

Факультет «Комп'ютерні технології і системи»
Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Дослідження підходів реактивного програмування у мові DART»
за освітньою програмою: 121 Інженерія програмного забезпечення
зі спеціальності: 121 Інженерія програмного забезпечення

Виконав: студент групи ПЗ2221



/Катерина КАЛІНЧЕНКО/

Керівник:



/Олександр ІВАНОВ/

Нормоконтролер:



/Світлана ВОЛКОВА/

Засвідчую, що у цій роботі немає запозичень з
праць інших авторів без відповідних посилань.

Студент



Дніпро – 2024 рік

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems
Department Computer information technology

Explanatory Note to Master's Thesis

on the topic: «Research on the reactive programming via DART programming language»

according to educational curriculum **121 software engineering**
in the Speciality: **121 software engineering**

Done by the student of the group PZ2221-M: / Kateryna KALINCHENKO /

Scientific Supervisor: / Oleksandr IVANOV /

Normative controller: / Svitlana VOLKOVA/

Dnipro – 2024

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: магістр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____/Вадим ГОРЯЧКІН/
(підпис)
Дата _____

ЗАВДАННЯ

на кваліфікаційну роботу для здобуття ступеня магістра
студентці Калінченко Катерині Олегівні

1. Тема роботи: «Дослідження підходів реактивного програмування у мові DART»

Керівник роботи: Іванов Олександр Петрович, доцент
затверджені наказом № 1196 ст від 05.12.2022

2. Строк подання студентом роботи: 23.01.2024 р.

3. Вихідні дані до роботи: _____

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):

4.1 Теоретичні положення: аналіз джерел, огляд підходів до розробки,
огляд основних теоретичних матеріалів, огляд майбутніх трендів
реальні кейси використання.

4.2 Опис розроблених практичних прикладів для аналізу:
короткий опис розроблених систем, опис функціональних вимог,
опис експлуатаційних вимог, огляд систем, програмування прикладів,
результати роботи кожної системи.

4.3 Практичне використання реактивного підходу:
розробка програм, діаграми класів, порівняння різниці між реактивним та
нереактивним підходом.

4.4 Дослідження можливостей реактивного підходу:
результати експериментів, аналіз результатів експериментів,
діаграми та графіки, що підтверджують висновки

КАЛЕНДАРНИЙ ПЛАН

№ з/ п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ	20.07.2023-02.08.2023	
2	Аналіз сучасного стану дослідження	02.08.2023-18.09.2023	
3	Обґрунтування загальної методики проведення наукового дослідження	18.09.2023-25.10.2023	
4	Розробка інструментальних засобів для проведення дослідження реактивного програмування	25.10.2023-05.11.2022	30%
5	Дослідження ефективності реактивного програмування	05.11.2023-15.11.2023	
6	Аналіз результатів та оформлення пояснювальної записки	15.11.2023-10.12.2023	60%
7	Розробка демонстраційних матеріалів	13.12.2023-20.12.2023	100%
8	Подання кваліфікаційної роботи до кафедри	11.01.2024	
9	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	23.01.2024	

Студент

(підпис)

Катерина КАЛІНЧЕНКО

(Ім'я ПРІЗВИЩЕ)

Керівник роботи

(підпис)

Олександр ІВАНОВ

(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра складається з вступу, переліку умовних познач, символів, скорочень і термінів, 4 розділів, загальних висновків та рекомендацій, списку використаної літератури:

- вступ. Складається з 3 сторінок;
- розділ 1 теоретичні положення. Складається з 27 сторінок;
- розділ 2 постановка задачі. Складається з 12 сторінок;
- розділ 3 практичне використання реактивного підход. Складається з 9 сторінок;
- розділ 4 дослідження можливостей реактивного підходу. Складається з 9 сторінок;
- загальні висновки та рекомендації. Складається з 4 сторінки;
- список літератури. Складає 1 сторінки;
- додатки – містить текст програми.

Кількість таблиць: 3 штуки.

Кількість рисунків: 29 штук.

Кількість джерел: 10 штук.

Об'єктом цього дослідження є програм-прикладів, що демонструють основні підходи та відмінності реактивного та нереактивного програмування.

Метою роботи є аналіз та оцінка ефективності реактивного програмування.

Методи дослідження: порівняння програм-прикладів, їх характеристик та показників.

Результати та їх новизна: у ході дослідження реактивного програмування з використанням бібліотеки RxDart було проведено оцінку продуктивності та ефективності обробки асинхронних подій у реальному часі. Новизною роботи є чітке визначення оптимальних стратегій реагування на зміни у потоках даних при зміні зовнішніх умов. Результати вказують на покращення адаптивності системи до динамічного середовища, що максимізує продуктивність реактивного програмування при використанні RxDart. Це може послужити основою для подальшого розвитку та оптимізації реактивних програм у сфері розробки на мові Dart. Рівень новизни -

розширення і розуміння оптимальних підходів до реактивного програмування в контексті даної бібліотеки.

Ключові слова: ПОТОКИ, АСИНХРОННІСТЬ, СПОСТЕРЕЖУВАНІЙ, РЕАКТИВНІ ПАТЕРНИ, РЕАКТИВНА РОЗРОБКА, RX, RXDART.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ.....	9
ВСТУП.....	9
РОЗДІЛ 1 ТЕОРЕТИЧНІ ПОЛОЖЕННЯ.....	11
1.1 Загальні положення про реактивне програмування.....	11
1.2 Основи RXDart.....	14
1.3 Observable та Observer Pattern.....	17
1.4 Streams and Stream Controllers.....	19
1.5 Оператори в RXDart.....	22
1.6 Обробка помилок та стратегія повторних спроб.....	29
1.7 Потоки та планувальники (schedulers).....	31
1.8 Майбутні тренди та внески спільноти.....	32
Висновки до першого розділу.....	34
РОЗДІЛ 2 ПОСТАНОВКА ЗАВДАННЯ.....	36
2.1 Визначення основних завдань та проблем, які розв'язує RXDart.....	36
2.2 Власні дослідження та використання RXDart.....	37
2.2.1 Приклад 1. Searchable data.....	37
2.2.2 Приклад 2. Sports scoreboard.....	39
2.2.3 Приклад 3. Todo list.....	41
2.2.4 Приклад 4. User login.....	43
2.2.5 Приклад 5. Weather app.....	45
Висновки до другого розділу.....	46
РОЗДІЛ 3 ПРАКТИЧНЕ ВИКОРИСТАННЯ РЕАКТИВНОГО ПІДХОДУ.....	48
3.1 Вступ.....	48
3.2 Аналіз різниці імплементацій.....	48
3.2.1 Searchable data.....	49
3.2.2 Sports scoreboard.....	51
3.2.3 Todo list.....	53
3.2.4 User login.....	53
3.2.5 Weather app.....	55
Висновки до третього розділу.....	57
РОЗДІЛ 4 ДОСЛІДЖЕННЯ МОЖЛИВОСТЕЙ РЕАКТИВНОГО ПІДХОДУ.....	59
4.1 Вступ.....	59
4.2 Порівняльний аналіз та практична цінність методики.....	60
4.3 Результати та графічні представлення.....	60
Висновки до четвертого розділу.....	65
ЗАГАЛЬНІ ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ.....	66

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....	69
ДОДАТКИ.....	70

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційна система.

ПЗ – програмне забезпечення.

RxDart – Реалізація реактивного програмування для Dart, надаючи інтерфейс для створення та обробки асинхронних подій та потоків даних.

API – інтерфейс програмування застосунків.

BLoC – Бізнес-логіка в компоненті реактивного програмування.

RxUI – Взаємодія з інтерфейсом користувача в реактивному стилі.

RxFlutter – Інтеграція Rx у фреймворк Flutter для реактивної розробки мобільних додатків.

ВСТУП

Реактивне програмування є підходом до розробки програмного забезпечення, який розглядає дані та події як потоки, дозволяючи ефективно реагувати на зміни. У контексті мови програмування Дарт та бібліотеки RXDart, дослідження фокусується на вивченні можливостей та особливостей реактивного програмування.

Реактивне програмування представляє собою підхід до створення програмного забезпечення, який розглядає дані та події як потоки, що дозволяє ефективно реагувати на зміни. Цей метод стає особливо актуальним у контексті мови програмування Дарт та використання бібліотеки RXDart. Дослідження цієї теми спрямоване на ретельне вивчення можливостей та особливостей реактивного програмування в даному програмному середовищі.

У сучасному програмуванні реактивний підхід визначається як стратегія розробки, яка акцентує увагу на тому, щоб реагувати на події, що стаються в реальному часі. У випадку мови програмування Дарт та використання бібліотеки RXDart, ця стратегія набуває особливого значення через їхні специфічні можливості та функціональність.

Одним з основних аспектів, на якому зосереджується це дослідження, є вивчення внутрішньої структури RXDart та аналіз алгоритмів, які забезпечують його функціональність реактивного програмування. Цей процес передбачає розгляд внутрішніх механізмів обробки подій та взаємодії з потоками даних відповідно до принципів реактивності.

Ця робота має на меті розкрити основні аспекти реактивного програмування в контексті мови Дарт та RXDart, визначити його переваги та недоліки, а також надати практичні рекомендації для його застосування.

Звідси можна визначити конкретні завдання для виконання дослідження метою розкриття основних аспектів реактивного програмування в контексті Дарт та RXDart: визначення його переваг і недоліків, дослідження функціональних можливостей RXDart, визначити його ключові характеристики та виявити, які завдання призначений вирішувати цей підхід.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ПОЛОЖЕННЯ

1.1 Загальні положення про реактивне програмування

Реактивне програмування виступає як трансформаційна парадигма у сфері розробки програмного забезпечення, представляючи революційний підхід для керування асинхронними подіями та операціями. На відміну від традиційної парадигми імперативного програмування, яка характеризується послідовністю команд (спочатку виконується Дія 1, після неї Дія 2 і так далі), реактивне програмування ретельно розроблено для навігації в тонкощах динамічних даних, що постійно змінюються (наприклад користувацьке введення з клавіатури), реагуючи на події в реальному часі. Це комплексне дослідження має на меті розглянути фундаментальні принципи, мотивацію, переваги та недоліки, які лежать в основі реактивного програмування, створюючи ґрунтовну основу для розуміння його ключової ролі у формуванні сучасної розробки програмного забезпечення.

Реактивне програмування розвинулося у відповідь на зростаючі вимоги до сучасних програм. З розвитком технологій додатки стають складнішими, що вимагає від розробників зусиль для боротьби зі складнощами обробки асинхронних, одночасних операцій і оновлення даних у реальному часі. Реактивне програмування з властивою йому здатністю безпроблемно керувати динамічними даними та подіями стало елегантним рішенням для вирішення цих проблем.

В основі реактивного програмування лежить “зміна парадигми” (paradigm shift), яка виходить за межі традиційних підходів до розробки програмного забезпечення. Замість того, щоб дотримуватися заздалегідь визначеної послідовності кроків, реактивне програмування обертається навколо реакції на події, коли вони розгортаються. Це дає розробникам змогу моделювати та обробляти складні асинхронні взаємодії більш інтуїтивно та ефективно. Даний підхід являє собою відхід від жорстких методологій, орієнтованих на потоки керування, запроваджуючи більш декларативний стиль, де розробники визначають, що має статися, коли відбуваються конкретні події.

Однією з основних мотивацій для впровадження реактивного програмування є його здатність оптимізувати розробку додатків із динамічними та інтерактивними інтерфейсами користувача. Взаємодія користувача, наприклад натискання кнопок, надсилання форм або отримання даних, за своєю суттю є асинхронними та керованими подіями. Реактивне програмування забезпечує елегантний спосіб керування цими асинхронними подіями, завдяки чому воно добре підходить для додатків, де швидкість реагування та оновлення в реальному часі є вирішальними.

Реактивне програмування базується на ідеї асинхронної обробки подій. Асинхронна обробка означає, що обробка однієї події не блокує обробку іншої події.

У перших програмах із графічним інтерфейсом інтерфейс користувача оновлювався лише після певної дії користувача (наприклад натискання кнопки). Серцем програми був так званий цикл подій. Це був нескінченний цикл що відповідав за обробку введених користувачем даних і оновлення інтерфейсу користувача. Але оскільки програма синхронно чекала (тобто блокувала) введення користувача, нічого іншого не могло статися. А синхронне очікування події, яка не настає, означає, що вся програма зависає, а графічний інтерфейс перестає реагувати.

Рішення полягає в тому, щоб відокремити обробку подій від оновлення GUI. Відокремлення здійснюється шляхом введення черги подій і паралельної обробки подій. Цей підхід добре відомий як модель виробника та споживача (producer and consumer model). Потік, який обробляє дані користувача, розміщує події в черзі, а потік-споживач бере події з початку черги та обробляє їх. Потік споживача також може діяти як виробник, розміщуючи запити на оновлення GUI у чергу, оброблену потоком оновлення GUI. Такий ланцюжок виробник/споживач призводить до реактивних потоків.

Сьогодні парадигма реактивного програмування також поширилася на хмарні програми. Хмарні сервіси часто реалізуються як набір мікросервісів. Це невеликі компоненти, які слабо пов'язані та взаємодіють один з одним через асинхронну передачу повідомлень. Парадигма реактивного програмування ідеально підходить для таких мікросервісів.

Ключові концепції реактивного програмування

1. Об'єкти спостереження та спостерігачі (Observables and Observers)

Основним поняттям у реактивному програмуванні є Observable. Спостережуваний (observable) — це послідовність значень або подій, які можна асинхронно спостерігати з часом. Спостерігачі (observables) підписуються на ці спостережувані об'єкти та реагують на нові значення. Такий розподіл завдань спрощує керування асинхронними операціями та сприяє створенню більш модульної кодової бази, яку легше підтримувати.

2. Потоки даних (Streams)

У реактивному програмуванні дані розглядаються як безперервні потоки подій. Незалежно від того, чи це дані користувача, дані датчиків або зовнішні відповіді API, ці потоки представлені як об'єкти спостереження. Цей потоковий підхід дозволяє розробникам моделювати та маніпулювати даними більш природним та ефективним способом.

3. Декларативне програмування

Реактивне програмування охоплює декларативний стиль програмування, де розробники виражають бажані результати та перетворення, не вказуючи покрокові процедури. Це призводить до більш лаконічного та читабельного коду, оскільки фокус зміщується з опису потоку керування на визначення зв'язків між різними потоками даних.

4. Асинхронні операції

Асинхронні операції, такі як мережеві запити, файли вводу/виводу або взаємодії з користувачем, поширені в сучасних програмах. Реактивне програмування забезпечує безпроблемний спосіб обробки цих асинхронних завдань за допомогою спостережуваних для представлення асинхронних подій і реагування на них, усуваючи складнощі, пов'язані зі зворотними викликами або традиційними моделями потоків.

На даний момент існує кілька доступних фреймворків реактивного програмування. Першою бібліотекою реактивного програмування була RxJava, яка була представлена в 2013 році.

Існує також ініціатива зі стандартизації API реактивних потоків. Ініціатива називається Reactive Streams. Він визначає набір правил для обробки асинхронного потоку з неблокуючим зворотним тиском (non-blocking back pressure).

Реактивне програмування являє собою зміну парадигми, що стосується складнощів сучасної розробки програмного забезпечення. Зосереджуючись на принципах реактивності, спостережуваності і декларативного програмування, цей підхід забезпечує ефективний засіб керування асинхронними та керованими подіями операціями. Переваги швидкої реакції, компонування та обробки помилок роблять реактивне програмування переконливим вибором для створення динамічних та інтерактивних програм.

1.2 Основи RXDart

У постійно змінюваному ландшафті розробки Dart пошуки ефективних рішень для обробки асинхронних операцій і керування складними потоками даних спонукали та спонукають розробників використовувати можливості реактивного програмування. Це дослідження поглиблюється в основні аспекти RXDart, розкриваючи його основні принципи та проливаючи світло на те, як реактивний підхід легко справляється з тонкощами асинхронного програмування в екосистемі Dart.

Оскільки Dart продовжує набирати популярність для створення кросплатформних програм, потреба в парадигмі реактивного програмування стає все більш очевидною. Реактивне програмування в основному обертається навколо концепції реагування на зміни та події, забезпечуючи більш раціональний підхід до керування асинхронними завданнями.

За своєю суттю RXDart є реалізацією ReactiveX (Reactive Extensions) для Dart. ReactiveX — це широко поширена парадигма, яка розширює шаблон спостерігача для підтримки послідовностей даних і подій. RXDart дає змогу

розробникам моделювати сценарії як реактивні потоки, підвищуючи читабельність коду та зручність обслуговування.

Основні концепти RXDart.

1) Observables. Streams of Data and Events.

Основною концепцією RXDart є поняття Observables. Спостережуваний представляє потік значень або подій, які можна спостерігати з часом. Цей підхід, орієнтований на потік, дозволяє розробникам моделювати динамічні потоки даних у більш природний та ефективний спосіб. Спостережувані служать джерелами даних, випромінюючи значення спостерігачам, які підписалися на ці потоки. У сфері RXDart Observables стають основою для керування асинхронними подіями та реагування на них.

2) Observers and Subscriptions.

Спостерігачі, сутності, які підписалися на Observables, відіграють ключову роль у парадигмі реактивного програмування. Вони реагують на значення, повернені Observables, і визначають логіку, яка має виконуватися, коли ці події відбуваються. Підписки, механізм зв'язку Observables і Observers, встановлюють з'єднання, яке забезпечує потік даних. Такий розподіл завдань забезпечує модульну та чисту архітектуру, що покращує масштабованість і зручність обслуговування програм Dart.

3) Subjects. Bridging the Gap.

RXDart представляє концепцію суб'єктів, які діють і як спостережувані, і як спостерігачі. Ця подвійна природа робить суб'єктів універсальними компонентами реактивного програмування. Вони можуть повертати нові значення та підписуватися на інші Observables, діючи як мости, що полегшують зв'язок між різними частинами програми Dart. Теми забезпечують додатковий рівень гнучкості, дозволяючи розробникам легко інтегрувати реактивні принципи в різні аспекти своєї кодової бази.

Асинхронне програмування є поширеною проблемою в розробці Dart, особливо в таких сценаріях, як мережеві запити, введення користувачами або оновлення в реальному часі. RXDart вирішує ці проблеми, представляючи

концепцію потоків і контролерів потоків. Потоки представляють послідовності даних, які розгортаються з часом, тоді як контролери потоку керують потоком цих даних. Ця абстракція спрощує представлення асинхронних завдань і керування ними, пропонуючи розробникам узгоджений і виразний спосіб обробки асинхронного програмування.

Особлива перевага RXDart полягає в його багатому наборі операторів, призначених для перетворення, фільтрації та об'єднання потоків. Ці оператори дають змогу розробникам без зусиль маніпулювати потоками даних, забезпечуючи високий рівень виразності в коді Dart. Незалежно від того, чи йдеться про відображення значень, фільтрацію подій чи об'єднання потоків, ці оператори стають важливими інструментами для створення складних реактивних робочих процесів. Оператори RXDart роблять внесок у елегантність і читабельність коду Dart, пропонуючи потужні засоби перетворення потоку.

RXDart визнає важливість надійної обробки помилок в асинхронному програмуванні. Він повністю інтегрує механізми обробки помилок на різних етапах потоку даних, забезпечуючи стійкість і цілісність програм. Крім того, RXDart допомагає в ефективному управлінні ресурсами, запобігаючи витокам пам'яті та оптимізуючи використання ресурсів у програмах Dart.

RXDart постає маяком у величезному морі розробки Dart, проводячи розробників через тонкощі реактивного програмування. Повна інтеграція в екосистему Dart у поєднанні з багатим набором основних принципів і концепцій позиціонує RXDart як трансформаційний інструмент для керування асинхронним програмуванням. Елегантність, з якою RXDart обробляє потоки, Observables і асинхронні робочі процеси, дозволяє розробникам створювати динамічні та адаптивні додатки Dart.

У наступних розділах цього дослідження ми глибше заглибимося в практичне застосування RXDart, досліджуючи реальні приклади та тематичні дослідження, які демонструють універсальність бібліотеки та її вплив на розвиток Dart. Від реагування користувальницького інтерфейсу до мережових запитів і керування

станом, RXDart виявляється фундаментальним активом у формуванні майбутнього ландшафту програм Dart.

1.3 Observable та Observer Pattern

У багатьох завданнях створення програмного забезпечення розробники більш-менш очікують, що інструкції, які вони пишуть, виконуватимуться та завершуватимуться поступово, одна за раз, у тому порядку, в якому вони їх написали. Але в ReactiveX багато інструкцій можуть виконуватися паралельно, а їхні результати пізніше фіксуються в довільному порядку «спостерігачами». Замість того, щоб викликати метод, розробники визначають механізм для отримання та перетворення даних у формі «Observable», а потім підписують спостерігача на нього, після чого попередньо визначений механізм запускає дію зі спостерігачем, який стоїть на сторожі, вловлюючи та реагуючи на його випромінювання (emit), коли вони будуть готові.

Перевага цього підходу полягає в тому, що коли у розробника є багато завдань, які не залежать одне від одного, розробник може розпочати їх усі одночасно, а не чекати, поки кожне закінчиться, перш ніж розпочинати наступне — таким чином на виконання усього пакета завдань потрібно стільки ж часу, скільки й для виконання найдовшого завдання в пакеті.

У ReactiveX спостерігач підписується на Observable. Потім цей спостерігач реагує на будь-який об'єкт або послідовність об'єктів, які видає Observable. Цей патерн полегшує виконання паралельних операцій, оскільки йому не потрібно блокуватися, чекаючи, поки об'єкт спостереження випустить об'єкти, а замість цього він створює вартового у вигляді спостерігача, який готовий відреагувати належним чином, коли об'єкт спостереження випустить їх у майбутньому.

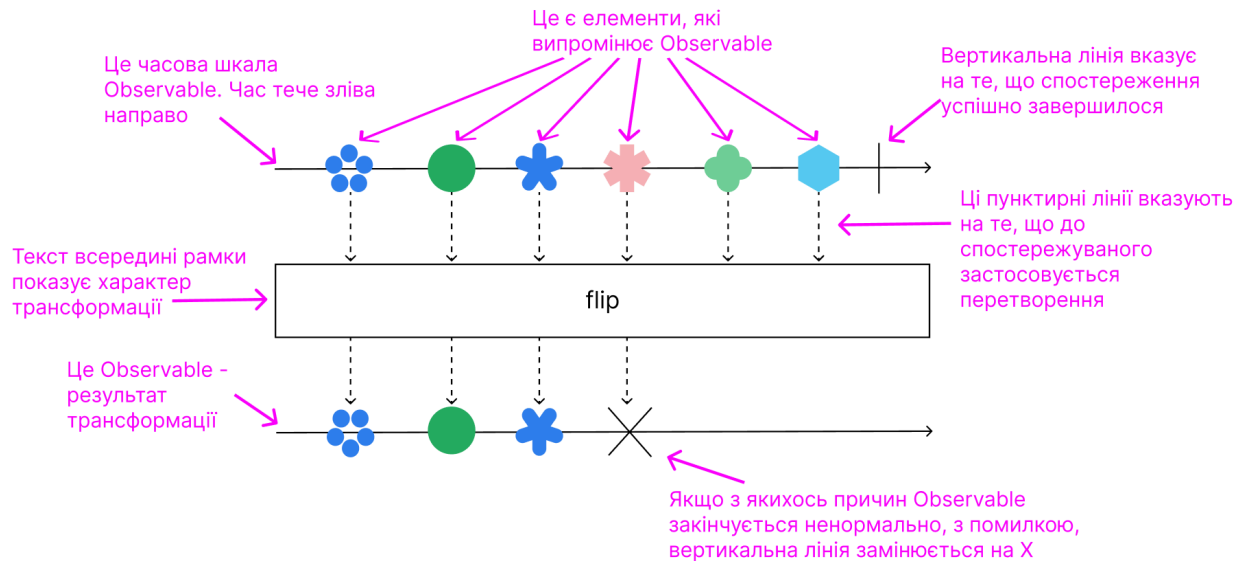


Рисунок 1.1 – Діаграма, що представле спостережувані об'єкти та перетворення спостережуваних об'єктів

При звичайному виклику методу - тобто не асинхронному, паралельному виклику, типовому для ReactiveX - потік виглядає приблизно так:

```
// Викликати метод.
// Зберегти значення, що повертається з цього методу, у змінній.
returnVal = someMethod(itsParameters);
// Використовувати цю змінну та її нове значення для чогось корисного.
```

В асинхронній моделі потік виглядає приблизно так:

- 1) Визначити метод, який робить щось корисне зі значенням, що повертається з асинхронного виклику; цей метод є частиною спостерігача.
- 2) Визначити сам асинхронний виклик як Observable.
- 3) Приєднати спостерігача до цього Observable, підписавши його (це також ініціює дії Observable).
- 4) Продовжувати робити свої справи; щоразу, коли виклик повернеться, метод спостерігача почне оперувати зі значенням або значеннями, що повертаються - елементами, які випромінює спостережуваний об'єкт.

Виглядає це приблизно так:

```
// Визначає, але не викликає обробник onNext Subscriber'a
def myOnNext = { it -> do something useful with it };
// Визначає, але не викликає, the Observable
def myObservable = someObservable(itsParameters);
// Підписує Subscriber на Observable, та викликає Observable
myObservable.subscribe(myOnNext);
```

Кожна мовна реалізація ReactiveX має власні особливості іменування. Не існує канонічного стандарту іменування, хоча між реалізаціями є багато спільних рис.

Крім того, деякі з цих імен мають різні значення в інших контекстах або здаються незручними в ідіоматиці конкретної мови реалізації.

Наприклад, існує шаблон імен на основі події onEvent (наприклад, onNext, onCompleted, onError). У деяких контекстах такі імена вказують на методи, за допомогою яких реєструються обробники подій. Однак у ReactiveX вони називають самі обробники подій.

Observable починає випромінювати свою послідовність елементів в залежності від спостережуваного. "Гарячий" observable може почати випромінювати елементи відразу після створення, і тому будь-який observer, який пізніше підпишеться на цю observable, може почати спостерігати послідовність десь посередині. З іншого боку, "холодний" observable чекає, поки на нього не підпишеться observer, перш ніж почне випромінювати елементи, і тому такий спостерігач гарантовано побачить всю послідовність з самого початку.

У деяких реалізаціях ReactiveX існує також так званий "підключуваний (connectable)" Observable. Такий Observable не починає випромінювати елементи, доки не буде викликано його метод Connect, незалежно від того, чи підписався на нього якийсь спостерігач, чи ні.

1.4 Streams and Stream Controllers

Реактивне програмування в Dart розкриває свій справжній потенціал завдяки маніпуляціям з потоками, і RXDart стоїть на передовій, пропонуючи надійний фреймворк для управління асинхронними потоками даних. У цьому розділі ми

заглиблюємося у фундаментальні концепції потоків і ключову роль, яку відіграють контролери потоків у бібліотеці RXDart, розуміючи, як вони спільно створюють реактивний ландшафт у додатках Dart.

Потоки в основі реактивного програмування

Потоки в контексті реактивного програмування - це послідовності даних або подій, які розгортаються в часі. Вони забезпечують універсальну абстракцію для обробки асинхронних завдань, таких як вхідні дані користувача, мережеві запити або оновлення в реальному часі. У RXDart потоки стають будівельними блоками для моделювання та реагування на динамічні потоки даних, дозволяючи розробникам застосовувати більш інтуїтивний та виразний підхід до асинхронного програмування.

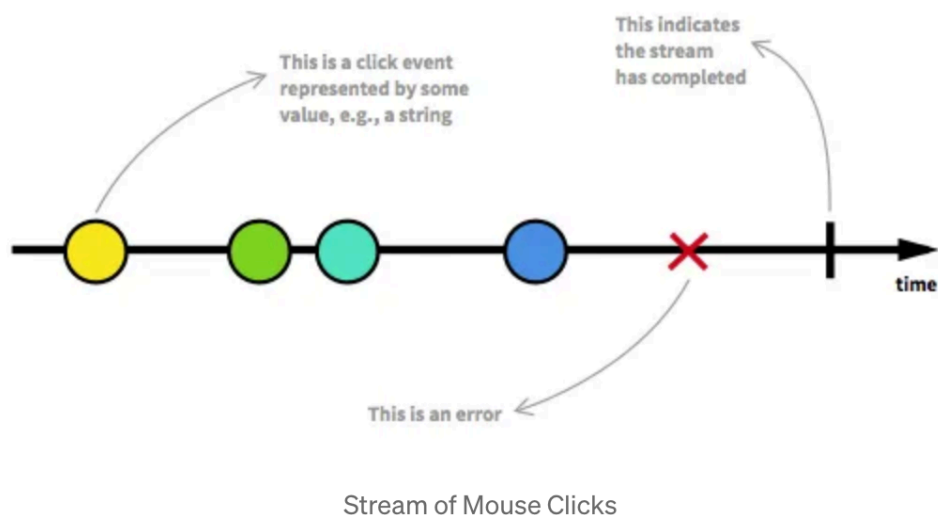


Рисунок 1.2 – Діаграма, що представляє потік об'єктів [1]

Представлення потоків у RXDart

RXDart представляє потоки за допомогою концепції спостережуваних об'єктів (Observables). Спостережуваний об'єкт (Observable), як випливає з назви, є сутністю, яку можна спостерігати в часі, випромінюючи значення або події під час проходження потоку. Об'єкти спостереження слугують джерелом даних у RXDart, а розробники підписуються на ці об'єкти спостереження, щоб реагувати на випромінювані значення. Цей підхід, орієнтований на потік, легко узгоджується з

парадигмою реактивного програмування, сприяючи створенню чистої та модульної архітектури в додатках Dart.

Роль контролерів потоків

У той час як потоки забезпечують представлення асинхронних потоків даних, контролери потоків є диригентами, які організовують симфонію подій. У RXDart контролер потоку є механізмом для управління та контролю потоку даних у потоці. Він діє як міст між Observables та Observers, дозволяючи розробникам здійснювати контроль над емісією та підпискою значень у потоці. Контролери потоку вводять додатковий рівень абстракції, полегшуючи маніпулювання та координацію даних у більш організований спосіб.

Створення та керування потоками

У RXDart створення та керування потоками полегшується завдяки використанню контролерів потоків. Розробники можуть створювати екземпляри контролерів потоків, які слугують як джерелом, так і слухачем потоку. Контролер потоку інкапсулює логіку випромінювання нових значень у потік та обробки підписок від спостерігачів. Таке розділення завдань покращує читабельність та модульність коду, полегшуючи розуміння та підтримку оркестрування асинхронних потоків даних.

Початок випромінювання даних

Процес починається з викиду значень у потік. Використовуючи контролер потоку, розробники можуть виштовхувати нові дані або події в потік, викликаючи каскад реакцій спостерігачів, підписаних на цей потік. Ця емісія значень може бути синхронною або асинхронною, пропонуючи гнучкість у роботі з різними типами джерел даних у додатках Dart.

Підписка спостерігачів

Спостерігачі - суб'єкти, підписані на потік, відіграють вирішальну роль у реактивному ландшафті. Коли спостерігач підписується на потік, керований контролером потоку, він висловлює зацікавленість в отриманні та реагуванні на значення, випромінювані цим потоком. Підписка формує життєво важливий зв'язок між об'єктами спостереження і спостерігачами, дозволяючи потоку даних досягти

свого призначення. Можливість динамічно підписуватись та відписуватись надає розробникам тонкий контроль над тривалістю життя їхніх реактивних компонентів.

Обробка backpressure та помилок

Контролери потоку в RXDart дають розробникам можливість керувати backpressure (тобто сценарієм, коли швидкість випромінюваних значень перевищує швидкість, з якою спостерігачі можуть їх обробляти). Включаючи такі стратегії, як буферизація або скидання (dropping), розробники можуть витончено керувати потоком даних, забезпечуючи стабільність і швидкість реакції своїх додатків. Крім того, контролери потоку полегшують інтеграцію механізмів обробки помилок, дозволяючи м'яко обробляти помилки в потоці, ще більше підвищуючи надійність додатків Dart.

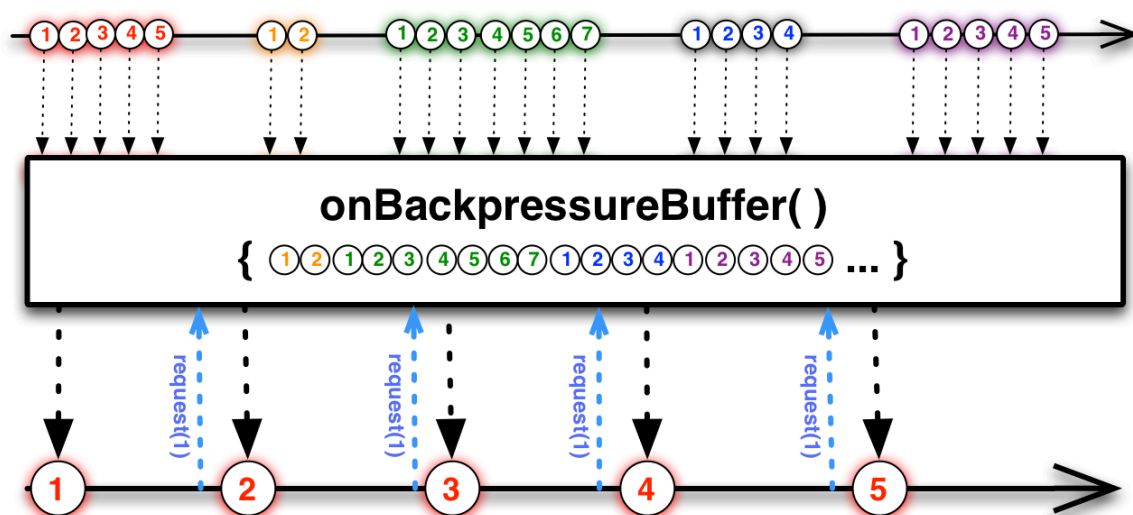


Рисунок 1.3 – Приклад backpressure [2]

1.5 Оператори в RXDart

Кожна мовна реалізація ReactiveX має свій набір операторів. Хоча між реалізаціями існує багато перетинів, є також деякі оператори, які реалізовані лише в певних реалізаціях. Крім того, кожна реалізація має тенденцію називати свої оператори так, щоб вони нагадували назви подібних методів, які вже знайомі з інших контекстів цієї мови.

Більшість операторів оперують зі спостережуваним і повертають спостережуване. Це дозволяє застосовувати ці оператори один за одним, ланцюжком.

Кожен оператор у ланцюжку модифікує спостережуваний об'єкт, який є результатом роботи попереднього оператора.

Існують патерни, наприклад, патерн “Будівельник”, в якому різноманітні методи певного класу працюють з об'єктом цього ж класу, модифікуючи цей об'єкт за допомогою роботи методу. Ці патерни також дозволяють з'єднувати методи у подібний спосіб. Але якщо у патерні Builder порядок, в якому методи з'являються у ланцюжку, зазвичай не має значення, то у випадку з операторами Observable порядок має значення.

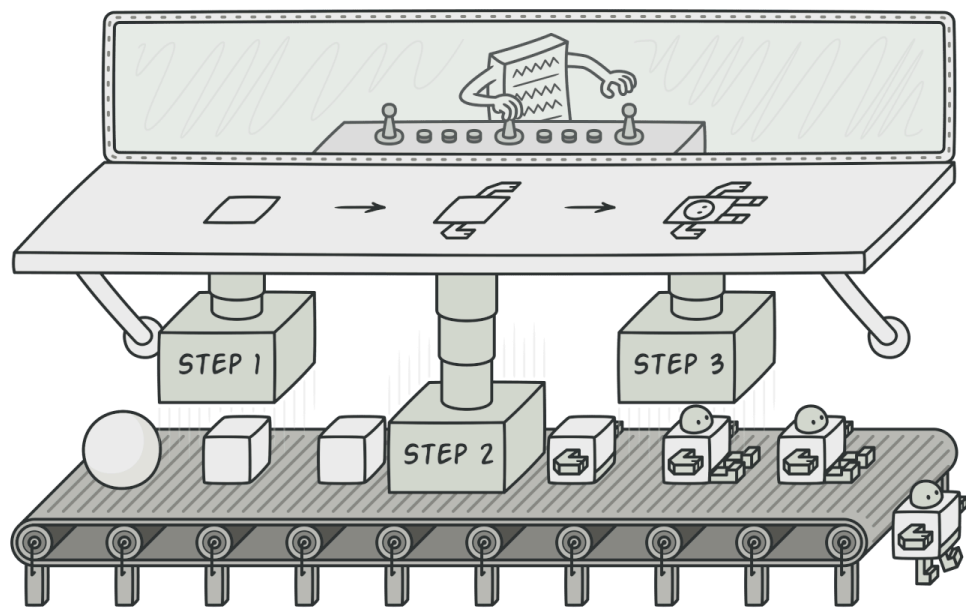


Рисунок 1.4 – Паттерн “Будівельник” [3]

Ланцюжок операторів Observable не працює незалежно від початкового Observable, з якого починається ланцюжок, але вони працюють по черзі, кожен з них працює з Observable, створеним оператором, що стоїть безпосередньо перед ним у ланцюжку.

Категорії операторів [4].

Створення спостережуваних даних

- 1) Create – створити спостережуваний об'єкт з нуля шляхом програмного виклику методів об'єкта.

- 2) Defer – не створювати спостережуваний об'єкт, доки спостерігач не підпишеться, і створювати новий спостережуваний об'єкт для кожного спостерігача.
- 3) Empty/Never/Throw – створювати спостережувані, які мають дуже точну і обмежену поведінку.
- 4) From – перетворити деякий інший об'єкт або структуру даних у спостережуваний об'єкт.
- 5) Interval – створити Observable, який видає послідовність цілих чисел, розділених певним часовим інтервалом.
- 6) Just – перетворити об'єкт або набір об'єктів на спостережуваний об'єкт, який видає цей або ці об'єкти.
- 7) Range – створити Observable, який випромінює діапазон послідовних цілих чисел.
- 8) Repeat – створити спостережуваний об'єкт, який видає певний елемент або послідовність елементів багаторазово.
- 9) Start – створити спостережуваний об'єкт, який випромінює значення функції, що повертається.
- 10) Timer – створити Observable, який виводить один елемент після заданої затримки.

Перетворення спостережуваних даних

- 1) Buffer – періодично збирати елементи зі спостережуваного об'єкта у пакети і випромінювати ці пакети замість того, щоб випромінювати елементи по одному.
- 2) FlatMap – перетворює елементи, що випромінюються спостережуваним об'єктом, у спостережувані об'єкти, а потім згладжує випромінювання з них в один спостережуваний об'єкт.
- 3) GroupBy – розбити об'єкт спостереження на набір об'єктів спостереження, кожен з яких випромінює окрему групу елементів від вихідного об'єкта спостереження, впорядковану за ключем

- 4) Map – трансформувати елементи, які випромінює спостережуваний об'єкт, застосовуючи функцію до кожного елемента.
- 5) Scan – послідовно застосовує функцію до кожного елемента, випромінюваного спостережуваним об'єктом, і випромінює кожне наступне значення.
- 6) Window – періодично розбивати елементи зі спостережуваного об'єкта на вікна спостережуваного об'єкта і випромінювати ці вікна замість того, щоб випромінювати елементи по одному.

Фільтрування спостережуваних

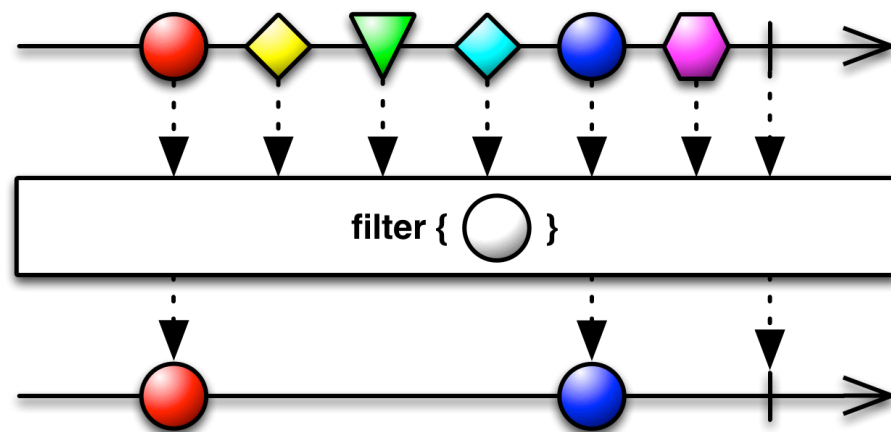


Рисунок 1.5 – Фільтр об'єктів [4]

- 1) Debounce – випускати елемент зі спостережуваного об'єкта лише тоді, коли минув певний проміжок часу, протягом якого він не випускав іншого елемента.
- 2) Distinct – виключити дублікати елементів, що випромінюються об'єктом спостереження.
- 3) ElementAt – виводити лише елемент n , який випромінює спостережуваний об'єкт.
- 4) Filter – виводити тільки ті елементи з Observable, які пройшли перевірку на предикат.
- 5) First – виводити тільки перший елемент або перший елемент, який відповідає умові, з Observable.
- 6) IgnoreElements – не виводити жодного елемента зі спостережуваного об'єкта, але віддзеркалювати його сповіщення про завершення роботи.

- 7) Last – виводити лише останній елемент, випущений об'єктом спостереження.
- 8) Sample – випускати останній елемент, випущений об'єктом спостереження через певні проміжки часу.
- 9) Skip – ігнорувати перші n елементів, випущених об'єктом спостереження
- 10) SkipLast – вилучити останні n елементів, випромінюваних об'єктом спостереження
- 11) Take – видавати лише перші n елементів, випущених об'єктом спостереження.
- 12) TakeLast – видавати лише останні n елементів, виданих об'єктом спостереження.

Поєднання спостережуваних даних

- 1) And/Then/When – об'єднати набори елементів, що були відправлені двома або більше об'єктами спостереження за допомогою посередників Pattern та Plan.
- 2) CombineLatest – коли елемент випромінюється будь-яким із двох об'єктів спостереження, об'єднати останній елемент, випромінюваний кожним об'єктом спостереження, за допомогою вказаної функції і випромінювати елементи на основі результатів цієї функції.
- 3) Join – об'єднати елементи, випромінювані двома спостережуваними об'єктами, коли елемент від одного з них випромінюється протягом часового вікна, визначеного відповідно до елемента, випромінюваного іншим об'єктом.
- 4) Merge – об'єднати декілька спостережуваних об'єктів в один шляхом злиття їхніх викидів.

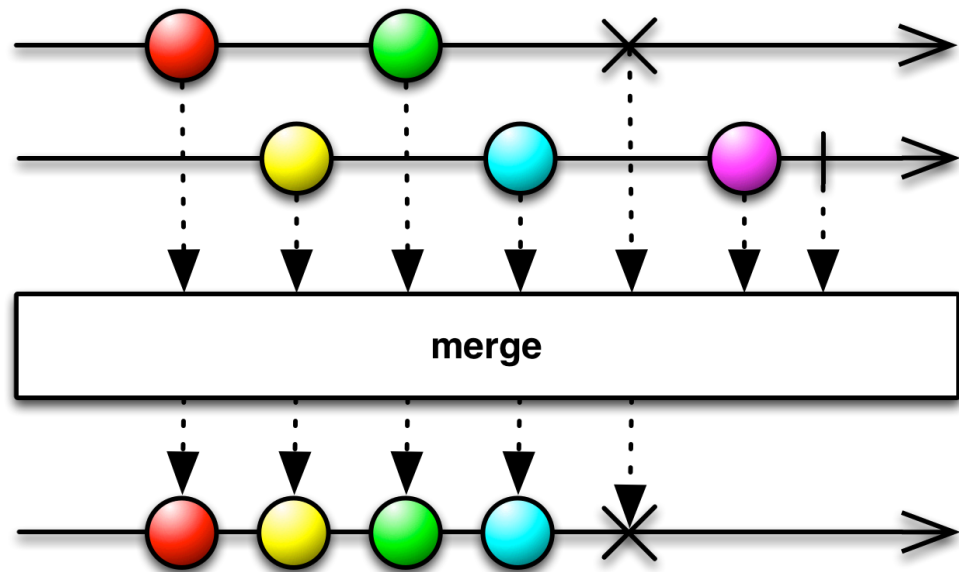


Рисунок 1.6 – Merge об'єктів [5]

- 5) **StartWith** – випромінювати певну послідовність елементів перед початком випромінювання елементів від джерела Observable.
- 6) **Switch** – перетворити об'єкт спостереження, який видає об'єкти спостереження, на єдиний об'єкт спостереження, який видає елементи, що були видані останнім з цих об'єктів спостереження.
- 7) **Zip** – об'єднати викиди декількох спостережуваних об'єктів за допомогою певної функції і випускати окремі елементи для кожної комбінації на основі результатів цієї функції.

Оператори для обробки помилок

- 1) **Catch** – відновитися після сповіщення `onError`, продовживши послідовність без помилок.
- 2) **Retry** – якщо джерело Observable надсилає сповіщення `onError`, повторно підписатися на нього в надії, що воно завершиться без помилок.

Умовні і логічні оператори

- 1) **All** – визначити, чи всі елементи, випущені спостережуваним об'єктом, відповідають певним критеріям.

- 2) `Amb` – за наявності двох або більше спостережень-джерел визначати всі елементи, випромінювані лише першим із цих спостережень, яке випромінює елемент.
- 3) `Contains` – визначити, чи випускає спостережуваний об'єкт певний елемент чи ні.
- 4) `DefaultIfEmpty` – видати елементи з джерела `Observable` або елемент за замовчуванням, якщо джерело `Observable` не випускає нічого.
- 5) `SequenceEqual` – визначити, чи два обсерватори видають однакову послідовність елементів.
- 6) `SkipUntil` – відкидати елементи, які видає `Observable`, доки другий `Observable` не видасть елемент.
- 7) `SkipWhile` – відкидати елементи, видані спостережуваним об'єктом, доки вказана умова не стане хибною.
- 8) `TakeUntil` – відкидати елементи, що видаються об'єктом спостереження, після того, як другий об'єкт спостереження видасть елемент або завершить свою роботу.
- 9) `TakeWhile` – відкидати елементи, повернені спостережуваним об'єктом після того, як вказана умова стане хибною.

З'єднувальні оператори, які можна спостерігати

- 1) `Connect` – наказати підключуваному `Observable` почати надсилати елементи своїм підписникам
- 2) `Publish` – перетворити звичайний `Observable` на підключуваний `Observable`.
- 3) `RefCount` – змусити підключуваний спостережуваний об'єкт поводитися як звичайний спостережуваний об'єкт.
- 4) `Replay` – гарантує, що всі спостерігачі побачать ту саму послідовність відправлених елементів, навіть якщо вони підписалися після того, як `Observable` почав відправляти елементи.

Оператори для перетворення спостережуваних

- 1) `To` – перетворити `Observable` в інший об'єкт або структуру даних.

1.6 Обробка помилок та стратегія повторних спроб

Обробка помилок і стратегії повторних спроб є невід'ємними частинами парадигми адаптивного програмування. Розуміння того, як ефективно обробляти помилки і досягати ефективного управління завданнями за допомогою механізмів потоків і планувальника RXDart, має важливе значення для створення надійних і високопродуктивних Dart-додатків.

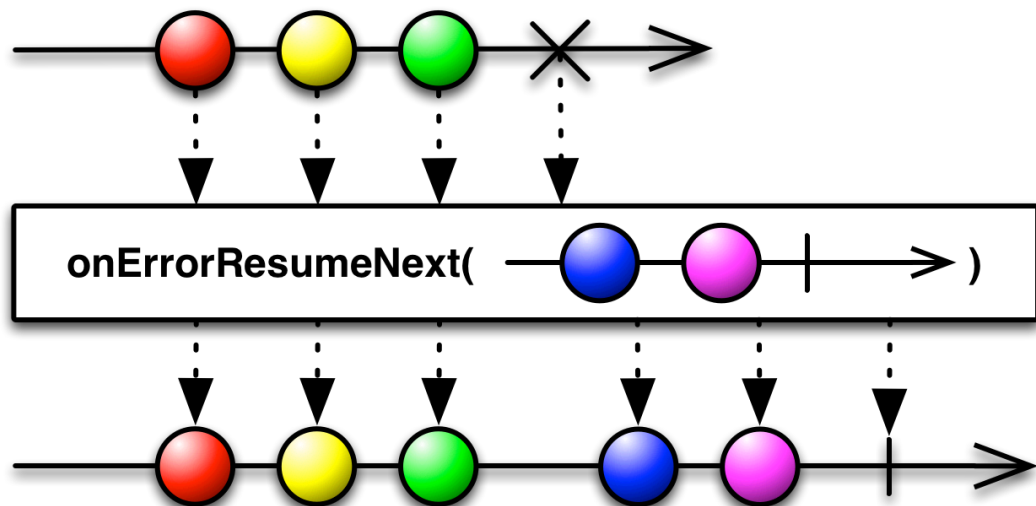


Рисунок 1.7 – Приклад обробки помилок [6]

Реактивне програмування наголошує на обробці асинхронних та керованих подіями операцій, в яких потоки відіграють ключову роль. Threads - це найменші одиниці, які виконують комп'ютерні інструкції, і їх правильне використання в асинхронному середовищі може підвищити продуктивність і швидкість реакції програми. У RXDart роль потоків полягає не тільки у виконанні коду, але й в обробці паралельних завдань та ізоляції помилок.

Планувальник - це механізм координації та управління виконанням паралельних завдань. В адаптивному програмуванні роль планувальника є вирішальною, оскільки він гарантує, що завдання виконуються в заздалегідь визначеному порядку і в певний час. Планувальник не тільки покращує читабельність програми, але й ефективно уникає паралельних конфліктів та боротьби за ресурси. У RXDart планувальник розроблений таким чином, щоб витончено обробляти паралельні завдання, забезпечуючи стабільність і ремонтпридатність програми.

У RXDart обробка потоків в основному досягається за допомогою асинхронного програмування. Оскільки Dart є однопотоковою мовою виконання, RXDart розумно використовує механізм циклу обробки подій у своїй структурі для забезпечення асинхронного виконання завдань. Завдяки циклу обробки подій RXDart може ефективніше обробляти асинхронні операції, мінімізуючи проблеми безпеки потоків.

RXDart надає абстракцію планувальника, щоб дозволити розробникам краще керувати паралельними завданнями. Планувальник дозволяє розробнику визначити порядок виконання, пріоритет і затримку завдань. У складних Dart-додатках правильне використання планувальників може оптимізувати продуктивність програми і гарантувати, що завдання виконуються у бажаний спосіб.

Помилки неминучі в асинхронних операціях, і RXDart вирішує цю проблему, впроваджуючи надійні механізми обробки помилок. Розробники можуть перехоплювати та обробляти помилки за допомогою операторів, наданих RXDart, гарантуючи, що програми обробляються належним чином, коли стикаються з винятковими ситуаціями. Крім того, RXDart надає гнучку стратегію повторних спроб, яка дозволяє розробникам вирішувати, чи потрібно повторювати невдалу спробу в кожному конкретному випадку, і якщо так, то скільки разів.

Обробка помилок і стратегія повторних спроб є ключовими темами в RXDart, які мають безпосередній вплив на стабільність і продуктивність програми. Отримавши більш глибоке розуміння ролі потоків, важливості планувальника та механізмів обробки помилок і повторних спроб RXDart, розробники можуть краще створювати надійні та ефективні Dart-додатки. У все більш складних прикладних сценаріях ефективне управління паралельними завданнями і обробка помилок стануть ключовим фактором забезпечення успішної роботи додатків. Добре продумані механізми багатопотоковості та планувальника RXDart надають розробникам потужні інструменти для вирішення проблем асинхронного програмування і гарантують, що Dart-додатки будуть добре працювати за будь-яких умов.

1.7 Потоки та планувальники (schedulers)

Реактивне програмування - це парадигма, яка має справу з асинхронними та керованими подіями операціями, і в цій парадигмі потоки та планувальники відіграють вирішальну роль.

Реактивне програмування робить акцент на перетворенні асинхронних та керованих подій операцій у потоки для більш природної та інтуїтивно зрозумілої роботи з даними. І потоки відіграють ключову роль у цій парадигмі. Нитки - це найменші одиниці, які виконують комп'ютерні інструкції, і їх правильне використання в асинхронному середовищі може підвищити продуктивність і швидкість реакції програми.

У RXDart обробка потоків в основному досягається за допомогою асинхронного програмування. Оскільки Dart є однопоточною мовою виконання, RXDart розумно використовує механізм циклу обробки подій у своїй структурі для забезпечення асинхронного виконання завдань. Завдяки циклу обробки подій, RXDart може ефективніше обробляти асинхронні операції, уникаючи при цьому проблем з безпекою потоків, наскільки це можливо.

RXDart надає гнучкий набір механізмів співпраці потоків, щоб гарантувати, що при обробці потоків даних не відбувається блокування або погіршення продуктивності. Використовуючи асинхронні оператори, розробники можуть краще контролювати виконання потоків, досягаючи таким чином хорошої продуктивності у високо паралельних сценаріях.

Планувальник - це механізм для координації та управління виконанням паралельних завдань. В адаптивному програмуванні порядок і час виконання завдань є критично важливими. Планувальники гарантують, що завдання виконуються у визначеному порядку та у визначений час, надаючи контекст та механізми контролю виконання завдань. Це важливо для покращення читабельності програми та уникнення одночасних конфліктів і боротьби за ресурси.

RXDart надає абстракцію планувальника, що дозволяє розробникам краще керувати паралельними завданнями. Планувальник дозволяє розробникам визначати порядок виконання, пріоритет і час затримки завдань. У складних Dart-додатках

правильне використання планувальників може оптимізувати продуктивність програми і гарантувати, що завдання будуть виконуватися в потрібний спосіб.

Планувальник в RXDart розроблений таким чином, щоб бути гнучким і потужним. Він не тільки надає заздалегідь визначені планувальники, але також підтримує розробників у налаштуванні планувальника відповідно до конкретних бізнес-вимог. Ця гнучкість дозволяє вибрати найбільш підходящий планувальник для різних сценаріїв і гарантує ефективне управління паралельними завданнями.

Планувальник можна використовувати не тільки для керування порядком виконання завдань, але й для оптимізації завдань. У RXDart, вибравши різні планувальники, можна ефективно керувати виконанням завдань, так що оптимізація завдань може бути досягнута в різних сценаріях. Наприклад, планувальник затримки може бути використаний для відкладення виконання завдання відповідно до конкретних бізнес-вимог.

У RXDart обробка потоків реалізована через асинхронну природу мови Dart. Використовуючи асинхронні конструкції, такі як Future та Stream, RXDart може ефективно обробляти асинхронні завдання в однопотоковому середовищі. У той же час, RXDart надає оператори для роботи з потоками, забезпечуючи належне управління потоками в адаптивному програмуванні.

RXDart надає ряд попередньо визначених планувальників, які охоплюють різні сценарії використання. Ці планувальники включають ImmediateScheduler, NewThreadScheduler і т.д., що дозволяє розробникам вибрати правильний планувальник для своїх потреб.

1.8 Майбутні тренди та внески спільноти

RXDart, бібліотека реактивного програмування, стрімко розвивається, надихаючи екосистеми Dart і Flutter завдяки постійній еволюції та внеску спільноти.

Оскільки мобільні додатки та веб-розробка продовжують розвиватися, парадигма адаптивного програмування продовжує адаптуватися і розвиватися. Нові тенденції включають більш надійне управління станом, більш ефективну асинхронну обробку і більш розумне управління потоками даних. У RXDart ці

тенденції можуть бути відображені в більш багатих операторах, більш гнучких механізмах обробки даних і більш розумному дизайні планувальника.

У майбутніх версіях RXDart може з'явитися низка нових функцій, які відповідатимуть новим тенденціям. Це може включати більше операторів потоку даних для спрощення складних завдань обробки потоку даних і більш досконалі механізми обробки помилок для підвищення стабільності роботи програми. Крім того, можливим напрямком розвитку є краща інтеграція з іншими функціями екосистеми Dart і Flutter.

Досліджуючи нові тенденції, RXDart може стати більш глибоко інтегрованим у майбутні ідеї адаптивного програмування. Це може включати в себе більш розумну обробку потоку даних з використанням методів машинного навчання для прогнозування поведінки користувача заздалегідь і оптимізації потоку обробки даних. Така еволюція надасть розробникам більш високий рівень абстракції, що зробить асинхронне програмування більш інтуїтивно зрозумілим.

Спільнота RXDart завжди була однією з рушійних сил його розвитку. Через GitHub, форуми та інші соціальні медіа-платформи члени спільноти беруть активну участь в обговореннях, ставлять запитання та діляться своїм досвідом використання системи. Активність спільноти є життєво важливою для постійного вдосконалення та оновлення RXDart.

Стабільні та попередні версії RXDart постійно випускаються, і кожна нова версія приносить ряд поліпшень і виправлень. Спільнота розробників колективно сприяє постійному оновленню RXDart. Нові оновлені версії RXDart можуть включати підтримку останньої версії Dart, оптимізацію продуктивності та деякі корисні нові функції.

Спільнота RXDart робить внесок не лише у код, але й у документацію, підручники та підтримку інших розробників. Сила спільноти з відкритим вихідним кодом полягає в об'єднанні зусиль багатьох розробників, і спільнота RXDart не є винятком. Ділячись своїм досвідом та надаючи допомогу, учасники спільноти працюють разом, щоб підштовхнути RXDart у кращому напрямку.

Для сприяння спілкуванню та співпраці, спільнота RXDart може організовувати різні заходи та конференції. Це не тільки допомагає розробникам отримати глибше розуміння використання та розвитку RXDart, але також стимулює інновації та нові ідеї. Заходи спільноти також є важливим способом мотивувати більше людей приєднатися до спільноти RXDart.

Як одна з основних бібліотек в екосистемі Dart та Flutter, RXDart відіграє ключову роль в обробці асинхронних операцій та потоків даних. Очікується, що в майбутньому, коли Dart і Flutter продовжать розвиватися, RXDart продовжить займати важливе місце в цій екосистемі. Його спрощена модель асинхронного програмування та потужні адаптивні механізми управління потоками даних продовжуватимуть полегшувати роботу розробників.

По мірі того, як Flutter стає все більш широко використовуваним у розробці мобільних додатків, попит на адаптивне програмування продовжує зростати. RXDart, як один з інструментів, що обирають розробники Flutter для роботи з асинхронністю та потоками даних, ймовірно, стане більш глибоко інтегрованим у фреймворк Flutter у майбутньому, забезпечуючи більш інтуїтивно зрозумілий та потужний досвід розробки.

Майбутній розвиток RXDart буде продиктований не тільки технологічними тенденціями, але також буде залежати від відгуків та потреб спільноти розробників. Очікується, що завдяки постійним оновленням та вдосконаленням, RXDart продовжить лідирувати в галузі адаптивного програмування. Його відкрита природа також надає можливість розробникам брати участь у формуванні майбутнього RXDart, забезпечуючи його подальший успіх та актуальність у динамічному світі розробки Dart та Flutter.

Висновки до першого розділу

В даному розділі детально обговорюються концепції спостережуваних об'єктів і шаблонів спостерігачів, пояснюється, як їх можна використовувати в RXDart для моделювання і обробки динамічних потоків даних. Це робить більш природним управління та реагування на події в Dart-додатках. Поглиблено

розглядаються ролі потоків і контролерів потоків у RXDart, що є важливою частиною асинхронного програмування.

Цей розділ дає всебічне уявлення про важливість RXDart у розробці Dart, від базових концепцій до просунутих додатків, дозволяючи читачам повністю зрозуміти і використовувати цей потужний інструментарій адаптивного програмування.

РОЗДІЛ 2 ПОСТАНОВКА ЗАВДАННЯ

2.1 Визначення основних завдань та проблем, які розв'язує RXDart

RXDart є потужним інструментом для програмування в мові Dart, який ідеально впроваджує концепції ReactiveX (RX, або Reactive Programming). Описані програми, які імплементують та не імплементують зазначений вище реактивний підхід, надають відмінну можливість виявити, як саме RXDart вирішує конкретні завдання та долає проблеми у заданому контексті.

Наприклад, коли постає проблема, що управління асинхронним кодом та подіями ускладнює програмний процес та робить код програми важкозрозумілим. RXDart, як вирішення, забезпечує простоту та зрозумілість у коді, зокрема, значно полегшує обробку подій.

Іншим проблемним контекстом інколи може бути управління станом програми (наприклад, при використанні архітектур, що створюють один чи декілька глобальних станів програми, до яких має доступ майже кожен модуль). Розробнику може бути заскладно відслідковувати та управляти змінами стану, особливо у великих програмах. В цьому випадку застосування реактивного підходу для автоматизації відслідковування та реагування на зміни стану, спрощує управління програмою. З якогось боку, це може виглядати як синтаксичний цукор, але демонстрація розроблених прикладів з та без використання реактивного програмування розвіє цей сумнів.

Іншими прикладами проблематичних контекстів є ефективне використання ресурсів та паралельна обробка завдань у асинхронному середовищі, складнощі при об'єднанні та трансформації різних потоків даних, оновлення та синхронізація інтерфейсу користувача в реальному часі.

RXDart стає невід'ємною частиною вирішення конкретних завдань, що надає ефективний та зручний інструментарій для реалізації реактивного підходу у програмуванні. У подальших пунктах цього розділу роботи розглянуто приклади додатків, що дозволяють зрозуміти, наскільки ефективним або неефективним є використання цієї бібліотеки у конкретному контексті.

2.2 Власні дослідження та використання RXDart

2.2.1 Приклад 1. Searchable data

Програма розроблена для ефективного управління та відображення елементів, зокрема для виконання операцій пошуку. Кожен елемент представлений класом `Item`, що має атрибут `ім'я`.

Функціональні вимоги:

- 1) управління елементами – програма повинна забезпечувати можливість додавання, видалення та редагування елементів;
- 2) пошук елементів – потрібно забезпечити можливість швидкого та ефективного пошуку елементів за їх іменем;
- 3) фільтрація та відображення результатів пошуку – при введенні запиту у поле пошуку програма повинна фільтрувати елементи та відображати в режимі реального часу результати пошуку.

Експлуатаційні вимоги:

- 1) введення пошукового запиту – користувач повинен мати можливість вводити пошуковий запит у відповідне текстове поле;
- 2) відображення відфільтрованих результатів – відфільтровані результати пошуку повинні динамічно відображатися у `ListView.builder`;
- 3) реалізація пошукового механізму – метод `search` повинен ефективно фільтрувати елементи відповідно до введеного пошукового запиту;
- 4) адаптація результатів пошуку до змін – результати пошуку повинні автоматично оновлюватися при зміні пошукового запиту;
- 5) інтерфейс користувача – інтерфейс користувача повинен бути інтуїтивно зрозумілим та забезпечувати зручний доступ до функціоналу пошуку та відображення результатів.

Результати роботи:

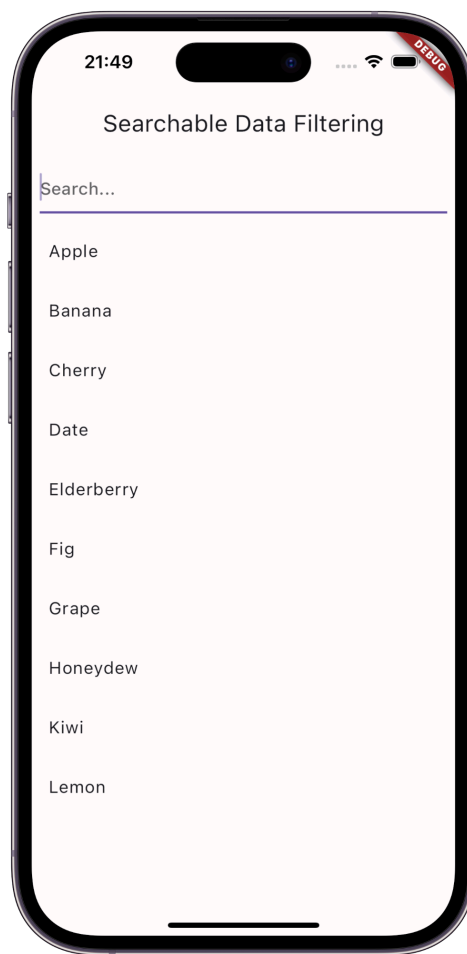


Рисунок 2.1 – Приклад “пошук елементів”.

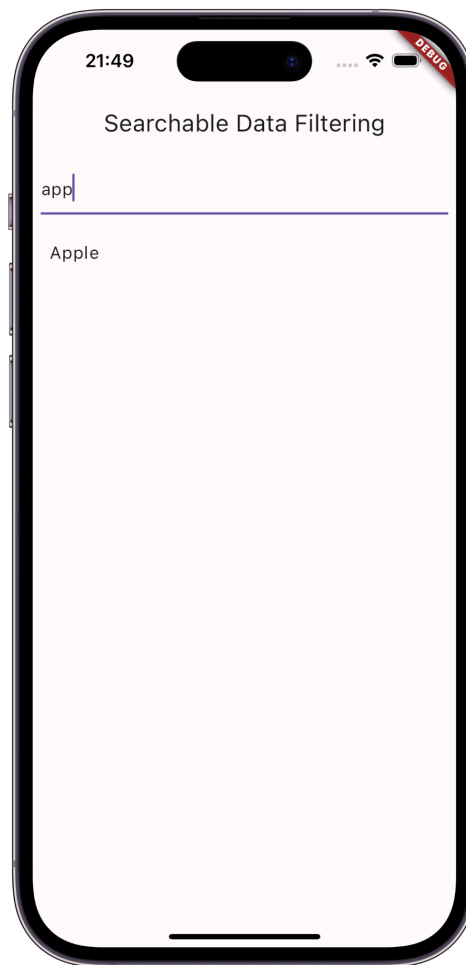


Рисунок 2.2 – Виконання пошуку елементів.

2.2.2 Приклад 2. Sports scoreboard

Програма призначена для відслідковування та відображення результатів спортивних подій в режимі реального часу. Оновлення результатів подій автоматично відображається у реальному часі.

Функціональні вимоги:

- 1) одержання результатів спортивних подій – програма повинна здійснювати одержання та оновлення результатів спортивних подій;
- 2) оновлення результатів у реальному часі – результати спортивних подій повинні оновлюватись автоматично та в реальному часі;
- 3) відображення індикатора завантаження – при очікуванні нових даних програма повинна відображати індикатор завантаження.

Експлуатаційні вимоги:

- 1) відображення результатів на інтерфейсі – результати спортивних подій повинні бути відображені на інтерфейсі користувача;
- 2) динамічне оновлення інтерфейсу – інтерфейс повинен автоматично оновлюватися при отриманні нових результатів спортивних подій.
- 3) моделювання періодичних оновлень – метод `fetchLiveScores` повинен моделювати періодичні оновлення результатів (наприклад, кожні 5 секунд).
- 4) генерація випадкових результатів – метод `_generateRandomScore` повинен генерувати випадкові результати для демонстраційних цілей.
- 5) динамічне відображення назви та результатів подій – на інтерфейсі повинні динамічно відображатися назва та результат кожної спортивної події.

Результати роботи:

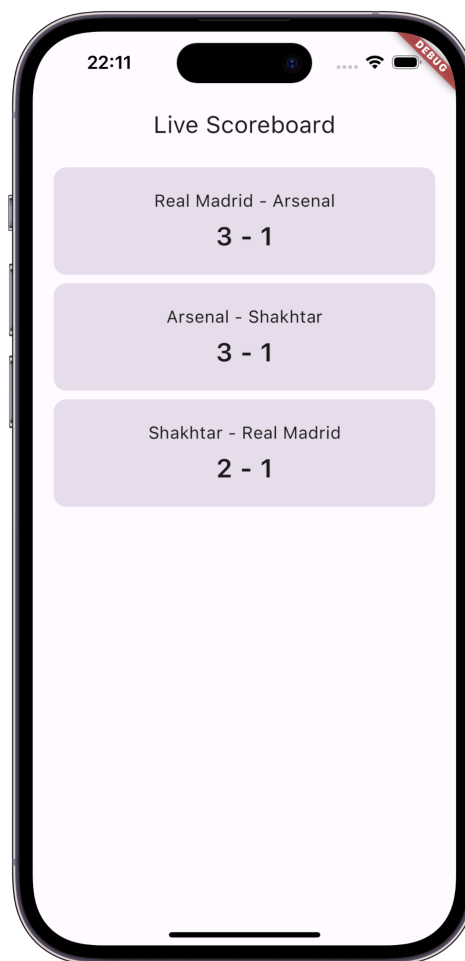


Рисунок 2.3 – приклад “оновлення в реальному часі”.

2.2.3 Приклад 3. Todo list

Програма реалізує функціональність управління списком справ (to-do list).

Кожна зміна у списку та його стані відображається в реальному часі.

Функціональні вимоги:

- 1) додавання нової справи – користувач може додати нову справу, вводячи текст у текстовому полі та натискаючи клавішу Enter або кнопку "Додати";
- 2) відмітка виконання справи – кожную справу можна позначити як виконану, торкнувшись відповідного чекбоксу;
- 3) видалення справи – справу можна видалити, тривалий час торкаючись її в інтерфейсі.

Експлуатаційні вимоги:

- 1) відображення списку справ – список справ відображається на інтерфейсі користувача;
- 2) динамічне оновлення інтерфейсу – інтерфейс автоматично оновлюється при зміні списку справ.
- 3) додаткові можливості – програма повинна показувати індикатор завантаження, коли дані обробляються, і відображати список справ, коли дані отримані.
- 4) інтерактивність інтерфейсу – користувач може взаємодіяти з програмою, додаючи, відзначаючи як виконані та видаляючи справи.

Результати роботи:

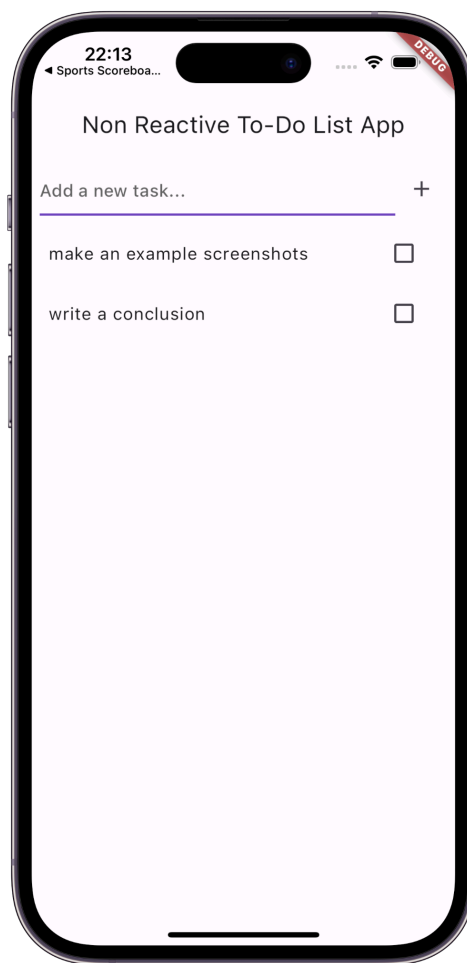


Рисунок 2.4 – пример “список справ”.

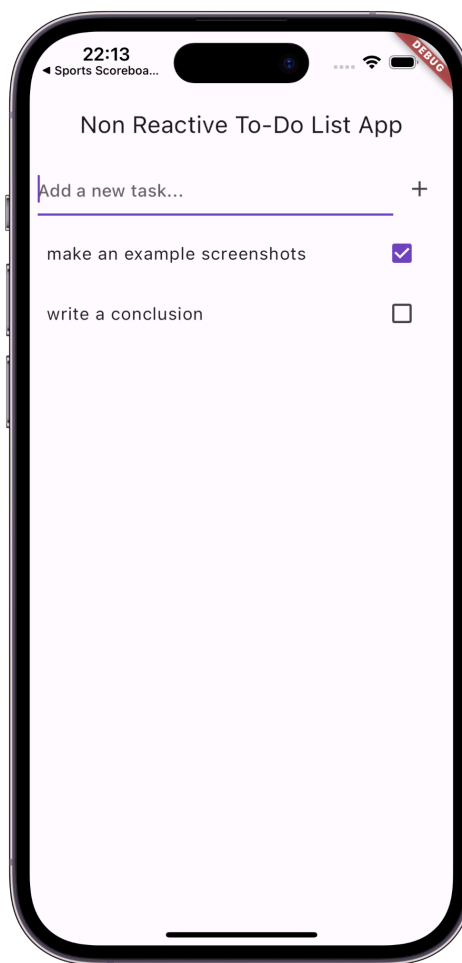


Рисунок 2.5 – приклад “список справ” з відміченим пунктом.

2.2.4 Приклад 4. User login

Програма призначена для управління потоком аутентифікації користувачів.

Функціональні вимоги:

- 1) введення та валідація даних – програма повинна дозволяти користувачеві вводити дані електронної пошти та пароля. Валідація електронної пошти та пароля повинна відбуватися в реальному часі.
- 2) аутентифікація користувача – після введення валідних даних користувача та натискання кнопки входу, повинна відбутися імітація процесу входу в систему.
- 3) модульність та розширюваність – програма повинна бути готовою до розширення та змін в майбутньому, зокрема, заміни імітації входу на фактичний механізм аутентифікації.

Експлуатаційні вимоги:

- 1) актуалізація логіки аутентифікації – користувач повинен замінити метод `_mockLogin` на реальну логіку аутентифікації у власному додатку.
- 2) додавання залежностей – користувач повинен додати необхідні залежності для валідації текстових полів та роботи з формою у файл `pubspec.yaml`.

Результати роботи:

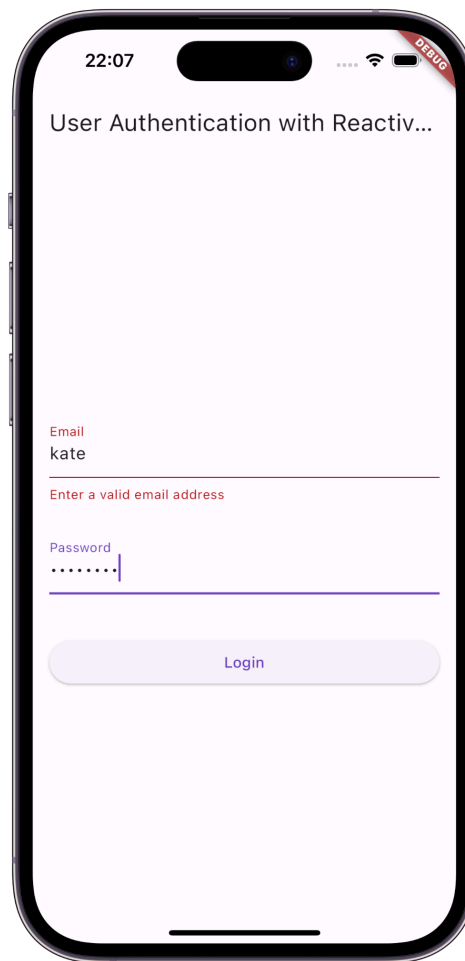


Рисунок 2.6 – Приклад “авторизація користувача”.

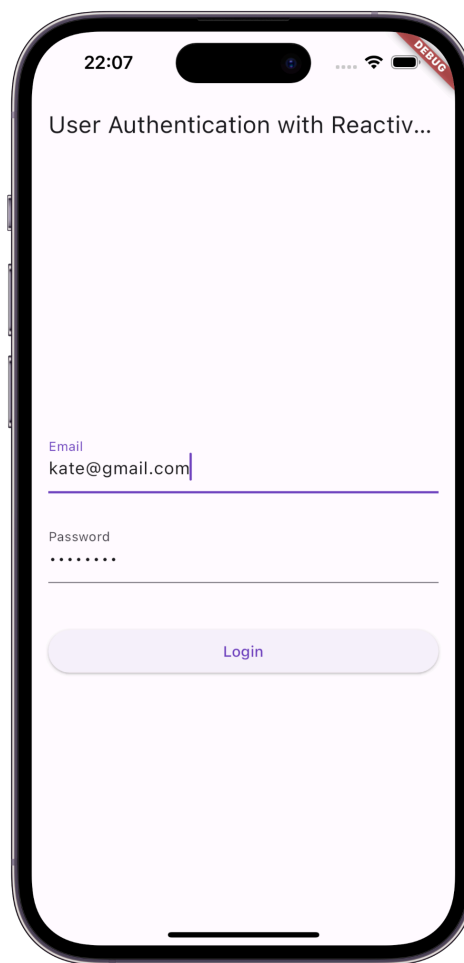


Рисунок 2.7 – виконання авторизації користувача.

2.2.5 Приклад 5. Weather app

Програма розроблена для отримання та відображення інформації про погоду.

Функціональні вимоги:

- 1) отримання та відображення даних погоди – програма повинна забезпечувати отримання та відображення актуальних даних про погоду.
- 2) періодичне оновлення даних – забезпечити періодичне оновлення даних про погоду, щоб користувач завжди отримував актуальну інформацію;
- 3) відправлення запиту до API – повинен здійснюватися запит до API для отримання інформації про погоду.
- 4) відображення індикаторів – програма повинна відображати індикатор завантаження під час очікування даних та повідомлення про помилку в разі виникнення помилок.

- 5) ініціалізація та оновлення потоку даних – програма повинна автоматично оновлювати дані погоди періодично.

Експлуатаційні вимоги:

- 1) доступ до API ключа – користувач повинен замінити 'YOUR_API_KEY' на власний ключ API для OpenWeatherMap;
- 2) інтерфейс користувача – інтерфейс користувача повинен чітко відображати стан завантаження, помилок та актуальні погодні дані.

Результати роботи:



Рисунок 2.8 – Приклад “Погода”.

Висновки до другого розділу

У цьому розділі було проаналізували різноманітні проблеми та завдання, з якими може зіткнутися розробник програмного забезпечення, та визначили, як RXDart може вирішувати ці проблеми та полегшувати вирішення завдань у програмуванні.

Описані приклади програм, які використовують та ті самі програми, що не використовують `RxDart`, ілюструють значний вплив цього інструменту на покращення розробки програмного забезпечення. Завдяки розгляду конкретних сценаріїв, ми демонструємо, як `RxDart` стає важливим елементом для вирішення реальних завдань, надаючи ефективний та зручний інструментарій для реалізації реактивного підходу у програмуванні.

Загалом, `RxDart` виявляється необхідним інструментом для вирішення проблем асинхронного програмування та реактивної обробки даних в мові `Dart`. Дослідження цього розділу дозволяє визначити його переваги та використання в реальних сценаріях, надаючи рекомендації для ефективного використання та подальших досліджень в цій області.

РОЗДІЛ 3 ПРАКТИЧНЕ ВИКОРИСТАННЯ РЕАКТИВНОГО ПІДХОДУ

3.1 Вступ

У цьому розділі представлено практичне використання реактивного програмування в контексті розроблення п'яти програм, кожна з яких імплементована двома способами: з використанням бібліотеки RxDart та без неї. Дослідження зосереджене на аналізі діаграм класів для кожної програми з метою виявлення переваг та обмежень реактивного підходу у порівнянні з традиційним нереактивним програмуванням.

Цей розділ спрямований на вивчення впливу реактивного підходу на структуру та функціональність розроблених програм. Для досягнення цієї мети, проводиться докладний аналіз діаграм класів, що дозволяє визначити взаємодію компонентів, оптимізацію операцій та ефективність коду в обох варіантах реалізації.

Таблиця 3.1 – Таблиця порівняння кількості класів.

	RX	No RX
Searchable data	4	3
Sports Scoreboard	5	5
Todo list	4	3
User interface	4	3
Weather app	5	5
Всього	22	19

3.2 Аналіз різниці імплементацій

Для аналізу найліпше підходить діаграма класів.

Діаграма класів (англ. Class diagram) – це статичне представлення структури моделі в UML. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм. [7]

Використання діаграм класів має кілька переваг:

1. візуалізація структури. Діаграми класів допомагають візуалізувати структуру системи, зробити її більш зрозумілою та легко сприйнятливою;
2. виявлення проблем. Діаграми класів дозволяють виявити можливі проблеми або недоліки в структурі системи на ранніх етапах розробки;
3. підвищення продуктивності. Використання діаграм класів дозволяє покращити комунікацію між розробниками та забезпечити однозначне розуміння структури системи;
4. модульність та розширюваність. Діаграми класів допомагають визначити модулі та їх взаємозв'язки, що сприяє покращенню модульності та розширюваності системи. [8]

3.2.1 Аналіз діаграм класів для прикладу Searchable data

У данному порівняльному аналізі розглядаються дві діаграми класів, які представляють альтернативні підходи до реалізації програми, що має за мету виконання пошуку в списку з декількох елементів. Одна діаграма відображає реактивний підхід, зокрема використання бібліотеки RxDart та BehaviorSubject, тоді як інша діаграма ілюструє нереактивний підхід.

Важливість такого порівняльного аналізу полягає в тому, щоб з'ясувати, як обидва підходи впливають на структуру та організацію програмного коду в контексті конкретного завдання – пошуку в списку. Структура програми та взаємодія між класами може відрізнитися в залежності від того, чи використовується реактивний підхід, чи нереактивний.

Цей порівняльний аналіз має на меті допомогти зрозуміти переваги та недоліки обох підходів у конкретному контексті застосування, а також визначити, який підхід може бути більш відповідним для конкретних вимог та умов проекту з точки зору читабельності, ефективності та підтримки функціональності пошуку в списку.

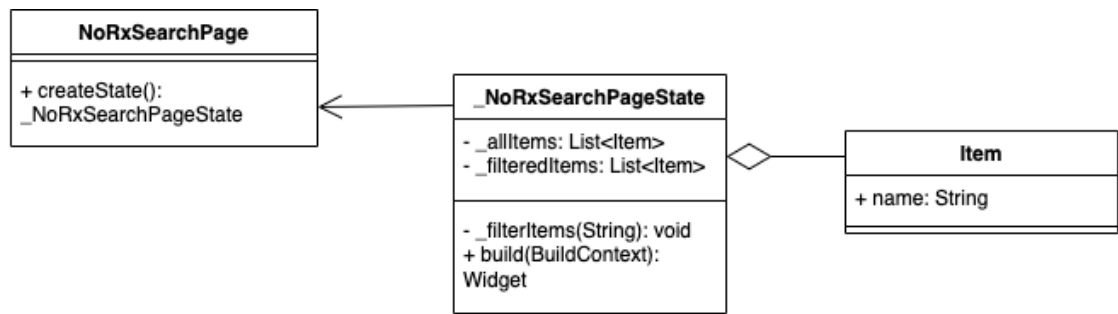


Рисунок 3.1 – Діаграма класів для нереактивного підходу.

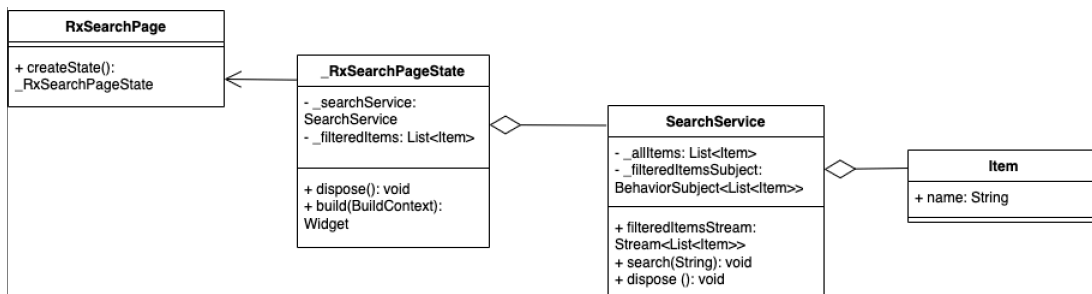


Рисунок 3.2 – Діаграма класів для реактивного підходу.

На представлених діаграмах класів очевидно, що для успішної імплементації реактивного підходу доводиться введення додаткового класу для обробки потоків даних, а саме використання класу, який взаємодіє з `BehaviorSubject` і надає необхідний механізм реактивного програмування. Цей додатковий клас реалізований за допомогою бібліотеки `RxDart`, що надає інструменти та абстракції для зручного та ефективного управління потоками подій.

Такий підхід ілюструє необхідність використання спеціалізованих інструментів для реалізації реактивних властивостей, які виходять за межі стандартного обсягу мов програмування. Введення цього додаткового класу визначає архітектурну компоненту, яка взаємодіє із зовнішнім середовищем, сприяючи ефективній обробці асинхронних подій та створенню реактивної логіки в програмі.

Отже, на діаграмах класів виокремлюється потреба у введенні спеціалізованого класу, що використовує `BehaviorSubject`, і цей момент стає ключовим у контексті реалізації реактивного підходу. Інтеграція з бібліотекою

RxDart визначає необхідність використання засобів, що надаються цією бібліотекою для ефективної роботи з потоками даних та підтримки реактивної функціональності.

3.2.2 Аналіз діаграм класів для прикладу Sports scoreboard

В даному навчальному прикладі розглядається альтернативний підхід у роботі з потоками даних у мові програмування, який полягає в використанні StreamController замість BehaviorSubject із бібліотеки Rx. Перехід від BehaviorSubject до StreamController може відобразити аспекти гнучкості та різноманітності в роботі з поточковим програмуванням.

У цьому контексті, розглядається вибір альтернативного інструменту для управління потоками даних та розбирається його застосування на прикладі конкретного сценарію. Дослідження пропонує зрозуміти, як StreamController може взаємодіяти з потоками даних та як він порівнюється із BehaviorSubject в контексті реактивного програмування.

Відштовхуючись від даного прикладу, можна долучити додаткові знання та розглянути аспекти ефективності, гнучкості та простоти реалізації при використанні StreamController у порівнянні із BehaviorSubject. Цей вступ створений для сприяння розумінню та оцінки вибору між StreamController та BehaviorSubject при роботі з потоками даних у програмуванні.

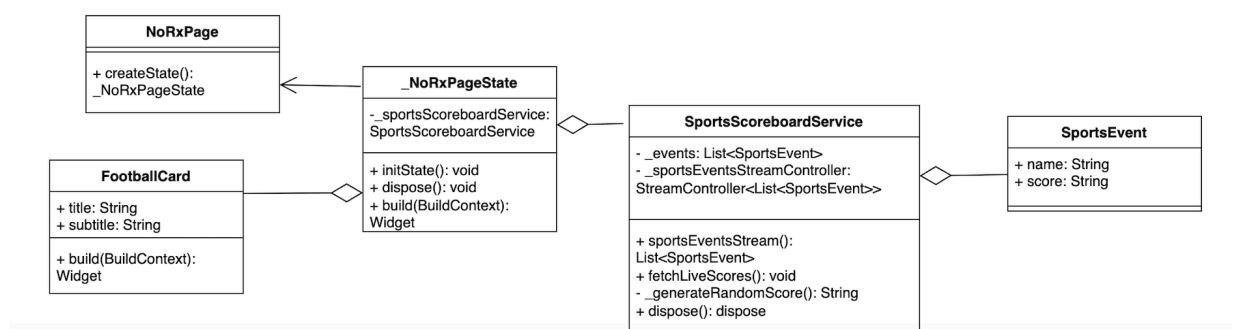


Рисунок 3.3 – Діаграма класів для реактивного підходу.

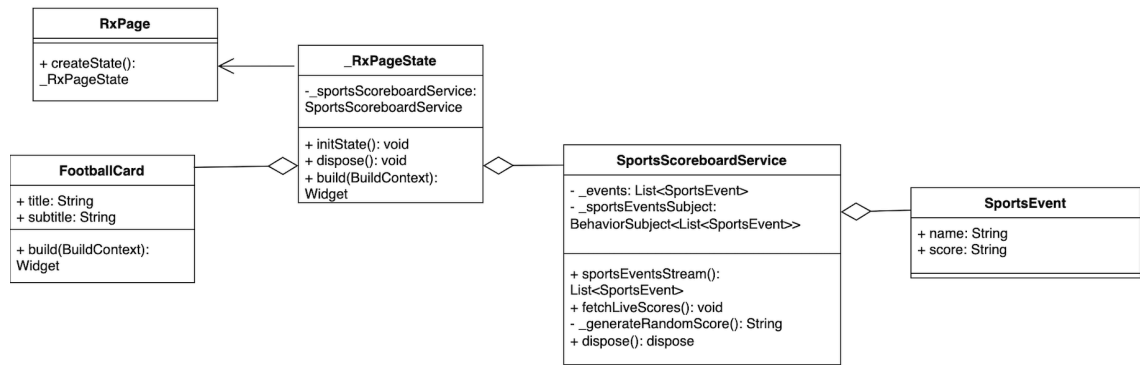


Рисунок 3.4 – Діаграма класів для реактивного підходу.

Підсумовуючи вивчальний приклад використання StreamController замість BehaviorSubject у контексті реактивного програмування, можна зробити висновок, що суттєвої різниці між цими двома реалізаціями не виявлено. Обидва інструменти дозволяють ефективно управляти потоками даних та забезпечують необхідний рівень гнучкості та контролю над реактивними операціями.

У виборі між StreamController та BehaviorSubject може впливати наявність конкретних вимог чи переваг розробленого проекту. Однак, якщо глядіти на їхні базові можливості та принципи роботи, обидва варіанти можуть бути успішно використані в реактивному програмуванні.

Оцінка вибору між цими реалізаціями повинна ґрунтуватися на конкретних вимогах та контексті використання у проекті. З одного боку, StreamController може виглядати привабливим у певних сценаріях, з іншого - BehaviorSubject може мати свої переваги в інших умовах.

Усупереч вибору конкретного інструменту, важливо враховувати особливості проекту, його масштаб та вимоги, щоб максимально використовувати переваги реактивного програмування. Таким чином, висновок підтверджує, що у даному контексті суттєвої різниці між реалізаціями StreamController та BehaviorSubject не виявлено, і вибір може залежати від конкретних потреб та умов використання у конкретному проекті.

3.2.3 Аналіз діаграм класів для прикладу Todo list

Даний приклад демонструє, що при зміні вимог попри те, що класів більше, програма, що імплементована реактивно буде легше модифікувати. Це досягається через те, що компоненти інтерфейсу не мають сильного зв'язку із бізнес логікою.

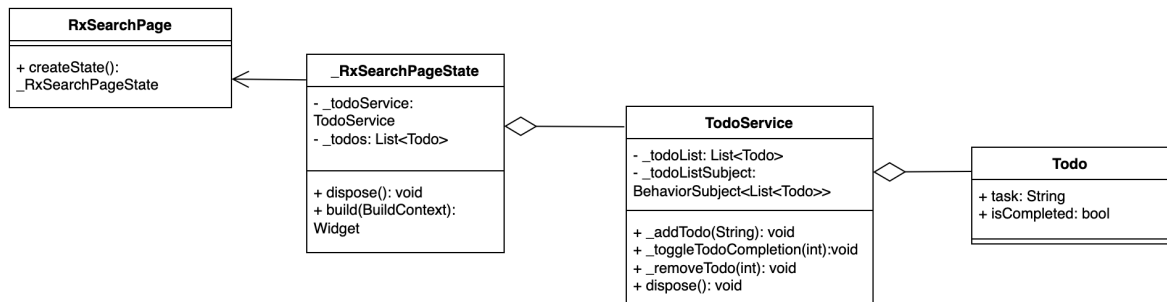


Рисунок 3.5 – Діаграма класів для реактивного підходу.

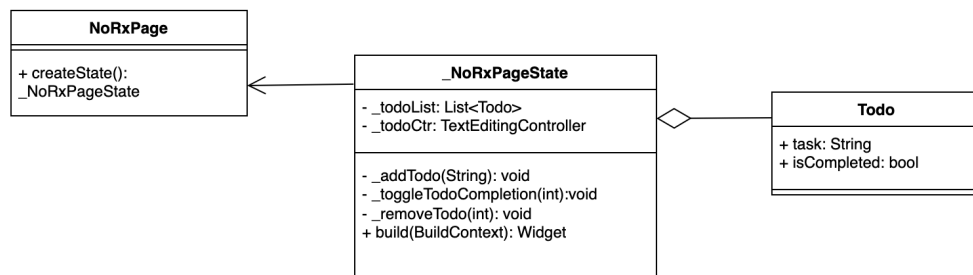


Рисунок 3.6 – Діаграма класів для нереактивного підходу.

3.2.4 Аналіз діаграм класів для прикладу User login

Цей приклад є цікавим тим, що не дивлячись на рекомендації використовувати реактивний підхід з користувацькими інтерфейсами, нереактивний підхід виграє.

Приклади імплементації є майже ідентичними. Клас **AuthService** відповідає за обробку електронної пошти та пароля користувача, а також за ключ форми та логіку валідації. Методи `validateEmail` та `validatePassword` використовуються для валідації відповідно електронної пошти та пароля. Метод `submit` викликається, коли користувач натискає кнопку входу. Він перевіряє, чи форма є валідною, а потім викликає `_mockLogin` для імітації процесу входу в систему. Метод `_mockLogin` імітує виклик API для аутентифікації користувача. У реальному застосуванні його слід замінити на фактичну логіку аутентифікації. У користувацькому інтерфейсі

використовуються віджети `TextFormField` з валідаторами для електронної пошти та пароля. Кнопка входу вимкнена, поки обидва поля не є валідними.

Єдиною відмінністю є те, що нереактивний підход дозволяє використовувати `Stateless` віджет замість `Stateful`. Тобто в дереві віджетів буде відсутній стейт форми логіна тим самим не буде займатись пам'ять. Оновлення сторінки повністю зав'язане на ключ форми (`GlobalKey`).

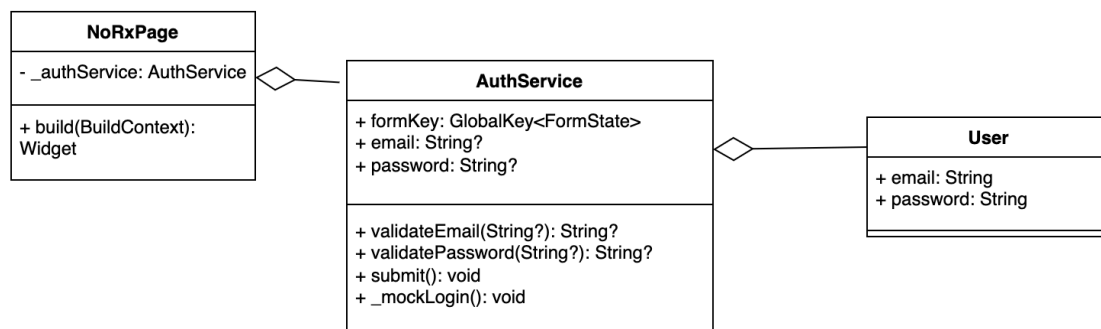


Рисунок 3.7 – Діаграма класів для нереактивного підходу.

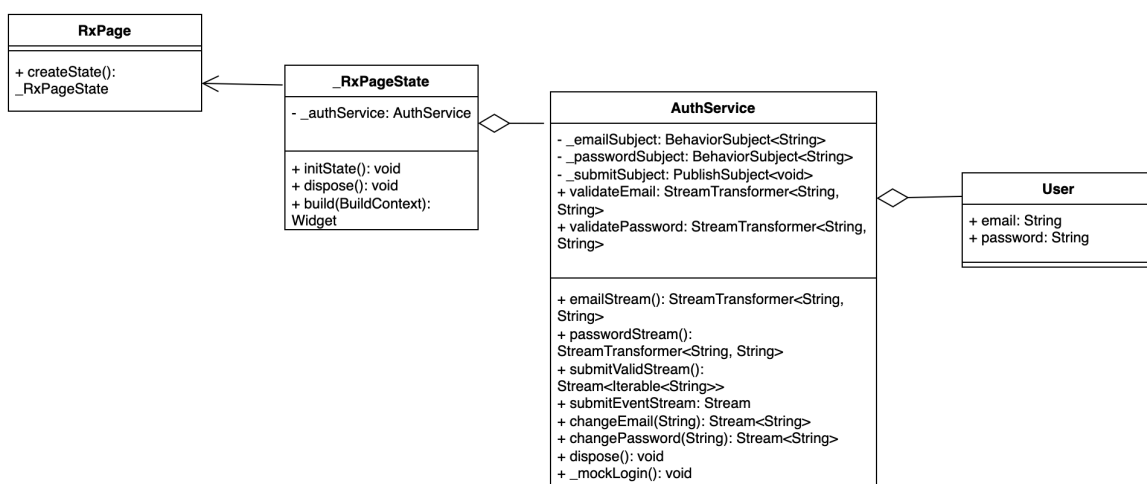


Рисунок 3.8 – Діаграма класів для реактивного підходу.

Нереактивний сервіс виявляється значно об'ємнішим у порівнянні з його реактивним аналогом. Хоча рекомендації часто вказують на використання реактивного підходу при роботі з користувацькими інтерфейсами, у даному конкретному випадку нереактивний підхід виявився більш ефективним.

Цей приклад надає цікавий аспект, оскільки, незважаючи на загальні рекомендації на користь реактивного програмування у взаємодії зі споживачами

(користувачьким інтерфейсом), нереактивний підхід в даному випадку виявився переважаючим. Така ситуація, імовірно, обумовлена конкретними характеристиками та вимогами проекту, які забезпечують більшу ефективність у використанні нереактивних методологій.

Важливо відзначити, що, незважаючи на перевагу нереактивного підходу у даному контексті, ця перевага може втратити актуальність при подальших модифікаціях чи розширеннях проекту. Зміни у вимогах, обсязі даних чи структурі програми можуть вплинути на ефективність реактивного підходу, і розгляд його переваг та недоліків повинен бути проведений у контексті конкретних потреб та еволюції проекту. Такий аналіз може служити важливим керівництвом для вибору оптимального підходу при подальшому розвитку програмного забезпечення.

3.2.5 Аналіз діаграм класів для прикладу Weather app

В даному конкретному випадку можна визначити, що впровадження сервісного шару дозволяє в повній мірі абстрагувати реалізацію та легко імплементувати перехід до нереактивного підходу, не потребуючи значних модифікацій. Цей приклад ілюструє гнучкість та розширюваність архітектури, де сервісний шар виступає як проміжний інтерфейс між реактивним та нереактивним кодом.

Інтеграція сервісного шару визначає можливість взаємозаміни різних підходів без ризику втрати функціональності. Введення цього проміжного рівня дозволяє програмістам здійснювати перехід між реактивним та нереактивним кодом, зберігаючи стабільність та надійність системи.

Цей концептуальний підхід до структури коду відзначається його абстракційною природою, яка дозволяє ефективно розширювати та модернізувати систему відповідно до змінних вимог. Перехід до нереактивного підходу залишається прозорим та безпечним завдяки використанню сервісів як інтерфейсного шару, що ілюструє його важливу роль у забезпеченні гнучкості та адаптивності програмного забезпечення.

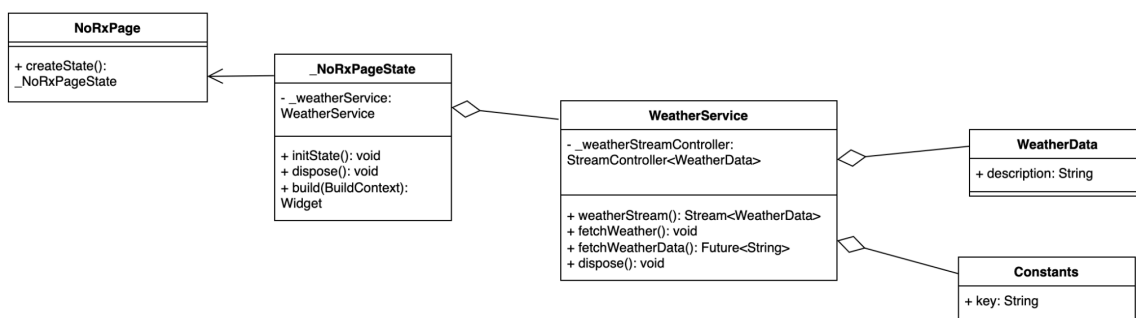


Рисунок 3.9 – Діаграма класів для нереактивного підходу.

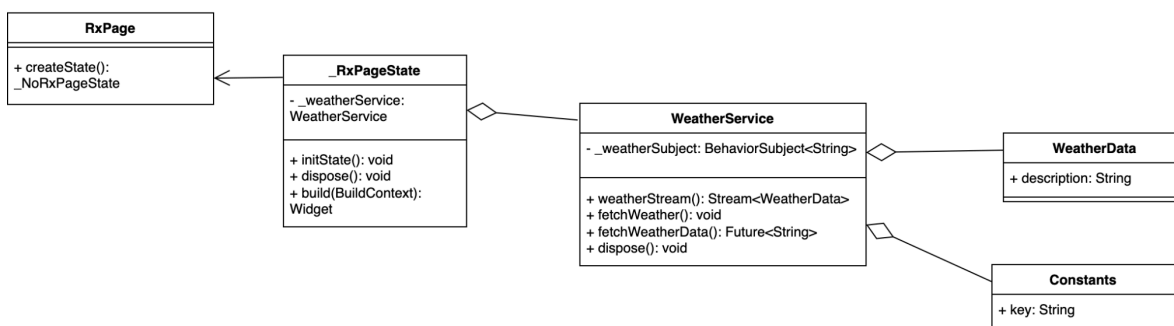


Рисунок 3.10 – Діаграма класів для реактивного підходу.

У підсумку, аналіз класу **WeatherService** відображає важливість введення сервісних шарів в архітектурі програмного забезпечення для забезпечення гнучкості та адаптивності. Реактивний підхід, використаний у даному класі, дозволяє ефективно управляти потоком даних та автоматизувати оновлення інформації. Застосування **BehaviorSubject**, як частини реактивного підходу, створює зручний механізм для розповсюдження даних погоди серед підписників.

Проте, ключовою особливістю є можливість використання сервісу для інкапсуляції логіки та обробки даних, що дозволяє спростити заміну реактивного підходу на нереактивний, не вносячи значних змін у реалізацію. Цей аспект визначає гнучкість та стабільність архітектурного рішення, особливо в умовах змінюваних вимог та еволюції проекту.

Використання сервісного шару в розробці програмного забезпечення підкреслює важливість заведення проміжного рівня абстракції, який служить інтерфейсом для взаємодії зі стороною даних. Це стає ключовим компонентом для реалізації змін у стратегії обробки даних без значного впливу на зовнішній фасад.

системи. Введення сервісу, як у прикладі класу `WeatherService`, стає гарантом того, що в будь-який момент часу можна зручно адаптувати програму до нових вимог чи змінити підхід до роботи з даними.

Такий підхід допомагає побудувати масштабовані та легко адаптовані системи, що реагують на зміни вимог та технологічні тенденції. Також він відкриває шлях для вибору найбільш підходящого підходу (реактивного чи нереактивного) у залежності від специфіки завдань, забезпечуючи розробникам максимальну гнучкість у виборі технологій та стратегій розробки.

Висновки до третього розділу

В даному розділі надано ретельний аналіз діаграм класів для п'яти розроблених програм, які були реалізовані з використанням реактивного підходу (за допомогою бібліотеки `RxDart`) та без нього. Оцінювання включало в себе порівняння структури класів, взаємодії компонентів та ходу виконання програм для визначення переваг та обмежень обох підходів.

Використання `RxDart` дозволяє ефективно моделювати асинхронні операції, спрощуючи структуру програми. Реактивний підхід відзначається більш гнучкою та зручною взаємодією між компонентами програми завдяки використанню потоків подій. Результати аналізу діаграм класів підтверджують, що використання реактивного підходу може сприяти поліпшенню структури програм та полегшити їх розробку.

Проаналізовані діаграми класів дозволили визначити, як кожен підхід впливає на архітектурну структуру програм. Реактивний підхід, зокрема за допомогою `RxDart`, спрощує обробку асинхронних подій та робить код більш читабельним. Взаємодія між об'єктами та компонентами програми стає більш зручною завдяки використанню потоків та спостережувачів, що може позитивно вплинути на продуктивність розробки.

Однак важливо зауважити, що різниця у структурі класів не завжди є значущою, адже вона може бути впливована конкретними особливостями кожного проекту та його вимогами. В контексті даного дослідження виявлено, що використання реактивного підходу разом із абстракціями не завжди суттєво впливає

на структуру класів, зокрема при введенні абстракцій та використанні схожих програм.

Результати аналізу також підтверджують, що величина програми та використання абстракцій може знижувати суттєвість впливу реактивного підходу на архітектурну структуру. Отже, вибір між реактивним та нереактивним підходами може залежати від конкретних вимог та характеристик проекту.

Загалом, розгляд аналізу діаграм класів та реактивного підходу підкреслює важливість обрання ефективного методу розробки відповідно до конкретних потреб та вимог проекту. Враховуючи ретельний аналіз структури класів, можна прийти до обґрунтованого вибору між реактивним та нереактивним підходами, оптимізуючи продуктивність та надійність програмного продукту.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ МОЖЛИВОСТЕЙ РЕАКТИВНОГО ПІДХОДУ

4.1 Вступ

Реактивне програмування, зокрема в контексті використання підходу RX (Reactive Extensions), стає все більш важливим інструментом в розробці програмного забезпечення. Цей підхід визначається своєю здатністю ефективно управляти асинхронністю, подіями та потоками даних. Саме це і робить його особливо корисним у сучасних динамічних системах та їх проектуванні. Дослідження можливостей реактивного підходу, зокрема способами, що представлені в бібліотеці RXDart, визначається потребою розкрити та оцінити його роль у розв'язанні реальних завдань.

У даному розділі представлено конкретні науково обґрунтовані пропозиції та проекти, спрямовані на дослідження базових можливостей реактивного програмування. Були розглянуті та проаналізовані ідеї та наукові розробки, що визначають основні та базові перспективи застосування реактивного підходу у сучасній розробці ПЗ.

В цьому розділі проаналізовано сутність запропонованих ідей та висвітлено їхню наукову вартість. Саме результати наукових досліджень виводяться на передній план, дозволяючи виявити практичну цінність підходу RX. Додатково, порівняльний аналіз між реактивним підходом та нереактивним підходом (без використання бібліотеки RXDart) допомагає встановити конкурентні переваги та обґрунтувати ефективність підходу, що досліджується.

У цьому розділі надано детальний аналіз характеристик розроблених програм (детальний опис яких був викладений в попередньому розділі), інструкції щодо використання та представлені результати проведених досліджень та чисельних експериментів. Розрахунки, графіки та діаграми використовуються для ілюстрації та визначення практичних вигод запропонованого реактивного підходу.

Загальна мета цього розділу — розкрити потенціал реактивного програмування з використанням RXDart та довести, що запропоновані науково обґрунтовані інновації сприяють покращенню ефективності розробки програмного забезпечення та розв'язанню сучасних завдань у програмуванні.

4.2 Порівняльний аналіз та практична цінність методики

В мові програмування Дарт є попередньо реалізований механізм обробки асинхронних подій. Є попередньо створена абстракція Streams.

Таблиця 4.1 – Таблиця порівняння Stream та RX [9].

Ситуація	RX Observable	Dart Stream
Обробка помилок за замовчуванням	Observable закінчує роботу	Помилка повертається, stream все ще активний
Холодний Observable	Багато підписників можуть слухати один і той самий холодний observable	Тільки один підписник
Гарячий Observable	Так	Так (broadcast stream)
Чи Publish, Behavior, Replay Subjects гарячі?	Так	Так
Чи є Single, Maybe, Completable?	Так	Так (але вимагають підключення стороннього розширення)
Чи є підтримка backpressure?	Так	Так
Чи можна повертати null?	Так	Так
Чи синхронні за замовчуванням?	Так	Ні
Чи можна призупинити та відновити підписку?	Ні	Так

Тобто між RX Observable та Dart Streams є різниця, до того ж в RxDart бібліотеці вже міститься безліч способів для обробки потоків.

4.3 Результати та графічні представлення

Візуалізація метрик на графіках може суттєво полегшити сприйняття результатів та допомагати в порівняльному аналізі реактивного та нереактивного підходів. Варто зазначити, що всі дані проведені на одній машині (комп'ютері та

телефонах). Слід зауважити, що при використанні інших девайсів можуть виникнути невеликі відмінності в результатах, однак сутність експерименту залишається недоторканою. Такі відмінності можуть бути пов'язані з конфігурацією обладнання чи іншими параметрами, але загальний висновок та основні відкриття залишаються невідзмінними. Важливо враховувати, що деталі експерименту можуть залежати від конкретних умов, але це не впливає на обґрунтованість та консистентність отриманих результатів.

Час виконання операцій.

Збір метрик часу виконання операцій є важливою складовою для оцінки ефективності програмного коду. Це дозволяє визначити, як швидко програма виконує певні операції та порівняти продуктивність між різними підходами, такими як реактивний та нереактивний код.

Час виконання цих операцій вимірюється в `elapsed ticks`, так як операції виконуються менше ніж за мілісекунду.

Лінійні графіки використовуються для візуалізації часу виконання операцій у коді. Кожна лінія графіку представляє весь процес в цілому. Порівняння лінійних графіків показує, як кожен підхід впливає на швидкодію в конкретних операціях та допомагає визначити, який код є більш ефективним з точки зору часу виконання.

Такий аналіз метрик є корисним для оптимізації програмного коду, покращення його продуктивності та визначення оптимальних стратегій в реалізації операцій.

Rx and NoRx

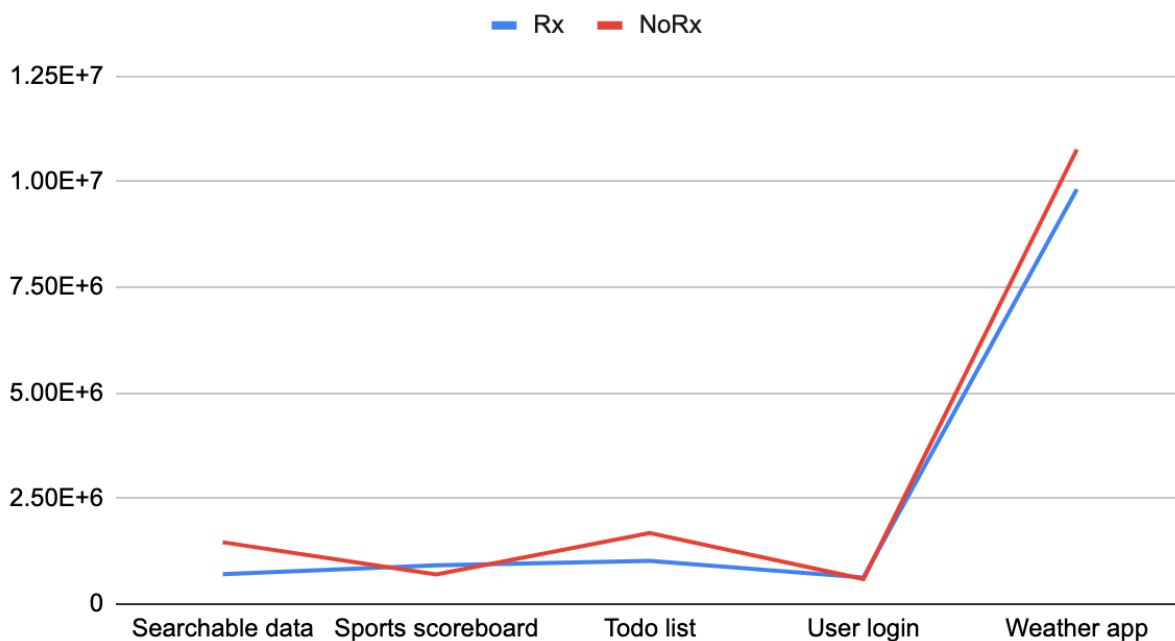


Рисунок 4.1 – Лінійний графік, що порівнює час виконання операцій.

Використання ресурсів.

Для проведення наступного тесту був використаний інструментальний засіб, відомий як “Dart dev tools”. Цей інспектор є вбудованим в інтегроване середовище розробки (IDE), що значно спрощує процес тестування та аналізу коду. Його легкість використання дозволяє швидко та ефективно визначати можливі проблеми та недоліки в програмному коді.

Використання інспектора надає можливість виявлення потенційних проблем, таких як стиль коду, невикористані змінні, чи інші аспекти, які можуть впливати на ефективність та читабельність коду. Інспектор також забезпечує можливість автоматичного виявлення невідповідностей до стандартів програмування та надає рекомендації з їх виправлення [10].

Використання вбудованого інспектора в IDE дозволяє зробити процес аналізу коду більш доступним та зручним, сприяючи покращенню якості та стабільності програмного продукту.

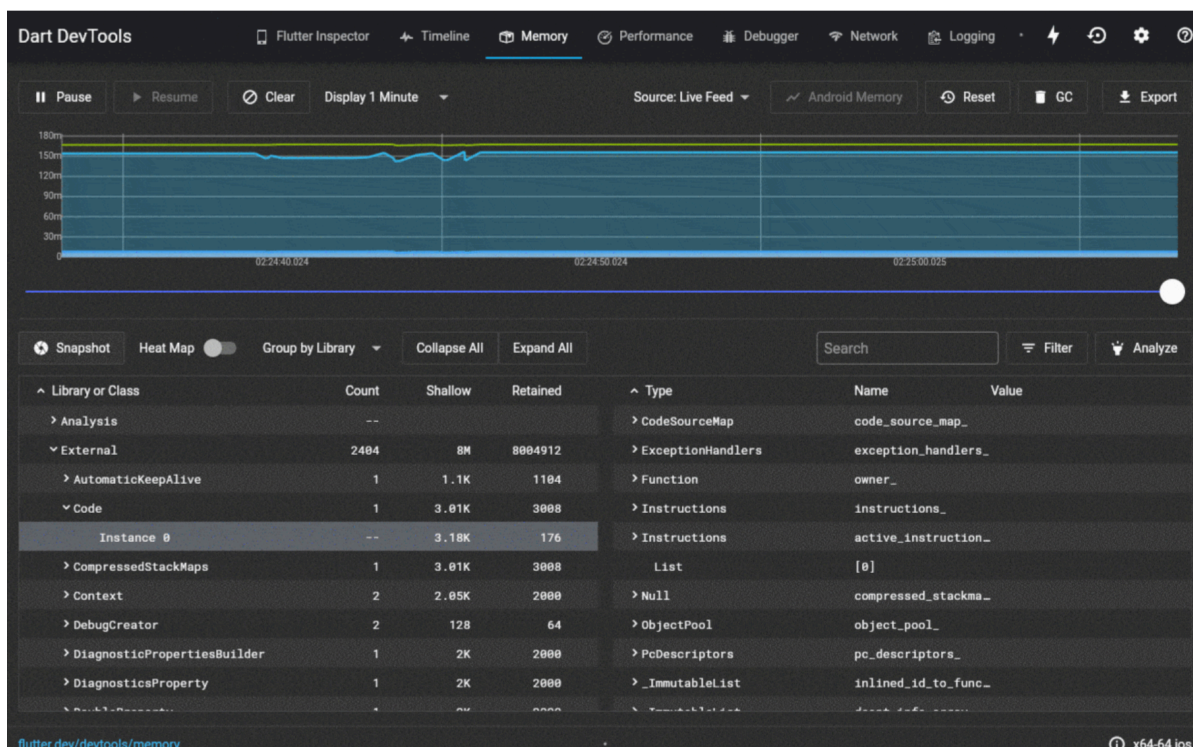


Рисунок 4.2 – Приклад інспектора з метриками [10].

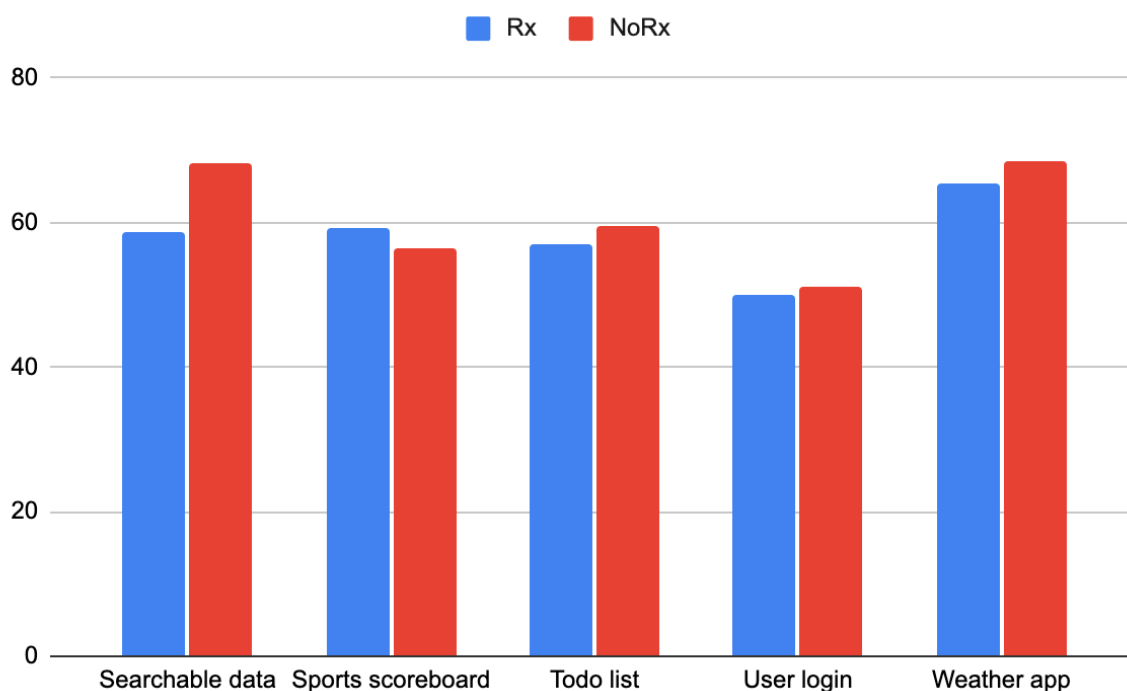


Рисунок 4.3 – Стовпчиковий графік, що порівнює використання ресурсів пам'яті.

Кількість помилок.

Метрика “Кількість помилок” дозволяє оцінити стійкість та надійність програмного продукту, а також вказати на можливі області для поліпшення.

Проте важливо зазначити, що цей розділ може бути неактуальним, оскільки майже всі виявлені помилки вже оброблені. Така ситуація може виникнути в результаті активної роботи з усунення помилок під час розробки, попередження їх виникнення за допомогою вбудованих інструментів, чи зростання досвіду команди.

Враховуючи низьку кількість активних помилок і важкість їх викликання, метрика “Кількість помилок” може служити підтвердженням високої якості коду та ефективності процесу виявлення та виправлення помилок.

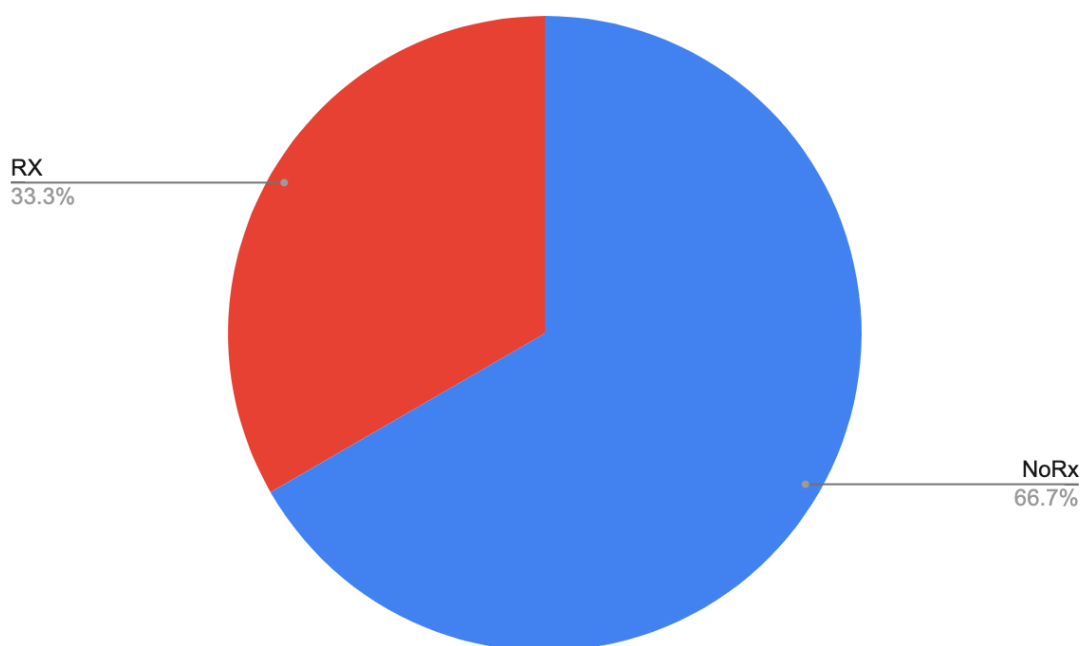


Рисунок 4.4 – Графік, що демонструє кількість помилок.

Кількість ліній коду.

Кількість ліній коду впливає на різні аспекти розробки та обслуговування програмного продукту.

Зазвичай, менша кількість ліній коду полегшує читання та розуміння програми. Це може бути важливою функцією для того, щоб код був доступніший та обслуговувався командою розробників. Менше ліній коду може сприяти швидшій розробці, особливо при використанні ефективних абстракцій та готових рішень. Більша кількість ліній коду може збільшити ймовірність виникнення помилок, але це також залежить від якості та структури коду.

У даному дослідженні виявлено, що різниця у кількості ліній коду між реактивним та нереактивним підходами не є суттєвою завдяки схожості програм та введеним абстракціям.

Таблиця 4.2 – Таблиця порівняння рядків коду.

	RX	No RX
Searchable data	132	102
Sports Scoreboard	176	173
Todo list	151	126
User interface	155	120
Weather app	127	129
Всього	741	650

Висновки до четвертого розділу

В цьому розділі було докладно розглянуто потенціал реактивного програмування, зокрема в контексті використання бібліотеки RXDart, яка є важливим інструментом у сфері розробки програмного забезпечення.

У контексті вимірювання часу виконання операцій було застосовано лінійні графіки для порівняння часу виконання конкретних операцій у реактивному та нереактивному коді. Це дозволяє визначити ефективність обох підходів та виявити можливі різниці у продуктивності. Зазначено, що різниця в часі виконання може бути визначена параметрами конкретної машини, але сутність експерименту залишається недоторканою.

Дослідження також включало аналіз кількості виявлених помилок в програмному коді. Зазначено, що розділ "Кількість помилок" може стати неактуальним, оскільки майже всі виявлені помилки вже оброблені. Така ситуація може свідчити про високу якість та стабільність коду, що обробляє помилки активно під час розробки, попереджуючи їх виникнення.

Наслідком аналізу кількості ліній коду було визначено, що різниця у кількості ліній коду між реактивним та нереактивним підходами не є суттєвою. Вплив

кількості ліній коду на процеси розробки та обслуговування може виявитися в швидкості читання та розуміння коду, а також в ефективності процесів розробки.

На основі цих досліджень та аналізу результатів представлених програм, які використовують реактивний підхід, ми можна сформулюємо наступні висновки.

Використання RXDart у програмній розробці дозволяє ефективно управляти асинхронністю, подіями та потоками даних. Результати наших досліджень демонструють практичну цінність цього підходу в розв'язанні реальних завдань.

Порівняльний аналіз з нереактивним підходом визначив незначні конкурентні переваги реактивного програмування, особливо при використанні RXDart.

Детальний опис та аналіз розроблених програм, їх характеристики, інструкції щодо використання та результати експериментів надали зрозумілу картину практичного застосування реактивного підходу.

Також був розкритий потенціал реактивного програмування, продемонстрований за допомогою RXDart та розглянуто, що досліджуваний підхід має значний вплив на покращення ефективності розробки програмного забезпечення.

Узагальнюючи, дослідження засвідчує важливість використання інструментів для аналізу коду та тестування продуктивності. Також визначено, що при правильному обслуговуванні та вдосконаленні коду різниця у кількості помилок може стати невеликою, а вплив кількості ліній коду на ефективність може бути зменшений за рахунок введених абстракцій та схожості програм. Всі ці аспекти є важливими для створення надійного та ефективного програмного продукту.

ЗАГАЛЬНІ ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

Аналіз реактивного та нереактивного підходів у розробці програм – це важливий етап для визначення оптимального методу під час створення програмного продукту.

В першому розділі була детально розглянута теорія даного підходу. Розгляд теоретичних аспектів в розробці програм виявляється ключовим етапом, оскільки він визначає базові принципи, концепції та методології, які лягають в основу побудови програмних рішень. Розгляд теорії має кілька важливих аспектів. Теоретичний аналіз допомагає розуміти різні парадигми та підходи в програмуванні. Це важливо для вибору найбільш підходящого методу під час розробки програми. Знання теорії дозволяє визначити найкращі практики та дотримуватися стандартів у програмуванні. Це сприяє створенню якісного, ефективного та безпечного програмного продукту.

У цьому дослідженні були розглянуті та проаналізовані приклади, які охоплюють різноманітні сфери від управління елементами до відслідковування результатів спортивних подій та роботи зі списками справ. Висновки, які можна зробити, допоможуть зрозуміти переваги та обмеження обох підходів у різних контекстах.

В результаті виконаної роботи можна сказати, що покращення або втрата продуктивності при використанні Rx Dart буде залежати від того, наскільки добре використовується модель реактивного програмування для роботи з асинхронними операціями, подіями та потоками даних. Якщо додаток включає багато асинхронних і керованих подіями завдань, Rx Dart може забезпечити більш стислий і виразний спосіб управління цими сценаріями.

У ході аналізу та розробки діаграм класів для п'яти прикладів, порівняно реактивний та нереактивний підходи у розробці програм. Зауважено, що реактивний підхід може виглядати трошки більше та складніше на малих проектах, особливо коли мова йде про створення діаграм класів.

На малих прикладах, де важко виправдати використання реактивного підходу, може здатися, що він призводить до більшого обсягу коду та складнішої структури.

Однак, при введенні абстракцій та врахуванні рівня абстракції, нереактивний підхід може безпечно замінити реактивний, забезпечуючи при цьому менший обсяг коду та більшу простоту в розробці.

Важливо враховувати, що великий обсяг коду у реактивних програмах зазвичай пов'язаний із високою рівнем абстракції, яка полегшує управління асинхронними операціями та подіями. У випадках, коли абстракції можуть бути введені в нереактивний код, вигода від використання реактивного підходу може зменшитися.

Таким чином, вибір між реактивним та нереактивним підходами повинен базуватися на конкретних вимогах проекту, його розмірі та особливостях. У деяких випадках, де активне реагування на події та асинхронні операції не є критичними, можна впроваджувати нереактивний підхід, зберігаючи при цьому простоту та зрозумілість коду.

Однак, важливо зазначити, що використання Rx може призвести до певних накладних витрат через додаткові абстракції та механізми, які він привносить. Як і у випадку з будь-яким інструментом або бібліотекою, ефективність коду і конкретний випадок використання відіграватимуть значну роль у визначенні того, чи буде RxDart швидшим, ніж просто Dart для конкретної програми.

Після ретельного аналізу та розгляду п'яти розроблених прикладів програм можна визначити, що реактивний підхід має свої переваги, особливо в плані ефективності та продуктивності. Розроблені приклади включали різні сценарії від управління елементами до відслідковування результатів спортивних подій та роботи зі списками справ, де реактивний підхід виявився дещо більш ефективним.

У випадках, де важлива реакція на події в реальному часі та ефективне управління асинхронними операціями, реактивний підхід виявив свою перевагу. Використання бібліотек, таких як RxDart, дозволило створити більш динамічні та продуктивні програми, забезпечуючи відмінну швидкість та реалізацію асинхронних операцій.

Нереактивний підхід, хоч і є здатним робити схожі завдання, може виявитися менш ефективним у випадках великої асинхронності та потреби в активній реакції

на події. Однак важливо враховувати, що різниця у продуктивності може бути менш відчутною в менших проектах або в тих, де швидкість не є критичною.

Отже, наші приклади демонструють, що реактивний підхід може вигравати в плані продуктивності та ефективності, зокрема в умовах великої асинхронності та потреби у реальному часі реакції на події.

Таким чином, вплив використання Rx Dart на продуктивність у порівнянні з традиційним Dart буде залежати від вашого конкретного випадку використання і від того, наскільки добре використовується модель реактивного програмування для обробки асинхронних завдань у додатку.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. What exactly is Reactive Programming? — Part One: The Theory [Електронний ресурс] –
<https://levelup.gitconnected.com/okay-but-what-exactly-is-reactive-programming-64ca2aabd1aa>
2. Backpressure in Reactive Systems. Nicolas Fränkel [Електронний ресурс] –
<https://blog.frankel.ch/backpressure-reactive-systems/>
3. Design Patterns. Builder. Refactoring Guru [Електронний ресурс] –
<https://refactoring.guru/design-patterns/builder>
4. Operators. Офіційна документація RXDart [Електронний ресурс] –
<https://reactivex.io/documentation/operators/filter.html>
5. Merge. Офіційна документація RXDart [Електронний ресурс] –
<https://reactivex.io/documentation/operators/merge.html>
6. Catch. Офіційна документація RXDart [Електронний ресурс] –
<https://reactivex.io/documentation/operators/catch.html>
7. James Rumbaugh, Ivar Jacobson, Grady Booch (1999). The unified modeling language reference manual (англ.). Addison Wesley Longman Inc. ISBN 0-201-30998-X.
8. ООП. Діаграми класів [Електронний ресурс] –
<https://ua5.org/oop/392-diagrami-klasiv.html>
9. Офіційна документація RXDart [Електронний ресурс] –
<https://github.com/ReactiveX/rxdart#rx-observables-vs-dart-streams>
10. Офіційна документація Flutter by Google [Електронний ресурс] –
<https://docs.flutter.dev/tools/devtools/overview>

ДОДАТКИ

ДОДАТОК А
Технічне завдання

ЗАТВЕРДЖУЮ

Проректор
Українського державного
університету науки і технології
Анатолій РАДКЕВИЧ

ДОСЛІДЖЕННЯ ПІДХОДІВ РЕАКТИВНОГО
ПРОГРАМУВАННЯ У МОВІ DART

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01328-01-ЛЗ

Завідувач кафедри КІТ

_____Вадим ГОРЯЧКІН

Керівник розробки

_____Олександр ІВАНОВ

Виконавець

_____Катерина КАЛІНЧЕНКО

Нормоконтролер

_____Світлана ВОЛКОВА

ЗАТВЕРДЖЕНО

44165850.01328-01

ДОСЛІДЖЕННЯ ПІДХОДІВ РЕАКТИВНОГО
ПРОГРАМУВАННЯ У MOBI DART

Технічне завдання

Листів 19

2024

АНОТАЦІЯ

Документ 44165850.01328-01-ЛЗ «Дослідження підходів реактивного програмування у мові DART» входить до складу програмної документації на програму, що реалізує додатки, що демонструють приклади використання реактивного програмування.

У даному документі представлене технічне завдання.

ЗМІСТ

1. ВВЕДЕННЯ.....	4
2. ПІДСТАВИ ДЛЯ РОЗРОБКИ.....	5
3. ПРИЗНАЧЕННЯ РОЗРОБКИ.....	5
4. ВИМОГИ ДО ПРОГРАМИ АБО ПРОГРАМНОГО ПРОДУКТУ.....	12
4.1 Вимоги до функціональних характеристик.....	12
4.2 Вимоги до надійності.....	8
4.3 Умови експлуатації.....	9
4.4 Вимоги до складу і параметрів технічних засобів.....	10
4.5 Вимоги до інформаційної і програмної сумісності.....	9
4.6 Вимоги до маркування і упаковки.....	9
4.7 Вимоги до транспортування і зберігання.....	10
5. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ.....	10
6. СТАДІЇ ТА ЕТАПИ РОЗРОБКИ.....	11
7. ПОРЯДОК КОНТРОЛЮ ПРИЙМАННЯ.....	12
8. БІБЛІОГРАФІЧНИЙ СПИСОК.....	13

1. ВВЕДЕННЯ

Система реактивного програмування Rx Dart впроваджує інноваційний підхід до розробки, зосереджений на використанні пакету RxDart для підтримки реактивного програмування в Dart. Це дозволяє створювати динамічні, ефективні та легко читабельні програми за допомогою потоків подій.

У конкретному випадку дослідження реактивного програмування за допомогою RxDart включає такі аспекти:

- 1) Використання потоків подій.
- 2) Асинхронна обробка.
- 3) Побудова потоків даних.
- 4) Управління станом.

Застосування RxDart для реактивного програмування дозволяє створювати високопродуктивні та зручні в управлінні системи.

Область застосування: програмний продукт призначений для проведення дослідження в реактивного програмування.

Додаток — програмне забезпечення, призначене для роботи на комп'ютерах.

Програма — це тексти будь-яких програм на мові програмування або в об'єктному коді, придатні для виконання на електронно-обчислювальних машинах.

Програмне забезпечення (ПЗ) — це сукупність програм і пов'язаних з ними даних певного призначення, що придатні для виконання на електроннообчислювальних машинах.

RxDart — Реалізація реактивного програмування для Dart, надаючи інтерфейс для створення та обробки асинхронних подій та потоків даних.

API — інтерфейс програмування застосунків.

BLoC — Бізнес-логіка в компоненті реактивного програмування.

RxUI — Взаємодія з інтерфейсом користувача в реактивному стилі.

RxFlutter — Інтеграція Rx у фреймворк Flutter для реактивної розробки мобільних додатків.

44165850.01328-01-ЛЗ

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ ректора Українського державного університету науки і технології Радкевич А.В. «Про затвердження тем та призначення керівників дипломних проектів» №1196 ст від 05.12. 2022 року.

Тема проекту: «Дослідження підходів реактивного програмування у мові DART».

Керівник дипломного проекту: Олександр ІВАНОВ.

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення дослідження щодо застосування RXDart у розробці реактивних мобільних додатків полягає у дослідженні та оцінці ефективності RXDart у досягненні наступних цілей:

- Покращена швидкість реагування. Підхід реактивного програмування RXDart дозволяє створювати програми, які миттєво реагують на введення користувача та зміни даних, забезпечуючи безперебійну та привабливу взаємодію з користувачем.
- Покращена продуктивність. Асинхронний характер RXDart дозволяє розробникам обробляти складні операції з даними, не блокуючи основний потік інтерфейсу користувача, гарантуючи, що додатки залишаються чуйними навіть під час виконання інтенсивних завдань.
- Знижена складність розробки. Декларативний і потоковий API RXDart спрощує розробку реактивних і чуйних програм, скорочуючи загальний час і зусилля на розробку.

Експлуатаційне призначення дослідження щодо застосування RXDart у розробці реактивних мобільних додатків полягає в досягненні наступних цілей:

- Спроекувати та розробити п'ять мобільних додатків із використанням RXDart для різних функціональних можливостей, включаючи реактивну обробку введених даних користувачем, спортивне табло в прямому ефірі, інтерактивний список завдань, список з можливістю пошуку та програму погоди.
- Оцінити продуктивність, швидкість реагування та взаємодію з користувачем кожної програми на основі RXDart за допомогою тестування користувача та інструментів порівняльного аналізу продуктивності.
- Порівняти продуктивність і використання ресурсів додатків на основі RXDart з еквівалентними додатками, розробленими з використанням

44165850.01328-01-ЛЗ

традиційних підходів, таких як імперативне програмування або програмування, кероване подіями.

4. ВИМОГИ ДО ПРОГРАМИ АБО ПРОГРАМНОГО ПРОДУКТУ

4.1 Вимоги до функціональних характеристик

В ході дослідження має бути створено 5 додатків. В списку описані функціональні характеристики до відповідних застосунків.

TODO List.

Програма (мобільний додаток) повинна забезпечити можливість створення списку справ.

До неї належать такі основні сутності:

- Елемент списку (назва та поточний стан).

Вихідні дані:

- Список справ.

Дані вводяться з клавіатури телефона.

Weather app.

Програма (мобільний додаток) повинна забезпечити можливість перегляду погодних умов.

До неї належать такі основні сутності:

- Погодна умова.

Вихідні дані:

- Дані про погоду.

Дані отримані з сервера.

Searchable data list

Програма (мобільний додаток) повинна забезпечити можливість пошуку по списку.

До неї належать такі основні сутності:

- Елемент списку.

Вихідні дані:

- Список.

Дані попередньо згенеровані.

Live scoreboard.

Програма (мобільний додаток) повинна забезпечити можливість перегляду актуального рахунку спортивних матчів, дані автоматично оновлюються.

До неї належать такі основні сутності:

- Спортивна подія.

Вихідні дані:

- Поточний рахунок спортивних ігор.

Дані попередньо згенеровані.

Reactive user interface.

Програма (мобільний додаток) повинна забезпечити можливість авторизації користувача.

Дані вводяться з клавіатури телефона та передбачають валідацію на користувацьке введення.

4.2 Вимоги до надійності

Програмні системи повинні надійно працювати.

При збоях програмного забезпечення в системі має бути реалізована коректна обробка виняткових ситуацій.

Для контролю коректності вхідної інформації та захисту від помилок:

- Захист від помилок оператора (випадкових символів тощо).

Для вихідної інформації передбачити:

- Відсутність блокування даних.
- Постійне оновлення даних.
- Обов'язковий зворотній зв'язок на дії користувача.

4.3 Умови експлуатації

Програма використовується для проведення дослідження під час виконання дипломного проекту. Збереження архівних копій - на визначених носіях. Збереження резервних копій – на визначеному диску без перезапису. Працювати з програмою може людина, що має навички роботи з персональним комп'ютером, мобільними девайсами та ознайомлена з керівництвом користувача.

4.4 Вимоги до складу і параметрів технічних засобів

Система має розміщуватись на мобільному девайсі із такими мінімальними характеристиками:

- 8 ГБ оперативної пам'яті;
- 1 ГБ пам'яті на диску.

4.5 Вимоги до інформаційної і програмної сумісності

Програмний продукт розробляється для телефонів під керуванням операційної системи Android/IOS.

4.6 Вимоги до маркування і упаковки

Обов'язкове маркування на двох мовах(українська та англійська). Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних). Упаковка не повинна забруднювати навколишнє середовище своїм вмістом, особливо якщо це хімічно активні речовини: фарби, рідкі і сипучі пестициди, продукти нафтопереробки. На упаковці повинно бути вказана назва продукту, номер версії, мінімальні системні вимоги (рис.1). На зворотній стороні упаковки вказується розробник та його юридична адреса (рис.1).

Програмні додатки «Дослідження реактивного програмування» Мінімальні системні вимоги: – 8 ГБ оперативної пам'яті; – 1 ГБ пам'яті на жорсткому диску.	Кафедра «КІТ», УДУНТ м. Дніпро, вул. Лазаряна 2 2024
---	--

4.7 Вимоги до транспортування і зберігання

Системи містяться на носії у форматі .арк. Транспортування виконується у електронному вигляді або ж за допомогою фізичного носія (флешки).

5. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації повинні входити наступні документи:

- Текст програми;
- Керівництво користувача.

Вся програмна документація повинна задовольняти вимогам державного стандарту до оформлення програмний документів.

44165850.01328-01-ЛЗ

6. СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Стадія	Зміст	Строки виконання
Технічне завдання	Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.	04.09.23 – 10.09.23
Ескізний проект	Розробка ескізного проекту. Попередня розробка структури вхідних та вихідних даних. Уточнення методів рішення задачі	10.09.23 – 20.09.23
Робочий проект	Програмування та відладка програми.	20.09.23 - 24.09.23
	Тестування програми	24.09.23 - 30.09.23
	Розробка, узгодження і затвердження програмної документації.	30.09.23 - 01.10.23

7. ПОРЯДОК КОНТРОЛЮ ПРИЙМАННЯ

Контроль за виконанням роботи здійснює керівник розробки Олександр ІВАНОВ.

Прийом продукту здійснює комісія у складі приймальної комісії кафедри КТС.

8. БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с

ДОДАТОК Б
Текст програми

ЗАТВЕРДЖУЮ

Проректор
Українського державного
університету науки і технології
Анатолій РАДКЕВИЧ

ДОСЛІДЖЕННЯ ПІДХОДІВ РЕАКТИВНОГО
ПРОГРАМУВАННЯ У MOBI DART

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01328-01 12 01-ЛЗ

Завідувач кафедри КІТ

_____Вадим ГОРЯЧКІН

Керівник розробки

_____Олександр ІВАНОВ

Виконавець

_____Катерина КАЛІНЧЕНКО

Нормоконтролер

_____Світлана ВОЛКОВА

2024

ЗАТВЕРДЖЕНО

44165850.01328-01 12 01

ДОСЛІДЖЕННЯ ПІДХОДІВ РЕАКТИВНОГО
ПРОГРАМУВАННЯ У MOBI DART

Текст програми

Листів 41

2024

АНОТАЦІЯ

Документ 44165850.01328-01 12 01-ЛЗ «Дослідження підходів реактивного програмування у мові DART» входить до складу програмної документації на програму, що реалізує додатки, що демонструють приклади використання реактивного програмування.

У даному документі представлений опис програм. Програми написані на мові Dart. Програми призначена для систем під керуванням операційної системи IOS/Android.

ЗМІСТ

1. ТЕКСТ ПРОГРАМИ.....	5
------------------------	---

1. ТЕКСТ ПРОГРАММЫ

Текст файла searchable_data/main.dart:

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

```
import 'rx/rx_search_page.dart';
```

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

```
class MyApp extends StatelessWidget {  
  const MyApp({super.key});
```

```
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'Flutter Demo',  
      theme: ThemeData(  
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),  
        useMaterial3: true,  
      ),  
      home: const RxSearchPage(),  
    );  
  }  
}
```

Текст файла searchable_data/rx_search_page.dart.

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

```
import 'package:searchable_data/models/item.dart';
```

```
import 'search_service.dart';
```

```
class RxSearchPage extends StatefulWidget {  
  const RxSearchPage({super.key});
```

```

@override
State<RxSearchPage> createState() => _RxSearchPageState();
}

```

```

class _RxSearchPageState extends State<RxSearchPage> {
  final SearchService _searchBloc = SearchService();

```

```

@override
void dispose() {
  _searchBloc.dispose();
  super.dispose();
}

```

```

@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: const Text('Searchable Data Filtering'),
      ),
      body: Column(
        children: [
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: TextField(
              onChanged: (query) {
                _searchBloc.search(query);
              },
              decoration: const InputDecoration(
                hintText: 'Search...',
              ),
            ),
          ),
          Expanded(
            child: StreamBuilder<List<Item>>(<

```

```

stream: _searchBloc.filteredItemsStream,
builder: (context, snapshot) {
  if (snapshot.hasData) {
    return ListView.builder(
      itemCount: snapshot.data!.length,
      itemBuilder: (context, index) {
        return ListTile(
          title: Text(snapshot.data![index].name),
        );
      },
    );
  } else {
    return const Center(child: CircularProgressIndicator());
  }
},
),
),
],
),
),
);
}
}

```

Текст файлу searchable_data/search_service.dart

```

import 'package:rxdart/rxdart.dart';
import 'package:searchable_data/models/item.dart';

class SearchService {
  final List<Item> _items = [
    Item('Apple'),
    Item('Banana'),
    Item('Cherry'),
    Item('Date'),
    Item('Elderberry'),

```

```

    Item('Fig'),
    Item('Grape'),
    Item('Honeydew'),
    Item('Kiwi'),
    Item('Lemon'),
  ];

```

```

final BehaviorSubject<List<Item>> _filteredItemsSubject =
  BehaviorSubject<List<Item>>.seeded([]);

```

```

          Stream<List<Item>> get filteredItemsStream =>
    _filteredItemsSubject.stream;

```

```

    void search(String query) {
      List<Item> filteredItems = _items
                                                                    .where((item) =>
item.name.toLowerCase().contains(query.toLowerCase()))
      .toList();
      _filteredItemsSubject.sink.add(filteredItems);
    }

```

```

    void dispose() {
      _filteredItemsSubject.close();
    }
  }

```

Текст файла searchable_data/item.dart

```

class Item {
  final String name;

  Item(this.name);
}

```

Текст файла searchable_data/no_rx_search_page.dart

```

import 'package:flutter/material.dart';
import 'package:searchable_data/models/item.dart';

class NoRxSearchPage extends StatefulWidget {
  const NoRxSearchPage( {super.key});

  @override
  State<NoRxSearchPage> createState() => _NoRxSearchPageState();
}

class _NoRxSearchPageState extends State<NoRxSearchPage> {
  final List<Item> _allItems = [
    Item('Apple'),
    Item('Banana'),
    Item('Cherry'),
    Item('Date'),
    Item('Elderberry'),
    Item('Fig'),
    Item('Grape'),
    Item('Honeydew'),
    Item('Kiwi'),
    Item('Lemon'),
  ];

  List<Item> _filteredItems = [];

  void _filterItems(String query) {
    setState(() {
      _filteredItems = _allItems
        .where(
          (item) => item.name.toLowerCase().contains(query.toLowerCase())
        )
        .toList();
    });
  }

  @override

```

```

Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Searchable Data Filtering'),
    ),
    body: Column(
      children: [
        Padding(
          padding: const EdgeInsets.all(8.0),
          child: TextField(
            onChanged: (query) {
              _filterItems(query);
            },
            decoration: const InputDecoration(
              hintText: 'Search...',
            ),
          ),
        ),
        Expanded(
          child: ListView.builder(
            itemCount: _filteredItems.length,
            itemBuilder: (context, index) {
              return ListTile(
                title: Text(_filteredItems[index].name),
              );
            },
          ),
        ),
      ],
    ),
  );
}

```

Текст файла sports_scoreboard/lib/models/sport_events.dart

```

class SportsEvent {

```

```
final String name;
String score;
```

```
SportsEvent(this.name, this.score);
}
```

Текст файлу sports_scoreboard/lib/main.dart

```
import 'package:flutter/material.dart';
import 'package:sports_scoreboard/rx/rx_page.dart';
```

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

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});
```

```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const RxPage(),
    );
  }
}
```

Текст файлу sports_scoreboard/lib/widgets/football_card.dart

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

```
class FootballCard extends StatelessWidget {
  const FootballCard({
```

```

    super.key,
    required this.title,
    required this.subtitle,
  });

```

```

final String title;
final String subtitle;

```

```

@override
Widget build(BuildContext context) {
  return Card(
    elevation: 0,
    color: Theme.of(context).colorScheme.surfaceVariant,
    child: SizedBox(
      width: 300,
      height: 100,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text(
            title,
            style: const TextStyle(
              fontWeight: FontWeight.w400,
              fontSize: 16,
            ),
          ),
          const SizedBox(height: 4),
          Text(
            subtitle,
            style: const TextStyle(
              fontWeight: FontWeight.w600,
              fontSize: 24,
            ),
          ),
        ],
      ),
    ),
  );
}

```

```

    ),
  );
}
}

```

Текст файла sports_scoreboard/lib/rx/rx_page.dart

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

```
import 'package:sports_scoreboard/models/sport_events.dart';
import 'package:sports_scoreboard/rx/sports_scoreboard_service.dart';
import 'package:sports_scoreboard/widgets/football_card.dart';
```

```
class RxPage extends StatefulWidget {
  const RxPage({super.key});
```

```
  @override
  State<RxPage> createState() => _RxPageState();
}
```

```
class _RxPageState extends State<RxPage> {
  final SportsScoreboardService _sportsScoreboardService =
    SportsScoreboardService();
```

```
  @override
  void initState() {
    _sportsScoreboardService.fetchLiveScores();
    super.initState();
  }
```

```
  @override
  void dispose() {
    _sportsScoreboardService.dispose();
    super.dispose();
  }
```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Live Scoreboard'),
      elevation: 0,
    ),
    body: Padding(
      padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 8),
      child: Center(
        child: StreamBuilder<List<SportsEvent>>(
          stream: _sportsScoreboardService.sportsEventsStream,
          builder: (context, snapshot) {
            if (snapshot.hasData) {
              return ListView.builder(
                itemCount: snapshot.data!.length,
                itemBuilder: (context, index) {
                  return FootballCard(
                    title: (snapshot.data![index].name),
                    subtitle: (snapshot.data![index].score),
                  );
                },
              );
            } else {
              return const CircularProgressIndicator();
            }
          },
        ),
      ),
    ),
  );
}

```

Текст файлу sports_scoreboard/lib/rx/sports_scoreboard_service.dart

```
import 'dart:async';
import 'dart:math';
```

```
import 'package:rxdart/rxdart.dart';
import 'package:sports_scoreboard/models/sport_events.dart';
```

```
class SportsScoreboardService {
  final List<SportsEvent> _events = [
    SportsEvent('Real Madrid - Arsenal', '0 - 0'),
    SportsEvent('Arsenal - Shakhtar', '0 - 0'),
    SportsEvent('Shakhtar - Real Madrid', '0 - 0'),
  ];
```

```

                                final          _sportsEventsSubject          =
BehaviorSubject<List<SportsEvent>>.seeded([]);
```

```
Stream<List<SportsEvent>> get sportsEventsStream =>
  _sportsEventsSubject.stream;
```

```
void fetchLiveScores() {
  Timer.periodic(const Duration(seconds: 5), (timer) {
    for (var event in _events) {
      event.score = _generateRandomScore();
    }

    _sportsEventsSubject.add(List.from(_events));
  });
}
```

```
String _generateRandomScore() {
  final Random random = Random();
  return '${random.nextInt(5)} - ${random.nextInt(5)}';
}
```

```
void dispose() {
  _sportsEventsSubject.close();
}
```

```
}
}
```

Текст файла sports_scoreboard/lib/no_rx/no_rx.dart

```
import 'package:flutter/material.dart';
import 'package:sports_scoreboard/models/sport_events.dart';

import 'package:sports_scoreboard/no_rx/sport_scoreboard_service.dart';

class NoRxPage extends StatefulWidget {
  const NoRxPage({super.key});

  @override
  State<NoRxPage> createState() => _NoRxPageState();
}

class _NoRxPageState extends State<NoRxPage> {
  final SportsScoreboardService _sportsScoreboardService =
    SportsScoreboardService();

  @override
  void initState() {
    _sportsScoreboardService.fetchLiveScores(); // Start fetching live scores
    super.initState();
  }

  @override
  void dispose() {
    _sportsScoreboardService.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
```

```

return MaterialApp(
  home: Scaffold(
    appBar: AppBar(
      title: const Text('Live Sports Scoreboard'),
    ),
    body: Center(
      child: StreamBuilder<List<SportsEvent>>(
        stream: _sportsScoreboardService.sportsEventsStream,
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            return ListView.builder(
              itemCount: snapshot.data!.length,
              itemBuilder: (context, index) {
                return ListTile(
                  title: Text(snapshot.data![index].name),
                  subtitle: Text(snapshot.data![index].score),
                );
              },
            );
          } else {
            return const CircularProgressIndicator();
          }
        },
      ),
    ),
  );
}

```

Текст файлу sports_scoreboard/lib/no_rx/sport_scoreboard_service.dart

```

import 'dart:async';
import 'dart:math';

```

```

import 'package:sports_scoreboard/models/sport_events.dart';

```

```

class SportsScoreboardService {
  final List<SportsEvent> _events = [
    SportsEvent('Football', '0 - 0'),
    SportsEvent('Basketball', '0 - 0'),
    SportsEvent('Tennis', '0 - 0'),
  ];

  final StreamController<List<SportsEvent>> _sportsEventsStreamController =
    StreamController<List<SportsEvent>>.broadcast();

  Stream<List<SportsEvent>> get sportsEventsStream =>
    _sportsEventsStreamController.stream;

  void fetchLiveScores() {
    Timer.periodic(const Duration(seconds: 5), (timer) {
      for (var event in _events) {
        event.score = _generateRandomScore();
      }

      _sportsEventsStreamController.sink.add(List.from(_events));
    });
  }

  String _generateRandomScore() {
    final Random random = Random();
    return '${random.nextInt(5)} - ${random.nextInt(5)}';
  }

  void dispose() {
    _sportsEventsStreamController.close();
  }
}

```

Текст файлу todo_list/lib/models/todo.dart

```
class Todo {
  final String task;
  final bool isCompleted;

  Todo(this.task, this.isCompleted);
}
```

Текст файла todo_list/lib/main.dart

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

```
import 'no_rx/no_rx.dart';
import 'rx/rx_page.dart';
```

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

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});
```

```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const NoRxPage(),
    );
  }
}
```

Текст файла todo_list/lib/rx/rx_page.dart

```

import 'package:flutter/material.dart';
import 'package:todo_list/models/todo.dart';
import 'package:todo_list/rx/todo_service.dart';

class RxPage extends StatefulWidget {
  const RxPage({super.key});

  @override
  State<RxPage> createState() => _RxPageState();
}

class _RxPageState extends State<RxPage> {
  final TodoService _todoBloc = TodoService();
  final _todoCtr = TextEditingController();

  @override
  void dispose() {
    _todoBloc.dispose();
    _todoCtr.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Reactive To-Do List App'),
      ),
      body: Column(
        children: [
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: Row(
              children: [
                Expanded(
                  child: TextField(

```



```

        },
      );
    },
  );
} else {
  return const Center(child: CircularProgressIndicator());
}
},
),
),
],
),
);
}
}

```

Текст файла todo_list/lib/rx/todo_service.dart

```
import 'package:rxdart/rxdart.dart';
```

```
import 'package:todo_list/models/todo.dart';
```

```
class TodoService {
```

```
  final _todoListSubject = BehaviorSubject<List<Todo>>();
```

```
  final _todoList = <Todo>[];
```

```
  Stream<List<Todo>> get todoListStream => _todoListSubject.stream;
```

```
  void addTodo(String task) {
```

```
    _todoList.add(Todo(task, false));
```

```
    _todoListSubject.sink.add(_todoList);
```

```
  }
```

```
  void toggleTodoCompletion(int index) {
```

```
    _todoList[index] =
```

```
      Todo(_todoList[index].task, !_todoList[index].isCompleted);
```

```
    _todoListSubject.sink.add(_todoList);
```

```

}

void removeTodo(int index) {
  _todoList.removeAt(index);
  _todoListSubject.sink.add(_todoList);
}

void dispose() {
  _todoListSubject.close();
}
}

```

Текст файла todo_list/lib/no_rx/no_rx.dart

```

import 'package:flutter/material.dart';
import 'package:todo_list/models/todo.dart';

```

```

class NoRxPage extends StatefulWidget {
  const NoRxPage({super.key});

  @override
  State<NoRxPage> createState() => _NoRxPageState();
}

```

```

class _NoRxPageState extends State<NoRxPage> {
  final _todoList = <Todo>[];
  final _todoCtr = TextEditingController();

  void _addTodo(String task) {
    setState(() {
      _todoList.add(Todo(task, false));
    });
  }
}

```

```

void _toggleTodoCompletion(int index) {
  setState(() {

```

```

    _todoList[index] = Todo(
      _todoList[index].task,
      !_todoList[index].isCompleted,
    );
  });
}

```

```

void _removeTodo(int index) {
  setState(() {
    _todoList.removeAt(index);
  });
}

```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Non Reactive To-Do List App'),
    ),
    body: Column(
      children: [
        Padding(
          padding: const EdgeInsets.all(8.0),
          child: Row(
            children: [
              Expanded(
                child: TextField(
                  controller: _todoCtr,
                  onSubmitted: (String value) {
                    _addTodo(value);
                  },
                  decoration: const InputDecoration(
                    hintText: 'Add a new task...',
                  ),
                ),
              ),
            ],
          ),
        ),
      ],
    ),
  );
}

```

```

IconButton(
  icon: const Icon(Icons.add),
  onPressed: () {
    _addTodo(_todoCtr.text);
  },
),
],
),
),
Expanded(
  child: ListView.builder(
    itemCount: _todoList.length,
    itemBuilder: (context, index) {
      return ListTile(
        title: Text(_todoList[index].task),
        trailing: Checkbox(
          value: _todoList[index].isCompleted,
          onChanged: (bool? value) {
            _toggleTodoCompletion(index);
          },
        ),
        onLongPress: () {
          _removeTodo(index);
        },
      );
    },
  ),
),
],
),
);
}
}

```

```

import 'package:flutter/material.dart';
import 'package:user_interface/rx/rx_page.dart';

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const RxPage(),
    );
  }
}

```

Текст файла user_interface/lib/rx/auth_service.dart

```

import 'dart:async';

import 'package:rxdart/rxdart.dart';

class AuthService {
  final _emailSubject = BehaviorSubject<String>();
  final _passwordSubject = BehaviorSubject<String>();
  final _submitSubject = PublishSubject<void>();

  Stream<String> get emailStream =>
    _emailSubject.stream.transform(validateEmail);

```

```

Stream<String> get passwordStream =>
    _passwordSubject.stream.transform(validatePassword);
Stream<bool> get submitValidStream => CombineLatestStream.combine2(
    _emailSubject.stream,
    _passwordSubject.stream,
    (email, password) => true,
);
Stream<void> get submitEventStream => _submitSubject.stream;

Function(String) get changeEmail => _emailSubject.sink.add;
Function(String) get changePassword => _passwordSubject.sink.add;
void Function() get submit => () => _submitSubject.sink.add(null);

final validateEmail = StreamTransformer<String, String>.fromHandlers(
  handleData: (email, sink) {
    if (RegExp(r"^[a-zA-Z0-9+_.-]+@[a-zA-Z0-9.-]+$").hasMatch(email)) {
      sink.add(email);
    } else {
      sink.addError('Enter a valid email address');
    }
  },
);

final validatePassword = StreamTransformer<String, String>.fromHandlers(
  handleData: (password, sink) {
    if (password.length >= 6) {
      sink.add(password);
    } else {
      sink.addError('Password must be at least 6 characters long');
    }
  },
);

void dispose() {
  _emailSubject.close();
  _passwordSubject.close();
}

```

```

    _submitSubject.close();
  }

```

```

void mockLogin() {}
}

```

Текст файлу user_interface/lib/rx/rx_page.dart

```

import 'package:flutter/material.dart';
import 'package:user_interface/rx/auth_service.dart';

```

```

class RxPage extends StatefulWidget {
  const RxPage({super.key});

```

```

  @override
  State<RxPage> createState() => _RxPageState();
}

```

```

class _RxPageState extends State<RxPage> {
  final AuthService _authBloc = AuthService();

```

```

  @override
  void dispose() {
    _authBloc.dispose();
    super.dispose();
  }

```

```

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('User Authentication'),
      ),
      body: Padding(
        padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 80),
        child: Column(

```

```

crossAxisAlignment: CrossAxisAlignment.stretch,
children: <Widget>[
  StreamBuilder<String>(
    stream: _authBloc.emailStream,
    builder: (context, snapshot) {
      return TextField(
        onChanged: _authBloc.changeEmail,
        keyboardType: TextInputType.emailAddress,
        decoration: InputDecoration(
          labelText: 'Email',
          errorText:
            snapshot.hasError ? snapshot.error.toString() : null,
        ),
      );
    },
  ),
  const SizedBox(height: 20),
  StreamBuilder<String>(
    stream: _authBloc.passwordStream,
    builder: (context, snapshot) {
      return TextField(
        onChanged: _authBloc.changePassword,
        obscureText: true,
        decoration: InputDecoration(
          labelText: 'Password',
          errorText:
            snapshot.hasError ? snapshot.error.toString() : null,
        ),
      );
    },
  ),
  const SizedBox(height: 40),
  StreamBuilder<bool>(
    stream: _authBloc.submitValidStream,
    builder: (context, snapshot) {
      return ElevatedButton(

```

```

        onPressed: snapshot.hasData ? _authBloc.submit : null,
        child: const Text('Login'),
      );
    },
  ),
],
),
),
);
}
}

```

Текст файла user_interface/lib/no_rx/auth_service.dart

```

import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:user_interface/models/user.dart';

```

```

class AuthService {
  final GlobalKey<FormState> formKey = GlobalKey<FormState>();
  String? email;
  String? password;

  String? validateEmail(String? value) {
    if (value == null || value.isEmpty) {
      return 'Please enter your email';
    }
    return null;
  }

  String? validatePassword(String? value) {
    if (value == null || value.isEmpty) {
      return 'Please enter your password';
    }
    return null;
  }
}

```

```

void submit() {
  if (formKey.currentState!.validate()) {
    User user = User(email!, password!);
    _mockLogin(user);
  }
}

```

```

void _mockLogin(User user) {
  // Mock API call for login
  if (kDebugMode) {
    print('Logging in with email: ${user.email}, password: ${user.password}');
  }
}

```

Текст файла user_interface/lib/no_rx/no_rx_page.dart

```

import 'package:flutter/material.dart';
import 'package:user_interface/no_rx/auth_service.dart';

```

```

class NoRxPage extends StatelessWidget {
  NoRxPage({super.key});

```

```

  final AuthService _authBloc = AuthService();

```

```

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('User Authentication with Reactive UI'),
        ),
        body: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Form(

```

```

key: _authBloc.formKey,
child: Column(
  children: <Widget>[
    TextFormField(
      onChanged: (value) {
        _authBloc.email = value;
      },
      keyboardType: TextInputType.emailAddress,
      decoration: const InputDecoration(
        labelText: 'Email',
      ),
      validator: _authBloc.validateEmail,
    ),
    const SizedBox(height: 20),
    TextFormField(
      onChanged: (value) {
        _authBloc.password = value;
      },
      obscureText: true,
      decoration: const InputDecoration(
        labelText: 'Password',
      ),
      validator: _authBloc.validatePassword,
    ),
    const SizedBox(height: 40),
    ElevatedButton(
      onPressed: _authBloc.submit,
      child: const Text('Login'),
    ),
  ],
),
),
),
),
);
}

```

```
}
```

Текст файла user_interface/lib/models/user.dart

```
class User {
  final String email;
  final String password;

  User(this.email, this.password);
}
```

Текст файла weather_app/lib/core/constants.dart

```
class Constants {
  static const key = 'key'; // generated key.
}
```

Текст файла weather_app/lib/main.dart

```
import 'package:flutter/material.dart';
import 'package:weather_app/no_rx/no_rx.dart';
import 'package:weather_app/rx/rx.dart';
```

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

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});
```

```
@override
```

```
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
```

```

        useMaterial3: true,
      ),
      home: const NoRxPage(),
    );
  }
}

```

Текст файла weather_app/lib/rx/rx.dart

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

```
import 'package:weather_app/rx/weather_service.dart';
```

```
class RxPage extends StatefulWidget {
  const RxPage({super.key});

```

```

  @override
  State<RxPage> createState() => _RxPageState();
}

```

```

class _RxPageState extends State<RxPage> {
  final WeatherService _weatherService = WeatherService();

```

```

  @override
  void initState() {
    _weatherService.fetchWeather();
    super.initState();
  }

```

```

  @override
  void dispose() {
    _weatherService.dispose();
    super.dispose();
  }

```

```

  @override

```

```

Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: const Text('Real-Time Weather App'),
      ),
      body: Center(
        child: StreamBuilder<String>(
          stream: _weatherService.weatherStream,
          builder: (context, snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting) {
              return const CircularProgressIndicator();
            } else if (snapshot.hasError) {
              return Text('Error: ${snapshot.error}');
            } else {
              return Text('Weather: ${snapshot.data}');
            }
          },
        ),
      ),
    ),
  );
}

```

Текст файла weather_app/lib/rx/weather_service.dart

```

import 'package:rxdart/rxdart.dart';
import 'dart:async';
import 'dart:convert';
import 'package:http/http.dart' as http;

```

```

import '../core/constants.dart';

```

```

class WeatherService {
  final _weatherSubject = BehaviorSubject<String>();

```

```

Stream<String> get weatherStream => _weatherSubject.stream;

void fetchWeather() async {
  while (true) {
    await Future.delayed(
      const Duration(seconds: 5)); // Simulating periodic updates
    String weatherData =
      await fetchWeatherData(); // Fetch weather data from API
    _weatherSubject.sink.add(weatherData); // Send weather data to the stream
  }
}

Future<String> fetchWeatherData() async {
  final response = await http.get(
    Uri.parse(
      'https://api.openweathermap.org/data/2.5/weather?q=London&appid=${Constants.key}',
    ),
  );
  if (response.statusCode == 200) {
    return json.decode(response.body)['weather'][0]['description'];
  } else {
    throw Exception('Failed to load weather data');
  }
}

void dispose() {
  _weatherSubject.close();
}
}

Текст файлу weather_app/lib/no_rx/no_rx.dart
import 'dart:async';

```

```

import 'dart:convert';
import 'package:flutter/foundation.dart';
import 'package:http/http.dart' as http;
import 'package:weather_app/core/constants.dart';
import 'package:weather_app/models/weather_data.dart';

class WeatherService {
  final _weatherStreamController = StreamController<WeatherData>();

  Stream<WeatherData> get weatherStream =>
    _weatherStreamController.stream;

  void fetchWeather() async {
    while (true) {
      await Future.delayed(const Duration(seconds: 5));
      try {
        String weatherDescription = await fetchWeatherData();
        WeatherData weatherData = WeatherData(weatherDescription);
        _weatherStreamController.add(weatherData);
      } catch (e) {
        if (kDebugMode) {
          print('Error fetching weather data: $e');
        }
      }
    }
  }

  Future<String> fetchWeatherData() async {
    final response = await http.get(Uri.parse(
      'https://api.openweathermap.org/data/2.5/weather?q=London&appid=${Constants.key}'));
    if (response.statusCode == 200) {
      return json.decode(response.body)['weather'][0]['description'];
    } else {
      throw Exception('Failed to load weather data');
    }
  }
}

```

```

    }
  }

  void dispose() {
    _weatherStreamController.close();
  }
}

```

Текст файлу weather_app/lib/no_rx/no_rx.dart

```

import 'package:flutter/material.dart';
import 'package:weather_app/models/weather_data.dart';
import 'package:weather_app/no_rx/weather_service.dart';

```

```

class NoRxPage extends StatefulWidget {
  const NoRxPage({super.key});

  @override
  State<NoRxPage> createState() => _NoRxPageState();
}

class _NoRxPageState extends State<NoRxPage> {
  final WeatherService _weatherService = WeatherService();

  @override
  void initState() {
    super.initState();
    _weatherService.fetchWeather();
  }

  @override
  void dispose() {
    _weatherService.dispose();
    super.dispose();
  }
}

```

```

@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: const Text('Real-Time Weather App'),
      ),
      body: Center(
        child: StreamBuilder<WeatherData>(
          stream: _weatherService.weatherStream,
          builder: (context, snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting) {
              return const CircularProgressIndicator();
            } else if (snapshot.hasError) {
              return Text('Error: ${snapshot.error}');
            } else {
              return Text('Weather: ${snapshot.data?.description}');
            }
          },
        ),
      ),
    ),
  );
}

```

Текст файла weather_app/lib/no_rx/weather_service.dart

```

import 'dart:async';

```

```

import 'dart:convert';
import 'package:flutter/foundation.dart';
import 'package:http/http.dart' as http;
import 'package:weather_app/core/constants.dart';
import 'package:weather_app/models/weather_data.dart';

```

```

class WeatherService {
    final _weatherStreamController = StreamController<WeatherData>();

    Stream<WeatherData> get weatherStream =>
        _weatherStreamController.stream;

    void fetchWeather() async {
        while (true) {
            await Future.delayed(const Duration(seconds: 5));
            try {
                String weatherDescription = await fetchWeatherData();
                WeatherData weatherData = WeatherData(weatherDescription);
                _weatherStreamController.add(weatherData);
            } catch (e) {
                if (kDebugMode) {
                    print('Error fetching weather data: $e');
                }
            }
        }
    }

    Future<String> fetchWeatherData() async {
        final response = await http.get(Uri.parse(
            'https://api.openweathermap.org/data/2.5/weather?q=London&appid=${Constants.key}
        ));
        if (response.statusCode == 200) {
            return json.decode(response.body)['weather'][0]['description'];
        } else {
            throw Exception('Failed to load weather data');
        }
    }

    void dispose() {
        _weatherStreamController.close();
    }
}

```

```
}  
}
```

Текст файла weather_app/lib/models/weather_data.dart

```
class WeatherData {  
  final String description;  
  
  WeatherData(this.description);  
}
```

ДОДАТОК В

Керівництво користувача

ЗАТВЕРДЖУЮ

Проректор
Українського державного
університету науки і технології
Анатолій РАДКЕВИЧ

ДОСЛІДЖЕННЯ ПІДХОДІВ РЕАКТИВНОГО
ПРОГРАМУВАННЯ У МОВІ DART

Керівництво користувача
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01328-01 ІЗ 01-ЛЗ

Завідувач кафедри КІТ

_____Вадим ГОРЯЧКІН

Керівник розробки

_____Олександр ІВАНОВ

Виконавець

_____Катерина КАЛІНЧЕНКО

Нормоконтролер

_____Світлана ВОЛКОВА

2024

ЗАТВЕРДЖЕНО

44165850.01328-01 ІЗ 01

ДОСЛІДЖЕННЯ ПІДХОДІВ РЕАКТИВНОГО
ПРОГРАМУВАННЯ У MOBI DART

Керівництво користувача

Листів 11

2024

АНОТАЦІЯ

Документ 44165850.01328-01 ІЗ 01-ЛЗ «Дослідження підходів реактивного програмування у мові DART» входить до складу програмної документації на програму, що реалізує додатки, що демонструють приклади використання реактивного програмування.

У даному документі представлений опис програм. Програми написані на мові Dart. Програми призначена для систем під керуванням операційної системи IOS/Android.

ЗМІСТ

1. ВВЕДЕННЯ.....	5
2. ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ.....	6
3. ПІДГОТОВКА ДО РОБОТИ.....	8
4. ОПИС ОПЕРАЦІЙ.....	9
5. АВАРІЙНІ СИТУАЦІЇ.....	10
6. РЕКОМЕНДАЦІЇ ЩОДО ЗАСТОСУВАННЯ.....	11
7. БІБЛІОГРАФІЧНИЙ СПИСОК.....	12

1. ВВЕДЕННЯ

Система реактивного програмування Rx Dart впроваджує інноваційний підхід до розробки, зосереджений на використанні пакету RxDart для підтримки реактивного програмування в Dart. Це дозволяє створювати динамічні, ефективні та легко читабельні програми за допомогою потоків подій.

У конкретному випадку дослідження реактивного програмування за допомогою RxDart включає такі аспекти:

- 1) Використання потоків подій.
- 2) Асинхронна обробка.
- 3) Побудова потоків даних.
- 4) Управління станом.

Застосування RxDart для реактивного програмування дозволяє створювати високопродуктивні та зручні в управлінні системи.

Область застосування: програмний продукт призначений для проведення дослідження в реактивного програмування.

Додаток — програмне забезпечення, призначене для роботи на комп'ютерах.

Програма – це тексти будь-яких програм на мові програмування або в об'єктному коді, придатні для виконання на електронно-обчислювальних машинах.

Програмне забезпечення (ПЗ) – це сукупність програм і пов'язаних з ними даних певного призначення, що придатні для виконання на електроннообчислювальних машинах.

RxDart – Реалізація реактивного програмування для Dart, надаючи інтерфейс для створення та обробки асинхронних подій та потоків даних.

API – інтерфейс програмування застосунків.

BLoC – Бізнес-логіка в компоненті реактивного програмування.

RxUI – Взаємодія з інтерфейсом користувача в реактивному стилі.

RxFlutter – Інтеграція Rx у фреймворк Flutter для реактивної розробки мобільних додатків.

2. ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

Функціональне призначення дослідження щодо застосування RXDart у розробці реактивних мобільних додатків полягає у дослідженні та оцінці ефективності RXDart у досягненні наступних цілей:

- Покращена швидкість реагування. Підхід реактивного програмування RXDart дозволяє створювати програми, які миттєво реагують на введення користувача та зміни даних, забезпечуючи безперебійну та привабливу взаємодію з користувачем.
- Покращена продуктивність. Асинхронний характер RXDart дозволяє розробникам обробляти складні операції з даними, не блокуючи основний потік інтерфейсу користувача, гарантуючи, що додатки залишаються чуйними навіть під час виконання інтенсивних завдань.
- Знижена складність розробки. Декларативний і потоковий API RXDart спрощує розробку реактивних і чуйних програм, скорочуючи загальний час і зусилля на розробку.

Експлуатаційне призначення дослідження щодо застосування RXDart у розробці реактивних мобільних додатків полягає в досягненні наступних цілей:

- Спроекувати та розробити п'ять мобільних додатків із використанням RXDart для різних функціональних можливостей, включаючи реактивну обробку введених даних користувачем, спортивне табло в прямому ефірі, інтерактивний список завдань, список з можливістю пошуку та програму погоди.
- Оцінити продуктивність, швидкість реагування та взаємодію з користувачем кожної програми на основі RXDart за допомогою тестування користувача та інструментів порівняльного аналізу продуктивності.

- Порівняти продуктивність і використання ресурсів додатків на основі RXDart з еквівалентними додатками, розробленими з використанням традиційних підходів, таких як імперативне програмування або програмування, кероване подіями.

Збереження архівних копій - на визначених носіях. Збереження резервних копій – на визначеному диску без перезапису. Знищення копій за регламентом;

Система має розміщуватись на мобільному девайсі з такими мінімальними характеристиками:

- 8 ГБ оперативної пам'яті;
- 1 ГБ пам'яті на жорсткому диску.

Умови експлуатації зображені на рис. 1:

Пора року	Категорія робіт згідно з ГОСТ 12.1-005-88	Температура повітря, град.С	Відносна вологість повітря, %	Швидкість руху повітря, м/с
		оптимальна	оптимальна	оптимальна
Холодна	легка-1 а	22 - 24	40 - 60	0,1
	легка-1 б	21 - 23	40 - 60	0,1
Тепла	легка-1 а	23 - 25	40 - 60	0,1
	легка-1 б	22 - 24	40 - 60	0,2

Рисунок 1 – Умови експлуатації

Програма призначена для систем під керуванням операційної системи Android/IOS.

3. ПІДГОТОВКА ДО РОБОТИ

Для початку роботи програмних засобів їх необхідно запустити за допомогою відповідних файлів. Попередньо необхідно завантажити файли на мобільний пристрій.

4. ОПИС ОПЕРАЦІЙ

Після запуску кожного програмного засобу на екрані з'являється головний екран (кожен із додатків складається тільки з одного екрану).

Функція 1:

Найменування – «Обробник вибору компонента кнопки».

Умови – Користувачу необхідно натиснути на компонент кнопки.

Підготовчі дії – відображення компоненту кнопки.

Основні дії в послідовності, що вимагається – користувач натискає на компонент кнопки. В залежності від вибору кнопки користувачем, буде викликана відповідна функція.

Заключні дії – викликання необхідної функції, яка описана в обробнику подій.

Необхідні ресурси для виконання операції – відкриття додатку.

Функція 2:

Найменування – «Взаємодія з текстовим полем».

Умови – Користувачу необхідно ввести дані у текстове поле.

Підготовчі дії – відображення текстового поля.

Основні дії в послідовності, що вимагається – користувач вводить дані до текстового поля.

Заключні дії – викликання необхідної функції, яка зчитає дані з текстового поля.

Для завершення роботи програми необхідно натиснути на кнопку пункту меню «Exit».

5. АВАРІЙНІ СИТУАЦІЇ

Якщо програмний засіб під час роботи буде поводити некоректно чи із помилкою, від користувача потребується перезавантажити програмний засіб для відновлення коректної роботи програмного засобу.

Якщо програмний засіб буде запускатися з пристрою який не відповідає необхідним технічним характеристикам буде виведено необхідне повідомлення та викликане завершення роботи програмного засобу.

6. РЕКОМЕНДАЦІЇ ЩОДО ЗАСТОСУВАННЯ

Після встановлення програмного засобу, для його запуску необхідно натиснути на файл.

Після запуску програмного засобу, відобразиться головний екран, для ознайомлення чи вибору моделі для подальшого застосування, він містить компоненти, які відповідають за певні дії, що описані на екрані.

7. БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с