МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ

КАФЕДРА МУЛЬТИМЕДІЙНИХ СИСТЕМ

Розробка мобільних застосунків за допомогою технологій Flutter та Dart

Текстова частина до курсової роботи

за спеціальністю «Комп’ютерні науки та інформаційні технології» - 122

Керівник курсової роботи
к-т фізичко-математичних наук, доцент
Жежерун О.П.
«____»_____ 2020 року

Виконав студент ФІ-4
Гомілко О.О.
«____»_____ 2020 року

Київ 2020

Календарний план виконання роботи

Тема: Розробка мобільних застосунків за допомогою технологій Flutter та Dart

№	Назва етапу	Дата	Примітка
1.	Вибір теми курсової роботи	18.10.2019	
2.	Пошук тематичної літератури	25.10.2019	
3.	Ознайомлення з літературою	01.11.2019	
4.	Вивчення документації Flutter	05.11.2019	
5.	Ознайомлення з особливостями та синтаксисом Dart	05.12.2019	
6.	Перегляд навчальних відеоматеріалів по Flutter	15.01.2020	
7.	Написання першого розділу текстової частини курсової роботи	25.01.2020	
8.	Створення першого додатку «Hello, World!»	01.02.2020	
9.	Створення додатку «Quiz»	05.02.2020	
10.	Написання другого розділу текстової частини курсової роботи	17.02.2020	
11.	Написання додатку «Мій гаманець»	01.03.2020	
12.	Написання третього розділу текстової частини курсової роботи	27.03.2020	
13.	Написання висновків до курсової роботи	01.04.2020	
14.	Створення презентації	03.04.2020	
15.	Захист курсової роботи	19.04.2020	

Зміст

Вступ

Дана курсова робота присвячена вивченню технологій Flutter та Dart для створенню мобільних додатків.

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

Метою курсової 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. Технологічне майбутнє

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 безкінечних 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 2020 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 задовільнити 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.

Розділ 2. Flutter та Dart

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 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.

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.

Рисунок 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.

2.2.1. 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:

Рисунок 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.

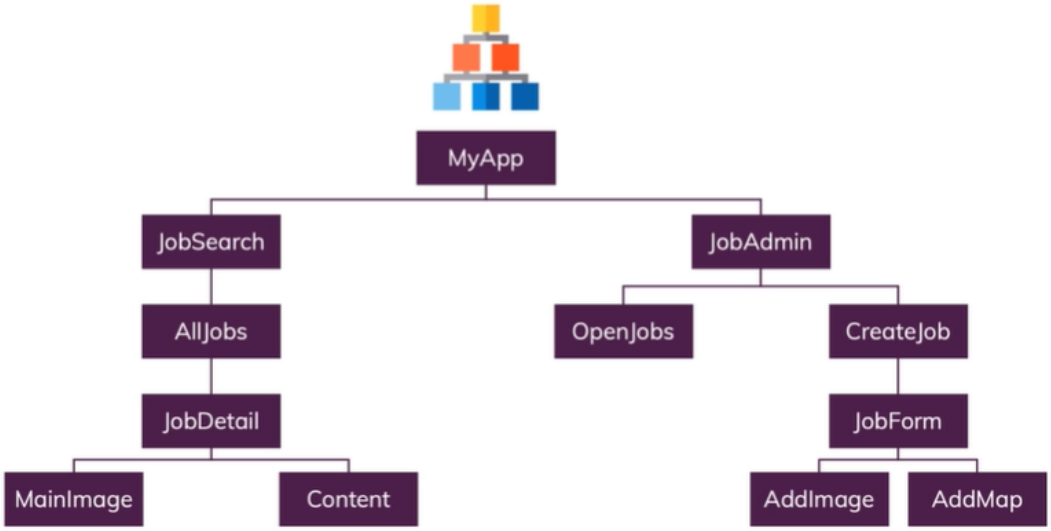

Рисунок 3. Дерево з віджетів

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.

2.2. 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 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.

2.3. 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	Reach 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

2.4. Переваги

1. 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 в співпраці розробників та дизайнерів, 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.

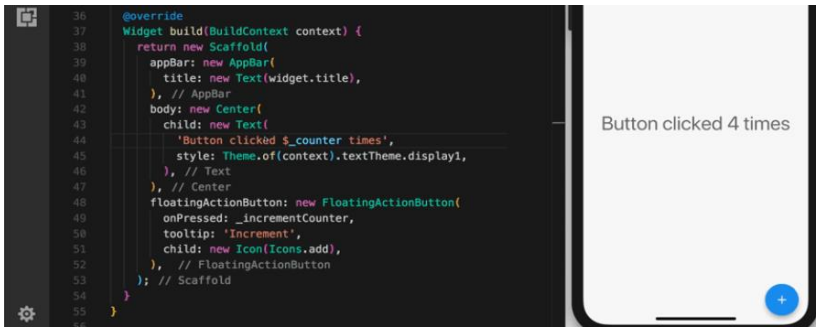

Рисунок 4. Hot-reload. До змін

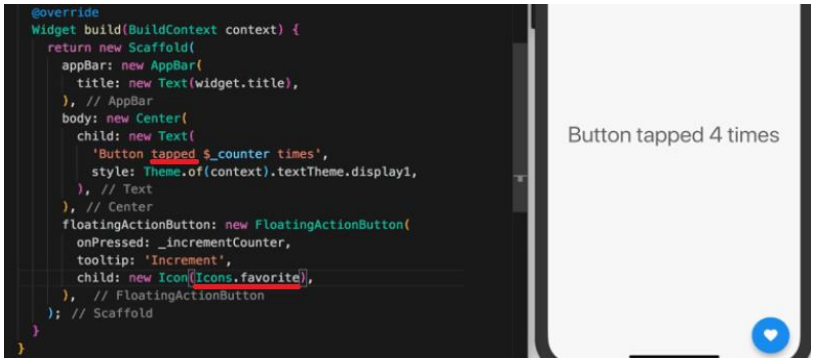

Рисунок 5. Hot-reload. Відразу відображені зміни без необхідності перезапуску додатку

2. 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

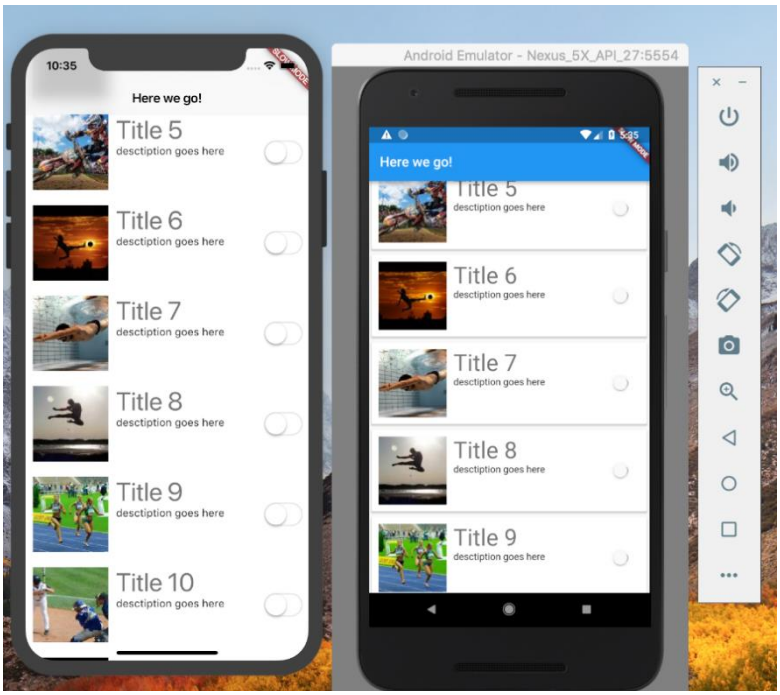

Рисунок 6. Можливість створювати додаток для двох платформ

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.

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 варіанта коду на кожному кроці.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

8. 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.

9. 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.

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

11. 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.

2.6. Недоліки

1. 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.

2. 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.

3. 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.

4. 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.

2.7. Приклад додатку. Стандартний застосунок “Hello, World!”

Розглянемо приклад стандартного додатку «Hello, World!» з якого, зазвичай, і починають вивчення нової мови програмування, щоб краще познайомитись з особливостями нових технологій.

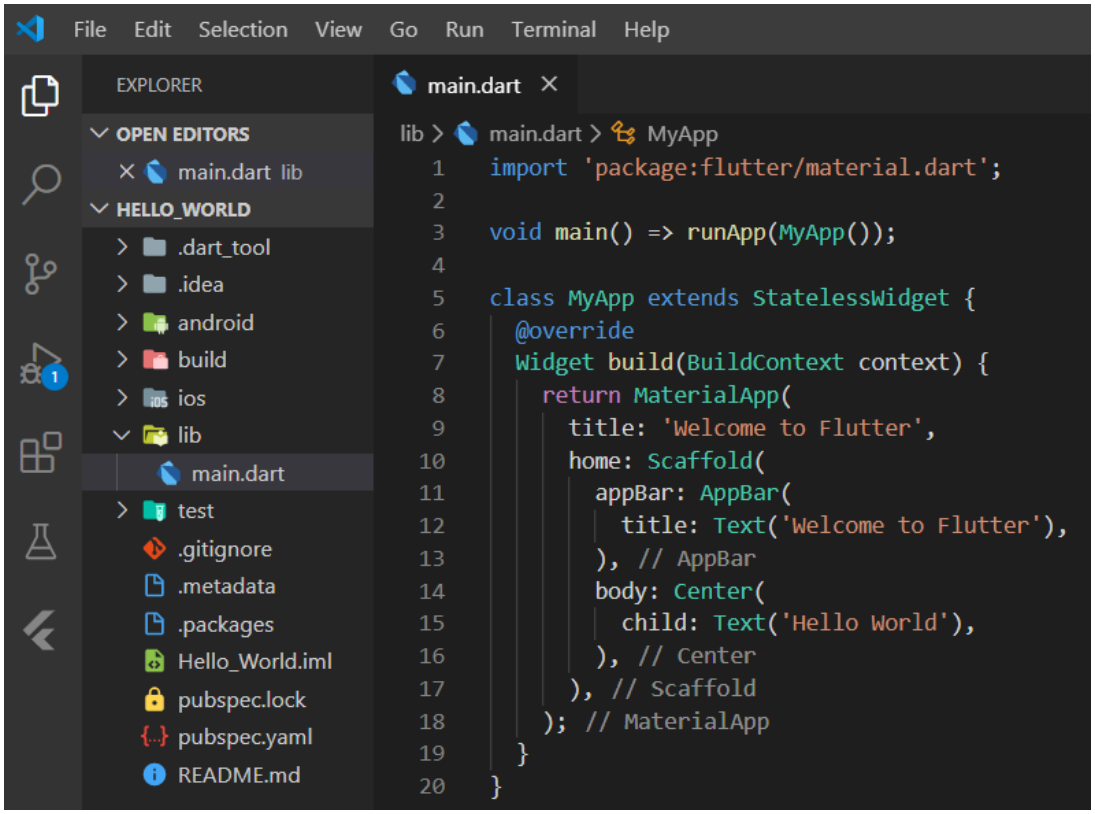

Рисунок 7. Лістинг до програми "Hello, World!"

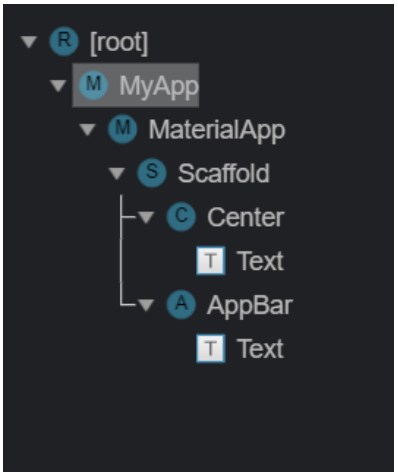

Рисунок 8. Widget Tree додатку "Hello, World"

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.

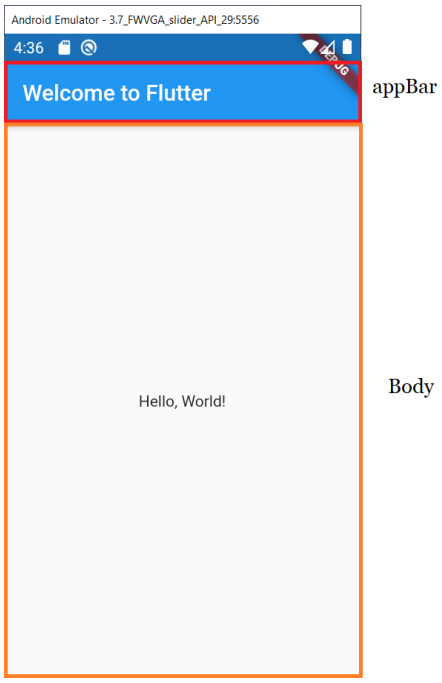

Рисунок 9. Додаток "Hello, World!"

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)

Розділ 3. Практична частина

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

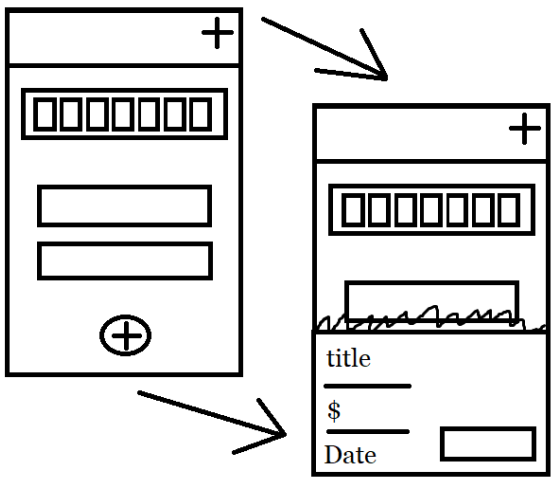

Рисунок 10. Схема додатку "Мій гаманець"

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 на 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.

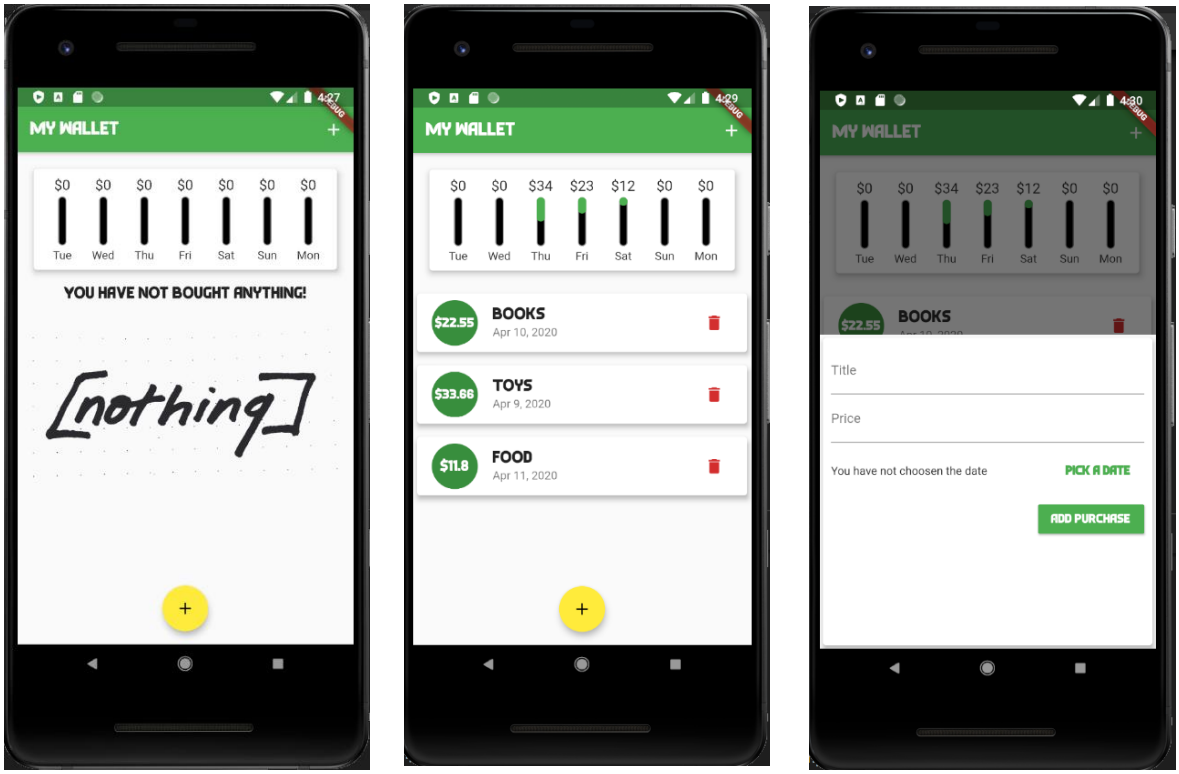

Рисунок 11. Додаток "Мій гаманець"

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.

```
  @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
  }
}
```

Рисунок 12. Лістинг віджету "IndicatorsChart"

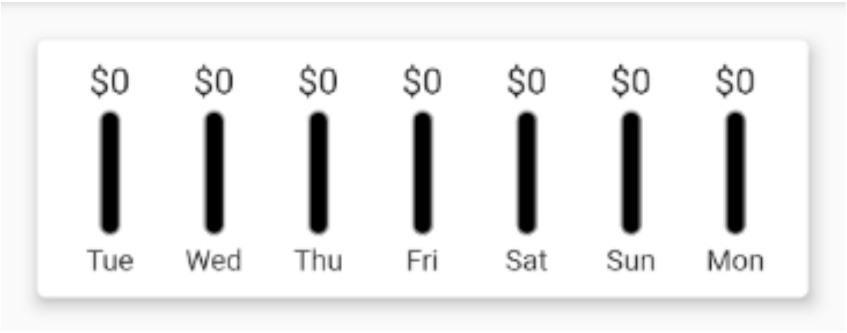

Рисунок 13. Результат роботи коду

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 mainAxisAlignment (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.

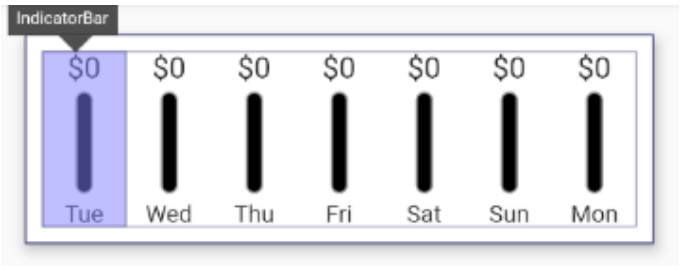

Рисунок 14. Віджет "IndicatorBar"

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
}
```

Рисунок 15. 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, constraints); 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: constraints.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 відштовуємося ми від constraints, 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 ChartBat) незалежно від розміру девайсу, на якому він використовуватиметься.

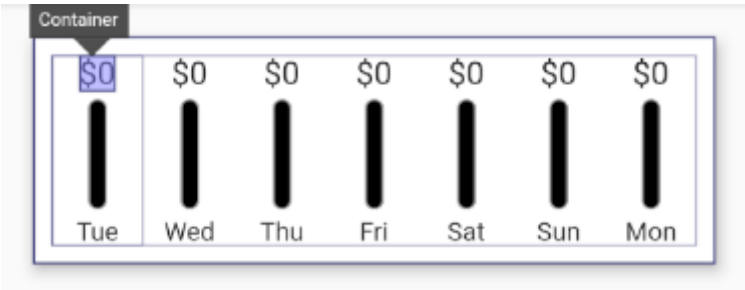

Рисунок 16. Віджет Container

Наступний нащадок – віджет SizedBox, для якого визначено лише те, що він займатиме 5% від загального розміру всього віджету: height: constraints.maxHeight * 0.05, тобто, це пусте місце, такий собі невидимий

віджет, який додано лише для того, щоб гарно розмежувати складові нашого віджету Column.

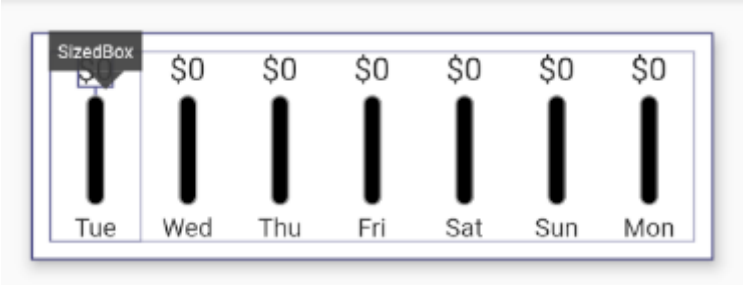

Рисунок 17. Віджет SizedBox

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

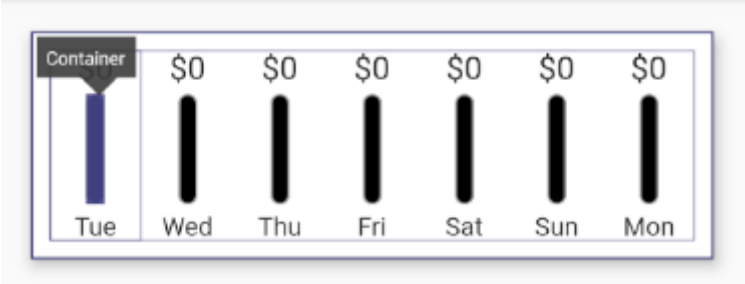

Рисунок 18. Віджет Container - 1

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.

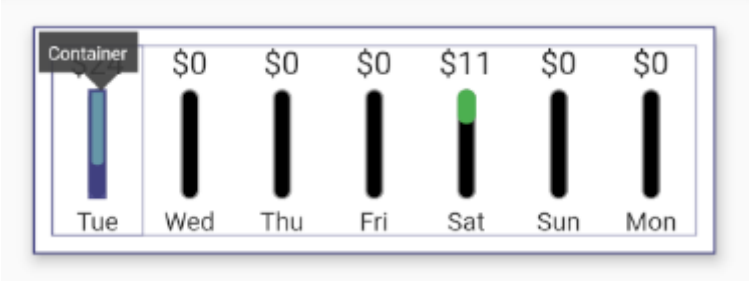

Рисунок 19. Віджет Container – 2

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.

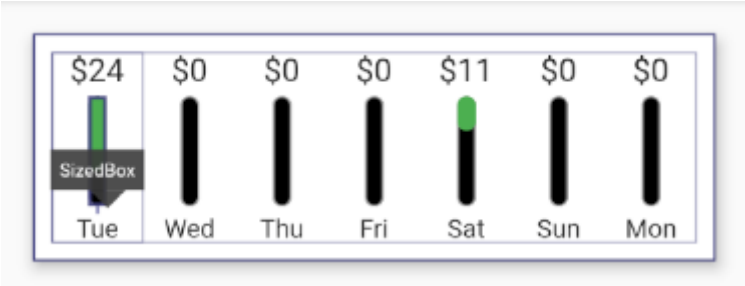

Рисунок 20. Віджет SizedBox

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.

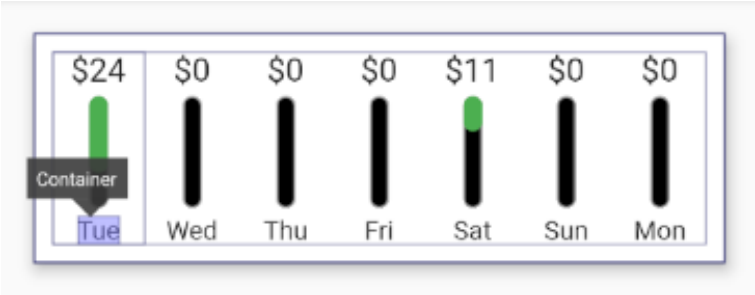

Рисунок 21. Віджет Container – день тижня

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 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:

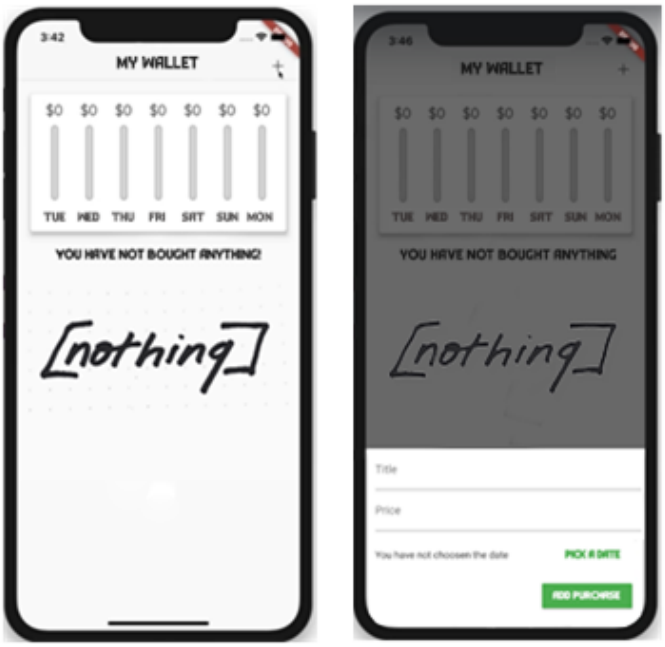

Рисунок 22. Додаток на iOS

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.

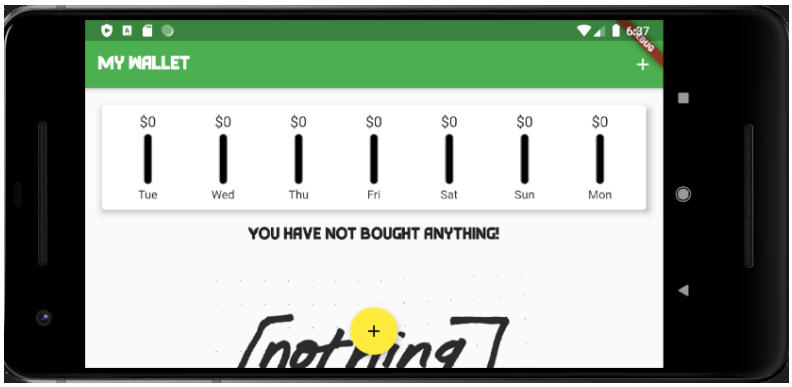

Рисунок 23. Додаток у landscape режимі

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.

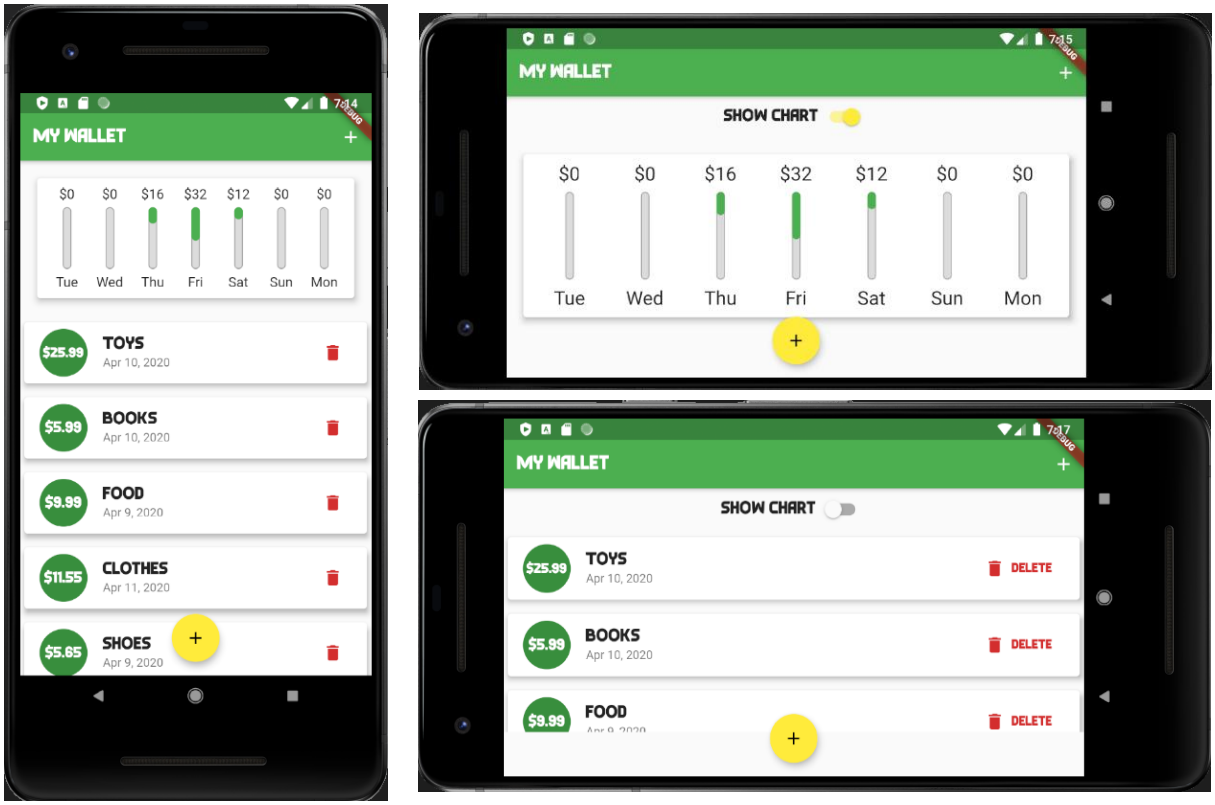

Рисунок 24. Зміни на девайсі при різному положенні екрану

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 зручнішого

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
```

Рисунок 25. 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.

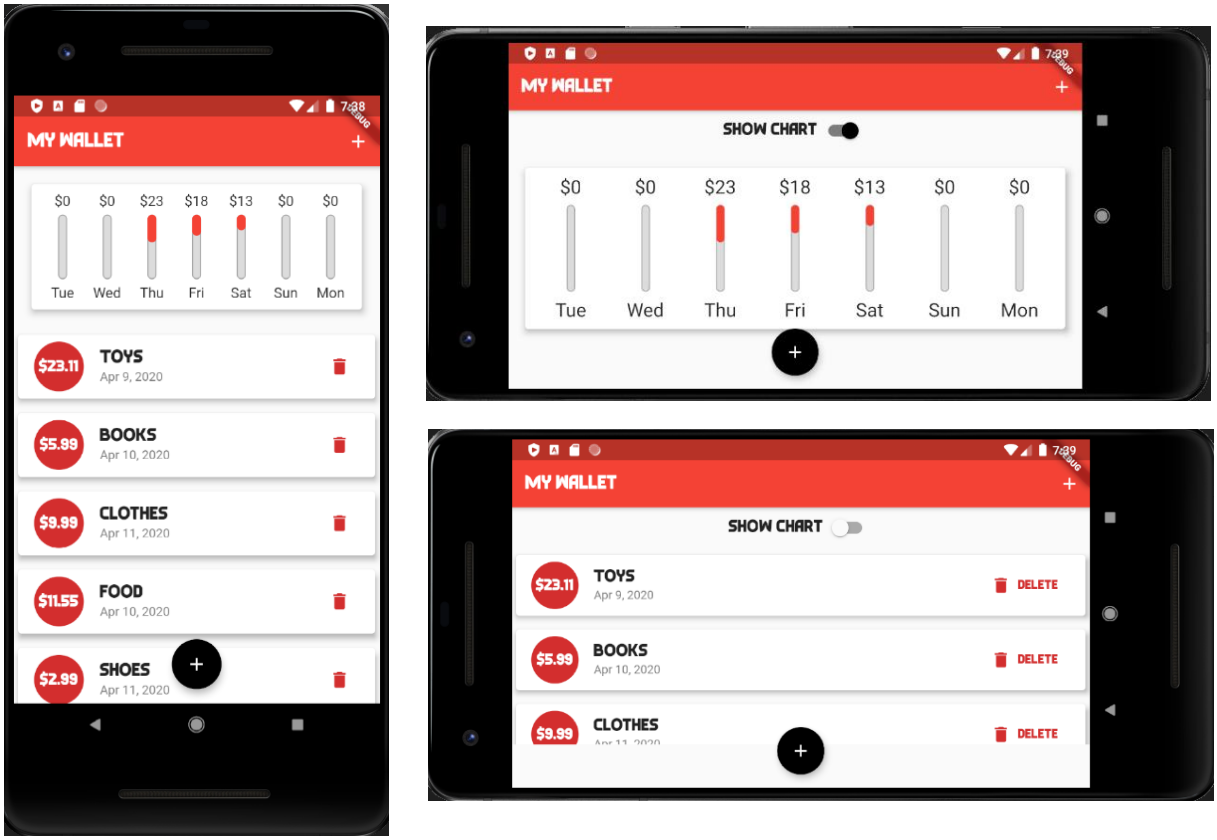

Рисунок 26. Додаток після певних змін у коді

3.8. Можливості відлагоджування

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

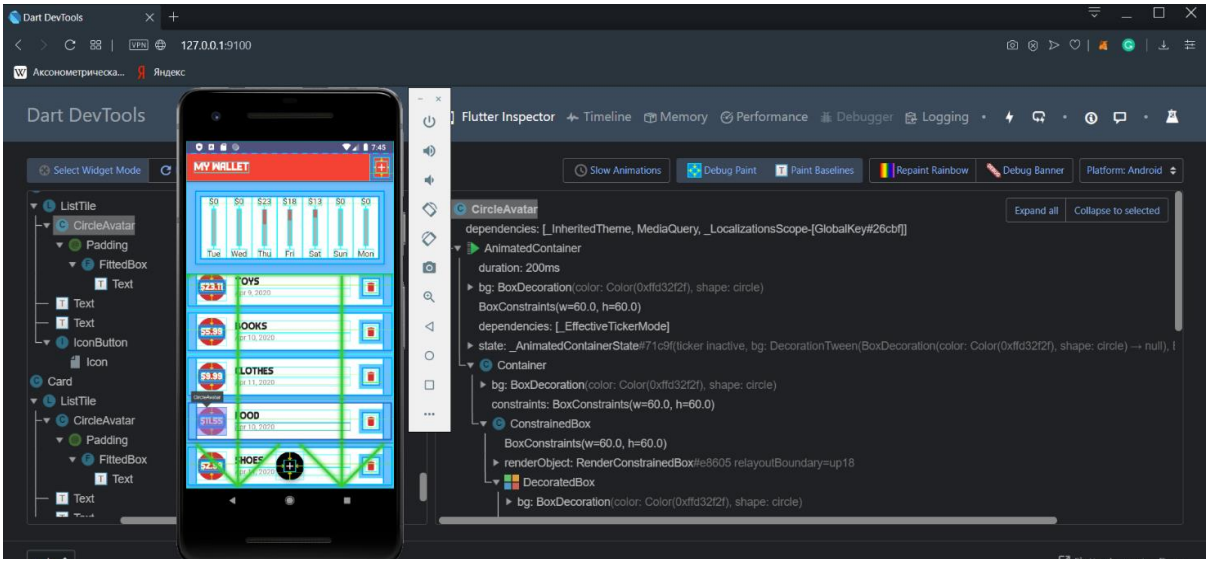

Рисунок 27. Dart: DevTools

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.

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 процесі написання курсової роботи було дослідження нову технологію розробки мобільних додатків за допомогою фреймворку Flutter від компанії Google. Було досліджено принцип роботи інструменту у поєднанні з мовою програмування Dart.

Було проведено дослідження щодо переваг та недоліків 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.

Було 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.

Було також 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.

Було також 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.

Список використаних джерел:

Google, 2017-2020, Flutter documentation: https://flutter.dev/docs

Станислав Термоса, 2018, Про Flutter, кратко, основы: https://habr.com/ru/post/430918/

Ihor Feoktistov, 2018, Top 8 Flutter Advantages and Why You Should Try Flutter on Your Next Project: https://relevant.software/blog/top-8-flutter-advantages-and-why-you-should-try-flutter-on-your-next-project/

Code Magic, 2018, What is Flutter? Benefits and limitations: https://blog.codemagic.io/what-is-flutter-benefits-and-limitations/

Academind, 2018, Flutter Tutorial for Beginners: https://www.youtube.com/watch?v=GLSG_Wh_YWc&list=PL55RiY5tL51qKxC472MY2ayxJTze5Qb7T

Hackernoon, Wm Leler, 2018, Why Flutter Uses Dart: https://hackernoon.com/why-flutter-uses-dart-dd635a054ebf

K S Kuppusamy, 2017, What makes Dart an easy, scalable and multi-purpose programming language: https://opensourceforu.com/2017/06/dart-easy-scalable-multi-purpose-programming-language/

Анна Гуляева, 2018, Как начать работать с Flutter: https://apptractor.ru/develop/coding/kak-nachat-rabotat-s-flutter.html

Beginning Flutter: A Hands On Guide to App Development, Marco L. Napoli

Flutter for Beginners, Alessandro Biessek

Google, 2017-2020, Examples of apps: https://flutter.github.io/samples/#

Google, 2017-2020, Flutter by example: https://www.flutterbyexample.com

Flutter уроки, 2018: https://flutter.su