легко дослідити властивості та поведінку будь-якого із віджетів. Дуже корисно мати можливість не тільки відлагодити логіку роботи додатку, але і його графічну складову, що і дозволяє зробити Dart: DevTools.

Висновки

У результаті проведеного дослідження розроблено комплексний додаток «Мій гаманець», розглянуто принципи побудови віджетів, використано «вкладе-

ні» віджети та наочно продемостровано, що таке дерево віджетів. Зроблено висновок про те, як легко можна зробити додаток адаптивним, юзер-френдлі та легко змінювати його стиль. Також, було продемонстровано можливості відлагодження, які надає Flutter у разі виникнення труднощі під час розробки.

Висновки

У результаті проведеного дослідження було розглянуто нову технологію розробки мобільних додатків за допомогою фреймворку Flutter від компанії Google. Було розглянуто принцип роботи інструменту у поєднанні з мовою програмування Dart.

Виявлено переваги та недоліки нового фреймоворку, а також здійснено порівняльну характеристику відносно відомих аналогів, у наслідок чого було зроблено висновки, що Flutter переважає в тому, що дозволяє створювати біль гнучкі додатки, контролює та відмальовує кожен піксель, а також не залежить від платформи для якої розробляється додаток.

Розроблено додаток на прикладі якого було наочно досліджено особливості роботи фреймворку, механізми розробки мобільних додатків для обох платформ Android та iOS на основі єдиної кодової бази, також було наочно розглянуто особливості відображення контенту на екрані залежно від орієнтації пристрою задля зручнішого користування додатком.

Розглянуто особливості графічної розробки, можливості використання вбудованого функціоналу, а також використання вбудованого функціоналу для створення власних віджетів, які є ядром при розробці додатку на Flutter.

Проведено оцінку роботи функціоналу фреймворку для відлагодження роботи графічної складової та загальної роботи додатку. Dart: DevTools дозволяє легко віднаходити «проблемні» місця під час побудови графічного інтерфейсу за допомогою віджетів, а також допомагає дослідити властивості будь-якого із віджетів на екрані та побачити загальне дерево віджетів додатку.

Літературні джерела

1. Flutter [Електронний ресурс] // flutter.dev - 2017-2021. - Режим доступу до ресурсу: https://docs.flutter.dev/
2. Термоса С. Про Flutter, кратко, основы [Електронний ресурс] / Термоса С. – 2018. - Режим доступу до ресурсу: https://habr.com/ru/post/430918/
3. Feoktistoc I. Top 8 Flutter Advantages and Why You Should Try Flutter on Your Next Project [Електронний ресурс] / Feoktistoc I. - Режим доступу до ресурсу: https://relevant.software/blog/top-8-flutter-advantages-and-why-you-should-try-flutter-on-your-next-project/
4. What is Flutter? Benefits and limitations [Електронний ресурс] // codemagic.io. – 2018.- Режим доступу до ресурсу: https://blog.codemagic.io/what-is-flutter-benefits-and-limitations/
5. Flutter Tutorial for Beginners: [Електронний ресурс] // https://www.youtube.com/Academind. – 2018. - Режим доступу до ресурсу: https://www.youtube.com/watch?v=GLSG_Wh_YWc&list=PL55RiY5tL51qKxC472MY2ayxJTze5Qb7T
6. Leler W. Why Flutter Uses Dart [Електронний ресурс] // hackernoon.com – 2018. - Режим доступу до ресурсу: https://hackernoon.com/why-flutter-uses-dart-dd635a054ebf
7. Kuppusamy K.S. What makes Dart an easy, scalable and multi-purpose programming language: [Електронний ресурс] // opensourceforu.com. – 2017. – Режим доступу до ресурсу: https://www.opensourceforu.com/2017/06/dart-easy-scalable-multi-purpose-programming-language/
8. Гуляева А. Как начать работать с Flutter: [Електронний ресурс] // apptractor.ru – 2018 – Режим доступу до ресурсу: https://apptractor.ru/develop/coding/kak-nachat-rabotat-s-flutter.html
9. Napoli M. Beginning Flutter: A Hand on Guide to App Development / Marco Napoli, 2019. – 528 с.
10. Biessek A. Flutter for Beginners / Alessandro Biessek, 2019. – 512 с.

11. Flutter [Електронний ресурс] // flutter.github.io. - 2017 - Режим доступу до ресурсу: https://flutter.github.io/samples/#
12. Rap P. Beginning App Development with Flutter: Create Cross-Platform Mobile Apps / Rap Payne, 2019. – 309 с.
13. Moore K. Flutter Apprentice // K. Moore, M. Katz, V. Ngo, 2019. – 508 с.
14. Miola A. Flutter Complete Reference: Create beautiful, fast and native apps for any device // Miola Albero, 2020. – 408 с.
15. Windmill E. Flutter in Action // Eric Windmill, 2019. – 255 с.
16. Freitas E. Flutter Succinctly // Ed Freitas, 2019. – 129 с.
17. Mainkar P. Google Flutter Mobile Development Quick Start Guide: Get up and running with iOS and Android mobile app development // P. Mainkar, S. Giordano, 2019. – 152 с.
18. Clow M. Learn Google Flutter Fast: 65 Example Apps // Mark Clow, 2019. – 476 с.
19. Zammetti F. Practical Flutter: Improve your Mobile Developemnt with Google’s Latest Open-Source SDK // Frank Zammetti, 2019. – 416 с.
20. Zaccagnino C. Programming Flutter: Native, Cross-Platform Apps the Easy Way // Carmine Zaccagnino, 2020. – 275 с.
21. Balbaert I. Dart Cookbook // Ivo Balbaert, 2014. – 246 с.
22. Sikore M. Dart Essentials // Martin Sikora, 2015. – 234 с.
23. Strom C. Dart for Hipsters // Chris Strom, 2014. – 144 с.
24. Buckett C. Dart in Action // Chris Buckett, 2013. – 424 с.
25. Mitchell D. Dart: Scalable Application Development // D. Mitchell, S. Akopkokhyants, I. Balbaert, 2017. – 1298 с.
26. Walrath K. Dart: Up and Running // K. Walrath, S. Ladd, 2012. – 152 с.
27. Akopkokhyants S. Mastering Dart // Sergey Akopkokhyants, 2014. – 346 с.
28. Bracha G. The Dart Programming Language // Gilad Bracha, 2015. – 224 с.
29. Belchin M. Web Programming with Dart // M. Belchin, P. Juberias, 2014. – 472 с.

30. Sells C. What’s new in Flutter 2.2. [Електронний ресурс] / Sells Chris – 2021. - Режим доступу до ресурсу: https://medium.com/flutter/whats-new-in-flutter-2-2-fd00c65e2039

ДОДАТОК 1
Код програми

main.dart

```
import 'dart:io';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import './models/purchase.dart';
import './widgets/new_purchase.dart';
import './widgets/purchase_list.dart';
import './widgets/indicatorsChart.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'My Wallet',
      theme: ThemeData(
          primarySwatch: Colors.green,
          accentColor: Colors.yellow,
          fontFamily: 'TypeSauce',
          textTheme: ThemeData.light().textTheme.copyWith(
                title: TextStyle(
                  fontFamily: 'TypeSauce',
                  fontSize: 18,
                ),
                button: TextStyle(color: Colors.white),
              ),
          appBarTheme: AppBarTheme(
            textTheme: ThemeData.light().textTheme.copyWith(
                  title: TextStyle(
                    fontFamily: 'TypeSauce',
                    fontSize: 20,
                  ),
                ),
          )),
      home: MyHomePage(),
    );
  }
}
```

```
class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  final List<Purchase> _userPurchases = [];
  bool _showChart = false;

  List<Purchase> get _purchases {
    return _userPurchases.where((pr) {return pr.date.isAfter(DateTime.now().subtract(Duration(days: 7),),);}).toList();
  }

  void _toAddNewPurchase(
    String prTitle, double prPrice, DateTime pickedDate) {
    final newPr = Purchase(title: prTitle, price: prPrice, date: pickedDate, id: DateTime.now().toString(),);
    setState(() {_userPurchases.add(newPr);
    });
  }

  void _beginNewPurchase(BuildContext cpr) {
    showModalBottomSheet(
      context: cpr,
      builder: (_) {
        return GestureDetector(
          onTap: () {},
          child: NewPurchase(_toAddNewPurchase),
          behavior: HitTestBehavior.opaque,
        );
      },
    );
  }

  void _deletePurchase(String id) {
    setState(() {_userPurchases.removeWhere((pr) => pr.id == id);});
  }

  @override
  Widget build(BuildContext context) {
    final mediaQuery = MediaQuery.of(context);
    final isLandscape = mediaQuery.orientation == Orientation.landscape;
    final PreferredSizeWidget appBar = Platform.isIOS
```

```
    ? CupertinoNavigationBar(
        middle: Text(
          'My Wallet',
        ),
        trailing: Row(
          mainAxisSize: MainAxisSize.min,
          children: <Widget>[
            GestureDetector(
              child: Icon(CupertinoIcons.add),
              onTap: () => _beginNewPurchase(context),
            ),
          ],
        ),
      )
    : AppBar(
        title: Text(
          'My Wallet',
        ),
        actions: <Widget>[
          IconButton(
            icon: Icon(Icons.add),
            onPressed: () => _beginNewPurchase(context),
          ),
        ],
      );
final prListWidget = Container(
  height: (mediaQuery.size.height -
          appBar.preferredSize.height -
          mediaQuery.padding.top) *
      0.7,
  child: PurchaseList(_userPurchases, _deletePurchase),
);
final pageBody = SafeArea(
  child: SingleChildScrollView(
    child: Column(
      // mainAxisAlignment: MainAxisAlignment.start,
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: <Widget>[
        if (isLandscape)
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Text(
                'Show Chart',
                style: Theme.of(context).textTheme.title,
```

```
            ),
            Switch.adaptive(
              activeColor: Theme.of(context).accentColor,
              value: _showChart,
              onChanged: (val) {
                setState(() {
                  _showChart = val;
                });
              },
            ),
          ],
        ),
        if (!isLandscape)
          Container(
            height: (mediaQuery.size.height -
                  appBar.preferredSize.height -
                  mediaQuery.padding.top) *
                0.3,
            child: IndicatorsChart(_purchases),
          ),
        if (!isLandscape) prListWidget,
        if (isLandscape)
          _showChart
              ? Container(
                  height: (mediaQuery.size.height -
                        appBar.preferredSize.height -
                        mediaQuery.padding.top) *
                      0.7,
                  child: IndicatorsChart(_purchases),
                )
              : prListWidget
      ],
    ),
  ),
);
return Platform.isIOS
    ? CupertinoPageScaffold(
        child: pageBody,
        navigationBar: appBar,
      )
    : Scaffold(
        appBar: appBar,
        body: pageBody,
        floatingActionButtonLocation:
            FloatingActionButtonLocation.centerFloat,
```

```
            floatingActionButton: Platform.isIOS
                ? Container()
                : FloatingActionButton(
                    child: Icon(Icons.add),
                    onPressed: () => _beginNewPurchase(context),
                  ),
          );
  }
}
```

purchase.dart

```
import 'package:flutter/foundation.dart';

class Purchase {
  final String id;
  final String title;
  final double price;
  final DateTime date;

  Purchase({
    @required this.id,
    @required this.title,
    @required this.price,
    @required this.date,
  });
}
```

adaptive_button.dart

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:flutter/cupertino.dart';

class AdaptiveFlatButton extends StatelessWidget {
  final String text;
  final Function function;

  AdaptiveFlatButton(this.text, this.function);

  @override
  Widget build(BuildContext context) {
    return Platform.isIOS
        ? CupertinoButton(
            child: Text(
```

```
            text,
            style: TextStyle(
              fontWeight: FontWeight.bold,
            ),
          ),
          onPressed: function,
        )
      : FlatButton(
          textColor: Theme.of(context).primaryColor,
          child: Text(
            text,
            style: TextStyle(
              fontWeight: FontWeight.bold,
            ),
          ),
          onPressed: function,
        );
  }
}
```

indicator_bar.dart

```
import 'package:flutter/material.dart';

class IndicatorBar extends StatelessWidget {
  final String text;
  final double totalSum;
  final double percentageOfTotal;

  IndicatorBar(this.text, this.totalSum, this.percentageOfTotal);

  @override
  Widget build(BuildContext context) {
    return LayoutBuilder(
      builder: (ctx, constraints) {
        return Column(
          children: <Widget>[
            Container(height: constraints.maxHeight * 0.15, child: FittedBox(child:
Text('\$${totalSum.toStringAsFixed(0)}'),
              ),
            ),
            SizedBox(height: constraints.maxHeight * 0.05,),
            Container(height: constraints.maxHeight * 0.6, width: 10, child: Stack(
              children: <Widget>[
                Container(
```

```
                decoration: BoxDecoration(
                  border: Border.all(color: Colors.grey, width: 1.0),
                  color: Color.fromRGBO(220, 220, 220, 1),
                  borderRadius: BorderRadius.circular(10),
                ),
              ),
              FractionallySizedBox(
                heightFactor: percentageOfTotal,
                child: Container(
                  decoration: BoxDecoration(
                    color: Theme.of(context).primaryColor,
                    borderRadius: BorderRadius.circular(10),
                  ),
                ),
              ),
            ],
          ),
        ),
        SizedBox(height: constraints.maxHeight * 0.05,
        ),
        Container( height: constraints.maxHeight * 0.15, child: FittedBox(child:
Text(text),
          ),
        ),
      ],
    );
  },
 );
 }
}
```

indicatorsChart.dart

```
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import './indicator_bar.dart';
import '../models/purchase.dart';

class IndicatorsChart extends StatelessWidget {
 final List<Purchase> recentPurchases;

 IndicatorsChart(this.recentPurchases);

 List<Map<String, Object>> get weekPurchase {
  return List.generate(7, (index) {
```

```
    final weekDay = DateTime.now().subtract(Duration(days: index),);
    var sumT = 0.0;

    for (var i = 0; i < recentPurchases.length; i++) {
      if (recentPurchases[i].date.day == weekDay.day && recentPurchases[i].date.month == weekDay.month && recentPurchases[i].date.year == weekDay.year) {
        sumT += recentPurchases[i].price;
      }
    }
    return {'day': DateFormat.E().format(weekDay).substring(0, 3), 'amount': sumT};}).reversed.toList();
    }

  double get totalSum {
    return weekPurchase.fold(0.0, (sum, item) {return sum + item['amount'];});
  }

  @override
  Widget build(BuildContext context) {
    return Card(
      elevation: 6,
      margin: EdgeInsets.all(18),
      child: Padding(
        padding: EdgeInsets.all(12),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceAround,
          children: weekPurchase.map((data) {
            return Flexible(
              fit: FlexFit.tight,
              child: IndicatorBar(
                data['day'],
                data['amount'],
                totalSum == 0.0
                    ? 0.0
                    : (data['amount'] as double) / totalSum,
              ),
            );
          }).toList(),
        ),
      ),
    );
  }
}
```

new_purchase.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import '../widgets/adaptive_button.dart';

class NewPurchase extends StatefulWidget {
  final Function addPurchase;

  NewPurchase(this.addPurchase);

  @override
  _NewPurchaseState createState() => _NewPurchaseState();
}

class _NewPurchaseState extends State<NewPurchase> {
  final _tlController = TextEditingController();
  final _priceController = TextEditingController();
  DateTime _datePicked;

  void _submitData() {
    if (_priceController.text.isEmpty) {return;}
    final titleEntered = _tlController.text;
    final priceEntered = double.parse(_priceController.text);

    if (titleEntered.isEmpty || priceEntered <= 0 || _datePicked == null) {return;}

    widget.addPurchase(titleEntered, priceEntered, _datePicked,);
    Navigator.of(context).pop();
  }

  void _presentDatePicker() {
    showDatePicker(context: context, initialDate: DateTime.now(), firstDate:
DateTime(2020), lastDate: DateTime.now(),).then((pickedDate) {
      if (pickedDate == null) {return;}
      setState(() {_datePicked = pickedDate;});});
  }

  @override
  Widget build(BuildContext context) {
    return SingleChildScrollView(
      child: Card(
        elevation: 7,
```

```
child: Container(
  padding: EdgeInsets.only(
    top: 12,
    left: 12,
    right: 12,
    bottom: MediaQuery.of(context).viewInsets.bottom + 10,
  ),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.end,
    children: <Widget>[
      TextField(
        decoration: InputDecoration(labelText: 'Title'),
        controller: _tlController,
        onSubmitted: (_) => _submitData(),
      ),
      TextField(
        decoration: InputDecoration(labelText: 'Price'),
        controller: _priceController,
        keyboardType: TextInputType.number,
        onSubmitted: (_) => _submitData(),
      ),
      Container(
        height: 70,
        child: Row(
          children: <Widget>[
            Expanded(
              child: Text(
                _datePicked == null
                    ? 'You have not chosen the date!'
                    : 'Picked Date: ${DateFormat.yMd().format(_datePicked)}',
              ),
            ),
            AdaptiveFlatButton('Pick a Date', _presentDatePicker)
          ],
        ),
      ),
      RaisedButton(
        child: Text('Add Purchase'),
        color: Theme.of(context).primaryColor,
        textColor: Theme.of(context).textTheme.button.color,
        onPressed: _submitData,
      ),
    ],
  ),
),
```

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

purchase_list.dart

```
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';
import '../models/purchase.dart';

class PurchaseList extends StatelessWidget {
  final List<Purchase> Purchases;
  final Function deletePr;

  PurchaseList(this.Purchases, this.deletePr);

  @override
  Widget build(BuildContext context) {
    return Purchases.isEmpty
        ? LayoutBuilder(builder: (ctx, constraints) {
            return Column(
              children: <Widget>[
                Text(
                  'You have not bought anything!',
                  style: Theme.of(context).textTheme.title,
                ),
                SizedBox(
                  height: 20,
                ),
                Container(
                    height: constraints.maxHeight * 0.5,
                    child: Image.asset(
                      'assets/images/nothing.png',
                      fit: BoxFit.cover,
                    )),
              ],
            );
          })
        : ListView.builder(
            itemBuilder: (ctx, index) {
              return Card(
                elevation: 7,
                margin: EdgeInsets.symmetric(
                  vertical: 8,
```

```
            horizontal: 5,
          ),
          child: ListTile(
            leading: CircleAvatar(
              radius: 30,
              child: Padding(
                padding: EdgeInsets.all(7),
                child: FittedBox(
                  child: Text('\$${Purchases[index].price}'),
                ),
              ),
            ),
            title: Text(
              Purchases[index].title,
              style: Theme.of(context).textTheme.title,
            ),
            subtitle: Text(
              DateFormat.yMMMd().format(Purchases[index].date),
            ),
            trailing: MediaQuery.of(context).size.width > 450
                ? FlatButton.icon(
                    icon: Icon(Icons.delete),
                    label: Text('Delete'),
                    textColor: Theme.of(context).errorColor,
                    onPressed: () => deletePr(Purchases[index].id),
                  )
                : IconButton(
                    icon: Icon(Icons.delete),
                    color: Theme.of(context).errorColor,
                    onPressed: () => deletePr(Purchases[index].id),
                  ),
          ),
        );
      },
      itemCount: Purchases.length,
    );
  }
}
```

ДОДАТОК 2
Копії публікацій за темою магістерської роботи

V Міжнародна науково-практична конференція «Мехатронні системи: інновації та інжиніринг» | Інновації та інжиніринг мехатронних, електротехнічних та електромеханічних систем

УДК 519.246.8(075.8)

ІННОВАЦІЙНІ РІШЕННЯ ДЛЯ СТВОРЕННЯ КРОСПЛАТФОРМЕННИХ МОБІЛЬНИХ ДОДАТКІВ

О.О. Гомілко, магістрант
Київський національний університет технологій та дизайну
Т.І. Демківська, кандидат технічних наук, доцент
Київський національний університет технологій та дизайну

Ключові слова: мобільна розробка, кросплатформеність, android, iOS, flutter, dart.

Головною метою є дослідження технологій Flutter та Dart для розробки кросплатформенних мобільних додатків. Акцент зроблено на наступних моментах:

- дослідження архітектури Flutter;
- дослідження можливості написання одного програмного коду одночасно для Android та iOS платформ;
- дослідження можливості розробки нативних додатків для Android та iOS платформ;
- дослідження «сильних» сторін розробки;
- порівняння з існуючими аналогами для отримання порівняльної характеристики.

Flutter – це безкоштовний фреймворк з відкритим кодом, що був створений компанією Google та випущений у травні 2017 року як аналог ReactNative від Facebook. У кількох словах, цей фреймворк дозволяє створювати нативні мобільні застосунки на основі лише однієї бази коду. Це означає, що ви можете використовувати одну мову програмування та одну кодову базу для створення двох різних додатків (для iOS та Android). Flutter складається з двох важливих частин:

- SDK (Software Development Kit): набір інструментів, які допоможуть вам розробити ваші програми. Сюди входять інструменти для компіляції вашого коду в нативний машинного коду (код для iOS та Android).
- Framework (UI бібліотека на основі віджетів): набір елементів інтерфейсу користувачів (кнопки, поля введення тексту, повзунки тощо), які ви можете персоналізувати для власних потреб.

Основна ідея побудови UI додатку використовуючи Flutter – це побудова інтерфейсу за допомогою написання коду. Ви завжди будуєте дерево з віджетів у вашому застосунку. У вас не буде drag-and-drop інтерфейсу для додавання кнопок чи тексту на екран, який бачить юзер, натомість ви будете писати лише код.

Віджетом називається абсолютно кожний елемент у Flutter. Не важливо, чи це текст, кнопка, іконка або ж навіть поле для введення тексту – усі ці елементи є віджетами (рис. 1). Розглянемо приклад:

Верхня панель – це віджет, що містить у собі інші менші віджети (дерево віджетів). Назви, поля для вводу, поле для прикріплення документів, кнопка для відправлення – усе це віджети. Абсолютно весь додаток буде побудований з віджетів, навіть уся сторінка є віджетом, та і весь додаток «загорнутий» у віджет. Що ж таке віджет? Це шматок коду, написаний розробником, що виконує певну інструкцію, щоби відобразити потрібний елемент на екрані користувача.

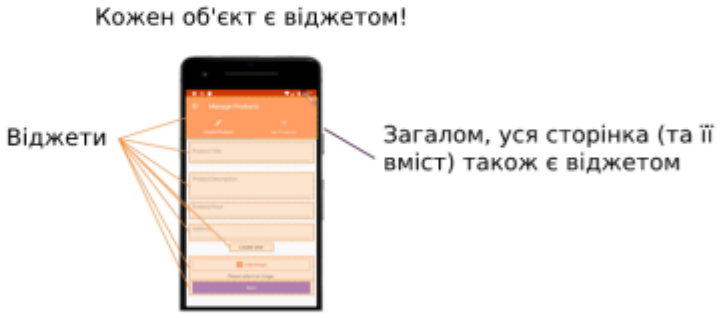


Рисунок 1 – Віджети у Flutter

Використовуючи віджети, ми будуємо так зване дерево із віджетів, де коренем дерева є наш додаток, а його сини – це відповідно інші віджети, які будуть відповідати за те, щоб відобразити необхідну нам сторінку.

Застосунки, що створюються на основі Flutter використовуються мову програмування Dart. Мова була створена компанією Google у жовтні 2011 року. Dart – це об'єктно орієнтована, строго типізована мова програмування.

Dart зосереджується на розробці інтерфейсу, і ви можете використовувати його для створення мобільних застосунків та вебдодатків.

Код, написаний на Dart, використовує Flutter фреймворк – це набір віджетів (вбудованих у Flutter, а також ваші власні), що необхідно скомпілювати для додатків на Android та iOS. Flutter компілює Dart код у нативний код для кожної із цих платформ за допомогою FlutterSDK. Як результат ви отримаєте додаток для кожної з платформ на основі вашогокоду. Flutter не використовує платформені примітиви. Наприклад, вам необхідно додати кнопку. Це не означає, що при компіляції Flutter створює нативний еквівалент кнопки для Andoird та iOS, натомість Flutter має власний механізм, що дозволяє контролювати на рендерити кожен піксель на екрані, що відображається користувачеві. Це надає Flutter повний контроль над інтерфейсом.

Список використаних джерел

1. Design Patterns: Elements of Reusable Object-Oriented Software [Книга]: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides – 1994.

2. Clean Architecture: A Craftsman's Guide to Software Structure and Design [Книга]: Robert C. Martin – 2017.

ГОМІЛКО О.О., ДЕМКІВСЬКА Т.І.

РОЗРОБКА МОБІЛЬНИХ ДОДАТКІВ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЙ FLUTTER ТА DART

HOMILKOO.O., DEMKIVSKA T.I

MOBILE APP DEVELOPMENT USING FLUTTER AND DART TECHNOLOGIES

The evolution is happening extremely fast right now. It gets harder to surprise society with new solutions. In order to prevent bureaucracy, waste of time and make our life easier, a lot of procedures and operations were automated.

Mobile apps happened to cover a lot of our needs. They are very popular and widely used almost in every sphere of our life. Having a mobile app on your smartphone may cover your different needs from entertainment to creating your nutrition plan.

Based on this significant role of mobile apps, the question of their development is highly relevant. In order to widen up the possibilities for mobile app development, I decided to study Flutter and Dart technologies. I did a research study on their advantages and their unique features of developing native apps for both Android and iOS platforms using one codebase. To make it more practical, I created the applied app showing all the development, graphics features, and possibilities Flutter and Dart provide us with.

Вступ

Розвиток технологій відбувається настільки швидко, що здивувати людство доволі складно. Як же сподобатись кінцевому користувачеві та привернути увагу до свого товару з ціллю збільшити попит? В еру нескінченний інформаційних війн буває дуже важко знайти необхідну інформацію та вирішення своєї проблеми, адже інформації, товару та пропозицій настільки багато, що заблукати та зробити неправильний вибір дуже легко.

Сьогодні більшу частину свого життя можна помістити у компактний смартфон, адже величезний спектр потреб може задовольнити каталог додатків, що нам пропонують в PlayMarket або ж в AppleStore.

Додаток для знайомств онлайн, застосунок, що допомагає вести здоровий образ життя та підтримувати водний баланс, чи навіть банально гра-квест - усе це може завжди бути під рукою та стати в нагоді при найменшій потребі, а це економить час та вирішує багато проблем за лічені секунди.

Постановка завдання

Головною метою є дослідження технологій Flutter та Dart для розробки мобільних додатків. Акцент зроблено на наступних моментах:

- дослідження архітектури Flutter;
- дослідження можливості написання одного програмного кодуодночасно для Android та iOS платформ;
- дослідження можливості розробки нативних додатків для Android та iOS платформ;

Інформаційні технології в науці, виробництві та підприємництві
Київський національний університет технологій та дизайну

ГОМІЛКО О.О., ДЕМКІВСЬКА Т.І.

РОЗРОБКА МОБІЛЬНИХ ДОДАТКІВ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЙ FLUTTER ТА DART

HOMILKOO.O., DEMKIVSKA T.I

MOBILE APP DEVELOPMENT USING FLUTTER AND DART TECHNOLOGIES

The evolution is happening extremely fast right now. It gets harder to surprise society with new solutions. In order to prevent bureaucracy, waste of time and make our life easier, a lot of procedures and operations were automated.

Mobile apps happened to cover a lot of our needs. They are very popular and widely used almost in every sphere of our life. Having a mobile app on your smartphone may cover your different needs from entertainment to creating your nutrition plan.

Based on this significant role of mobile apps, the question of their development is highly relevant. In order to widen up the possibilities for mobile app development, I decided to study Flutter and Dart technologies. I did a research study on their advantages and their unique features of developing native apps for both Android and iOS platforms using one codebase. To make it more practical, I created the applied app showing all the development, graphics features, and possibilities Flutter and Dart provide us with.

Вступ

Розвиток технологій відбувається настільки швидко, що здивувати людство доволі складно. Як же сподобатись кінцевому користувачеві та привернути увагу до свого товару з ціллю збільшити попит? В еру нескінченний інформаційних війн буває дуже важко знайти необхідну інформацію та вирішення своєї проблеми, адже інформації, товару та пропозицій настільки багато, що заблукати та зробити неправильний вибір дуже легко.

Сьогодні більшу частину свого життя можна помістити у компактний смартфон, адже величезний спектр потреб може задовольнити каталог додатків, що нам пропонують в PlayMarket або ж в AppleStore.

Додаток для знайомств онлайн, застосунок, що допомагає вести здоровий образ життя та підтримувати водний баланс, чи навіть банально гра-квест - усе це може завжди бути під рукою та стати в нагоді при найменшій потребі, а це економить час та вирішує багато проблем за лічені секунди.

Постановка завдання

Головною метою є дослідження технологій Flutter та Dart для розробки мобільних додатків. Акцент зроблено на наступних моментах:

- дослідження архітектури Flutter;
- дослідження можливості написання одного програмного кодуодночасно для Android та iOS платформ;
- дослідження можливості розробки нативних додатків для Android та iOS платформ;

- дослідження «сильних» сторінрозробки;
- порівняння з існуючими аналогами для отриманняпорівняльної характеристики.

У кінцевому результаті застосуємо отримані навички для розробки мобільного додатку за допомогою Flutterта Dart, який наочно продемонструє можливості фреймворку.

Основна частина

Flutter – це безкоштовний фреймворк з відкритим кодом, що був створений компанією Google та випущений у травні 2017 року як аналог ReactNative від Facebook. У кількох словах, цей фреймворк дозволяє створювати нативні мобільні застосунки на основі лише однієї бази коду. Це означає, що ви можете використовувати одну мову програмування та одну кодову базу для створення двох різних додатків (для iOS та Android). Flutter складається з двох важливих частин:

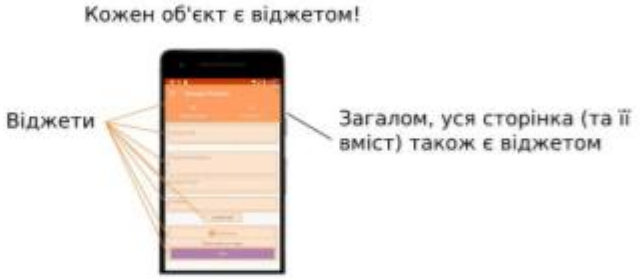
- SDK (SoftwareDevelopmentKit): набір інструментів, які допоможуть вам розробити ваші програми. Сюди входять інструменти для компіляції вашого коду в нативний машинного коду (код для iOS та Android).
- Framework (UI бібліотека на основі віджетів): набір елементів інтерфейсу користувачів (кнопки, поля введення тексту, повзунки тощо), які ви можете персоналізувати для власних потреб.

Основна ідея побудови UI додатку використовуючи Flutter – це побудова інтерфейсу за допомогою написання коду. Ви завжди будуєте дерево з віджетів у вашому застосунку. У вас не буде drag-and-drop інтерфейсу для додавання кнопок чи тексту на екран, який бачить юзер, натомість ви будете писати лише код.

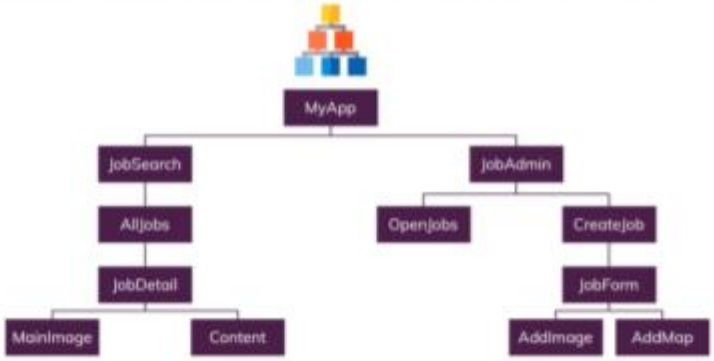
Віджетом називається абсолютно кожний елемент у Flutter. Не важливо, чи це текст, кнопка, іконка або ж навіть поле для введення тексту – усі ці елементи є віджетами.

Розглянемо приклад:

Верхня панель – це віджет, що містить у собі інші менші віджети (дерево віджетів). Назви, поля для вводу, поле для прикріплення документів, кнопка для відправлення – усе це віджети. Абсолютно весь додаток буде побудований з віджетів, навіть уся сторінка є віджетом, та і весь додаток «загорнутий» у віджет. Що ж таке віджет? Це шматок коду, написаний розробником, що виконує певну інструкцію, щоби відобразити потрібний елемент на екрані користувача.



Використовуючи віджети, ми будуємо так зване дерево із віджетів, де коренем дерева є наш додаток, а його сини – це відповідно інші віджети, які будуть відповідати за те, щоб відобразити необхідну нам сторінку.



Застосунки, що створюються на основі Flutter використовуються мову програмування Dart. Мова була створена компанією Google у жовтні 2011 року. Dart – це об'єктноорієнтована, строго типізована мова програмування.

Якщо говорити про синтаксис, то це така собі суміш Java, Javascript та C#, тож якщо у вас був досвід програмування хоча б на одній з зазначених мов, то труднощів виникнути у вас не повинно.

Dart зосереджується на розробці інтерфейсу, і ви можете використовувати його для створення мобільних застосунків та вебдодатків.

Код, написаний на Dart, використовує Flutter фреймворк – це набір віджетів (вбудованих у Flutter, а також ваші власні), що необхідно скомпілювати для додатків на Android та iOS. Flutterкомпілює Dart код у нативний код для кожної із цих платформ за допомогою FlutterSDK. Як результат виотримаєте додаток для кожної з платформ на основі вашогокоду. Flutter не використовує платформені примітиви. Наприклад, вам необхідно додати кнопку. Це не означає, що при компіляції Flutter

створює нативний еквівалент кнопки для Andoird та iOS, натомість Flutter має власний механізм, що дозволяє контролювати на рендерити кожен піксель на екрані, що відображається користувачеві. Це надає Flutter повний контроль над інтерфейсом.

Висновки

Досліджено нову технологію розробки мобільних додатків за допомогою фреймворку Flutter від компанії Google а також принцип роботи інструменту у поєднанні з мовою програмування Dar . Проведено дослідження щодо переваг та недоліків нового фреймворку, а також здійснено порівняльну характеристику відносно відомих аналогів.

Розроблено додаток, на прикладі якого було наочно досліджено особливості роботи фреймворку, механізми розробки мобільних додатків для обох платформ Android та iOS на основі єдиної кодової бази, також було розглянуто особливості відображення контенту на екрані залежно від орієнтації пристрою задля зручнішого користування додатком.

Розглянуто особливості графічної розробки, можливості використання вбудованого функціонала, а також використання вбудованого функціонала для створення власних віджетів, які є ядром при розробці додатку на Flutter.

ДОДАТОК З
Презентація дипломної магістерської роботи

Розробка мобільних додатків за допомогою технологій Flutter та Dart

Виконав студент МгІТ-2-20
Гомілко О.О.
Керівник: Демківська Т.І.

Постановка задачі

Сьогодні більшу частину свого життя можна помістити у компактний смартфон, адже величезний спектр потреб може задовільнити каталог додатків, що нам пропонують в Play Market або ж в Apple Store, саме тому мобільні додатки все частіше стають звичним способом для вирішення проблем та посідають значну роль у нашому житті.

Мета: познайомитись з новими технологіями, дослідити переваги та недоліки, застосувати отримані навички на практиці.

Актуальність: мобільні додатки набувають все більшої популярності, а якщо існує зручний та простий інструмент для створення гарних додатків, то чому б не дослідити його.

Предмет дослідження: фреймворк Flutter та мова програмування Dart.

Переваги Flutter

- Hot-reload
- Одна кодова база для двох платформ
- Менше тестування
- Красивий дизайн
- Висока продуктивність

Недоліки Flutter

- Нова технологія – відсутність об’ємної спільноти
- Масивний розмір файлу
- Ніяких сторонніх бібліотек

Додаток «My Wallet»