НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення комп’ютеризованих систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Управління потоками даних у бібліотеці Project-Reactor*

Виконав: студент IV курсу, групи *ІП-71 Михайлов Дмитро Ігорович*
(прізвище, ім’я, по батькові) (підпис)

Керівник *ст. вик. Халус Олена Андріївна*
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.*
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Рецензент:

к.т.н., доц. каф. АУТС Писаренко А.В.
посада,науковий ступінь,вчене звання,прізвище,ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення комп’ютеризованих систем*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2021 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Михайлову Дмитру Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «Управління потоками даних у бібліотеці Project-Reactor»
керівник проєкту Халус Олена Андріївна, ст. вик.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “11” травня 2021 р. №1139-с

2. Термін подання студентом проєкту «08» *червня 2021 року*

3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення
3) Розгортання та впровадження програмного забезпечення
4) Методика випробувань програмного продукту

5. Перелік графічного матеріалу

1) Схема структурна бізнес-процесів

2) Схема структурна послідовності

3) Схема структурна класів програмного забезпечення для Project-Reactor

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «14» *березня 2021 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*21.03.2021*	
2.	*Аналіз існуючих методів розв'язання задачі*	*21.03.2021*	
3.	*Постановка та формалізація задачі*	*26.03.2021*	
4.	*Аналіз вимог до програмного забезпечення*	*26.03.2021*	
5.	*Алгоритмізація задачі*	*02.04.2021*	
6.	*Моделювання програмного забезпечення*	*16.04.2021*	
7.	*Обґрунтування використовуваних технічних засобів*	*16.04.2021*	
8.	*Розробка архітектури програмного забезпечення*	*23.04.2021*	
9.	*Розробка програмного забезпечення*	*30.04.2021*	
10.	*Налагодження програми*	*07.05.2021*	
11.	*Виконання графічних документів*	*15.05.2021*	
12.	*Оформлення пояснювальної записки*	*15.05.2021*	
13.	*Подання ДП на попередній захист*	*17.05.2021*	
14.	*Подання ДП рецензенту*	*01.06.2021*	
15.	*Подання ДП на основний захист*	*08.06.2021*	

Студент _____ Дмитро МИХАЙЛОВ
(підпис)

Керівник _____ Олена ХАЛУС
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	A4		Завдання на дипломний проєкт	2	
2	A4	КПІ.ІП-7120.045490.02.81	Пояснювальна записка	79	
3	A4	КПІ.ІП-7120.045490.03.91	Технічне завдання	12	
4	A4	КПІ.ІП-7120.045490.04.51	Програма та методика тестування	6	
5	A4	КПІ.ІП-7120.045490.05.13	Опис програми	75	
6	A3	КПІ.ІП-7120.045490.06.99	Схема структурна бізнес-процесів	1	
7	A3	КПІ.ІП-7120.045490.06.99	Схема структурна послідовності	1	
8	A3	КПІ.ІП-7120.045490.06.99	Схема структурна класів програмного забезпечення для Project-Reactor	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-7120.045490.01.90		
Розробн.		Михайлов Д.І.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Халус О.А.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-71	
Н/контр.		Ліщук К.І.					
В.о.зав.ка.		Павлов О.А.					

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломної роботи викладена на 79 сторінках, містить 15 рисунків, 38 таблиць та 12 джерел.

Даний дипломний проект присвячено розширенню способів управління потоками даних у бібліотеці Project-Reactor шляхом створення додаткового оператора Prefetch разом зі створенням додатку для демонстрації роботи програмного забезпечення.

Метою розробки роботи є створення нового способу управління потоками даних бібліотеки Project-Reactor, який дозволить краще налаштувати обробку потоків даних; та перевірка розробленого способу на прикладі демонстраційного проекту дистанційного замовлення.

У першому розділі проведений аналіз предметної області та існуючих технічний рішень. Розроблені та описані функціональні та нефункціональні вимоги до програмного забезпечення та демонстраційного проекту.

У розділі моделювання та конструювання програмного забезпечення були описані розроблювані бізнес процеси, на основі яких була розроблена та описана архітектура демонстраційного проекту. Прописані основні технології, деталі реалізації та сутності кінцевого продукту.

У третьому розділі прописані компоненти та функціональність, які потребували тестування, типи та методи проведених тестів, і результати тестування розробленого способу управління потоками даних.

У розділі впровадження та супровід програмного забезпечення описано процес збірки та розгортання програмного забезпечення та демонстраційного проекту.

КЛЮЧОВІ СЛОВА: РЕАКТИВНЕ ПРОГРАМУВАННЯ, PROJECT-REACTOR, РЕАКТИВНІ ПОТОКИ, РЕАКТИВНА СИСТЕМА, МІКРОСЕРВІСИ.

ABSTRACT

Structure and scope of thesis. The explanatory note of the thesis project is set out on 79 pages and contains 15 figures, 38 tables and 12 sources.

This diploma project is dedicated to the expanding of the data flow management in the Project-Reactor library by creating and additional Prefetch operator with the creation of an application to demonstrate the work of the software.

The purpose of the project is to create a new way of managing data flows of the Project-Reactor library, which will better configure the processing of data flows, and verification of the created way using a demonstration project of remote ordering.

The first section analyzes the subject area and existing technical solutions. Functional and non-functional requirements for software and demonstration project are created and described.

The software modeling and design section describes developed business processes, and, based on them, the demonstration project architecture is developed and described. The basic technologies, realization details and final product essence are described.

The third section describes components and functionality that require testing, types and methods of preformed tests and test results of the developed data flow management method.

The software implementation and maintenance section describe the process of software and demonstration project assembling and deploying.

KEY WORDS: REACTIVE PROGRAMMING, PROJECT-REACTOR, REACTIVE FLOWS, REACTIVE SYSTEM, MICROSERVICES.

Пояснювальна записка
до дипломного проєкту

на тему: «*Управління потоками даних у бібліотеці Project-Reactor*»

Київ – 2021 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Реактивні системи – це архітектурний стиль, який дозволяє будувати розподілені, інтерактивні системи з можливостями відновлення після збоїв.

Реактивне програмування – одна з парадигм програмування, яка орієнтована на обробці потоків даних.

Upstream оператор – оператор, який знаходиться вище по потоку обробки даних.

Downstream оператор - оператор, який знаходиться нижче по потоку обробки даних.

JWT (JSON Web Token) – стандарт токена доступу, який оснований на JSON. Використовується для створення даних, де підпис та шифрування є необов’язковими.

Project–Reactor – open-source бібліотека для мови програмування Java для реалізації моделей реактивного програмування, яка базується на специфікації реактивних потоків разом із стандартами створення реактивних додатків.

ВСТУП

Кожного року кількість пристроїв підключених до мережі Інтернет значно збільшується. Разом із цим збільшується і об’єм трафіку, який споживається. Це означає, що сучасним вебзастосункам необхідно обробляти все більшу кількість клієнтських запитів. Також, з роками вимоги користувачів стають більш жорсткими, час затримки запитів повинен бути мінімальним і сягати в деяких випадках менше, ніж 100 мілісекунд. IT гігантам із кількістю користувачів понад 500 мільйонів (LinkedIn, Twitter) дуже важко дотримуватись даних вимог.

Для адаптації бізнесу до обробки потоку великої кількості даних у режимі, близькому до реального часу, необхідно проводити ряд комплексних дій щодо створення застосунку з архітектурою, спроможною на це. Попередньо популярні архітектурні патерни, такі як Monolithic app, SOA, можуть виявитися не спроможними на це, або потребувати надмірної кількості обчислювальних ресурсів. Прикладом вирішення даної проблеми може стати розробка реактивної системи, яка відповідає вимогам Reactive Manifesto.

Реактивні системи створені відповідно до Reactive Manifesto є більш гнучкими, менше зв’язними та краще масштабуються. Завдяки цьому легше розширювати їх функціонал та вносити в нього зміни. Також, вони є більш відмовостійкими, бо виникнення помилок під час виконання для них не є причиною до повної відмови. Такі реактивні системи відрізняються ефективним та швидким зворотнім зв’язком з користувачем.

Наявність переваг, описаних вище, дозволяє зменшити час виходу на ринок та збільшити продуктивність розробників. Прикладом цього може слугувати компанія Walmart Canada. Компанія перебудувала вебзастосунок та мобільний додаток відповідно до специфікацій реактивної системи, і завдяки цьому змогла зменшити затримку запитів при високому навантаженні з 6-7 секунд до 2.4 секунди, а також значно зменшити потреби в обчислювальних ресурсах. Дані перетворення на новій платформі збільшили продажі на 235%,

час збирання проекту зменшився на 40%, а вартість експлуатації зменшилася на 50%.

Для створення реактивних систем можуть використовуватися як вбудовані можливості мов програмування, так і сторонні бібліотеки, такі як Project-Reactor, RxJava, RxJS або Rx.NET. Основна задача цих бібліотек зробити обробку потоків даних зручною та ефективною. Отже, саме від них залежить швидкодія вебзастосунку при створенні реактивних систем. Для підвищення швидкодії використовують конкурентне виконання програми, яке потребує додаткових структур даних та підходів для коректного результату. Вони приносять додаткові витрати на час виконання, які необхідно мінімізувати. Дана проблема також зустрічається в бібліотеці Project-Reactor, яка використовується для мови програмування Java.

Мета розробки: створення нового способу керування потоками даних для бібліотеки Project-Reactor, який дозволить покращити налаштування обробки потоків даних, та перевірити його вплив на швидкодію у демонстраційному проекті.

Завдання розробки:

– створення засобу для управління потоками даних у бібліотеці Project-Reactor;

– створення сервісу дистанційного замовлення для демонстрації розробленого способу для Project-Reactor.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Маніфест Реактивних Систем – це документ, що містить ряд характеристик, відповідаючи яким система може називатися реактивною. Ці характеристики були обрані на основі розробок великих компаній з метою узагальнення отриманого ними досвіду та стандартизації термінології.

Згідно маніфесту, реактивна система повинна мати такі властивості: відгучність, стійкість, гнучкість та вміння основуватися на обміні повідомленнями.

Відгучна система – це система, яка своєчасно відповідає кінцевому користувачеві, якщо це взагалі можливо. Така поведінка робить взаємодію передбачуваною та забезпечує стабільну якість обслуговування.

Стійка система – це система, що буде доступною навіть у разі відмови. Дана характеристика відноситься до усіх реактивних систем, а не тільки до високодоступних та критично важливих застосунків, так як втрата цієї властивості веде до порушення відгучності системи. Стійкість системи досягається шляхом реплікації, ізоляції та делегування.

Система не повинна мати вузьких місць та конкурувати за ресурси. Отже, вона має бути бути гнучкою. Гнучкість означає здатність системи автоматично регулювати пропускну здатність шляхом збільшення і зменшення кількості ресурсів у відповідь на зміни навантаження.

Реактивні системи засновані на асинхронному обміні повідомленнями, що дозволяє встановити межі компонентів системи і забезпечити слабку зв’язність між ними. Ці межі також дозволяють перетворювати і передавати інформацію про збої у вигляді повідомлень. Відкритий обмін повідомленнями робить можливим регулювання навантаження, гнучкість та управління потоком, для чого в системі створюються та відслідковуються черги повідомлень; в разі необхідності, використовується зворотний тиск.

Обробка потоків даних, обсяг яких не визначений заздалегідь, вимагає особливої обережності в асинхронній системі. Найпоширеніша проблема полягає в тому, що споживання ресурсів необхідно ретельно контролювати, аби швидке джерело даних не перевантажувало кінцевого споживача.

Для того, щоб приймаюча сторона не була змушена буферизувати довільні обсяги даних, необхідно забезпечувати зворотній тиск, що дозволяє обмежувати черги, які є посередниками між потоками. Переваги асинхронної обробки зведуться нанівець, якщо сигнали зворотного тиску будуть синхронними.

З метою сумісності бібліотек та застосунків, що базуються на обробці асинхронних потоків із зворотнім тиском, що не блокується, були розроблені Reactive Streams Specifications для різних мов програмування. Reactive Streams Specifications for the JVM - стандарт для мови програмування Java.

Reactive Streams Specifications визначає ряд компонентів, таких як Publisher, Subscriber, Subscription, Processor, а також поведінку та способи взаємодії між ними. Крім того, специфікація містить набір тестів The Technology Compatibility Kit (TCK) для перевірки відповідності їй.

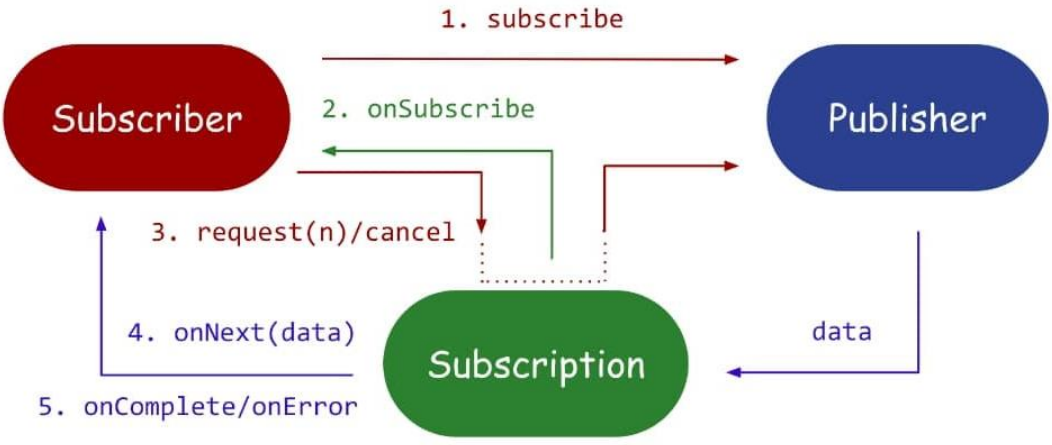


Рисунок 1.1 – Взаємодія компонентів Reactive Streams

Взаємодію основних компонентів Reactive Streams можна спостерігати на Рисунку 1.1. Клас Publisher представляє собою джерело даних, клас Subscriber – споживача даних, а клас Subscription – спеціальний клас для управління механізмом зворотного тиску. Послідовність подій при взаємодії даних класів наступна.

Крок 1 – у метод subscribe екземпляра класу Publisher передається екземпляр класу Subscriber для виконання підписки.

Крок 2 – Publisher після виконання внутрішньої логіки підписки викликає метод onSubscribe екземпляра класу Subscriber та передає йому екземпляр Subscription.

Крок 3 – Subscriber може запросити певну кількість елементів в Publisher за допомогою методу request екземпляра класу Subscription, або скасувати підписку за допомогою методу cancel екземпляра класу Subscription.

Крок 4 – запитана кількість елементів в Publisher буде передаватися до Subscriber за допомогою метода onNext, зберігаючи порядок.

Крок 5 – елементи будуть передаватися до досягнення термінальних станів помилки або закінчення елементів. При виникненні помилки Publisher викличе метод onError екземпляра Subscription, а при закінченні елементів – метод onComplete.

Для проміжної обробки потоку елементів від Publisher реалізовані спеціальні оператори. Кожен оператор має унікальну логіку. Це можуть бути як оператори map, switchMap, за допомогою яких можна перетворювати елементи, так і більш складні оператори publishOn, subscribeOn для передачі елементів на окремих потоках. Кожен оператор реалізує Publisher, Subscriber та Subscription для зберігання поведінки взаємодії цих компонентів.

Project-Reactor, RxJava та деякі інші бібліотеки повністю реалізовують встановлену поведінку специфікацій Reactive Streams та активно беруть участь в розробці нових вимог.

1.2 Змістовний опис і аналіз предметної області

Насущними задачами у реактивному програмуванні є питання в сфері оптимізації та поліпшення синтаксису взаємодії операторів. Прикладами таких досліджень може бути реактивна I/O взаємодія, злиття операторів (operator fusion), прозоре виконання віддалених запитів. У даній роботі основна увага приділена оптимізації шляхом злиття операторів.

Злиття операторів (operator fusion) це об'єднання двох або більше послідовних операторів з метою зменшення накладних витрат (час, пам'ять) на потік даних. Основні підходи оптимізації даного способу засновані на тому, що:

– багато послідовностей операторів починаються з статичних джерел повідомлень (наприклад *just(), from(T[])*), які не потребують додаткових дій для потокобезпеченої взаємодії;

– деякі оператори можуть перевикористати свої внутрішні компоненти, наприклад черги;

– деякі оператори можуть визначати, використали вони значення чи відкинули його, уникаючи додаткових запитів елементів по одному.

Життєвий цикл реактивних систем можна розділити на 3 стадії:

– час збірки – час, коли створюються екземпляри об'єктів операторів до виникнення підписки;

– час підписки – час, коли Subscriber підписується на послідовність операторів і викликає ланцюжок внутрішніх підписок в них;

– час виконання – час, коли створюються нові значення до виникнення термінальних станів помилки (error) чи закінчення (complete) потоку.

Кожен окремий етап в життєвому циклі надає різний набір можливостей оптимізації. Зазвичай виділяють 2 типи злиття операторів:

– макро-злиття – це оптимізації, що відбуваються в основному на етапі збірки. Їх суть полягає в заміні двох або більше наступних операторів одним, що скорочує накладні витрати на етапі підписки;

– мікро-злиття – це оптимізації, що відбуваються в основному на етапі підписки. Вони ґрунтуються на спільному використанні ресурсів або внутрішніх структур.

Первинно, ідея мікро-злиття полягає в тому, що ті операторі, які у ланцюжку стоять вище та використовують чергу, та ті, що стоять нижче та використовують чергу, можуть використовувати одну і ту ж чергу. Це дозволяє економити на алокації та на операціях синхронізації потоків, таких як drain-loop, work-in-progress, оновлення atomics змінних.

Всі вищеописані операції оптимізації можуть бути застосовані до бібліотеки Project-Reactor та дозволяють значно підвищити продуктивність операторів в ній.

Метою даної роботи є зміна поведінки операції prefetch у бібліотеці Project-Reactor. Операція prefetch є частиною операторів concatMap, flatMap, limitRate, publishOn. Всі ці оператори містять в собі внутрішню чергу для асинхронної обробки елементів. Сенсом операції prefetch є попередній запит заданої кількості елементів у ланцюжка операторів, що стоїть вище, під час етапу підписки.

Операція prefetch була компромісом, зробленим на початковому етапі проекту, який складно змінити на поточній стадії проекту. Він дозволяє підвищити продуктивність шляхом зменшення витрат на час очікування нових елементів у сценаріях, коли споживачі запитують великі обсяги даних, але джерело повідомлень працює ефективніше на менших розмірах запитів. Також, оператори, що реалізують операцію prefetch, дозволяють автоматично наповнювати заново чергу при досягненні ліміту в 25% від розміру черги за замовчуванням. Дані оптимізації здійсненні ціною неможливості більш тонкого налаштування цієї поведінки.

Причиною для зміни операції prefetch став ряд факторів. В першу чергу, це негативно позначається на випадках, коли джерело повідомлень має обмежені ресурси, наприклад база даних або мережева взаємодія. У деяких випадках після підписки та запиту деякого об’єму даних у ресурсу, кінцевий споживач повідомлень може взагалі не запросити елементів, таким чином марно витративши ресурси джерела повідомлень. Іншою проблемою стала не очевидна поведінка операторів, що реалізують операції prefetch для користувачів бібліотеки. На форумах міститься багато запитань про надлишкові запити через операцію prefetch.

Рішенням цієї ситуації є винесення логіки операції prefetch в окремий оператор з додаванням можливостей більш тонкого налаштування кількості запитуваних елементів. Це зробить управління потоками даних більш ефективним та більш зрозумілим, таким чином вирішивши існуючі проблеми.

Створення оператора prefetch потребує внесення змін до concatMap, flatMap, limitRate та publishOn. Після цього вони не повинні виконувати логіку операції prefetch і при цьому зберегти існуючі оптимізації.

Для перевірки впливу оператора prefetch на швидкодію обробки потоків даних додатково необхідно реалізувати демонстраційний проект з використанням бібліотеки Project-Reactor, який дозволить перевірити оператор при роботі з джерелами даних із обмеженнями. Тому в даній дипломній роботі буде розроблено частину системи сервісу дистанційних замовлень, яка дозволить запровадити дистанційні замовлення в умовах карантину. Рішення повинно бути побудоване на базі архітектурного паттерну мікросервісів, з дотриманням вимог реактивних систем та використовуючи бібліотеку Project-Reactor.

1.3 Аналіз успішних ІТ-проектів

1.3.1 Аналіз відомих технічних рішень

Спільнота бібліотеки Project-Reactor активно стежить за новими можливостями для проведення оптимізацій, так як ця бібліотека є частиною екосистеми фреймворка Spring для мови програмування Java. Цей фреймворк дозволяє організувати обробку реактивним способом починаючи з мережевого взаємодії (reactor-netty) і закінчуючи роботою з базою даних (R2DBC, Cassandra).

Прикладом макро-злиття може бути заміна оператора спеціальним оператором. Це добре демонструє випадок при ланцюжку операторів *just().SubscribeOn()* або *just().PublishOn()*, коли *just()* створює одне єдине значення. Оператори *subscribeOn()* і *publishOn()* створені для неблокуючої обробки повідомлень і вимагають для цього додаткового створення черг, ініціалізації Workers і Atomic змінних, що є зайвими в разі коли джерело подій містить єдине значення. Тому доцільно замінити ці оператори спеціальним оператором, що мінімізує накладні витрати. Дана оптимізація міститься в Project-Reactor і в RxJava.

Прикладом мікро-злиття може бути спеціальний вид Subscriber – *ConditionalSubscriber*. Даний вид Subscriber реалізовують оператори *filter(), distinct()* і їм подібні. Основна відмінність цих операторів полягає в можливості відкидання частини значень, що призведе до запитів нових елементів по одному. Численні виклики нових елементів по одному можуть призвести до ланцюжка атомарних інкрементацій лічильників, що негативно відобразиться на продуктивності. Вирішенням цієї проблеми є реалізація інтерфейсу *ConditionalSubscriber,* який представляє собою наявність методу *tryOnNext()*. Даний метод повертає логічне значення успішності перевірки елемента і відправки його далі по ланцюжку операторів, що дозволяє уникнути зайвих запитів елементів по одному.

1.4 Аналіз вимог до програмного забезпечення

Оператор prefetch повинен дозволити налаштувати наступні функції:

– розмір асинхронної черги;

– нижній ліміт кількості елементів в черзі, при якому необхідно зробити запит на нові;

– етап життєвого циклу, на якому необхідно зробити попередній запит елементів.

Враховуючи варіанти використання prefetch, які необхідно перевірити, а також появи у великої кількості магазинів та ресторанів потреби у доставці під час карантину, були виокремлені сутності та сформовані функції для програмного забезпечення для дистанційного замовлення.

Користувач – сутність звичайного користувача, який може переглядати каталог товарів та створити аккаунт для замовлення.

Клієнт – сутність користувача, який має аккаунт в системі і може зробити замовлення, а потім відслідковувати статус зміни замовлення.

CRM-система – сутність існуючого програмного забезпечення, для якого необхідно запровадити дистанційне замовлення.

Програмне забезпечення повинно мати наступні функції:

– аутентифікація користувачів;

– перегляд каталогу товарів за категоріями;

– сортування товарів за ціною;

– зберігання обраних товарів у кошику;

– створення замовлення;

– попередня оплата замовлення;

– можливість відслідковувати зміну статусу замовлення.

1.4.1 Розроблення функціональних вимог

Найпоширенішим і досить зручним способом опису функціональних вимог є діаграма прецедентів (Use Case diagram). Варіанти використання оператора prefetch зображені на Рисунку 1.2.

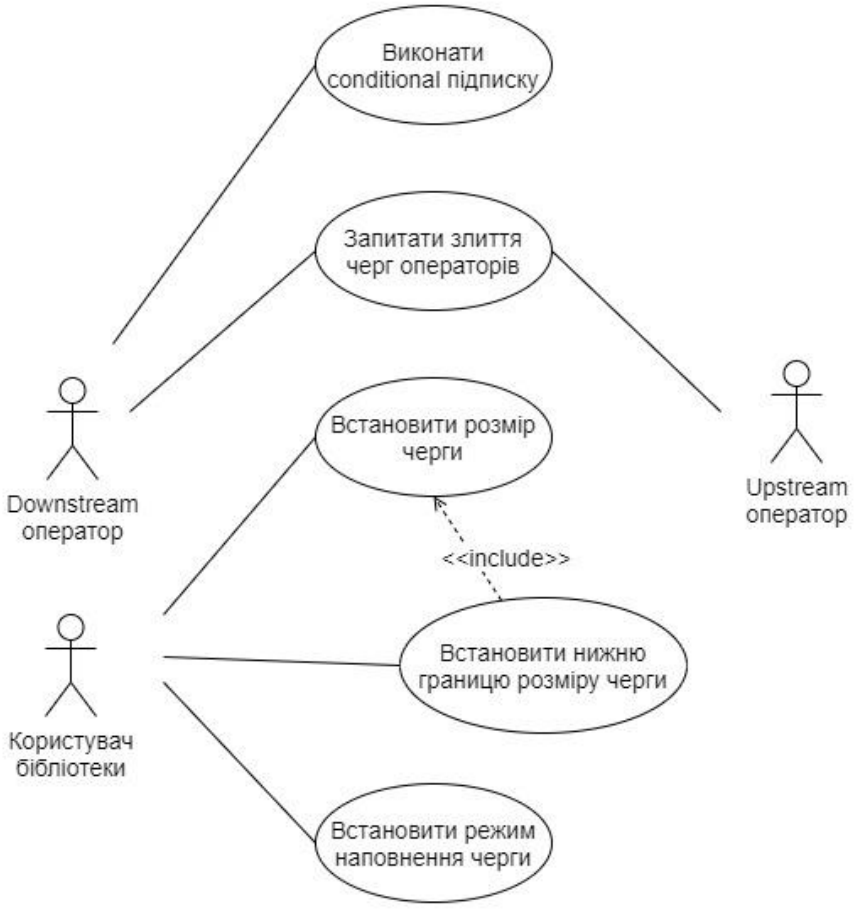


Рисунок 1.2 – Схема структурна варіантів використання оператора prefetch

Таблиця 1.1 – Варіант використання UC001

Назва	Встановити розмір черги
Опис	Користувач бібліотеки обирає розмір черги для оператора prefetch.
Учасники	Користувач бібліотеки
Передумови	Немає
Постумови	Оператор ініціалізувався та передав посилання на PrefetchPublisher до downstream оператору.

Продовження таблиці 1.1

Основний сценарій	Користувач бібліотеки за допомогою додатного цілочислового параметра встановлює розмір черги. Оператор ініціалізує асинхронну чергу заданого розміру.
Розширення сценаріїв	При відсутньому параметру розміру встановлюється розмір за замовчуванням у 32 елементи
Сценарій виключення	Якщо запит містить некоректне значення параметра, користувач отримує відповідну помилку.

Таблиця 1.2 – Варіант використання UC002

Назва	Встановити нижню границю розміру черги
Опис	Користувач бібліотеки обирає нижню границю розміру черги для оператора prefetch, при якій робиться запит на нові елементи.
Учасники	Користувач бібліотеки
Передумови	Встановлено розмір черги.
Постумови	Оператор ініціалізувався та передав посилання на PrefetchPublisher до downstream оператору.
Основний сценарій	Користувач бібліотеки за допомогою додатного цілочислового параметра встановлює нижню границю розміру черги, яка повинна бути менше розміру черги. Оператор ініціалізує асинхронну чергу та зберігає нижню границю.
Розширення сценаріїв	При відсутньому параметру границі встановлюється значення за замовчуванням у 25% від розміру черги.
Сценарій виключення	Якщо запит містить некоректне значення параметра, користувач отримує відповідну помилку.

Таблиця 1.3 – Варіант використання UC003

Назва	Встановити режим наповнення черги
Опис	Користувач бібліотеки обирає з переліку можливих значень режим наповнення черги оператора prefetch.
Учасники	Користувач бібліотеки
Передумови	Немає
Постумови	Оператор ініціалізувався, передав посилання на PrefetchPublisher до downstream оператору й у необхідний етап життєвого циклу робить запит елементів.
Основний сценарій	Користувач бібліотеки із запропонованого переліку встановлює режим наповнення черги за допомогою параметра. Оператор зберігає дане значення та робить попередній запит елементів у відповідний етап життєвого циклу оператора.

Таблиця 1.4 – Варіант використання UC004

Назва	Запитати злиття черг операторів
Опис	Downstream оператор робить запит на злиття операторів (operator fusion).
Учасники	Downstream оператор, upstream оператор
Передумови	Немає
Постумови	Оператор prefetch відповідно до обраного типу злиття змінює поведінку.
Основний сценарій	Downstream оператор робить запит на злиття з оператором prefetch. В залежності від типу upstream оператора обирається ASYNC або SYNC злиття, або злиття забороняється.

Таблиця 1.5 – Варіант використання UC005

Назва	Виконати conditional підписку
Опис	Downstream оператор під час підписки передає Subscriber, який реалізує інтерфейс ConditionalSubscriber.
Учасники	Downstream оператор
Передумови	Downstream оператор є conditional оператором
Постумови	Оператор prefetch ініціалізувався та передає події з дотриманням контракту conditional операторів.
Основний сценарій	Оператор prefetch під час етапу підписки отримує ConditionalSubscriber, з ким надалі буде взаємодіяти за контрактом conditional операторів.

На основі вище описаних варіантів використання оператора prefetch сформовано функціональні вимоги, зазначені у Таблиці 1.6.

Таблиця 1.6 – Опис функціональних вимог оператора prefetch

Ідентифікатор	Назва	Опис
REQ001	Конфігурування оператора	Бібліотека надає можливість встановити розмір черги, її нижню границю та режим наповнення черги.
REQ002	Передача подій у режимі ASYNC злиття черг.	Оператор prefetch передає елементи нижче по потоку з дотриманням контракту взаємодії ASYNC fusion.
REQ003	Передача подій у режимі SYNC злиття черг.	Оператор prefetch передає елементи нижче по потоку з дотриманням контракту взаємодії SYNC fusion.

Продовження таблиці 1.6

REQ004	Передача подій у режимі злиття з Conditional оператором.	Оператор prefetch передає елементи нижче по потоку з дотриманням контракту взаємодії з Conditional Subscriber.

На Рисунку 1.3 зображена матриця трасування функціональних вимог та сценаріїв використання оператора prefetch.

	REQ001	REQ002	REQ003	REQ004
UC001	■			
UC002	■			
UC003	■			
UC004		■	■	
UC005				■

Рисунок 1.3 – Матриця трасування оператора prefetch

Діаграма прецедентів, яка зображує варіанти використання сервісу дистанційного замовлення зображена на Рисунку 1.4.

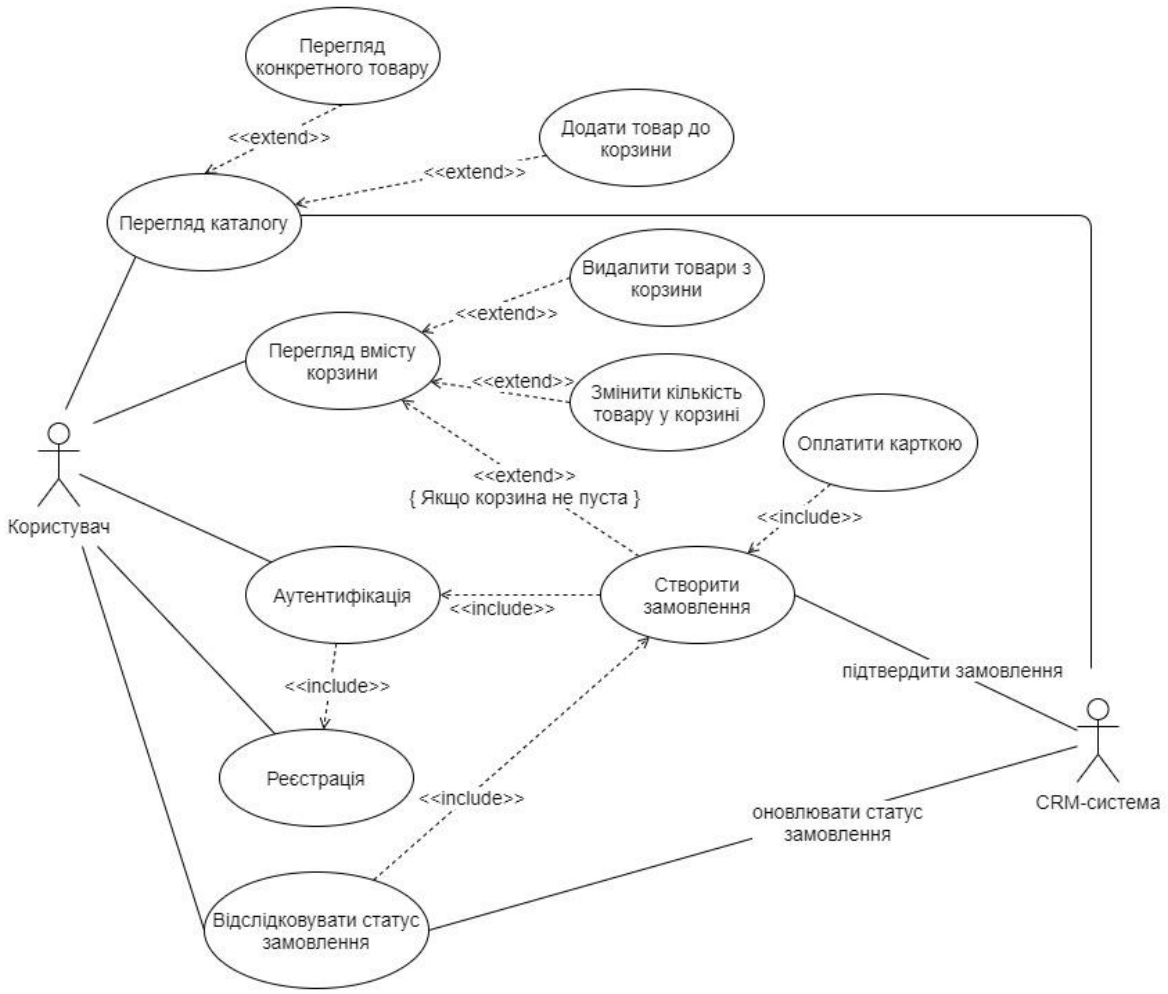


Рисунок 1.4 – Схема структурна варіантів використання сервісу дистанційного замовлення

Таблиця 1.7 – Варіант використання UC101

Назва	Перегляд каталогу товарів
Опис	Користувач обирає порядок сортування, категорію та переглядає перелік товарів у каталозі.
Учасники	Користувач
Передумови	Немає
Постумови	Користувач отримав перелік товарів у каталозі, який відповідає вказаним параметрам.
Основний сценарій	Користувач надсилає запит з параметрами порядку сортування та обраною категорією.

Продовження таблиці 1.7

Розширення сценаріїв	При відсутності параметра порядку сортування, товари в каталозі знаходяться в довільному порядку. При відсутності параметра обраної категорії, користувач отримує товари з усіх категорій.
Сценарій виключення	Якщо запит містить некоректні параметри, користувач отримує відповідь з повідомленням про помилку в параметрах.

Таблиця 1.8 – Варіант використання UC102

Назва	Перегляд конкретного товару
Опис	Користувач переглядає конкретну інформацію про товар.
Учасники	Користувач
Передумови	Користувач має ідентифікатор товару.
Постумови	Користувач отримав повну інформацію про товар.
Основний сценарій	Користувач надсилає запит із вказанням ідентифікатора товару.
Сценарій виключення	Якщо товару з обраним ідентифікатором не існує, користувач отримує відповідь з повідомленням про помилку перегляду неіснуючого товару.

Таблиця 1.9 – Варіант використання UC103

Назва	Додати товар до корзини
Опис	Клієнт додає обраний товар до корзини.
Учасники	Клієнт
Передумови	Користувач виконав аутентифікацію. Користувач має ідентифікатор товару.
Постумови	Товар доданий до корзини.

Продовження таблиці 1.9

Основний сценарій	Клієнт робить запит із вказанням ідентифікатора товару та його кількості. Система перевіряє наявність товару та додає до корзини користувача.
Сценарій виключення	Якщо товару з обраним ідентифікатором не існує, користувач отримує відповідь із повідомленням про помилку додавання до корзини неіснуючого товару.

Таблиця 1.10 – Варіант використання UC104

Назва	Реєстрація
Опис	Користувач може створити аккаунт у системі.
Учасники	Користувач
Передумови	Немає
Постумови	Аккаунт клієнта створено в системі
Основний сценарій	Користувач надсилає запит з персональними даними (логін, пароль, адреса, тощо). Система створює аккаунт користувача.
Сценарій виключення	Якщо аккаунт з логіном користувача вже існує, він отримує відповідь з помилкою про це. Якщо користувач не надіслав обов’язкові персональні дані, він отримує відповідь з помилкою про це.

Таблиця 1.11 – Варіант використання UC105

Назва	Аутентифікація
Опис	Користувач може автентифікуватися за логіном та паролем.
Учасники	Користувач
Передумови	Немає
Постумови	Користувач увійшов у систему як клієнт.

Продовження таблиці 1.11

Основний сценарій	Користувач відправляє запит на сервіс аутентифікації з власним логіном та паролем. Система проводить автентифікацію клієнта.
Сценарій виключення	Сервіс аутентифікації виявив, що за відправленим логіном та паролем не існує користувача. Користувач отримує відповідь з повідомленням про помилку аутентифікації.

Таблиця 1.12 – Варіант використання UC106

Назва	Перегляд вмісту корзини
Опис	Клієнт переглядає вміст корзини для редагування або створення замовлення.
Учасники	Клієнт
Передумови	Користувач виконав аутентифікацію.
Постумови	Користувач отримав перелік товарів у корзині з їх кількістю.
Основний сценарій	Користувач надсилає запит системі для отримання переліку товарів у корзині.

Таблиця 1.13 – Варіант використання UC107

Назва	Видалити товари із корзини.
Опис	Клієнт видаляє декілька товарів з корзини.
Учасники	Клієнт
Передумови	Користувач виконав аутентифікацію.
Постумови	Вміст корзини було оновлено відповідно до запиту клієнта, видалені товари не присутні в корзині.
Основний сценарій	Користувач надсилає запит системі із переліком товарів для видалення із корзини.

Таблиця 1.14 – Варіант використання UC108

Назва	Змінити кількість товару в корзині
Опис	Клієнт редагує кількість одиниць товару в корзині.
Учасники	Клієнт
Передумови	Користувач виконав аутентифікацію.
Постумови	Кількість одиниць товару було оновлено відповідно до запиту клієнта.
Основний сценарій	Клієнт надсилає запит системі з переліком оновлених товарів та їх кількістю.
Розширення сценаріїв	Якщо після оновлення кількість одиниць товару рівна 0 – товар видаляється.
Сценарій виключення	Якщо оновлена кількість одиниць товару, який відсутній у корзині, клієнт отримує відповідь з помилкою про відсутність товару в корзині.

Таблиця 1.15 – Варіант використання UC109

Назва	Створити замовлення
Опис	Клієнт створює замовлення з товарами, присутніми в корзині.
Учасники	Клієнт, CRM-система.
Передумови	Користувач виконав аутентифікацію. Товари присутні в корзині клієнта.
Постумови	Замовлення клієнта створено та очікує на обробку місцем замовлення. Місце замовлення отримує повідомлення про створення замовлення.
Основний сценарій	Клієнт надсилає запит на створення замовлення із вказанням адреси доставки.

Продовження таблиці 1.15

Сценарій виключення	Якщо при створенні замовлення корзина з товарами пуста, клієнт отримує відповідь з помилкою про відсутність товарів в корзині.

Таблиця 1.16 – Варіант використання UC110

Назва	Оплатити карткою
Опис	Клієнт може попередньо оплатити замовлення карткою.
Учасники	Клієнт, CRM-система.
Передумови	Користувач виконав аутентифікацію. Замовлення створено і не завершено.
Постумови	Замовлення оплачено. Місце замовлення отримує повідомлення про оплату замовлення.
Основний сценарій	Клієнт за ідентифікатором замовлення оплачує замовлення. Система обробляє результат оплати та сповіщає місце замовлення про успішну оплату.
Сценарій виключення	Якщо оплата було відхилена, клієнт отримує відповідь з повідомленням про помилку оплати.

Таблиця 1.17 – Варіант використання UC111

Назва	Відслідковувати статус замовлення
Опис	Клієнт може відслідковувати зміну статусу замовлення
Учасники	Клієнт, CRM-система.
Передумови	Користувач виконав аутентифікацію. Замовлення створено.
Постумови	Клієнт отримав повідомлення про статус замовлення. Замовлення доставлено.

Продовження таблиці 1.17

Основний сценарій	Клієнт підписується на сповіщення про зміну статусу замовлення та отримує повідомлення при оновленні статусу системою управління місцем замовлення.
Сценарій виключення	Якщо клієнт відписався від сповіщень, статус фіксується системою для подальшого перегляду клієнтом.

На основі вище описаних варіантів використання сервісу дистанційного замовлення сформовано функціональні вимоги зазначені у Таблиці 1.18.

Таблиця 1.18 – Опис функціональних вимог сервісу дистанційного замовлення

Ідентифікатор	Назва	Опис
REQ101	Реєстрація	Користувач для створення замовлень може створити акаунт.
REQ102	Аутентифікація користувача	Для створення замовлень та відслідковування їх статусу користувач має пройти аутентифікацію.
REQ103	Перегляд конкретного товару	Користувач може переглянути детальну інформацію обраного ним товару.
REQ104	Перегляд всіх товарів	Користувач може переглянути каталог товарів.
REQ105	Перегляд товарів з категорії	Користувач може переглянути товари в каталозі за обраною категорію.
REQ106	Сортувати товари за ціною	Користувач має змогу обрати порядок товарів у каталозі за їх ціною.
REQ107	Отримати вміст корзини	Клієнт має змогу отримати перелік товарів з їх кількістю у його корзині.

Продовження таблиці 1.18

REQ108	Оновити вміст корзини	Клієнт може за допомогою команд оновлювати наявність та кількість товарів в корзині.
REQ109	Створити замовлення	Клієнт може здійснити замовлення товарів з корзини.
REQ110	Обробити замовлення	Місце замовлення має обробити замовлення клієнта, підтвердити або скасувати його.
REQ111	Оплатити замовлення	Клієнт має змогу зробити попередню оплату замовлення.
REQ112	Отримати статус замовлення	Клієнт може отримати поточне значення статусу замовлення.
REQ113	Оновити статус замовлення	Місце замовлення оновлює статус замовлення по мірі його зміни.
REQ114	Сповістити про зміну статусу замовлення	Клієнт може підписатися на повідомлення про зміну статусу замовлення.

На Рисунку 1.5 зображена матриця трасування функціональних вимог та сценаріїв використання сервісу дистанційного замовлення.

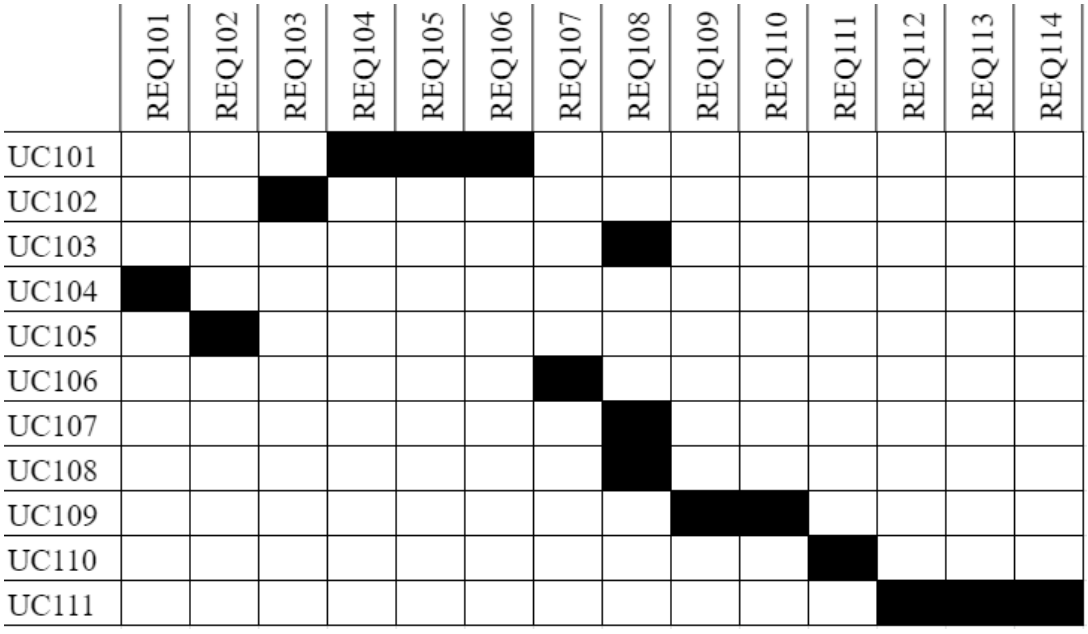


Рисунок 1.5 – Матриця трасування сервісу дистанційного замовлення

1.4.2 Розроблення нефункціональних вимог

Оператор prefetch є частиною бібліотеки Project-Reactor, а отже повинен відповідати всім нефункціональним вимогам, висунутим операторам бібліотеки. Оператор prefetch повинен:

- проходити TCK тест;
- проходити тести продуктивності (benchmark) бібліотеки Project-Reactor;
- бути тестованим для покриття оператора Unit тестами за допомогою засобів бібліотеки.

Сервіс дистанційного замовлення розробляється для перевірки нового способу керування потоком даних та для демонстрації переваг реактивних систем. Для цього сервіс повинен дотримуватися принципів Reactive Manifesto, щоб називатися реактивним. Також сервіс повинен відповідати нефункціональним вимогам:

- обмін повідомленнями між сервісами за допомогою бінарного протоколу RSocket прикладного рівня з можливістю зворотного тиску;
- обмін повідомленнями з клієнтом за допомогою бінарного протоколу RSocket;
- мати систему моніторингу ресурсів та трасування запитів.

1.4.3 Постановка комплексу завдань модулю

Розроблюване програмне забезпечення призначене для вирішення проблеми з неявною поведінкою в операції prefetch бібліотеки Project-Reactor в операторах concatMap, flatMap, limitRate та publishOn, що містять чергу для асинхронної обробки повідомлень із збереженням порядку.

Метою розробки є удосконалення способів управління потоками даних у бібліотеці Project-Reactor за допомогою створення нового оператора prefetch та перевірки впливу цього оператора на швидкодію бібліотеки на прикладі

мікросервісів реактивної системи для дистанційного замовлення. Для цього в даному дипломному проекті необхідно вирішити задачі:

- розробка оператора prefetch;
- видалення поведінки операції prefetch з операторів concatMap, flatMap, limitRate та publishOn;
- створити демонстраційний проект дистанційного замовлення з використанням бібліотеки Project-Reactor;
- створення засобів для перегляду метрик використаних ресурсів та для трасування запитів сервісу замовлень.

1.5 Висновки по розділу

Створення високопродуктивних систем з вимогами до масштабованості, відгучності та стійкості у наш час є досить розповсюдженим. Для розробки даних систем можна скористатися парадигмою реактивних систем, принципи яких описані у Reactive Manifesto. Тому у даному розділі був розглянутий засіб для створення реактивних систем – бібліотека Project-Reactor, та запропоновані способи його поліпшення.

В розділі був проведений аналіз існуючих технічних рішень, висунуті вимоги до програмного забезпечення та демонстраційного проекту. Вимоги представлені за допомогою структурних діаграм прецедентів, функціональних та нефункціональних вимог. На їх основі було поставлено комплекс завдань модулю.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

2.1.1 Моделювання програмного забезпечення для Project-Reactor

Для подальшого моделювання бізнес сценаріїв взаємодії нового оператора Prefetch із бібліотекою необхідно ввести абстракції бібліотеки, що використовуються для створення операторів. Вони представлені на діаграмі класів на Рисунку 2.1, а опис цих класів у Таблиці 2.1.

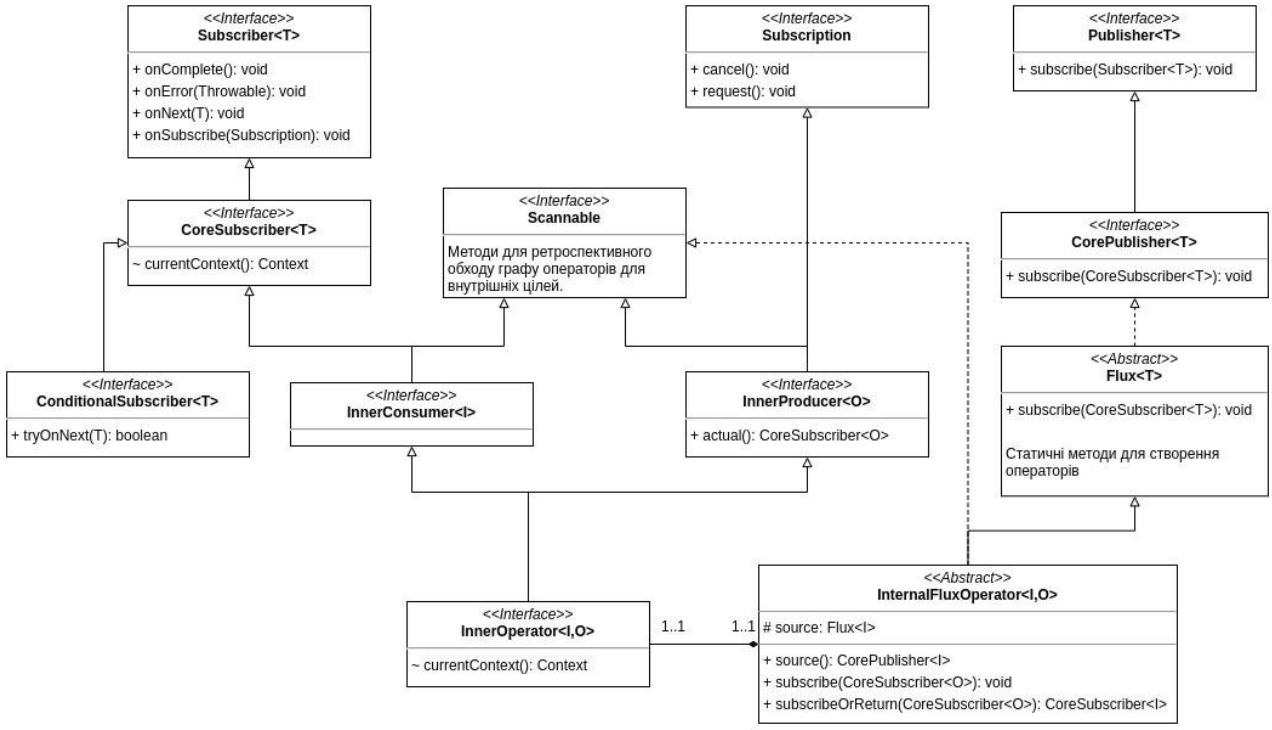


Рисунок 2.1 – Схема структурна класів бібліотеки Project-Reactor

Таблиця 2.1 – Опис внутрішніх класів бібліотеки Project-Reactor

Клас	Опис
Subscriber<T>	Інтерфейс, який задає контракт отримання даних та сигналів підписником.

Продовження таблиці 2.1

CoreSubscriber<T>	Інтерфейс, який дозволяє коректно працювати підписнику в умовах виконання на декількох потоках.
ConditionalSubscriber<T>	Інтерфейс, який вводить спеціальний контракт взаємодії з умовними операторами для зменшення кількості викликів.
Subscription	Інтерфейс, який задає контракт для виконання механізму зворотного тиску.
Scannable	Інтерфейс, який задає контракт для ретроспективного обходу графу операторів для внутрішніх цілей.
InnerConsumer<I>	Інтерфейс обгортка, який додає до CoreSubscriber поведінку Scannable.
InnerProducer<O>	Інтерфейс обгортка, який додає до Subscription поведінку Scannable.
InnerOperator<I,O>	Інтерфейс внутрішнього класу в InternalFluxOperator, який реалізує в собі одночасно Subscriber та Subscription.
Publisher<T>	Інтерфейс, який задає контракт взаємодії з джерелом даних.
CorePublisher<T>	Інтерфейс обгортка для виклику спеціальних хуків для керуванням часом життя елементів.

Продовження таблиці 2.1

Flux<T>	Абстрактний клас, який містить статичні методи для створення всіх Flux операторів, а також реалізує базову реалізацію subscribe.
InternalFluxOperator<I,O>	Абстрактний клас внутрішньої реалізації оператора.

Попередня реалізація злиття операторів та предзапиту елементів працювала наступним чином, представленим на діаграмі послідновності на Рисунку 2.2.

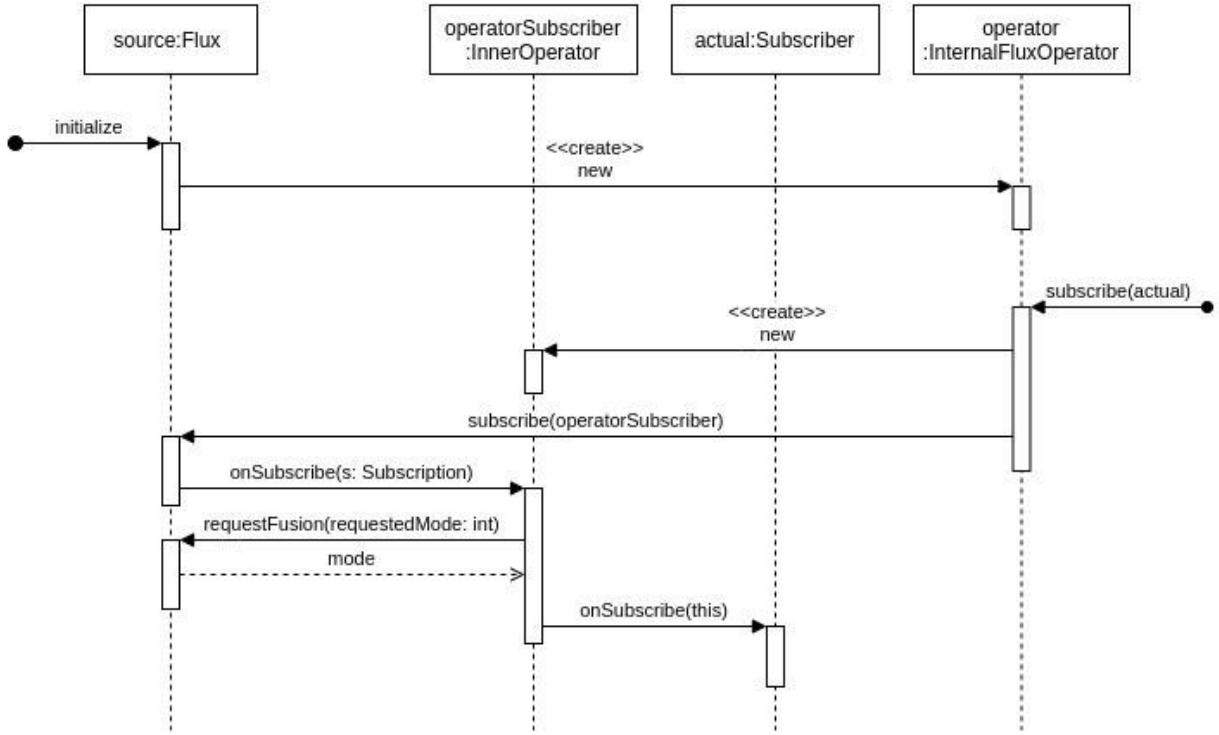


Рисунок 2.2 – Діаграма послідовності операцій злиття у бібліотеці Project-Reactor

Після завершення етапу підписки (отримання Subscription s за допомогою методу onSubscribe) оператор міг зробити запит на злиття із оператором, що стоїть вище за допомогою методу requestFusion, передавши у метод тип злиття (ANY, ASYNC), який підтримує. Оператор source, що стоїть вище, може прийняти злиття, повернувши тип злиття, який буде використовуватись далі (ASYNC, SYNC), або відхилити злиття, повернувши тип NONE. Після

завершення всіх допоміжних етапів оператор при підтримці функції передзапиту елементів робив запит елементів за допомогою методу request, не чекаючи запита від підписника actual.

Для підвищення ефективності управління потоками даних я пропоную ускладнити схему установки злиття, вводячи додаткові етапи, та переглянути необхідність відправки частини сигналів. Запропонований мною варіант зображений на діаграмі послідовностей у документі КПІ.ІП-7120.045490.06.99.СС “Схема структурна послідовності”. Оператор Prefetch повинен робити злиття декількома способами: downstream оператор підтримує злиття або downstream оператор не підтримує злиття.

Розглянемо перший варіант, коли downstream оператор підтримує злиття:

– як тільки PrefetchSubscriber отримав підписку від source за допомогою методу OnSubscribe, він повинен одразу надати оператору нижче себе як підписку;

– PrefetchSubscriber отримує запит на злиття від downstream оператора з підтримуваним типом злиття;

– якщо upstream оператор підтримує злиття:

  – PrefetchSubscriber намагається наскрізно злитися з оператором вище, відправивши тип злиття оператора нижче за допомогою метода requestFusion;

  – PrefetchSubscriber отримує від оператора source обраний ним тип злиття і фіксує його;

  – якщо обраний тип не дорівнює NONE, то він повертає даний тип;

  – якщо PrefetchSubscriber не зміг наскрізно злитися, то він намагається злитися з оператором нижче у режимі ASYNC і повертає результат;

– якщо upstream оператор не підтримує злиття:

– PrefetchSubscriber намагається злитися з оператором нижче у режимі ASYNC і повертає результат.

Розглянемо другий варіант, коли downstream оператор не підтримує злиття:

– як тільки PrefetchSubscriber отримав підписку від source за допомогою методу OnSubscribe, він повинен одразу надати оператору нижче себе як підписку;

– якщо upstream оператор підтримує злиття:

– PrefetchSubscriber намагається злитися з оператором вище за допомогою метода requestFusion, відправивши ANY тип злиття;

– PrefetchSubscriber отримує від оператора source обраний ним тип злиття і фіксує його.

Передзапит елементів виконується на різних етапах в залежності від обраного режиму передзапиту та встановленого режиму злиття.

– Передзапит елементів не потрібно виконувати, якщо оператор Prefetch злився з upstream оператором для зменшення кількості викликів.

– При режимі злиття EAGER, необхідно зробити передзапит елементів одразу після закінчення етапу Subscriptions життєвого циклу, тобто після передачі downstream оператору підписки.

– При режимі злиття LAZY, необхідно зробити передзапит елементів тільки після запиту у Prefetch оператора елементів, тобто після виклику downstream оператором метода request.

Як можна помітити запропонована схема набагато складніша за звичайну. За допомогою такого процесу злиття можна відкласти попередній запит елементів у source до моменту коли actual зробить запит на них. Це дозволить зменшити кількість відброшених елементів, якщо actual замість запиту на елементи скасує підписку. Також дана схема дозволяє зменшити кількість викликів методів та звернень до volatile полів за рахунок непотрібності робити виклики request у оператора вище при типі злиття ASYNC.

2.1.2 Моделювання програмного забезпечення сервісу дистанційного замовлення

Для моделювання бізнес процесів дистанційного замовлення був обраний спосіб за допомогою BPMN діаграм. Основний процес створення замовлення користувачем наведений у документі КПІ.ІП-7120.045490.06.99.СС “Схема структурна бізнес-процесів”. Він включає наступні стадії:

– користувач переглядає меню товарів;
– користувач додає обрані товари до корзини;
– користувач формує замовлення;
– замовлення має бути опрацьовано менеджером;
– якщо замовлення було скасовано, замовлення скасовується;
– якщо замовлення підтверджено, користувач оплачує замовлення;
– після оплати місце замовлення перевіряє статус оплати;
– якщо під час оплати виникла помилка, замовлення скасовується;
– після оплати користувач має змогу відслідковувати повідомлення про зміну статусу замовлення до його завершення.

2.2 Архітектура програмного забезпечення

Розробка оператора не передбачає внесення змін в архітектуру бібліотеки Project-Reactor. Необхідно використовувати створені абстракції для виконання наступних поставлених задач у межах існуючих модулей.

Для розробки сервісу дистанційного замовлення був обраний архітектурний підхід мікросервісів. Так як була поставлена задача розробки реактивної системи, то даний підхід наштовхує на створення продукту саме за схемою мікросервісів.

Архітектура мікросервісів надає ряд переваг при її доречному використанні. Завдяки розділу завдань по різних сервісах, падіння частини сервісів не виведе з ладу додаток повністю, що підвищує відмовостійкість системи. Також, незалежність сервісів між собою дає можливість масштабувати

кожен сервіс окремо. Reactive Manifesto якраз виділяю ці властивості як одні з ключових для реалізації, тому при належній якості розробки система може отримати всі переваги реактивних систем.

Для розробки кожного сервісу було обрано підхід трирівневої архітектури. Даний підхід дозволяє виділити 3 слої у додатку: шар представлення, шар бізнес логіки, шар даних. Це дозволить відділити домену модель від деталей реалізації та слоїв інфраструктури, що дозволить зменшити зчеплення модулів та підвищити зв’язність коду.

В цілому було виділено наступні сервіси, що представлені на Рисунку 2.3 у вигляді діаграми розгортання. Опис призначень сервісів представлено у Таблиці 2.2.

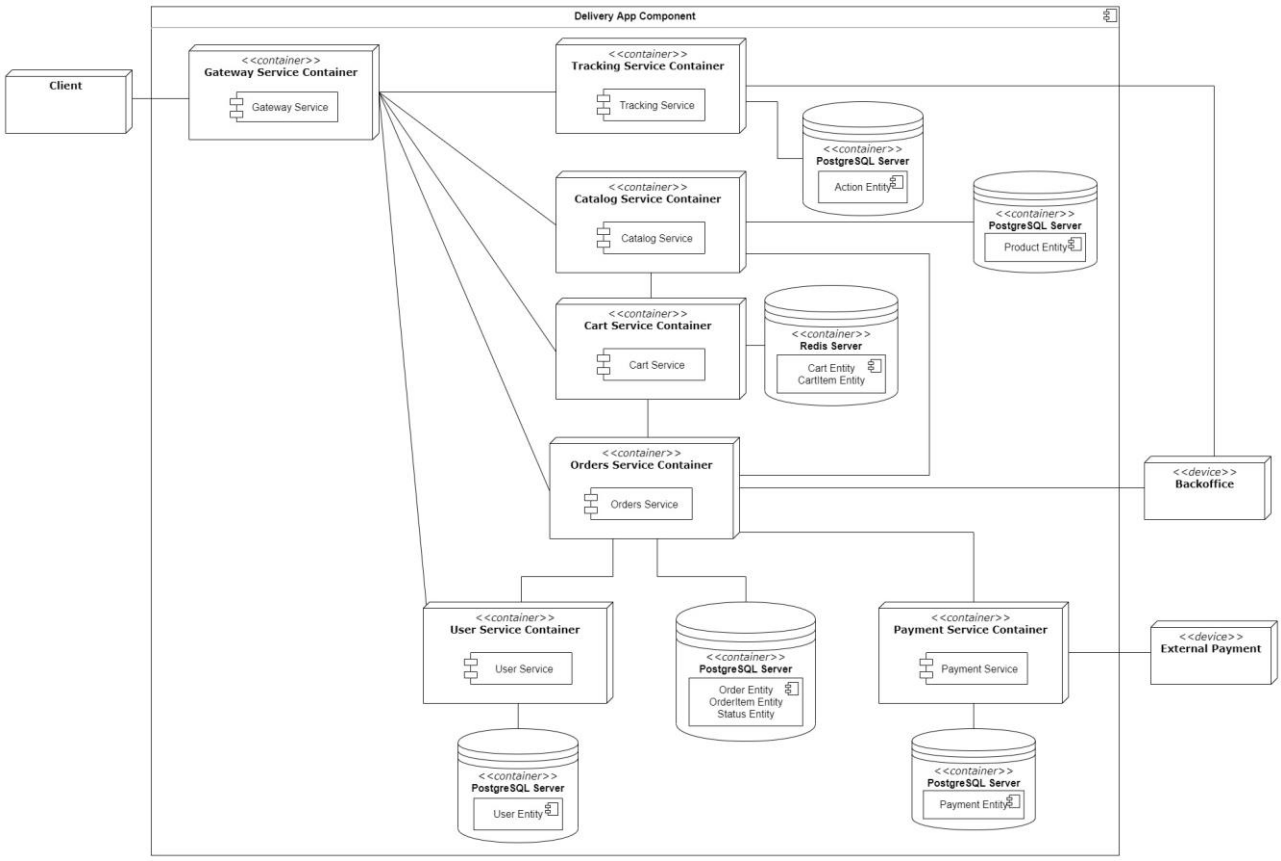


Рисунок 2.3 – Діаграма розгортання сервісу дистанційного замовлення

Таблиця 2.2 – Опис сервісів дистанційного замовлення

Назва сервісу	Опис
Gateway Service	Сервіс, який виконує функцію шлюзу та проводить аутентифікацію користувачів.
User Service	Сервіс, який зберігає інформацію про користувачів.
Catalog Service	Сервіс, який надає доступ до переліку товарів.
Cart Service	Сервіс, який надає можливість користувачам зберігати товари у корзині.
Orders Service	Сервіс, який надає можливість користувачам працювати з замовленнями.
Payment Service	Сервіс, який дозволяє інтегруватися додатку з різними плаіжними системами.
Tracking Service	Сервіс, який дозволяє користувачу відслідковувати актуальний статус замовлення.
Backoffice	Зовнішній сервіс, який вводить абстракцію системи управління рестораном з якою буде працювати додаток.
External Payment	Зовнішній сервіс, який вводить абстракцію зовнішньої системи оплати з якою буде працювати додаток.

2.3 Конструювання програмного забезпечення

2.3.1 Конструювання програмного забезпечення для Project-Reactor

При розробці програмного забезпечення для додавання нового способу керування потоком даних у бібліотеці Project Reactor були створені нові оператори та змінені існуючі. Зміни зображені у вигляді діаграми класів

представленої у документі КПІ.ІП-7120.045490.06.99.СС “Схема структурна класів програмного забезпечення для Project-Reactor”. Опис класів програмного забезпечення наведений у Таблиці 2.3, а опис методів у Таблицях 2.4 – 2.6.

Таблиця 2.3 – Опис класів програмного забезпечення

Клас	Опис
Fuseable	Інтерфейс флаг, який позначає що попередній оператор має можливість провести операцію злиття.
QueueSubscription	Інтерфейс, який дозволяє оператору нижче використовувати підписку як чергу при злитті для зменшення кількості викликів.
FluxPrefetch<T>	Клас нового оператора, який реалізує контракт Publisher та композує у собі Subscriber та Subscription.
PrefetchSubscriber<T>	Внутрішній клас FluxPrefetch, який одночасно реалізує контракт Subscriber та Subscription.
PrefetchConditionalSubscriber<T>	Внутрішній клас FluxPrefetch, який одночасно реалізує контракт Subscriber та Subscription і використовується, якщо оператор нижче є умовним для зменшенн кількості викликів.

Таблиця 2.4 – Опис методів класу FluxPrefetch

Назва методу	Призначення
getPrefetch	Метод для отримання кількості елементів, яку буде передзапитано у Subscription.
subscribeOrReturn	Метод для виконання етапу підписки і реалізації оптимізації роботи з умовними підписником.

Таблиця 2.5 – Опис методів класу PrefetchSubscriber

Назва методу	Призначення
onSubscribe	Метод Callback, за допомогою якого оператор отримує Subscription для запиту елементів. Також на даному етапі оператор намагається провести злиття з оператором вище, якщо це можливо і оператор нижче вже не злився з Prefetch.
onNext	Метод Callback, за допомогою якого оператор отримує дані для складання в чергу або null як сигнал при асинхронному злитті про можливість продовження отримання даних.
onError	Метод Callback, за допомогою якого оператор отримує помилку яка виникла вище за потоком і повинен зупинити передачу елементів і передати її далі.
onComplete	Метод Callback, за допомогою якого оператор отримує сигнал про закінчення даних і повинен передати його далі.
request	Метод, за допомогою якого оператор отримує кількість елементів яку далі необхідно передати оператору нижче.

Продовження таблиці 2.5

cancel	Метод, за допомогою якого оператор отримує сигнал про завершення роботи, який необхідно передати оператору вище.
drainAsync	Метод, за допомогою якого оператор обробляє дані з потоку типів ASYNC, NONE і передає елементи оператору нижче.
drainOutput	Метод, за допомогою якого оператор обробляє дані з потоку вище, якщо оператор злився з операторо нижче.
drainSync	Метод, за допомогою якого оператор обробляє дані з потоку типу SYNC і передає елементи оператору нижче.
drain	Метод для вибору стратегії обробки елементів drainAsync, drainOutput або drainSync.
checkTerminated	Метод для перевірки оператором досягнення термінальних станів, щоб зупинити передачу елементів нижче в разі необхідності.
clear	Метод для очищення черги, або при злитті з оператором вище, передачі оператору вище сигналу для очищення черги.
isEmpty	Метод для перевірки відсутності елементів у черзі.
poll	Метод для отримання оператором нижче елементів при злитті з ним.
requestFusion	Метод, за допомогою оператор нижче може зробити запит на злиття з оператором Prefetch, передавши підтримуваний тип (ANY, ASYNC).

Продовження таблиці 2.5

size	Метод для перевірки кількості елементів у черзі.
actual	Метод, за допомогою якого можна отримати підписника оператора нижче.

Таблиця 2.6 – Опис методів класу PrefetchConditionalSubscriber

Назва методу	Призначення
drainAsync	Метод, за допомогою якого оператор обробляє дані з потоку типів ASYNC, NONE і передає елементи оператору нижче, враховуючи контракт обміну повідомленнями з умовними операторами.
drainOutput	Метод, за допомогою якого оператор обробляє дані з потоку вище, якщо оператор злився з операторо нижче, враховуючи контракт обміну повідомленнями з умовними операторами.
drainSync	Метод, за допомогою якого оператор обробляє дані з потоку типу SYNC і передає елементи оператору нижче, враховуючи контракт обміну повідомленнями з умовними операторами.
poll	Метод для отримання оператором нижче елементів при злитті з ним, враховуючи контракт обміну повідомленнями з умовними операторами.

2.3.2 Конструювання програмного забезпечення сервісу дистанційного замовлення

Для розробки сервісу дистанційного замовлення необхідно обрати фреймворк для Java, який надає можливість працювати з реактивними

потоками, а також бути адаптований для роботи системи за схемою мікросервісів. Серед численної кількості фреймворків було обрано один із найпопулярніших – Spring Framework. Даний фреймворк має нативну підтримку роботи з протоколом RSocket, багато бібліотек для організації мікросервісів, а також дозволяє працювати з базами даних у реактивному стилі за допомогою R2DBC.

З великої кількості фреймворків для роботи з Spring Framework були обрані ті, які дозволяють працювати у неблокуючому реактивному стилі з даними. Для роботи з мережею у реактивному стилі було обрано Spring WebFlux з драйвером для протоколу RSocket. За цими же парматерами було обрано фреймворк для доступу до баз даних Spring Data R2DBC з необхідними драйверами. Для забезпечення безпеки як при роботі з мережею так і для зберігання даних було обрано фреймворк Spring Security. Для трасування запитів було використано Spring Sleuth з розподіленою системою Zipkin.

Для налаштування інфраструктури було обрано засіб контейнеризації Docker, а для оркестрації інструмент Docker Compose. Docker дозволяє описати процес збірки програмного забезпечення, а потім ізолювати його виконання. За допомогою Docker Compose можна декларативно описати інфраструктуру: налаштувати запуск образів, передачу файлів конфігурації, мережу і дії у разі відмови сервісів. Усе це дозволяє покращити процес розробки і зменшити кількість проблем з конфігурацією інфраструктури.

Представлення сутностей надана у вигляді ER-діаграми на Рисунку 2.4, а їх детальний опис та опис властивостей надано у Таблицях 2.6 – 2.7.

Рисунок 2.4 – ER-діаграма сервісу дистанційного замовлення

Таблиця 2.7 – Опис сутностей сервісу дистанційного замовлення

Сутність	Опис
User	Сутність, яка представляє інформації про користувача, а також дані для аутентифікації.
Product	Сутність, яка представляє товар у каталозі товарів.
Cart	Сутність корзини користувача.
CartItem	Сутність товару доданого у корзину.
Order	Сутність замовлення, яка надає усю необхідну інформацію про замовлення.
OrderItem	Сутність товару у замовленні.
Payment	Сутність оплати для історичного зберігання всіх транзакцій.
Action	Сутність події, яке відбувається із замовленням для відстеження користувачем його зміни.

Таблиця 2.8 – Опис властивостей сутностей

Сутність	Властивість	Опис
User	id	Первинний ключ таблиці Users. Тип: integer
User	id	Первинний ключ сутності User. Тип: UUID
User	name	Ім’я користувача. Тип: varchar
User	email	Пошта користувача. Тип: varchar
User	encoded_password	Захешований пароль користувача. Тип: varchar
User	authorities	Роль користувача. Тип: varchar
User	phone	Телефон користувача Тип: varchar
User	address	Адреса користувача за замовчуванням. Тип: varchar
User	created_at	Дата створення аккаунту користувача. Тип: timestamp
Product	id	Первинний ключ сутності Product. Тип: UUID.

Продовження таблиці 2.8

Product	name	Назва товару. Тип: varchar
Product	description	Детальний опис товару. Тип: varchar
Product	img_link	Посилання на зображення товару. Тип: varchar
Product	price	Вартість товару. Тип: BigDecimal
Product	currency	Валюта вартості товару. Тип: Currency
Product	available	Доступність товару. Тип: boolean
Order	id	Первинний ключ сутності Order. Тип: UUID
Order	user_id	Зовнішній ключ на сутність Користувача. Тип: UUID
Order	address	Адреса куди необхідно доставити замовлення. Тип: varchar
Order	created_at	Час створення замовлення. Тип: timestamp

Продовження таблиці 2.8

Order	status	Статус замовлення, можливі варіанти: – processing - обробка – canceled - скасовано – proceeding - в прогресі – completed - завершено Тип: varchar
Order	created_at	Час змінення статусу замовлення. Тип: timestamp
OrderItem	order_id	Зовнішній ключ на сутність Користувача. Тип: UUID
OrderItem	product_id	Зовнішній ключ на сутність Товару. Тип: UUID
OrderItem	price	Вартість продукту в замовленні. Тип: BigDecimal
OrderItem	quantity	Кількість товару у замовленні. Тип: integer

Опис класів, інтерфейсів шару доступу до даних описаний у Таблиці 2.8. У Таблиці 2.9 наданий опис класів контролерів, а у Таблиці 2.10 надано опис класів сервісів.

Таблиця 2.9 – Шар доступу до даних

Клас	Опис
UserRepository	Інтерфейс CRUD репозиторію для доступу до даних про користувача.

Продовження таблиці 2.9

OrderRepository	Інтерфейс CRUD репозиторію для доступу до даних про замовлення.
ActionRepository	Інтерфейс CRUD репозиторію для доступу до даних про зміну статусу замовлення.
PaymentRepository	Інтерфейс CRUD репозиторію для доступу до даних про статус оплати замовлення.
CartRepository	Інтерфейс репозиторію для зберігання корзини користувача.
CatalogRepository	Інтерфейс CRUD репозиторію для доступу до даних про доступні товари.

Таблиця 2.10 – Контролери застосування

Клас	Опис
AuthenticationController	Клас контролер, який дозволяє аутентифіковатися користувачу у системі.
UserController	Клас контролер, який надає можливість користувачу отримати або змінити інформацію про себе.
CatalogController	Клас контролер для перегляду каталогу товарів.
OrderController	Клас контролер, який дозволяє користувачу створити замовлення, скасувати та переглянути попередні замовлення.
PaymentController	Клас контролер для здійснення оплати замовлення.
TrackingController	Клас контролер, який дозволяє користувачу підписатися на повідомлення про замовлення, а службі доставки відправляти повідомлення про актуальний стан замолвення.
RountingController	Клас контролер Gateway сервісу для проксування повідомлень користувачів до відповідних сервісів.

Таблиця 2.11 – Класи шару View

AuthenticationService	Клас, що містить бізнес-логіку атуентифікації користувача.
UserService	Клас, що містить бізнес-логіку оновлення та зчитування даних про користувача.
OrderService	Клас, що містить бізнес-логіку пов’язану з створенням замовлення (перевірка наявності товару, перевірка статусу оплати, сповіщення Backoffice).
PaymantService	Клас, який абстрагує логіку роботи з різними платіжними системами.
DefaultPaymantService	Клас, який представляє mock реалізацію оплати для сервісу дистанційного замовлення.
CatalogService	Клас, який містить бізнес-логіку для перегляду переліку товарів у каталозі.
TrackingService	Клас, який дозволяє відслідковувати зміну статусу замовлення та сповіщати користувача та Order сервіс про його зміну.
RemoteAuthenticationManager	Клас, який дозволяє аутентифікавути запити користувача до різних сервісів у User сервісу.
SecurityConfiguration	Клас, який налаштовує Spring Security для безпечної взаємодії по мережі та шифрування паролів.

Продовження таблиці 2.10

JWTMetadataRSocketConnectorConfigurer	Клас для додавання мета-даних до JWT токену при взаємодії сервісів у внутрішній мережі.

2.4 Аналіз безпеки даних

Розробка програмного забезпечення для Project Reactor жодним чином не впливає на безпеку. Оператори бібліотеки не зберігають дані та не використовують мережу для передачі даних, тому не потрібно впроваджувати додаткових дій для налаштування безпеки.

Демонстраційний проект сервісу дистанційного замовлення навпаки, проводить обмін даними за мережею та зберігає їх у базах даних. Взаємодія користувача з сервісом відбувається за допомогою протоколів HTTPS та RSocket.

Протокол HTTPS - це розширення протоколу HTTP, який надає підтримку шифрування повідомлень. Для шифрування за замовчуванням використовується протокол рівня представлення TLS. Протокол TLS використовує асиметричне шифрування для обміну ключами і симетричне шифрування для шифрування відправлених повідомлень.

Протокол RSocket - це протокол прикладного рівня і може працювати над різними протоколами. За замовчуванням він використовує для передачі повідомлень протокол WebSocket, який має версію з підтримкою шифрування - WebSocket Secure. WebSocket Secure - протокол прикладного рівня, який використовують для встановлення дуплексного з’єднання. Для шифрування як і для HTTPS використовується протокол рівня представлення TLS.

Для зберігання вразливих даних використовується Spring Security. За допомогою цього фреймворку паролі хєшуються з використанням алгоритму bcrypt. bcrypt - KDF функція для хешування паролей, яка є одним з рекомендованих алгоритмів OWASP організацією.

2.5 Висновки по розділу

У даному розділі були змодельовані бізнес процеси для розроблюваного програмного забезпечення. Бізнес процеси були описані за допомогою діаграм послідовності та BPMN. Далі за висунутими бізнес вимогами була створена та описана архітектура застосунку. Опис архітектуру був здійснений з використанням діаграм розгортання. У розділі конструювання програмного забезпечення були описані технології, деталі реалізації та сутності кінцевого продукту. Це було зроблено з використанням діаграм класів та ER діаграм.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Розробка програмного забезпечення в контексті дипломного проекту, а саме розробка нового способу керування потоком даних у Project-Reactor підпорядковується життєвому циклу програмного забезпечення. У технічному завданні були висунуті функціональні та нефункціональні вимоги до кінцевого продукту. Саме тому необхідно створити тест-план та провести тестування програмного забезпечення для перевірки на відповідність.

Основними цілями створення тест-плану програмного забезпечення є:

а) описати об’єкти, які підлягають тестуванню;

б) визначити основні сценарії , методи та вхідні дані необхідні для перевірки;

в) обрати інструменти, середовище тестування та описати підготовчі дії.

В ході тестування необхідно провести аналіз якості розроблюваного програмного забезпечення. Потрібно перевірити такі характеристики продукту: функціональна придатність, сумісність та ефективність. Для перевірки даних якісних характеристик необхідно розробити:

- тести для перевірки поведінки оператора Prefetch в залежності від встановлених параметрів;
- тест сумісності з іншими операторами Project-Reactor;
- тест продуктивності оператора;
- навантажувальні тести демонстраційного проекту з новим способом керування потоком даних.

Для тестування поведінки оператора необхідно розробити модульні тести за принципом білої скриньки. Це дозволить перевірити оператор на функціональну придатність. Для перевірки оператора на сумісність необхідно

його інтегрувати в існуючі тести TCK у бібліотеці, провівши таким чином димове тестування. Для проведення навантажувальних та бенчмарк тестів необхідно розробити сценарії використання та провести їх на ізольованому середовищі для підвищення точності.

3.2 Опис процесів тестування

Для проведення модульного тестування було обрано фреймворк для тестів JUnit. Даний фреймворк один з найрозповсюдженіших в екосистемі Java. Він надає усю необхідну функціональність для проведення Unit тестування, а також з його використанням вже написані TCK тести в бібліотеці Project-Reactor.

Unit тести необхідні для запуску кожним розробником після внесення змін до модулю. Вони повинні надавати впевненості програмісту, що внесені зміни не змінили поведінку модуля. Саме тому середовище виконання модульних тестів – локальне.

Нижче в Таблицях 3.1 – 3.6 наведені тест плани для перевірки поведінки оператора Prefetch.

Таблиця 3.1 – TC001

Мета тесту	Перевірка можливості злиття з джерелом даних, яке підтримує тільки синхронне злиття.
Початковий стан	Ініціалізовано джерело даних з можливістю синхронного злиття.
Вхідні дані	Тип злиття (ANY, ASYNC, NONE).
Схема проведення тесту	Додати оператор Prefetch до потоку обробки даних. Підписатися на отриманий pipeline. Отримати Subscription та зробити запит на злиття з відповідним типом. Зробити запит на необмежену кількість елементів.

Продовження таблиці 3.1

Очікуваний результат	Не був досягнутий термінальний стан Error. Було встановлено синхронне злиття оператора Prefetch з джерелом, якщо запитаний тип злиття не дорівнює ASYNC. Було встановлено синхронне злиття оператора Prefetch з споживачем, якщо запитаний тип злиття – ANY, та асинхронне злиття, якщо запитаний тип злиття – ASYNC.
Стан програмного продукту після проведення випробувань	Subscriber отримав усі елементи джерела даних та був досягнутий термінальний стан Complete.

Таблиця 3.2 – TC002

Мета тесту	Перевірка можливості злиття з джерелом даних, яке підтримує тільки асинхронне злиття.
Початковий стан	Ініціалізовано джерело даних з можливістю асинхронного злиття.
Вхідні дані	Тип злиття (ANY, ASYNC, NONE).
Схема проведення тесту	Додати оператор Prefetch до потоку обробки даних. Підписатися на отриманий pipeline. Отримати Subscription та зробити запит на злиття з відповідним типом. Зробити запит на необмежену кількість елементів.

Продовження таблиці 3.2

Очікуваний результат	Не був досягнутий термінальний стан Error. Було встановлено асинхронне злиття оператора Prefetch з джерелом. Було встановлено асинхронне злиття оператора Prefetch з споживачем, якщо запитаний тип злиття не дорівнює NONE.
Стан програмного продукту після проведення випробувань	Subscriber отримав усі елементи джерела даних та був досягнутий термінальний стан Complete.

Таблиця 3.3 – TC003

Мета тесту	Перевірка можливості злиття з джерелом даних, яке не підтримує злиття.
Початковий стан	Ініціалізовано джерело даних без можливості злиття.
Вхідні дані	Тип злиття (ANY, ASYNC, NONE)
Схема проведення тесту	Додати оператор Prefetch до потоку обробки даних. Підписатися на отриманий pipeline. Отримати Subscription та зробити запит на злиття з відповідним типом. Зробити запит на необмежену кількість елементів.
Очікуваний результат	Не був досягнутий термінальний стан Error. Не було встановлено злиття оператора Prefetch з джерелом. Було встановлено асинхронне злиття оператора Prefetch з споживачем, якщо запитаний тип злиття не дорівнює NONE.

Продовження таблиці 3.3

Стан програмного продукту після проведення випробувань	Subscriber отримав усі елементи джерела даних та був досягнутий термінальний стан Complete.

Таблиця 3.4 – TC004

Мета тесту	Перевірка підтримки механізму зворотного тиску оператором Prefetch.
Початковий стан	
Вхідні дані	Джерело даних з підтримкою одного з трьох типів злиття (ANY, ASYNC, NONE).
Схема проведення тесту	Додати оператор Prefetch до потоку обробки даних. Підписатися на отриманий pipeline. Отримати Subscription та зробити запит на n елементів (n менше розміру джерела) у джерелі. Отримати n елементів і зробити запит на додаткові n елементів. Повторювати до досягнення термінального стану.
Очікуваний результат	Не був досягнутий термінальний стан Error. Subscriber отримував рівно n елементів після запиту, або менше n з сигналом про термінальний стан. Subscriber отримав усі елементи джерела даних та був досягнутий термінальний стан Complete.
Стан програмного продукту після проведення випробувань	Було досягнуто термінального стану Complete.

Таблиця 3.5 – ТС005

Мета тесту	Перевірка попереднього запиту елементів у відкладеному (LAZY) режимі оператора Prefetch.
Початковий стан	Ініціалізовано джерело даних без можливості злиття.
Вхідні дані	
Схема проведення тесту	Додати оператор Prefetch до потоку обробки даних. Підписатися на отриманий pipeline. Отримати Subscription і перевірити кількість елементів, яку запитав оператор Prefetch у джерела. Зробити запит на деяку кількість елементів (додатну) і перевірити кількість елементів, яку запитав оператор Prefetch у джерела перший раз. Відмінити підписку.
Очікуваний результат	Не був досягнутий термінальний стан Error. Оператор Prefetch до запиту на елементи не робив запитів на елементи у джерела. Після першого запиту на елементи оператор Prefetch зробив запит на кількість елементів, рівних розміру черги (за замовчуванням 256). Після скасування підписки не було отримано елементів.
Стан програмного продукту після проведення випробувань	Джерело даних закінчило відправку даних.

Таблиця 3.6 – ТС006

Мета тесту	Перевірка попереднього запиту елементів у нетерплячому (EAGER) режимі оператора Prefetch.
Початковий стан	Ініціалізовано джерело даних без можливості злиття.
Вхідні дані	
Схема проведення тесту	Додати оператор Prefetch до потоку обробки даних. Підписатися на отриманий pipeline. Отримати Subscription і перевірити кількість елементів, яку запитав оператор Prefetch у джерела. Відмінити підписку.
Очікуваний результат	Не був досягнутий термінальний стан Error. Оператор Prefetch до запиту на елементи зробив запит у джерела на кількість елементів, рівних розміру черги (за замовчуванням 256). Після скасування підписки не було отримано елементів.
Стан програмного продукту після проведення випробувань	Джерело даних закінчило відправку даних.

Для проведення тестів продуктивності оператора Prefetch був обран інструмент JMH (Java Microbenchmark Harness). JMH дозволяє створювати власні тест кейси, а потім виконувати їх та вимірювати час роботи кейсів з заданою точністю з використанням різних методів вимірів. Інструмент дозволяє багаторазово виконувати тест для оптимізації його за допомогою JIT (Just-in-time) компілятора, а також проведення більшої кількості вимірів для отримання більш репрезентативних результатів.

Для визначення продуктивності розробленого оператора Prefetch був обран критерій часу, за який оператор обробляє дані з джерела даних. Для цього необхідно перевірити вплив оператора на швидкодію при параметрах:

- розмір джерела даних (малі та наближені до нескінченних джерела);
- тип злиття з джерелом даних (ASYNC, SYNC, NONE);
- стратегія запиту елементів (по одному та необмежена);
- режим попереднього запиту елементів (LAZY, EAGER).

Для інтерпретації отриманих значень було обрано попередню реалізацію оператора PublishOn, який мав функцію попереднього запиту елементів. Отже комбінацію оператора Prefetch з новою реалізацією PublishOn можна порівняти з попереднім PublishOn.

Для покращення точності результатів тестів продуктивності необхідно виконати ряд додаткових дій. Тести необхідно запускати на ізольованому середовищі з мінімальним впливом на процес вимірів. Для цього була обрана віртуальна машина у GCP (Google Cloud Platform) на базі операційної системи Ubuntu 16.04 з 4 ядрами та 8 Гб RAM. Також, швидкість обробки малих джерел даних дуже мала, що викликає складнощі з отриманням точних результатів. Швидкість даної операції може займати наносекунди. Тому необхідно змінити метод виміру JMH на Throughput. Даний метод вимірює кількість операцій в заданий інтервал часу, тобто кількість оброблених джерел за проміжок часу. Для розроблених тест-планів було обрано проміжок часу у 5 секунд, як компромісне рішення між часом проведення бенчмарків та їх точністю.

Нижче в Таблицях 3.7 – 3.8 наведені тест плани для перевірки продуктивності оператора Prefetch.

Таблиця 3.7 – ТС007

Мета тесту	Перевірка швидкодії оператора Prefetch на джерелах даних різного розміру при всіх типах злиття.
Початковий стан	
Вхідні дані	Розмір джерела даних, тип підтримуваного злиття джерелом даних (ASYNC, SYNC, NONE), стратегія запиту елементів споживачем (One – по одному, Unbound – необмежена).
Схема проведення тесту	Згенерувати джерело даних відповідно розміру. Використати оператор або группу операторів для підтримки заданого типу злиття. Додати оператори Prefetch та PublishOn до потоку обробки даних. Створити аналогічний потік обробки даних, але використовуючи попередню реалізацію PublishOn. Підписатися на отримані pipelines з використанням заданого типу споживача. Провести заміри кількості оброблених джерел за 5 секунд кожним з pipelines.
Очікуваний результат	Жодного разу не було досягнуто термінального стану Error. Було отримано середні значення замірів швидкодії та похибка для встановлених параметрів.
Стан програмного продукту після проведення випробувань	JMH згенерував звіт і завершив роботу.

Таблиця 3.8 – ТС008

Мета тесту	Перевірка впливу режиму попереднього запиту елементів на швидкодію оператора Prefetch.
Початковий стан	
Вхідні дані	Розмір джерела даних, тип підтримуваного злиття джерелом даних (ASYNC, SYNC, NONE), стратегія запиту елементів споживачем (One – по одному, Unbound – необмежена), режим попереднього запиту елементів (LAZY, EAGER).
Схема проведення тесту	Згенерувати джерело даних відповідно розміру. Використати оператор або групу операторів для підтримки заданого типу злиття. Додати оператори Prefetch (з режимом попереднього запиту з параметрів) та PublishOn до потоку обробки даних. Підписатися на отриманий pipeline з використанням заданого типу споживача. Провести заміри кількості оброблених джерел за 5 секунд.
Очікуваний результат	Жодного разу не було досягнуто термінального стану Error. Було отримано середні значення замірів швидкодії та похибка для встановлених параметрів.
Стан програмного продукту після проведення випробувань	JMH згенерував звіт і завершив роботу.

Для перевірки працездатності розробленого сервісу дистанційного замовлення розроблено інтеграційний тест план усіх компонентів системи, який буде перевірятися мануально. План тестування наведено у Таблиці 3.9.

Таблиця 3.9 – ТС009

Мета тесту	Перевірка основного сценарію замовлення користувачем товару.
Початковий стан	Сервіс замовлення запущений та працює в штатному режимі.
Вхідні дані	Ім’я користувача, пароль.
Схема проведення тесту	Користувач аутентфікується у системі. Користувач робить запит на каталог товарів. Користувач обирає довільний товар і додає її у корзину в кількості 1 штук. Користувач створює замовлення. Користувач оплачує замовлення. Користувач відслідковує замовлення.
Очікуваний результат	Користувач отримав підтвердження про опрацювання замовлення, повідомлення про зміну статусу замовлення. Кінцеве отримане повідомлення про успішне завершення замовлення.
Стан програмного продукту після проведення випробувань	Процес доставки замовлення завершено.

Також досить важливим є перевірка впливу нового способу управління потоками даних у бібліотеці Project-Reactor на роботу з джерелами з обмеженнями. Тому для даної задачі необхідно розробити спеціальний тест план, наведений у Таблиці 3.10.

Таблиця 3.10 – ТС010

Мета тесту	Перевірка впливу відкладеного режиму передзапиту елементів на роботу з джерелами з обмеженими ресурсами.
Початковий стан	Сервіс замовлення запущений та працює в штатному режимі.
Вхідні дані	Ім’я користувача, пароль.
Схема проведення тесту	Користувач підключається до системи. Користувач підписується на запит на отримання товарів з каталогу. Користувач не робить запит елементів і одразу відписується.
Очікуваний результат	Сервіс каталогу не зробив передзапит товарів з бази даних.
Стан програмного продукту після проведення випробувань	Користувач відписався від потоку даних, потік даних зупинено.

3.3 Опис контрольного прикладу

Очевидно, що розроблене програмне забезпечення надає більше можливостей керувати потоком даних, але завжди аналогічні зміни впливають на швидкодію або на використання ресурсів. Зазвичай такі рішення є компромісними і мають бути обґрунтованими.

Кожен доданий оператор до потоку обробки також додає накладні витрати на обробку. Це виникає через слідування операторами вимог закладених у Reactive Streams Specification. Зазвичай кожен додатковий оператор витрачає на 5 - 10% часу більше в залежності від його типу.

Покращення керування потоками даних досяглося шляхом додавання нового оператора, а отже при необхідності кількість операторів в pipeline збільшиться. Можна очікувати, що це вплине на швидкодію обробки, тому тест кейси з Таблиць 3.7 та 3.8 були обрані в якості контрольного прикладу.

У тесті для перевірки швидкодії оператора Prefetch на джерелах даних різного розміру при всіх типах злиття використовувалися джерела даних розміром 100 та 100000 елементів. Тест запускався за допомогою JMH 10 разів для прогріву та 20 разів для вимірів. Результати представлені у вигляді гістограм на Рисунках 3.1 – 3.4.

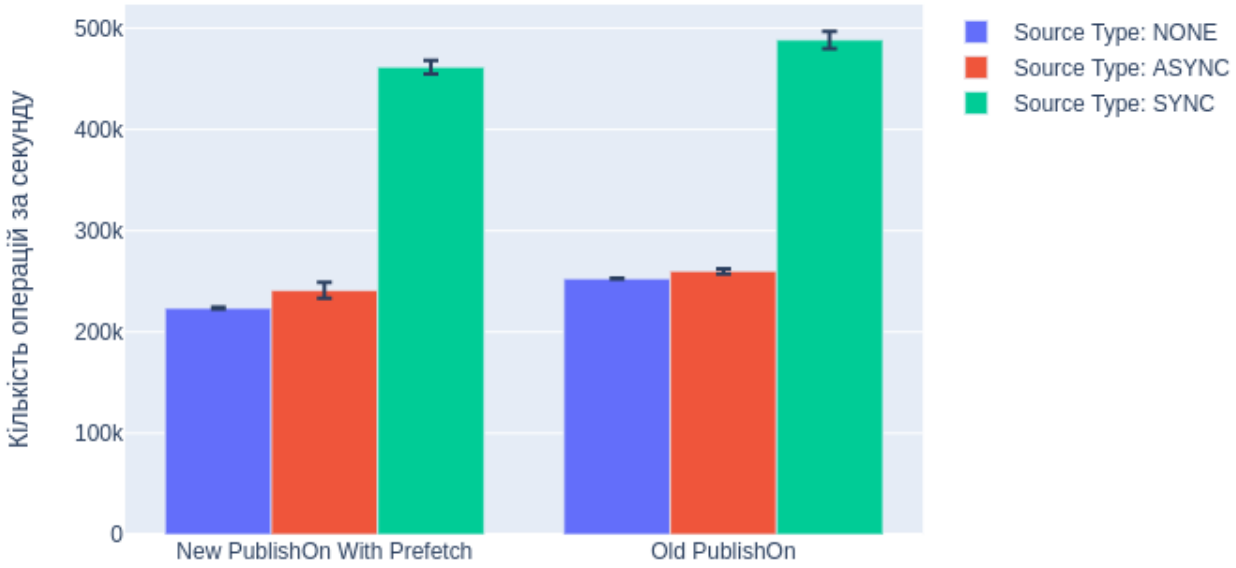


Рисунок 3.1 – Швидкодія операторів на джерелі розміру 100 елементів з стратегією запиту елементів по одному

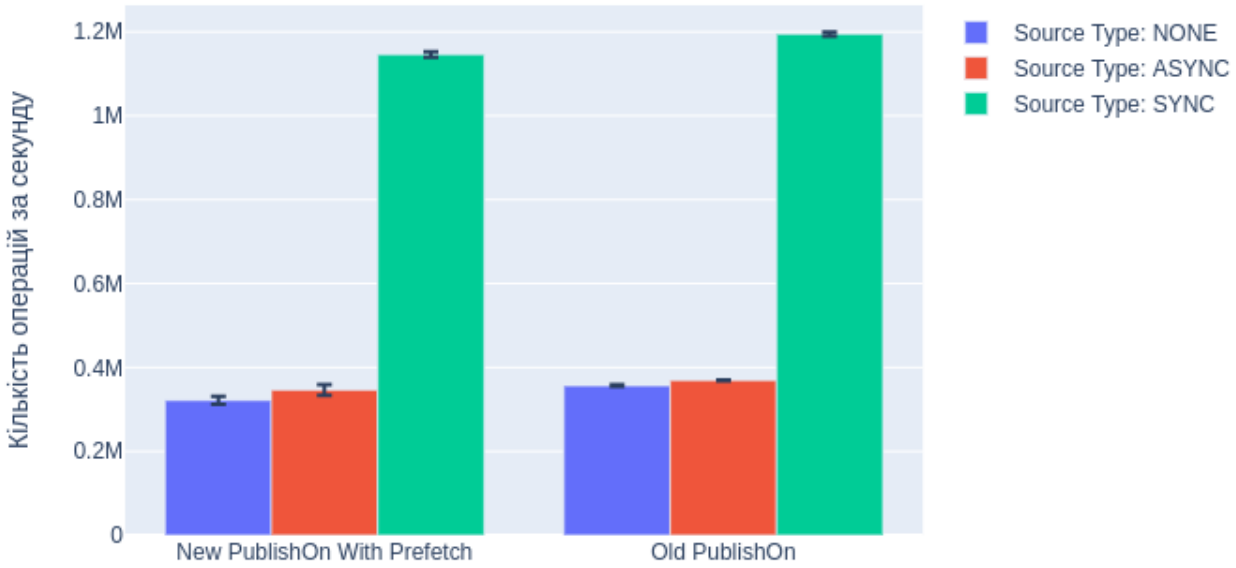


Рисунок 3.2 – Швидкодія операторів на джерелі розміру 100 елементів з запитом необмеженої кількості елементів

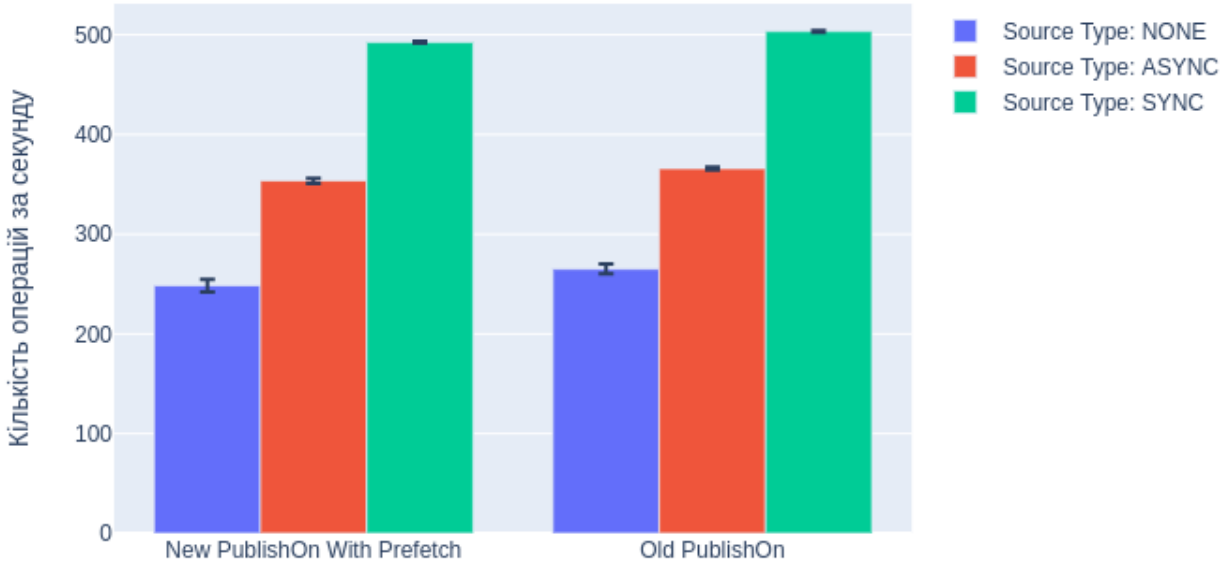


Рисунок 3.3 – Швидкодія операторів на джерелі розміру 100000 елементів з стратегією запиту елементів по одному

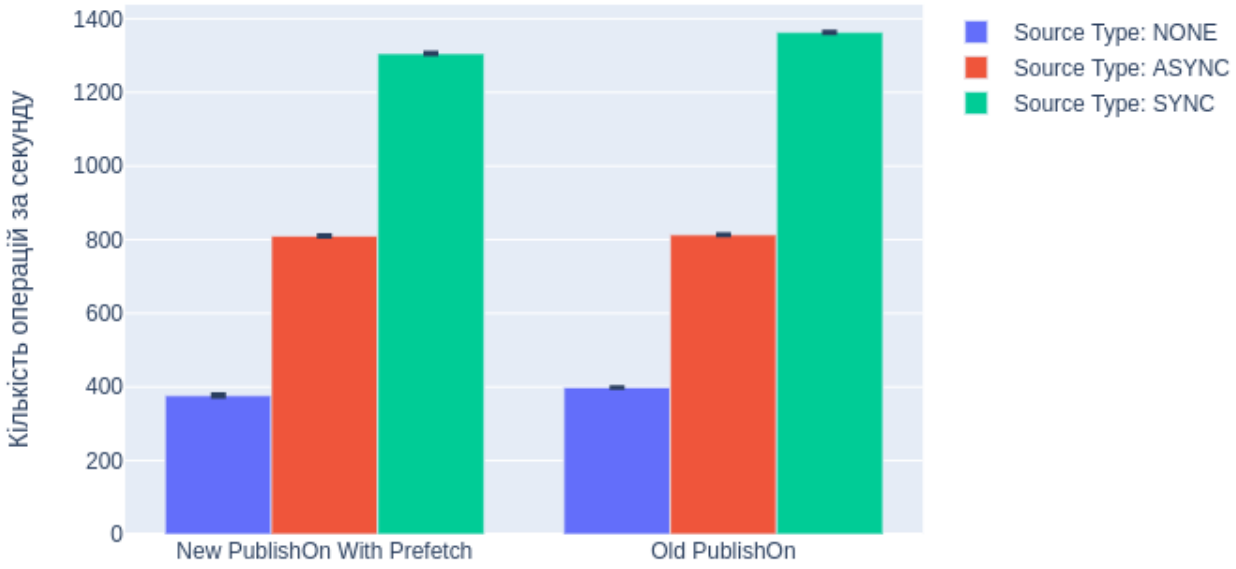


Рисунок 3.4 – Швидкодія операторів на джерелі розміру 100000 елементів з запитом необмеженої кількості елементів

За отриманими результатами вплив оператора Prefetch на малих розмірах джерела даних виявився більш значним ніж при роботі з великими обсягами даних. У найгіршому випадку він склав 11.55% і в середньому він склав 7%. На джерелах даних великого розміру вплив оператора виявився незначним, в середньому він склав близько 3%. Отримані результати можна вважати задовільними незважаючи на значний вплив для джерел малого розміру. Оператор Prefetch реалізує набагато складнішу логіку ніж переважна кількість операторів, а також не має сенсу на колекціях малого розміру. Отже результати можна вважати нормою.

У тесті для перевірки впливу режиму попереднього запиту елементів на швидкодію оператора Prefetch використовувалися джерела даних розміром 100 та 100000 елементів. Тест запускався за допомогою JMH 10 разів для прогріву та 20 разів для вимірів. Результати представлені у вигляді гістограм на Рисунках 3.5 - 3.6.

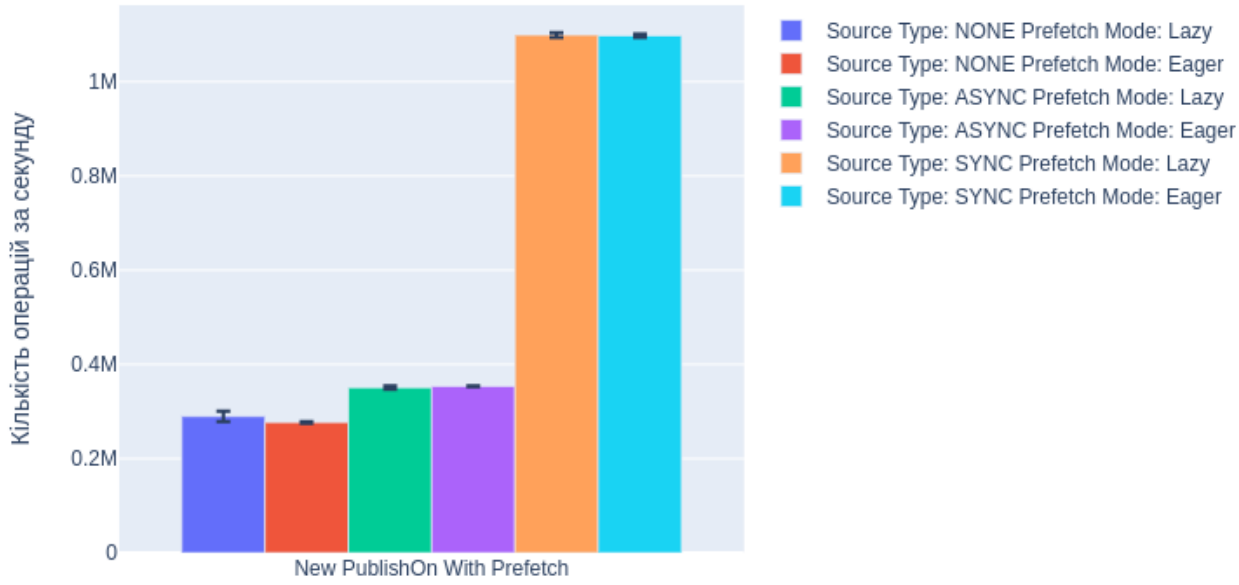


Рисунок 3.5 – Вплив режиму попереднього запиту елементів на швидкодію на джерелі розміром 100 елементів з запитом необмеженої кількості елементів

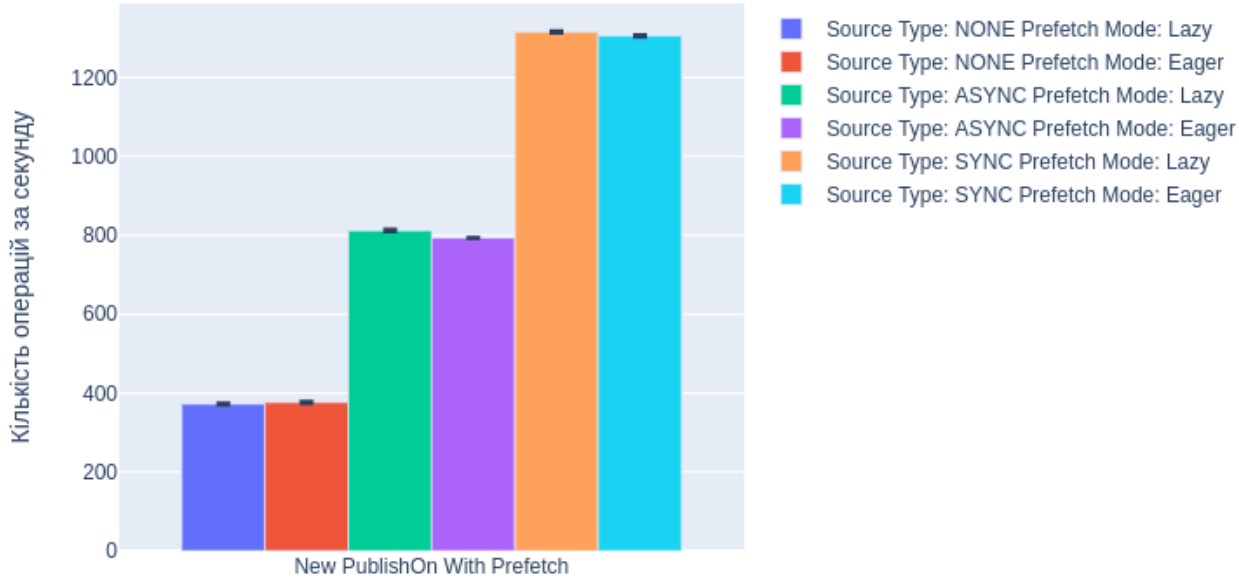


Рисунок 3.6 – Вплив режиму попереднього запиту елементів на швидкодію на джерелі розміром 100000 елементів з запитом необмеженої кількості елементів

За отриманими результатами вплив режиму попереднього запиту елементів на швидкодію виявився незначним на джерелах обох розмірів. У найгіршому випадку він склав 4%. Отримані результати можна вважати задовільними.

3.4 Висновок до розділу

У даному розділі були описані компоненти та функціональність програмного забезпечення, яка потребувала тестування. Були визначені типи та методи тестів для перевірки якісних характеристик продукту. Тести були представлені у вигляді тест-планів з детальним описом вимог до середовища, інструментів та вхідних даних. У контрольному прикладі були розглянуті основоположні тести, які дозволили дати оцінку до кінцевого продукту.

Провівши усі описані тести згідно з тест-планів ТС001-ТС010 було отримано позитивний результат за всіма кейсами. Отже можна вважати, що розроблений програмний продукт відповідає усім поставленим вимогам.

За результатами контрольного прикладу було зроблено висновок, що розроблене програмне забезпечення збільшує можливості керування потоком даних при цьому не маючи істотного впливу на швидкодію.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Розроблюваний оператор prefetch є частиною бібліотеки Project-Reactor, написаної для мови програмування Java. Для того, щоб користуватися можливостями бібліотеки, які включають в себе використання оператора Prefetch, необхідно провести ряд додаткових дій. Спочатку необхідно зібрати бібліотеку, використовуючи систему автоматичного збирання Gradle. Для цього необхідно виконати завдання build, який на виході дасть кінцевий артефакт. Потім для подальшого його використання необхідно імпортувати артефакт у файли проекту. Імпортування бібліотеки відбувається за допомогою менеджера залежностей. В залежності від обраного менеджера можуть змінюватися дії, але InteliJ IDEA дозволяє це зробити за допомогою вкладки “Залежності” для всіх розповсюджених менеджерів, а саме Gradle та Maven.

Для запуску демонстраційного проекта необхідно, щоб було встановлено програмне забезпечення для автоматизації розгортання: Docker версії 1.13 (або вище) та Docker Compose версії 3. Зібрати та запустити демонстраційний проект можна за допомогою виконаної 1 команди у корені проекта – docker-compose up. Дана команда запускає збірку проекта за допомогою Gradle, а потім з сформованих артефактів створює образи Docker. Саме дані образи запускаються та оркеструються Docker Compose. Перегляд логів кожного з мікросервісів можна спостерігати у консолі.

4.2 Робота з програмним забезпеченням

Після успішного імпортування бібліотеки можна користуватися її модулями та операторами, включаючи додатковий розроблений функціонал. Деталі роботи із бібліотекою прописані в офіційній документації бібліотеки Project-Reactor.

Після успішного запуску демонстраційного проекту за допомогою Docker Compose, застосунок очікує підключення клієнтів з використанням протоколу RSocket на порті 8080 за замовчуванням, а для перегляду трасування запитів необхідно перейти на порт 9411 у браузері. У разі виникнення збоїв Docker Compose автоматично виконає перезапуск сервісів.

4.3 Висновки по розділу

У даному розділі був описаний спосіб збірки за допомогою систем автоматичної збірки та підключення нових засобів для управління потоками даних у бібліотеці Project-Reactor у проект за допомогою засобів IDE InteliJ IDEA для подальшого використання.

Також у цьому розділі був розглянутий спосіб розгортання демонстраційного проекту дистанційного замовлення за допомогою засобів оркестрації, а також основні деталі роботи з ним.

ВИСНОВКИ

У ході виконання дипломного проекту було розроблене програмне забезпечення для управління потоками даних в бібліотеці Project-Reactor для мови програмування Java зі створенням додатку дистанційного замовлення для демонстрації роботи розробленого програмного забезпечення.

На початку роботи, проаналізувавши предметну область та існуючі технічні рішення, були висунуті основні вимоги до програмного забезпечення та демонстраційного проекту.

Наступним кором було моделювання бізнес процесів та бізнес вимог. Це дало змогу обрати необхідну архітектуру розроблюваного застосунку, створити її та детально описати.

Після розробки програмного забезпечення для управління потоками даних необхідно було провести тестування компонентів та системи загалом. Для надання оцінки кінцевого продукту були проведені основоположні тести, що були представлені у контрольному прикладі.

Результати тестування показали, що розроблюване програмне забезпечення збільшує можливості керування потоками даних. Притому, без істотного впливу на швидкодії обробки потоку повідомлень.

В кінці роботи описано процеси збірки та розгортання демонстраційного проекту дистанційного замовлення для подальшої демонстрації можливостей розробленого способу управління потоками даних в бібліотеці Project-Reactor.

ПЕРЕЛІК ПОСИЛАНЬ

1) The Reactive Manifesto [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://www.reactivemanifesto.org.

2) Karnok D. Advanced Reactive Java [Електронний ресурс] / David Karnok. – 2017. – Режим доступу до ресурсу: https://akarnokd.blogspot.com.

3) America's Largest Telecommunications Provider Goes Reactive [Електронний ресурс] – Режим доступу до ресурсу: https://www.lightbend.com/case-studies/americas-largest-telecommunications-provider-goes-reactive.

4) Reactive Stream [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.reactive-streams.org.

5) Maldini S. Reactor 3 Reference Guide [Електронний ресурс] / S. Maldini, S. Baslé. – 2021. – Режим доступу до ресурсу: https://projectreactor.io/docs/core/release/reference.

6) Overview of Docker Compose [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/compose.

7) ISO/IEC 25010, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models [Електронний ресурс]. – 2011. – Режим доступу до ресурсу: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en.

8) Web on Reactive Stack [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html.

9) Viglucci K. RSocket Protocol Specification [Електронний ресурс] / Kevin Viglucci. – 2021. – Режим доступу до ресурсу: https://rsocket.io/about/protocol.

10) Paluch M. Spring Data R2DBC - Reference Documentation [Електронний ресурс] / M. Paluch, J. Bryan, S. Cohen. – 2021. – Режим доступу до ресурсу: https://docs.spring.io/spring-data/r2dbc/docs/current/reference/html/#reference.

11) Spring Security Reference [Електронний ресурс] / [B. Alex, L. Taylor, R. Winch та ін.]. – 2021. – Режим доступу до ресурсу: https://docs.spring.io/spring-security/site/docs/current/reference/html5.

12) Spring Cloud Sleuth Reference Documentation [Електронний ресурс] / [A. Cole, S. Gibb, M. Grzejszczak та ін.]. – 2021. – Режим доступу до ресурсу: https://docs.spring.io/spring-cloud-sleuth/docs/current-SNAPSHOT/reference/html/.

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK

Ім'я користувача:
Попенко Володимир Дмитрович

ID перевірки:
1008231000

Дата перевірки:
08.06.2021 15:39:01 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
09.06.2021 18:21:59 EEST

ID користувача:
77149

Назва документа: Mykhailov_bachelor_ip71

Кількість сторінок: 67 Кількість слів: 10438 Кількість символів: 81362 Розмір файлу: 1.26 MB ID файлу: 1008303749

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

4.57%
Схожість

Найбільша схожість: 2.86% з Інтернет-джерелом (https://ela.kpi.ua/bitstream/123456789/39199/1/Shaverskyi_bakalavr.p.

3.94% Джерела з Інтернету	45	Сторінка 69
4.36% Джерела з Бібліотеки	291	Сторінка 70

0.32% Цитат

Цитати	2	Сторінка 71

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	2
Підозріле форматування	12 сторінок

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2021 р.

УПРАВЛІННЯ ПОТОКАМИ ДАНИХ У БІЛІОТЕЦІ PROJECT-REACTOR

Технічне завдання

КПІ.ІП-7120.045490.03.91

“ПОГОДЖЕНО”

Керівник проекту:

_____ О.А Халус

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____Д.І. Михайлов

Київ – 2021 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: «Управління потоками даних у бібліотеці Project-Reactor»

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного забезпечення «Управління потоками даних у бібліотеці Project-Reactor» КПІ.ІП-7120.045490.03.91, котре використовується для удосконалення способів управління компонентами бібліотеки та призначене для розробників неблокуючих систем на базі Project Reactor.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки «Управління потоками даних у бібліотеці Project-Reactor» є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут» ім.Ігоря Сікорського (КПІ ім. Ігоря Сікорського).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання її як частини бібліотеки Project-Reactor для управління асинхронними потоками даних у не блокуючих системах.

Метою розробки є поліпшення способів керування потоками даних у бібліотеці за рахунок розширення можливостей конфігурації потоків повідомлень та зменшення кількості необроблених повідомлень. Для перевірки швидкодії змін розроблюється демонстраційний проект дистанційного замовлення.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача бібліотеки Project Reactor:

– конфігурування розміру асинхронної черги повідомлень;

– конфігурування нижньої границі кількості елементів в черзі, при якому необхідно зробити запит на нові;

– конфігурування режиму попереднього наповнення черги.

4.1.1.2 Для користувача системи дистанційного замовлення:

– аутентифікація;

– перегляд каталогу товарів;

– зберігання обраних товарів у кошику;

– створення замовлення;

– оплата замовлення;

– можливість відслідковувати зміну статусу замовлення.

4.1.2 Розробку виконати на платформі Ubuntu 20.04.

4.1.3 Додаткові вимоги до демонстраційного проекту:

– запровадження системи моніторингу ресурсів та трасування запитів.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.2.3 Забезпечити цілісність інформації в базі даних демонстраційного проекту.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

Не висуваються.

4.3.2 Обслуговування

Вимоги до обслуговування не висуваються.

4.3.3 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

4.4.2 Мінімальна конфігурація технічних засобів:

4.4.2.1. Тип процесору ..... Intel Core i5 2.8GHz.

4.4.2.2. Об‘єм ОЗП ..... 4 ГБ.

4.4.2.3. Об‘єм жорсткого диску ..... 8 ГБ.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Linux.

4.5.2 Програмне забезпечення повинно бути розроблено на мові програмування Java з використанням бібліотеки Project-Reactor, фреймворку Spring та розгортатися у середовищі Docker Compose.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Згенерувати установчу версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1. Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2. Програмне забезпечення повинно мати довідникову систему

5.3. У склад супроводжувальної документації повинні входити наступні документи:

5.3.1. Пояснювальна записка не менше ніж на 70 аркушах формату А4 (без додатків 5.3.2 - 5.3.3).

5.3.2. Технічне завдання.

5.3.3. Програма та методика тестування.

5.4. Графічна частина повинна бути виконана на чотирьох аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1. Схема структурна бізнес-процесів.

5.4.2. Схема структурна послідовності.

5.4.3. Схема структурна класів програмного забезпечення для Project-Reactor.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Стадії та єтапи розробки наведені у таблиці 6.1.

Таблиця 6.1 - Стадії та етапи розробки

№	Назва етапу	Строк	Звітність
1	Вивчення рекомендованої літератури	21.03.2021	
2	Аналіз існуючих методів розв’язання задачі	21.03.2021	
3	Постановка та формалізація задачі	26.03.2021	Технічне завдання
4	Аналіз вимог до програмного забезпечення	26.04.2021	Схема структурна програмного забезпечення та специфікація компонентів
5	Алгоритмізація задачі	02.04.2021	
6	Моделювання програмного забезпечення	16.04.2021	
7	Обґрунтування використовуваних технічних засобів	16.04.2021	
8	Розробка архітектури програмного забезпечення	23.04.2021	Схема структурна компонентів програмного забезпечення
9	Розробка програмного забезпечення	30.04.2021	Тексти програмного забезпечення
10	Налагодження програми	07.05.2021	Програма та методика тестування

Продовження таблиці 6.1

11	Виконання графічних документів	15.05.2021	Графічний матеріал проекту
12	Оформлення пояснювальної записки	15.05.2021	Пояснювальна записка проекту
13	Подання ДП на попередній захист	17.05.2021	
14	Подання ДП рецензенту	01.06.2021	
15	Подання ДП на основний захист	08.06.2021	

7 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

7.1. Види випробувань

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2021 р.

УПРАВЛІННЯ ПОТОКАМИ ДАНИХ У БІЛІОТЕЦІ PROJECT-REACTOR

Програма та методика тестування

КПІ.ІП-7120.045490.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.А. Халус

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____ Д.І. Михайлов

Київ – 2021 року

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробувань є програмне забепечення для розширення можливостей управління потоками даних у бібліотеці Project-Reactor, розроблене на мові програмування Java у вигляді аддону до бібілотеки.

2 МЕТА ТЕСТУВАННЯ

Метою проведення тестування є аналіз якості розробленого програмного забезпечення. Для цього необхідно перевірити:

– функціональну працездатність розроблених компонентів;

– сумісність з іншими компонентами Project-Reactor;

– вплив розробки на швидкодію обробки елементів;

– вплив розробки на кількість оброблених повідомлень демонстраційною системою дистанційного замолвення;

– на відповідність вимогам Технічного завдання.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування поведінки розроблених компонетів для Project-Reactor необхідно розробити модульні тести за принципом білої скриньки. Це дозволить перевірити їх на функціональну придатність. Для перевірки компонетів на сумісність необхідно їх інтегрувати в існуючі тести ТСК у бібліотеці, провівши таким чином димове тестування. Для проведення навантажувальних та бенчмарк тестів необхідно розробити сценарії використання та провести їх на ізольованому середовищі для підвищення точності.

Для проведення оцінки впливу на кількість оброблюваних повідомлень демонстраційним проектом дистанційного замовлення спочатку необхідно перевірити його на відповідність функціональним вимогам за допомогою інтеграційних тестів, які покроють основні кейси. Потім необхідно провести навантажувальне тестування сервісу.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Модульне та інтеграційне тестування виконується за допомогою фремворку JUnit комопнентів для Project-Reactor. Бенчмарк тестування оператора проводиться з викристанням інструмента JMH. Для перевірки працедатності демонстраційного проекта використовується так само фреймворк JUnit для створення інтеграційних тестів. Для трасування запитів використовується розподілена система трасування Zipkin.

Порядок тестування наступний:

– модульне тестування компонентів для Project-Reactor;

– тестування сумісності розроблених компонентів з компонетами Project-Reactor;

– бенчмарк тести компонентів для Project-Reactor;

– інтеграційні тести сервісу дистанційного замовлення;

– навантажувальні тести сервісу дистанційнного змовлення.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2021 р.

УПРАВЛІННЯ ПОТОКАМИ ДАНИХ У БІЛІОТЕЦІ PROJECT-REACTOR

Опис програми

КПІ.ІП-7120.045490.05.13

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.А. Халус

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____ Д.І. Михайлов

Київ – 2021 року

Тексти програмного коду

Управління потоками даних у бібліотеці Project-Reactor

(Наименування програми (документа))

DVD-R

(Вид носія даних)

75 арк, 146 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ - 2021

1.1 Програмне забезпечення для управління потоками даних

```
package reactor.core.publisher;

import java.util.Objects;
import java.util.Queue;
import java.util.concurrent.atomic.AtomicIntegerFieldUpdater;
import java.util.concurrent.atomic.AtomicLongFieldUpdater;
import java.util.function.Supplier;

import org.reactivestreams.Subscriber;
import org.reactivestreams.Subscription;
import reactor.core.CoreSubscriber;
import reactor.core.Exceptions;
import reactor.core.Fuseable;
import reactor.util.annotation.Nullable;

final class FluxPrefetch<T> extends InternalFluxOperator<T, T> implements
Fuseable {

	final int prefetch;

	final int lowTide;

	final Supplier<? extends Queue<T>> queueSupplier;

	final PrefetchMode prefetchMode;

	enum PrefetchMode {
		EAGER, LAZY,
	}

	public FluxPrefetch(Flux<? extends T> source,
			int prefetch,
			int lowTide,
			Supplier<? extends Queue<T>> queueSupplier,
			PrefetchMode prefetchMode) {
		super(source);
		if (prefetch <= 0) {
			throw new IllegalArgumentException("prefetch > 0 required but
it was " + prefetch);
		}
		this.prefetch = prefetch;
		this.lowTide = lowTide;
		this.queueSupplier = Objects.requireNonNull(queueSupplier,
"queueSupplier");
		this.prefetchMode = prefetchMode;
	}

	@Override
	public Object scanUnsafe(Attr key) {
		if (key == Attr.RUN_STYLE) return Attr.RunStyle.SYNC;

		return super.scanUnsafe(key);
```

```
	}

	@Override
	public int getPrefetch() {
		return prefetch;
	}

	@Override
	@SuppressWarnings("unchecked")
	public CoreSubscriber<? super T> subscribeOrReturn(CoreSubscriber<? super
T> actual) {
		if (actual instanceof ConditionalSubscriber) {
			@SuppressWarnings("unchecked") ConditionalSubscriber<? super
T> cs =
					(ConditionalSubscriber<? super T>) actual;
			source.subscribe(new PrefetchConditionalSubscriber<>(cs,
					prefetch,
					lowTide,
					queueSupplier,
					prefetchMode));
			return null;
		}
		return new PrefetchSubscriber<>(actual,
				prefetch,
				lowTide,
				queueSupplier,
				prefetchMode);
	}

	static final class PrefetchSubscriber<T>
			implements QueueSubscription<T>, InnerOperator<T, T> {

		final CoreSubscriber<? super T> actual;

		final int prefetch;

		final int limit;

		final Supplier<? extends Queue<T>> queueSupplier;

		final PrefetchMode prefetchMode;

		Subscription s;

		Queue<T> queue;

		volatile boolean cancelled;

		volatile boolean done;

		Throwable error;

		volatile     long                                          requested;
		@SuppressWarnings("rawtypes")
		static final AtomicLongFieldUpdater<PrefetchSubscriber> REQUESTED =
```

```
	AtomicLongFieldUpdater.newUpdater(PrefetchSubscriber.class, "requested");

		volatile	int 					wip;
		@SuppressWarnings("rawtypes")
		static final AtomicIntegerFieldUpdater<PrefetchSubscriber> WIP =

	AtomicIntegerFieldUpdater.newUpdater(PrefetchSubscriber.class, "wip");

		volatile	int
discardGuard;
		@SuppressWarnings("rawtypes")
		static final AtomicIntegerFieldUpdater<PrefetchSubscriber>
DISCARD_GUARD =

	AtomicIntegerFieldUpdater.newUpdater(PrefetchSubscriber.class,
"discardGuard");

		int sourceMode = -1;

		int outputMode;

		boolean firstRequest = true;

		long produced;

		PrefetchSubscriber(CoreSubscriber<? super T> actual,
				int prefetch,
				int lowTide,
				Supplier<? extends Queue<T>> queueSupplier,
				PrefetchMode prefetchMode) {
			this.actual = actual;
			this.prefetch = prefetch;
			this.limit = Operators.unboundedOrLimit(prefetch, lowTide);
			this.queueSupplier = queueSupplier;
			this.prefetchMode = prefetchMode;

			REQUESTED.lazySet(this, Long.MIN_VALUE);
		}

		@Override
		public void onSubscribe(Subscription s) {
			if (Operators.validate(this.s, s)) {
				this.s = s;

				WIP.lazySet(this, 1);
				actual.onSubscribe(this);

				if (cancelled) {
					if (sourceMode == Fuseable.ASYNC) {
						// delegates discarding to the queue holder
to ensure there is no racing on draining from the SpScQueue
						queue.clear();
					}
					else {
```

```
                                    // discard MUST be happening only and only
if there is no racing on elements consumption
                                    // which is guaranteed by the WIP guard here
                                    Operators.onDiscardQueueWithClear(queue,
actual.currentContext(), null);
                              }
                              return;
                        }

                        // sourceMode == -1 if downstream was not calling
requestFusion
                        if (sourceMode == -1) {
                              // check if upstream if
                              // fuseable so we can fuse at least with the
upstream
                              if (s instanceof QueueSubscription) {
                                    @SuppressWarnings("unchecked")
QueueSubscription<T> fusion =
                                                (QueueSubscription<T>) s;
                                    int fusionMode =
fusion.requestFusion(Fuseable.ANY);

                                    if (fusionMode == Fuseable.SYNC) {
                                          sourceMode = Fuseable.SYNC;
                                          queue = fusion;
                                          done = true;
                                          firstRequest = false;

                                          // check if downstream requested
something
                                          if (this.wip == 1 &&
WIP.addAndGet(this, -1) == 0) {
                                                // exit if nothing was requested
yet
                                                return;
                                          }

                                          drainSync();
                                          return;
                                    }
                                    if (fusionMode == Fuseable.ASYNC) {
                                          sourceMode = Fuseable.ASYNC;
                                          queue = fusion;
                                          firstRequest = false;

                                          // check if something was requested or
delivered in the
                                          // meantime
                                          if (wip == 1 && WIP.addAndGet(this, -
1) == 0) {
                                                // exit if non of the mentioned
has happened yet
                                                return;
                                          }
```

```
                                    drainAsync();
                                    return;
                              }
                        }

                        sourceMode = Fuseable.NONE;
                        queue = queueSupplier.get();
                        if (prefetchMode == PrefetchMode.EAGER) {
                              firstRequest = false;

      s.request(Operators.unboundedOrPrefetch(prefetch));
                        }

                        if (wip == 1 && WIP.addAndGet(this, -1) == 0) {
                              return;
                        }

                        if (prefetchMode == PrefetchMode.LAZY) {
                              firstRequest = false;

      s.request(Operators.unboundedOrPrefetch(prefetch));
                        }

                        drainAsync();
                  }
                  else if (sourceMode == Fuseable.NONE && prefetchMode ==
PrefetchMode.EAGER) {
                        firstRequest = false;

      s.request(Operators.unboundedOrPrefetch(prefetch));
                  }
            }
      }

      @Override
      public void onNext(T t) {
            if (sourceMode == Fuseable.ASYNC) {
                  if (outputMode == Fuseable.ASYNC) {
                        actual.onNext(null);
                  }
                  else {
                        drain(null);
                  }
                  return;
            }

            if (done) {
                  Operators.onNextDropped(t, actual.currentContext());
                  return;
            }

            if (cancelled) {
                  Operators.onDiscard(t, actual.currentContext());
                  return;
            }
```

```
                if (!queue.offer(t)) {
                        Operators.onDiscard(t, actual.currentContext());
                        error = Operators.onOperatorError(s,

        Exceptions.failWithOverflow(Exceptions.BACKPRESSURE_ERROR_QUEUE_FULL),
                                        t,
                                        actual.currentContext());
                        done = true;
                }

                if (outputMode == Fuseable.ASYNC) {
                        actual.onNext(null);
                        return;
                }
                drain(t);
        }

        @Override
        public void onError(Throwable err) {
                if (done) {
                        Operators.onErrorDropped(err, actual.currentContext());
                        return;
                }
                error = err;
                done = true;

                if (sourceMode != Fuseable.NONE && sourceMode == outputMode) {
                        actual.onError(err);
                }
                else {
                        drain(null);
                }
        }

        @Override
        public void onComplete() {
                if (done) {
                        return;
                }
                done = true;

                if (sourceMode != Fuseable.NONE && sourceMode == outputMode) {
                        actual.onComplete();
                }
                else {
                        drain(null);
                }
        }

        @Override
        public void request(long n) {
                if (Operators.validate(n)) {
                        long previousState =
    Operators.addCapFromMinValue(REQUESTED, this, n);
```

```
                        // check if this is the first request from the
downstream
                        if (previousState == Long.MIN_VALUE) {
                              // check the mode and fusion mode
                              if (prefetchMode == PrefetchMode.LAZY) {
                                    if (sourceMode == -1) {
                                          // check if sourceMode was set
                                          if (WIP.getAndIncrement(this) == 0) {
                                                if (sourceMode == Fuseable.NONE)
{
                                                      firstRequest = false;

      s.request(Operators.unboundedOrPrefetch(this.prefetch));
                                                      drainAsync();
                                                }
                                                else if (sourceMode ==
Fuseable.ASYNC) {
                                                      drainAsync();
                                                }
                                                else {
                                                      drainSync();
                                                }
                                          }
                                          return;
                                    }
                                    else if (sourceMode == Fuseable.NONE) {
                                          firstRequest = false;

      s.request(Operators.unboundedOrPrefetch(this.prefetch));
                                    }
                              }
                        }

                        drain(null);
                  }
            }

            @Override
            public void cancel() {
                  if (cancelled) {
                        return;
                  }

                  cancelled = true;
                  s.cancel();

                  if (WIP.getAndIncrement(this) == 0) {
                        if (sourceMode == Fuseable.ASYNC) {
                              // delegates discarding to the queue holder to
ensure there is no racing on draining from the SpScQueue
                              queue.clear();
                        }
                        else if (outputMode == Fuseable.NONE) {
```

```
                        // discard MUST be happening only and only if
there is no racing on elements consumption
                        // which is guaranteed by the WIP guard here in
case non-fused output
                        Operators.onDiscardQueueWithClear(queue,
actual.currentContext(), null);
                    }
                }
            }

            protected void drainAsync() {
                final Subscriber<? super T> a = actual;
                final Queue<T> queue = this.queue;
                final int sourceMode = this.sourceMode;

                long emitted = produced;
                int missed = 1;
                for (; ; ) {
                    long requested = this.requested & Long.MAX_VALUE;

                    while (emitted != requested) {
                        T value;
                        try {
                            value = queue.poll();
                        }
                        catch (Throwable err) {
                            Exceptions.throwIfFatal(err);
                            s.cancel();
                            if (sourceMode == Fuseable.ASYNC) {
                                // delegates discarding to the queue
holder to ensure there is no racing on draining from the SpScQueue
                                queue.clear();
                            }
                            else {
                                // discard MUST be happening only and
only if there is no racing on elements consumption
                                // which is guaranteed by the WIP
guard here

      Operators.onDiscardQueueWithClear(queue, actual.currentContext(), null);
                            }

                            a.onError(Operators.onOperatorError(err,
actual.currentContext()));
                            return;
                        }

                        boolean empty = value == null;

                        if (checkTerminated(done, empty, value)) {
                            return;
                        }

                        if (empty) {
                            break;
```

```
                        }

                        a.onNext(value);
                        emitted++;

                        if (emitted == limit && sourceMode ==
Fuseable.NONE) {
                              if (requested != Long.MAX_VALUE) {
                                    requested = REQUESTED.addAndGet(this,
-emitted);
                              }
                              s.request(emitted);
                              emitted = 0L;
                        }
                  }

                  if (emitted == requested && checkTerminated(done,
queue.isEmpty(), null)) {
                        return;
                  }

                  int w = wip;
                  if (missed == w) {
                        produced = emitted;
                        missed = WIP.addAndGet(this, -missed);
                        if (missed == 0) {
                              break;
                        }
                  }
                  else {
                        missed = w;
                  }
            }
      }

      protected void drainOutput() {
            int missed = 1;
            for (; ; ) {
                  if (cancelled) {
                        // We are the holder of the queue, but we still
have to perform discarding under the guarded block
                        // to prevent any racing done by downstream
                        clear();
                        return;
                  }

                  actual.onNext(null);

                  if (done) {
                        Throwable err = error;
                        if (err != null) {
                              actual.onError(err);
                        }
                        else {
                              actual.onComplete();
```

```
                        }
                        return;
                    }

                    missed = WIP.addAndGet(this, -missed);
                    if (missed == 0) {
                        break;
                    }
                }
            }

            protected void drainSync() {
                final Subscriber<? super T> a = actual;
                final Queue<T> queue = this.queue;

                long emitted = produced;
                int missed = 1;
                for (; ; ) {
                    long requested = this.requested;

                    while (emitted != requested) {
                        T value;
                        try {
                            value = queue.poll();
                        }
                        catch (Throwable err) {
                            a.onError(Operators.onOperatorError(s, err,
actual.currentContext()));
                            return;
                        }

                        if (cancelled) {
                            Operators.onDiscard(value,
actual.currentContext());
                            Operators.onDiscardQueueWithClear(queue,
actual.currentContext(), null);
                            return;
                        }
                        if (value == null) {
                            a.onComplete();
                            return;
                        }

                        a.onNext(value);
                        emitted++;
                    }

                    if (cancelled) {
                        Operators.onDiscardQueueWithClear(queue,
actual.currentContext(), null);
                        return;
                    }

                    if (queue.isEmpty()) {
                        a.onComplete();
```

```
                        return;
                    }

                    int w = wip;
                    if (missed == w) {
                        produced = emitted;
                        missed = WIP.addAndGet(this, -missed);
                        if (missed == 0) {
                            break;
                        }
                    }
                    else {
                        missed = w;
                    }
                }

            }

            protected void drain(@Nullable Object dataSignal) {
                if (WIP.getAndIncrement(this) != 0) {
                    if (cancelled) {
                        if (sourceMode == Fuseable.ASYNC) {
                            // delegates discarding to the queue holder
to ensure there is no racing on draining from the SpScQueue
                            queue.clear();
                        }
                        else {
                            // discard given dataSignal since no more is
enqueued (spec guarantees serialised onXXX calls)
                            Operators.onDiscard(dataSignal,
actual.currentContext());
                        }
                    }
                    return;
                }

                if (outputMode != Fuseable.NONE) {
                    drainOutput();
                }
                else if (sourceMode == Fuseable.SYNC) {
                    drainSync();
                }
                else {
                    drainAsync();
                }
            }

            boolean checkTerminated(boolean done, boolean empty, @Nullable T
value) {
                if (cancelled) {
                    Operators.onDiscard(value, actual.currentContext());
                    if (sourceMode == Fuseable.ASYNC) {
                        // delegates discarding to the queue holder to
ensure there is no racing on draining from the SpScQueue
                        queue.clear();
```

```
                        }
                        else {
                              // discard MUST be happening only and only if
there is no racing on elements consumption
                              // which is guaranteed by the WIP guard here
                              Operators.onDiscardQueueWithClear(queue,
actual.currentContext(), null);
                        }
                        return true;
                  }
                  if (done) {
                        Throwable err = error;
                        if (err != null) {
                              Operators.onDiscard(value,
actual.currentContext());
                              if (sourceMode == Fuseable.ASYNC) {
                                    // delegates discarding to the queue holder
to ensure there is no racing on draining from the SpScQueue
                                    queue.clear();
                              }
                              else {
                                    // discard MUST be happening only and only
if there is no racing on elements consumption
                                    // which is guaranteed by the WIP guard here
                                    Operators.onDiscardQueueWithClear(queue,
actual.currentContext(), null);
                              }
                              actual.onError(err);
                              return true;
                        }
                        else if (empty) {
                              actual.onComplete();
                              return true;
                        }
                  }

                  return false;
            }

            @Override
            public void clear() {
                  if (sourceMode == Fuseable.ASYNC) {
                        queue.clear();
                        return;
                  }

                  // use guard on the queue instance as the best way to ensure
there is no racing on draining
                  // the call to this method must be done only during the ASYNC
fusion so all the callers will be waiting
                  // this should not be performance costly with the assumption
the cancel is rare operation
                  if (DISCARD_GUARD.getAndIncrement(this) != 0) {
                        return;
                  }
```

```
                int missed = 1;

                for (; ; ) {
                        Operators.onDiscardQueueWithClear(queue,
actual.currentContext(), null);

                        int dg = discardGuard;
                        if (missed == dg) {
                                missed = DISCARD_GUARD.addAndGet(this, -missed);
                                if (missed == 0) {
                                        break;
                                }
                        }
                        else {
                                missed = dg;
                        }
                }
        }

        @Override
        public boolean isEmpty() {
                return queue.isEmpty();
        }

        @Override
        @Nullable
        public T poll() {
                if (sourceMode == Fuseable.NONE) {
                        if (firstRequest && prefetchMode == PrefetchMode.LAZY) {
                                firstRequest = false;

    s.request(Operators.unboundedOrPrefetch(this.prefetch));
                        }

                        T value = queue.poll();
                        if (value != null) {
                                long p = produced + 1;
                                if (p == limit) {
                                        produced = 0;
                                        s.request(p);
                                }
                                else {
                                        produced = p;
                                }
                        }
                        return value;
                }
                else {
                        return queue.poll();
                }
        }

        public int requestFusion(int requestedMode) {
                if (s instanceof QueueSubscription) {
```

```
                        @SuppressWarnings("unchecked") QueueSubscription<T>
fusion =
                                    (QueueSubscription<T>) s;
                        int fusionMode = fusion.requestFusion(requestedMode);

                        if (fusionMode == Fuseable.SYNC) {
                              sourceMode = fusionMode;
                              outputMode = fusionMode;
                              queue = fusion;
                              done = true;
                              firstRequest = false;

                              WIP.lazySet(this, 0);
                              return fusionMode;
                        }
                        else if (fusionMode == Fuseable.ASYNC) {
                              sourceMode = fusionMode;
                              outputMode = fusionMode;
                              queue = fusion;
                              firstRequest = false;

                              WIP.lazySet(this, 0);
                              return fusionMode;
                        }
                  }

                  int fusionMode;
                  sourceMode = Fuseable.NONE;
                  queue = queueSupplier.get();

                  if ((requestedMode & Fuseable.ASYNC) != 0) {
                        outputMode = Fuseable.ASYNC;
                        fusionMode = Fuseable.ASYNC;
                  }
                  else {
                        fusionMode = Fuseable.NONE;
                  }

                  WIP.lazySet(this, 0);
                  return fusionMode;
            }

            @Override
            public int size() {
                  return queue.size();
            }

            @Override
            public CoreSubscriber<? super T> actual() {
                  return actual;
            }

            @Override
            public Object scanUnsafe(Attr key) {
                  if (key == Attr.PARENT) return s;
```

```
                if (key == Attr.CANCELLED) return cancelled;
                if (key == Attr.ERROR) return error;
                if (key == Attr.TERMINATED) return done;
                if (key == Attr.PREFETCH) return prefetch;
                if (key == Attr.REQUESTED_FROM_DOWNSTREAM) return requested;
                if (key == Attr.BUFFERED) return queue != null ? queue.size()
: 0;
                if (key == Attr.RUN_STYLE) return Attr.RunStyle.SYNC;

                return InnerOperator.super.scanUnsafe(key);
            }
        }

        static final class PrefetchConditionalSubscriber<T> extends
PrefetchSubscriber<T> {

            final ConditionalSubscriber<? super T> actual;

            long consumed;

            PrefetchConditionalSubscriber(ConditionalSubscriber<? super T>
actual,
                        int prefetch,
                        int lowTide,
                        Supplier<? extends Queue<T>> queueSupplier,
                        PrefetchMode prefetchMode) {
                this.actual = actual;
                this.prefetch = prefetch;
                this.limit = Operators.unboundedOrLimit(prefetch, lowTide);
                this.queueSupplier = queueSupplier;
                this.prefetchMode = prefetchMode;

                REQUESTED.lazySet(this, Long.MIN_VALUE);
            }

            @Override
            protected void drainAsync() {
                final ConditionalSubscriber<? super T> a = actual;
                final Queue<T> queue = this.queue;
                final int sourceMode = this.sourceMode;

                long emitted = produced;
                int missed = 1;
                long polled = consumed;
                for (; ; ) {
                    long requested = this.requested & Long.MAX_VALUE;

                    while (emitted != requested) {
                        T value;
                        try {
                            value = queue.poll();
                        }
                        catch (Throwable err) {
                            Exceptions.throwIfFatal(err);
                            s.cancel();
```

```
                                    if (sourceMode == Fuseable.ASYNC) {
                                          // delegates discarding to the queue
holder to ensure there is no racing on draining from the SpScQueue
                                          queue.clear();
                                    }
                                    else {
                                          // discard MUST be happening only and
only if there is no racing on elements consumption
                                          // which is guaranteed by the WIP
guard here

      Operators.onDiscardQueueWithClear(queue, actual.currentContext(), null);
                                    }

                                    a.onError(Operators.onOperatorError(err,
actual.currentContext()));
                                    return;
                              }

                              boolean empty = value == null;

                              if (checkTerminated(done, empty, value)) {
                                    return;
                              }

                              if (empty) {
                                    break;
                              }

                              if (a.tryOnNext(value)) {
                                    emitted++;
                              }

                              polled++;

                              if (polled == limit && sourceMode ==
Fuseable.NONE) {
                                    s.request(polled);
                                    polled = 0L;
                              }
                        }

                        if (emitted == requested && checkTerminated(done,
queue.isEmpty(), null)) {
                              return;
                        }

                        int w = wip;
                        if (missed == w) {
                              produced = emitted;
                              consumed = polled;
                              missed = WIP.addAndGet(this, -missed);
                              if (missed == 0) {
                                    break;
                              }
```

```
                        }
                        else {
                              missed = w;
                        }
                  }
            }

            @Override
            protected void drainOutput() {
                  int missed = 1;
                  for (; ; ) {
                        if (cancelled) {
                              // We are the holder of the queue, but we still
have to perform discarding under the guarded block
                              // to prevent any racing done by downstream
                              clear();
                              return;
                        }

                        actual.onNext(null);

                        if (done) {
                              Throwable err = error;
                              if (err != null) {
                                    actual.onError(err);
                              }
                              else {
                                    actual.onComplete();
                              }
                              return;
                        }

                        missed = WIP.addAndGet(this, -missed);
                        if (missed == 0) {
                              break;
                        }
                  }
            }

            @Override
            protected void drainSync() {
                  final ConditionalSubscriber<? super T> a = actual;
                  final Queue<T> queue = this.queue;

                  long emitted = produced;
                  int missed = 1;
                  for (; ; ) {
                        long requested = this.requested;

                        while (emitted != requested) {
                              T value;
                              try {
                                    value = queue.poll();
                              }
                              catch (Throwable err) {
```

```
                                    a.onError(Operators.onOperatorError(s, err,
actual.currentContext()));
                                    return;
                              }

                              if (cancelled) {
                                    Operators.onDiscard(value,
actual.currentContext());
                                    Operators.onDiscardQueueWithClear(queue,
actual.currentContext(), null);
                                    return;
                              }
                              if (value == null) {
                                    a.onComplete();
                                    return;
                              }

                              if (a.tryOnNext(value)) {
                                    emitted++;
                              }
                        }

                        if (cancelled) {
                              Operators.onDiscardQueueWithClear(queue,
actual.currentContext(), null);
                              return;
                        }

                        if (queue.isEmpty()) {
                              a.onComplete();
                              return;
                        }

                        int w = wip;
                        if (missed == w) {
                              produced = emitted;
                              missed = WIP.addAndGet(this, -missed);
                              if (missed == 0) {
                                    break;
                              }
                        }
                        else {
                              missed = w;
                        }
                  }

            }

            @Override
            protected void drain(@Nullable Object dataSignal) {
                  if (WIP.getAndIncrement(this) != 0) {
                        if (cancelled) {
                              if (sourceMode == Fuseable.ASYNC) {
                                    // delegates discarding to the queue holder
to ensure there is no racing on draining from the SpScQueue
```

```
                                    queue.clear();
                              }
                              else {
                                    // discard given dataSignal since no more is
enqueued (spec guarantees serialised onXXX calls)
                                    Operators.onDiscard(dataSignal,
actual.currentContext());
                              }
                        }
                        return;
                  }

                  if (outputMode != Fuseable.NONE) {
                        drainOutput();
                  }
                  else if (sourceMode == Fuseable.SYNC) {
                        drainSync();
                  }
                  else {
                        drainAsync();
                  }
            }

            @Override
            @Nullable
            public T poll() {
                  if (sourceMode == Fuseable.NONE) {
                        if (firstRequest && prefetchMode == PrefetchMode.LAZY) {
                              firstRequest = false;

      s.request(Operators.unboundedOrPrefetch(this.prefetch));
                        }

                        T value = queue.poll();
                        if (value != null) {
                              long c = consumed + 1;
                              if (c == limit) {
                                    consumed = 0;
                                    s.request(c);
                              }
                              else {
                                    consumed = c;
                              }
                        }
                        return value;
                  }
                  else {
                        return queue.poll();
                  }
            }
      }
}
```

```
package reactor;

import java.util.Arrays;
import java.util.HashMap;
import java.util.concurrent.TimeUnit;
import java.util.function.Function;

import org.openjdk.jmh.annotations.Benchmark;
import org.openjdk.jmh.annotations.BenchmarkMode;
import org.openjdk.jmh.annotations.Fork;
import org.openjdk.jmh.annotations.Measurement;
import org.openjdk.jmh.annotations.Mode;
import org.openjdk.jmh.annotations.OutputTimeUnit;
import org.openjdk.jmh.annotations.Param;
import org.openjdk.jmh.annotations.Scope;
import org.openjdk.jmh.annotations.Setup;
import org.openjdk.jmh.annotations.State;
import org.openjdk.jmh.annotations.Warmup;
import org.openjdk.jmh.infra.Blackhole;
import org.reactivestreams.Subscriber;
import org.reactivestreams.Subscription;
import reactor.core.CoreSubscriber;
import reactor.core.publisher.Flux;
import reactor.core.scheduler.Schedulers;

@BenchmarkMode(Mode.Throughput)
@Warmup(iterations = 10, time = 5)
@Measurement(iterations = 20, time = 5)
@OutputTimeUnit(TimeUnit.SECONDS)
@Fork(2)
@State(Scope.Benchmark)
public class FluxPrefetchOverheadBenchmark {

      @Param({"10", "100", "1000", "100000"})
      public int sourceSize;

      @Param({"NONE", "SYNC", "ASYNC"})
      public String sourceType;

      @Param({"true", "false"})
      public boolean prefetchMode;

      @Param({"One", "Unbound"})
      public String subscriberType;

      Flux<Integer> publishOnOldFlux;
      Flux<Integer> publishOnFlux;
      Flux<Integer> publishOnWithPrefetchFlux;
      Flux<Integer> prefetchFlux;

      Function<Blackhole, Subscriber<Integer>> subscriberSupplier;

      @Setup
      public void setup() {
            Flux<Integer> dataSource = getDataSource(sourceSize, sourceType);
```

```
            publishOnOldFlux = dataSource.publishOnOld(Schedulers.immediate());
            publishOnFlux = dataSource.publishOn(Schedulers.immediate());
            publishOnWithPrefetchFlux = dataSource.prefetch(prefetchMode)

.publishOn(Schedulers.immediate());
            prefetchFlux = dataSource.prefetch(prefetchMode);

            subscriberSupplier = getSubscriberSupplier(subscriberType);
      }

      private Flux<Integer> getDataSource(int size, String sourceType) {
            Integer[] array = sources.get(size);

            Flux<Integer> publisher = Flux.fromArray(array);

            switch (sourceType) {
                  case "ASYNC":
                        return publisher.onBackpressureBuffer();
                  case "SYNC":
                        return publisher;
                  case "NONE":
                  default:
                        return publisher.hide();
            }
      }

      private Function<Blackhole, Subscriber<Integer>>
getSubscriberSupplier(String subscriberType) {
            return (Blackhole bh) -> {
                  switch (subscriberType) {
                        case ("One"):
                              return new OneByOneSubscriber(bh);
                        case ("Unbound"):
                        default:
                              return new UnboundedSubscriber(bh);
                  }
            };
      }

      @Benchmark
      public void oldPublishOnPerformance(Blackhole bh) {
            Subscriber<Integer> s = subscriberSupplier.apply(bh);
            publishOnOldFlux.subscribe(s);
      }

      @Benchmark
      public void newPublishOnPerformance(Blackhole bh) {
            Subscriber<Integer> s = subscriberSupplier.apply(bh);
            publishOnFlux.subscribe(s);
      }

      @Benchmark
      public void newPublishOnWithPrefetchFlux(Blackhole bh) {
            Subscriber<Integer> s = subscriberSupplier.apply(bh);
```

```
        publishOnWithPrefetchFlux.subscribe(s);
    }

    @Benchmark
    public void prefetchFlux(Blackhole bh) {
        Subscriber<Integer> s = subscriberSupplier.apply(bh);
        prefetchFlux.subscribe(s);
    }

    static int[] sourceSizes = new int[]{10, 100, 1000, 100000};

    static HashMap<Integer, Integer[]> sources = new HashMap<>();

    static {
        for (int size : sourceSizes) {
            createSource(size);
        }
    }

    static private void createSource(int sourceSize) {
        Integer[] array = new Integer[sourceSize];
        Arrays.fill(array, 666);
        sources.put(sourceSize, array);
    }

    static class OneByOneSubscriber implements CoreSubscriber<Integer> {

        final Blackhole blackhole;

        Subscription subscription;

        OneByOneSubscriber(Blackhole blackhole) {
            this.blackhole = blackhole;
        }

        @Override
        public void onSubscribe(Subscription s) {
            subscription = s;
            subscription.request(1);
        }

        @Override
        public void onNext(Integer v) {
            blackhole.consume(v);
            subscription.request(1);
        }

        @Override
        public void onError(Throwable t) {
            blackhole.consume(t);
        }

        @Override
        public void onComplete() {
            blackhole.consume(true);
```

```
            }
        }

        static class UnboundedSubscriber implements CoreSubscriber<Integer> {

            final Blackhole blackhole;

            Subscription subscription;

            UnboundedSubscriber(Blackhole blackhole) {
                this.blackhole = blackhole;
            }

            @Override
            public void onSubscribe(Subscription s) {
                subscription = s;
                subscription.request(Long.MAX_VALUE);
            }

            @Override
            public void onNext(Integer v) {
                blackhole.consume(v);
            }

            @Override
            public void onError(Throwable t) {
                blackhole.consume(t);
            }

            @Override
            public void onComplete() {
                blackhole.consume(true);
            }
        }
    }
```

```
package reactor.core.publisher;

import java.util.Objects;
import java.util.Queue;
import java.util.concurrent.RejectedExecutionException;
import java.util.concurrent.atomic.AtomicIntegerFieldUpdater;
import java.util.concurrent.atomic.AtomicLongFieldUpdater;
import java.util.concurrent.atomic.AtomicReferenceFieldUpdater;

import org.reactivestreams.Subscriber;
import org.reactivestreams.Subscription;
import reactor.core.CoreSubscriber;
import reactor.core.Exceptions;
import reactor.core.Fuseable;
import reactor.core.scheduler.Scheduler;
```

```
import reactor.core.scheduler.Scheduler.Worker;
import reactor.util.annotation.Nullable;

/**
 * Emits events on a different thread specified by a scheduler callback.
 *
 * @param <T> the value type
 * @see <a href="https://github.com/reactor/reactive-streams-commons">Reactive-
Streams-Commons</a>
 */
final class FluxPublishOn<T> extends InternalFluxOperator<T, T> implements
Fuseable {

	final Scheduler scheduler;

	final boolean delayError;

	FluxPublishOn(Flux<T> source, Scheduler scheduler, boolean delayError) {
		super(source);
		this.scheduler = Objects.requireNonNull(scheduler, "scheduler");
		this.delayError = delayError;
	}

	@Override
	public Object scanUnsafe(Attr key) {
		if (key == Attr.RUN_ON) return scheduler;
		if (key == Attr.RUN_STYLE) return Attr.RunStyle.ASYNC;

		return super.scanUnsafe(key);
	}

	@Override
	@SuppressWarnings("unchecked")
	public CoreSubscriber<? super T> subscribeOrReturn(CoreSubscriber<? super
T> actual) {
		Worker worker = Objects.requireNonNull(scheduler.createWorker(),
				"The scheduler returned a null worker");

		if (actual instanceof ConditionalSubscriber) {
			ConditionalSubscriber<? super T> cs = (ConditionalSubscriber<?
super T>) actual;
			source.subscribe(new PublishOnConditionalSubscriber<>(cs,
					scheduler,
					worker,
					delayError));
			return null;
		}
		return new PublishOnSubscriber<>(actual, scheduler, worker,
delayError);
	}

	static final class PublishOnSubscriber<T>
			implements QueueSubscription<T>, Runnable, InnerOperator<T, T>
{
```

```
            final CoreSubscriber<? super T> actual;

            final Scheduler scheduler;

            final Worker worker;

            final boolean delayError;

            Subscription s;

            QueueSubscription<T> qs;

            volatile     T
value;
            @SuppressWarnings("rawtypes")
            static final AtomicReferenceFieldUpdater<PublishOnSubscriber,
Object> VALUE =

      AtomicReferenceFieldUpdater.newUpdater(PublishOnSubscriber.class,
                                  Object.class,
                                  "value");

            volatile boolean cancelled;

            volatile boolean done;

            Throwable error;

            volatile     long                                           requested;
            @SuppressWarnings("rawtypes")
            static final AtomicLongFieldUpdater<PublishOnSubscriber> REQUESTED
=

      AtomicLongFieldUpdater.newUpdater(PublishOnSubscriber.class,
"requested");

            volatile     int                                              wip;
            @SuppressWarnings("rawtypes")
            static final AtomicIntegerFieldUpdater<PublishOnSubscriber> WIP =

      AtomicIntegerFieldUpdater.newUpdater(PublishOnSubscriber.class, "wip");

            volatile     int
discardGuard;
            @SuppressWarnings("rawtypes")
            static final AtomicIntegerFieldUpdater<PublishOnSubscriber>
DISCARD_GUARD =

      AtomicIntegerFieldUpdater.newUpdater(PublishOnSubscriber.class,
"discardGuard");

            int sourceMode;

            long produced;
```

```
		boolean outputFused;

		PublishOnSubscriber(CoreSubscriber<? super T> actual,
				Scheduler scheduler,
				Worker worker,
				boolean delayError) {
			this.actual = actual;
			this.worker = worker;
			this.scheduler = scheduler;
			this.delayError = delayError;

			REQUESTED.lazySet(this, Long.MIN_VALUE);
		}

		@Override
		public void onSubscribe(Subscription s) {
			if (Operators.validate(this.s, s)) {
				this.s = s;

				if (s instanceof QueueSubscription) {
					@SuppressWarnings("unchecked")
QueueSubscription<T> f =
							(QueueSubscription<T>) s;

					int mode = f.requestFusion(Fuseable.ANY |
Fuseable.THREAD_BARRIER);

					if (mode == Fuseable.SYNC) {
						sourceMode = Fuseable.SYNC;
						qs = f;
						done = true;
					}
					else if (mode == Fuseable.ASYNC) {
						sourceMode = Fuseable.ASYNC;
						qs = f;
					}
				}
				actual.onSubscribe(this);
			}
		}

		@Override
		public void onNext(T t) {
			if (sourceMode == Fuseable.ASYNC) {
				trySchedule(this, null, null /* t always null */);
				return;
			}

			if (done) {
				Operators.onNextDropped(t, actual.currentContext());
				return;
			}

			if (cancelled) {
				Operators.onDiscard(t, actual.currentContext());
```

```
                        return;
                  }

                  if (!VALUE.compareAndSet(this, null, t)) {
                        Operators.onDiscard(t, actual.currentContext());
                        error = Operators.onOperatorError(s,

      Exceptions.failWithOverflow(Exceptions.BACKPRESSURE_ERROR_QUEUE_FULL),
                                    t,
                                    actual.currentContext());
                        done = true;
                  }
                  trySchedule(this, null, t);
            }

            @Override
            public void onError(Throwable t) {
                  if (done) {
                        Operators.onErrorDropped(t, actual.currentContext());
                        return;
                  }
                  error = t;
                  done = true;
                  trySchedule(null, t, null);
            }

            @Override
            public void onComplete() {
                  if (done) {
                        return;
                  }
                  done = true;
                  // WIP also guards, no competing onNext
                  trySchedule(null, null, null);
            }

            @Override
            public void request(long n) {
                  if (Operators.validate(n)) {
                        long previousState =
Operators.addCapFromMinValue(REQUESTED, this, n);

                        // check if this is the first request from the
downstream
                        if (previousState == Long.MIN_VALUE && this.sourceMode
== Fuseable.NONE) {
                              this.s.request(1);
                        }

                        // WIP also guards during request and onError is
possible
                        trySchedule(this, null, null);
                  }
            }
```

```
            @Override
            public void cancel() {
                  if (cancelled) {
                        return;
                  }

                  cancelled = true;
                  s.cancel();
                  worker.dispose();

                  if (WIP.getAndIncrement(this) == 0) {
                        if (sourceMode == Fuseable.ASYNC) {
                              // delegates discarding to the queue holder to
ensure there is no racing on draining from the SpScQueue
                              qs.clear();
                        }
                        else if (!outputFused) {
                              if (sourceMode == Fuseable.SYNC) {
                                    // discard MUST be happening only and only
if there is no racing on elements consumption
                                    // which is guaranteed by the WIP guard here
in case non-fused output
                                    Operators.onDiscardQueueWithClear(qs,
actual.currentContext(), null);
                              }
                              if (sourceMode == Fuseable.NONE) {
                                    @SuppressWarnings("unchecked") T v = (T)
VALUE.getAndSet(this, null);
                                    Operators.onDiscard(v,
actual.currentContext());
                              }
                        }
                  }
            }

            void trySchedule(@Nullable Subscription subscription,
                        @Nullable Throwable suppressed,
                        @Nullable Object dataSignal) {
                  if (WIP.getAndIncrement(this) != 0) {
                        if (cancelled) {
                              if (sourceMode == ASYNC) {
                                    // delegates discarding to the queue holder
to ensure there is no racing on draining from the SpScQueue
                                    qs.clear();
                              }
                              else {
                                    // discard given dataSignal since no more is
enqueued (spec guarantees serialised onXXX calls)
                                    Operators.onDiscard(dataSignal,
actual.currentContext());
                              }
                        }
                        return;
                  }
```

```
                try {
                        worker.schedule(this);
                }
                catch (RejectedExecutionException ree) {
                        if (sourceMode == ASYNC) {
                                // delegates discarding to the queue holder to
ensure there is no racing on draining from the SpScQueue
                                qs.clear();
                        }
                        else if (outputFused) {
                                // We are the holder of the queue, but we still
have to perform discarding under the guarded block
                                // to prevent any racing done by downstream
                                clear();
                        }
                        else {
                                if (sourceMode == Fuseable.SYNC) {
                                        // discard MUST be happening only and only
if there is no racing on elements consumption
                                        // which is guaranteed by the WIP guard here
in case non-fused output
                                        Operators.onDiscardQueueWithClear(qs,
actual.currentContext(), null);
                                }
                                if (sourceMode == Fuseable.NONE) {
                                        @SuppressWarnings("unchecked") T v = (T)
VALUE.getAndSet(this, null);
                                        Operators.onDiscard(v,
actual.currentContext());
                                }
                        }
                        actual.onError(Operators.onRejectedExecution(ree,
                                        subscription,
                                        suppressed,
                                        dataSignal,
                                        actual.currentContext()));
                }
        }

        void runAsync() {
                final Subscriber<? super T> a = actual;
                final Queue<T> queue = qs;

                long emitted = produced;
                int missed = 1;
                for (; ; ) {
                        long requested = this.requested & Long.MAX_VALUE;

                        while (emitted != requested) {
                        boolean d = done;
                        T v;
                        try {
                                v = queue.poll();
                        }
                        catch (Throwable ex) {
```

```
                                    Exceptions.throwIfFatal(ex);
                                    s.cancel();
                                    // delegates discarding to the queue holder
to ensure there is no racing on draining from the SpScQueue
                                    qs.clear();
                                    doError(a, Operators.onOperatorError(ex,
actual.currentContext()));
                                    return;
                              }

                              boolean empty = v == null;

                              if (checkTerminated(d, empty, a, v)) {
                                    return;
                              }

                              if (empty) {
                                    break;
                              }

                              a.onNext(v);
                              emitted++;
                        }

                        if (emitted == requested && checkTerminated(done,
queue.isEmpty(), a, null)) {
                              return;
                        }

                        int w = wip;
                        if (missed == w) {
                              produced = emitted;
                              missed = WIP.addAndGet(this, -missed);
                              if (missed == 0) {
                                    break;
                              }
                        }
                        else {
                              missed = w;
                        }
                  }
            }

            void runBackfused() {
                  int missed = 1;
                  for (; ; ) {
                        if (cancelled) {
                              // We are the holder of the queue, but we still
have to perform discarding under the guarded block
                              // to prevent any racing done by downstream
                              clear();
                              return;
                        }
                        boolean d = done;
```

```
                        actual.onNext(null);

                        if (d) {
                              Throwable e = error;
                              if (e != null) {
                                    doError(actual, e);
                              }
                              else {
                                    doComplete(actual);
                              }
                              return;
                        }

                        missed = WIP.addAndGet(this, -missed);
                        if (missed == 0) {
                              break;
                        }
                  }
            }

            void runOneByOne() {
                  final Subscriber<? super T> a = actual;

                  long emitted = produced;
                  int missed = 1;
                  for (; ; ) {
                        long requested = this.requested;

                        while (emitted != requested) {
                              boolean d = done;

                              @SuppressWarnings("unchecked") T v = (T)
VALUE.getAndSet(this, null);

                              boolean empty = v == null;

                              if (checkTerminated(d, empty, a, v)) {
                                    return;
                              }

                              if (empty) {
                                    break;
                              }
                              a.onNext(v);
                              emitted++;
                              s.request(1);
                        }

                        if (emitted == requested && checkTerminated(done, value
== null, a, null)) {
                              return;
                        }

                        int w = wip;
                        if (missed == w) {
```

```
                        produced = emitted;
                        missed = WIP.addAndGet(this, -missed);
                        if (missed == 0) {
                              break;
                        }
                  }
                  else {
                        missed = w;
                  }
            }
      }

      void runSync() {
            final Subscriber<? super T> a = actual;
            final Queue<T> queue = qs;

            long emitted = produced;
            int missed = 1;
            for (; ; ) {
                  long requested = this.requested;

                  while (emitted != requested) {
                        T v;
                        try {
                              v = queue.poll();
                        }
                        catch (Throwable ex) {
                              doError(a, Operators.onOperatorError(s, ex,
actual.currentContext()));
                              return;
                        }

                        if (cancelled) {
                              Operators.onDiscard(v,
actual.currentContext());
                              Operators.onDiscardQueueWithClear(queue,
actual.currentContext(), null);
                              return;
                        }
                        if (v == null) {
                              doComplete(a);
                              return;
                        }

                        a.onNext(v);
                        emitted++;
                  }

                  if (cancelled) {
                        Operators.onDiscardQueueWithClear(queue,
actual.currentContext(), null);
                        return;
                  }

                  if (queue.isEmpty()) {
```

```
                        doComplete(a);
                        return;
                    }

                    int w = wip;
                    if (missed == w) {
                        produced = emitted;
                        missed = WIP.addAndGet(this, -missed);
                        if (missed == 0) {
                            break;
                        }
                    }
                    else {
                        missed = w;
                    }
                }
            }

            @Override
            public void run() {
                if (outputFused) {
                    runBackfused();
                }
                else if (sourceMode == Fuseable.SYNC) {
                    runSync();
                }
                else if (sourceMode == Fuseable.ASYNC) {
                    runAsync();
                }
                else {
                    runOneByOne();
                }
            }

            void doComplete(Subscriber<?> a) {
                a.onComplete();
                worker.dispose();
            }

            void doError(Subscriber<?> a, Throwable e) {
                try {
                    a.onError(e);
                }
                finally {
                    worker.dispose();
                }
            }

            boolean checkTerminated(boolean d, boolean empty, Subscriber<?> a,
@Nullable T v) {
                if (cancelled) {
                    Operators.onDiscard(v, actual.currentContext());
                    if (sourceMode == ASYNC) {
                        // delegates discarding to the queue holder to
ensure there is no racing on draining from the SpScQueue
```

```
                        qs.clear();
                    }
                    return true;
                }
                if (d) {
                    if (delayError) {
                        if (empty) {
                            Throwable e = error;
                            if (e != null) {
                                doError(a, e);
                            }
                            else {
                                doComplete(a);
                            }
                            return true;
                        }
                    }
                    else {
                        Throwable e = error;
                        if (e != null) {
                            Operators.onDiscard(v,
actual.currentContext());
                            if (sourceMode == ASYNC) {
                                // delegates discarding to the queue
holder to ensure there is no racing on draining from the SpScQueue
                                qs.clear();
                            }
                            doError(a, e);
                            return true;
                        }
                        else if (empty) {
                            doComplete(a);
                            return true;
                        }
                    }
                }

                return false;
            }

            @Override
            public void clear() {
                if (sourceMode == Fuseable.ASYNC) {
                    qs.clear();
                    return;
                }

                // use guard on the queue instance as the best way to ensure
there is no racing on draining
                // the call to this method must be done only during the ASYNC
fusion so all the callers will be waiting
                // this should not be performance costly with the assumption
the cancel is rare operation
                if (DISCARD_GUARD.getAndIncrement(this) != 0) {
                    return;
```

```
                }

                int missed = 1;

                for (; ; ) {
                      Operators.onDiscardQueueWithClear(qs,
actual.currentContext(), null);

                      int dg = discardGuard;
                      if (missed == dg) {
                            missed = DISCARD_GUARD.addAndGet(this, -missed);
                            if (missed == 0) {
                                  break;
                            }
                      }
                      else {
                            missed = dg;
                      }
                }
          }

          @Override
          public boolean isEmpty() {
                return qs == null || qs.isEmpty();
          }

          @Override
          public T poll() {
                return qs == null ? null : qs.poll();
          }

          @Override
          public int requestFusion(int requestedMode) {
                if (sourceMode != Fuseable.NONE && (requestedMode &
Fuseable.ASYNC) != 0) {
                      outputFused = true;
                      return Fuseable.ASYNC;
                }
                return Fuseable.NONE;
          }

          @Override
          public int size() {
                return qs == null ? 0 : qs.size();
          }

          @Override
          public CoreSubscriber<? super T> actual() {
                return actual;
          }

          @Override
          @Nullable
          public Object scanUnsafe(Attr key) {
                if (key == Attr.REQUESTED_FROM_DOWNSTREAM) return requested;
```

```
                    if (key == Attr.PARENT) return s;
                    if (key == Attr.CANCELLED) return cancelled;
                    if (key == Attr.TERMINATED) return done;
                    if (key == Attr.ERROR) return error;
                    if (key == Attr.DELAY_ERROR) return delayError;
                    if (key == Attr.RUN_ON) return worker;
                    if (key == Attr.RUN_STYLE) return Attr.RunStyle.ASYNC;

                    return InnerOperator.super.scanUnsafe(key);
              }
        }

        static final class PublishOnConditionalSubscriber<T>
                    implements QueueSubscription<T>, Runnable, InnerOperator<T, T>
{

              final ConditionalSubscriber<? super T> actual;

              final Scheduler scheduler;

              final Worker worker;

              final boolean delayError;

              Subscription s;

              QueueSubscription<T> qs;

              volatile     T
value;
              @SuppressWarnings("rawtypes")
              static final
AtomicReferenceFieldUpdater<PublishOnConditionalSubscriber, Object>



                     VALUE =

        AtomicReferenceFieldUpdater.newUpdater(PublishOnConditionalSubscriber.cla
ss,
                                        Object.class,
                                        "value");

              volatile boolean cancelled;

              volatile boolean done;

              Throwable error;

              volatile     long
requested;
              @SuppressWarnings("rawtypes")
              static final AtomicLongFieldUpdater<PublishOnConditionalSubscriber>
REQUESTED =
```

```
      AtomicLongFieldUpdater.newUpdater(PublishOnConditionalSubscriber.class,
                                        "requested");

            volatile     int
wip;
            @SuppressWarnings("rawtypes")
            static final
AtomicIntegerFieldUpdater<PublishOnConditionalSubscriber> WIP =

      AtomicIntegerFieldUpdater.newUpdater(PublishOnConditionalSubscriber.class
, "wip");

            volatile     int
discardGuard;
            @SuppressWarnings("rawtypes")
            static final
AtomicIntegerFieldUpdater<PublishOnConditionalSubscriber> DISCARD_GUARD =

      AtomicIntegerFieldUpdater.newUpdater(PublishOnConditionalSubscriber.class
,
                                        "discardGuard");

            int sourceMode;

            long produced;

            boolean outputFused;

            PublishOnConditionalSubscriber(ConditionalSubscriber<? super T>
actual,
                        Scheduler scheduler,
                        Worker worker,
                        boolean delayError) {
                  this.actual = actual;
                  this.worker = worker;
                  this.scheduler = scheduler;
                  this.delayError = delayError;

                  REQUESTED.lazySet(this, Long.MIN_VALUE);
            }

            @Override
            public void onSubscribe(Subscription s) {
                  if (Operators.validate(this.s, s)) {
                        this.s = s;

                        if (s instanceof QueueSubscription) {
                              @SuppressWarnings("unchecked")
QueueSubscription<T> f =
                                          (QueueSubscription<T>) s;

                              int m = f.requestFusion(Fuseable.ANY |
Fuseable.THREAD_BARRIER);
```

```
                        if (m == Fuseable.SYNC) {
                              sourceMode = Fuseable.SYNC;
                              qs = f;
                              done = true;
                        }
                        if (m == Fuseable.ASYNC) {
                              sourceMode = Fuseable.ASYNC;
                              qs = f;
                        }
                  }
                  actual.onSubscribe(this);
            }
      }

      @Override
      public void onNext(T t) {
            if (sourceMode == ASYNC) {
                  trySchedule(this, null, null /* t always null */);
                  return;
            }

            if (done) {
                  Operators.onNextDropped(t, actual.currentContext());
                  return;
            }

            if (cancelled) {
                  Operators.onDiscard(t, actual.currentContext());
                  return;
            }

            if (!VALUE.compareAndSet(this, null, t)) {
                  Operators.onDiscard(t, actual.currentContext());
                  error = Operators.onOperatorError(s,

Exceptions.failWithOverflow(Exceptions.BACKPRESSURE_ERROR_QUEUE_FULL),
                              t,
                              actual.currentContext());
                  done = true;
            }
            trySchedule(this, null, t);
      }

      @Override
      public void onError(Throwable t) {
            if (done) {
                  Operators.onErrorDropped(t, actual.currentContext());
                  return;
            }
            error = t;
            done = true;
            trySchedule(null, t, null);
      }

      @Override
```

```
            public void onComplete() {
                  if (done) {
                        return;
                  }
                  done = true;
                  // WIP also guards, no competing onNext
                  trySchedule(null, null, null);
            }

            @Override
            public void request(long n) {
                  if (Operators.validate(n)) {
                        long previousState =
Operators.addCapFromMinValue(REQUESTED, this, n);

                        // check if this is the first request from the
downstream
                        if (previousState == Long.MIN_VALUE && this.sourceMode
== Fuseable.NONE) {
                              this.s.request(1);
                        }

                        // WIP also guards during request and onError is
possible
                        trySchedule(this, null, null);
                  }
            }

            @Override
            public void cancel() {
                  if (cancelled) {
                        return;
                  }

                  cancelled = true;
                  s.cancel();
                  worker.dispose();

                  if (WIP.getAndIncrement(this) == 0) {
                        if (sourceMode == Fuseable.ASYNC) {
                              // delegates discarding to the queue holder to
ensure there is no racing on draining from the SpScQueue
                              qs.clear();
                        }
                        else if (!outputFused) {
                              if (sourceMode == Fuseable.SYNC) {
                                    // discard MUST be happening only and only
if there is no racing on elements consumption
                                    // which is guaranteed by the WIP guard here
in case non-fused output
                                    Operators.onDiscardQueueWithClear(qs,
actual.currentContext(), null);
                              }
                              if (sourceMode == Fuseable.NONE) {
```

```
                                    @SuppressWarnings("unchecked") T v = (T)
VALUE.getAndSet(this, null);
                                    Operators.onDiscard(v,
actual.currentContext());
                              }
                        }
                  }
            }

            void trySchedule(@Nullable Subscription subscription,
                        @Nullable Throwable suppressed,
                        @Nullable Object dataSignal) {
                  if (WIP.getAndIncrement(this) != 0) {
                        if (cancelled) {
                              if (sourceMode == ASYNC) {
                                    // delegates discarding to the queue holder
to ensure there is no racing on draining from the SpScQueue
                                    qs.clear();
                              }
                              else {
                                    // discard given dataSignal since no more is
enqueued (spec guarantees serialised onXXX calls)
                                    Operators.onDiscard(dataSignal,
actual.currentContext());
                              }
                        }
                        return;
                  }

                  try {
                        worker.schedule(this);
                  }
                  catch (RejectedExecutionException ree) {
                        if (sourceMode == ASYNC) {
                              // delegates discarding to the queue holder to
ensure there is no racing on draining from the SpScQueue
                              qs.clear();
                        }
                        else if (outputFused) {
                              // We are the holder of the queue, but we still
have to perform discarding under the guarded block
                              // to prevent any racing done by downstream
                              clear();
                        }
                        else {
                              if (sourceMode == Fuseable.SYNC) {
                                    // discard MUST be happening only and only
if there is no racing on elements consumption
                                    // which is guaranteed by the WIP guard here
in case non-fused output
                                    Operators.onDiscardQueueWithClear(qs,
actual.currentContext(), null);
                              }
                              if (sourceMode == Fuseable.NONE) {
```

```
                                    @SuppressWarnings("unchecked") T v = (T)
VALUE.getAndSet(this, null);
                                    Operators.onDiscard(v,
actual.currentContext());
                              }
                        }
                        actual.onError(Operators.onRejectedExecution(ree,
                                    subscription,
                                    suppressed,
                                    dataSignal,
                                    actual.currentContext()));
                  }
            }

            void runAsync() {
                  final ConditionalSubscriber<? super T> a = actual;
                  final Queue<T> queue = qs;

                  long emitted = produced;
                  int missed = 1;
                  for (; ; ) {
                        long dropped = 0;
                        long requested = this.requested & Long.MAX_VALUE;

                        while (emitted != requested) {
                              boolean d = done;
                              T v;
                              try {
                                    v = queue.poll();
                              }
                              catch (Throwable ex) {
                                    Exceptions.throwIfFatal(ex);
                                    s.cancel();
                                    // delegates discarding to the queue holder
to ensure there is no racing on draining from the SpScQueue
                                    queue.clear();
                                    doError(a, Operators.onOperatorError(ex,
actual.currentContext()));
                                    return;
                              }
                              boolean empty = v == null;

                              if (checkTerminated(d, empty, a, v)) {
                                    return;
                              }

                              if (empty) {
                                    if (dropped != 0) {
                                          s.request(dropped);
                                    }
                                    break;
                              }

                              if (a.tryOnNext(v)) {
                                    emitted++;
```

```
                                    if (dropped != 0) {
                                          s.request(dropped);
                                          dropped = 0;
                                    }
                              }
                              else {
                                    dropped++;
                              }
                        }

                        if (emitted == requested && checkTerminated(done,
queue.isEmpty(), a, null)) {
                              return;
                        }

                        int w = wip;
                        if (missed == w) {
                              produced = emitted;
                              missed = WIP.addAndGet(this, -missed);
                              if (missed == 0) {
                                    break;
                              }
                        }
                        else {
                              missed = w;
                        }
                  }

            }

            void runBackfused() {
                  int missed = 1;
                  for (; ; ) {
                        if (cancelled) {
                              // We are the holder of the queue, but we still
have to perform discarding under the guarded block
                              // to prevent any racing done by downstream
                              clear();
                              return;
                        }
                        boolean d = done;

                        actual.onNext(null);

                        if (d) {
                              Throwable e = error;
                              if (e != null) {
                                    doError(actual, e);
                              }
                              else {
                                    doComplete(actual);
                              }
                              return;
                        }
```

```
                        missed = WIP.addAndGet(this, -missed);
                        if (missed == 0) {
                              break;
                        }
                  }
            }

            void runOneByOne() {
                  final ConditionalSubscriber<? super T> a = actual;

                  long emitted = produced;
                  int missed = 1;
                  for (; ; ) {
                        long requested = this.requested;

                        while (emitted != requested) {
                              boolean d = done;

                              @SuppressWarnings("unchecked") T v = (T)
VALUE.getAndSet(this, null);

                              boolean empty = v == null;

                              if (checkTerminated(d, empty, a, v)) {
                                    return;
                              }

                              if (empty) {
                                    break;
                              }

                              if (a.tryOnNext(v)) {
                                    emitted++;
                              }

                              s.request(1);
                        }

                        if (emitted == requested && checkTerminated(done, value
== null, a, null)) {
                              return;
                        }

                        int w = wip;
                        if (missed == w) {
                              produced = emitted;
                              missed = WIP.addAndGet(this, -missed);
                              if (missed == 0) {
                                    break;
                              }
                        }
                        else {
                              missed = w;
                        }
                  }
```

```
		}

		void runSync() {
			final ConditionalSubscriber<? super T> a = actual;
			final Queue<T> queue = qs;

			long emitted = produced;
			int missed = 1;
			for (; ; ) {
				long requested = this.requested;

				while (emitted != requested) {
					T v;
					try {
						v = queue.poll();
					}
					catch (Throwable ex) {
						doError(a, Operators.onOperatorError(s, ex,
actual.currentContext()));
						return;
					}

					if (cancelled) {
						Operators.onDiscard(v,
actual.currentContext());
						Operators.onDiscardQueueWithClear(queue,
actual.currentContext(), null);
						return;
					}
					if (v == null) {
						doComplete(a);
						return;
					}

					if (a.tryOnNext(v)) {
						emitted++;
					}
				}

				if (cancelled) {
					Operators.onDiscardQueueWithClear(queue,
actual.currentContext(), null);
					return;
				}

				if (queue.isEmpty()) {
					doComplete(a);
					return;
				}

				int w = wip;
				if (missed == w) {
					produced = emitted;
					missed = WIP.addAndGet(this, -missed);
					if (missed == 0) {
```

```
                                    break;
                              }
                        }
                        else {
                              missed = w;
                        }
                  }
            }

            @Override
            public void run() {
                  if (outputFused) {
                        runBackfused();
                  }
                  else if (sourceMode == Fuseable.SYNC) {
                        runSync();
                  }
                  else if (sourceMode == Fuseable.ASYNC) {
                        runAsync();
                  }
                  else {
                        runOneByOne();
                  }
            }

            void doComplete(Subscriber<?> a) {
                  a.onComplete();
                  worker.dispose();
            }

            void doError(Subscriber<?> a, Throwable e) {
                  try {
                        a.onError(e);
                  }
                  finally {
                        worker.dispose();
                  }
            }

            boolean checkTerminated(boolean d, boolean empty, Subscriber<?> a,
@Nullable T v) {
                  if (cancelled) {
                        Operators.onDiscard(v, actual.currentContext());
                        if (sourceMode == ASYNC) {
                              // delegates discarding to the queue holder to
ensure there is no racing on draining from the SpScQueue
                              qs.clear();
                        }
                        return true;
                  }
                  if (d) {
                        if (delayError) {
                              if (empty) {
                                    Throwable e = error;
                                    if (e != null) {
```

```
                                        doError(a, e);
                                    }
                                    else {
                                        doComplete(a);
                                    }
                                    return true;
                                }
                            }
                            else {
                                Throwable e = error;
                                if (e != null) {
                                    Operators.onDiscard(v,
actual.currentContext());
                                    if (sourceMode == ASYNC) {
                                        // delegates discarding to the queue
holder to ensure there is no racing on draining from the SpScQueue
                                        qs.clear();
                                    }
                                    doError(a, e);
                                    return true;
                                }
                                else if (empty) {
                                    doComplete(a);
                                    return true;
                                }
                            }
                        }

                        return false;
                    }

                    @Override
                    public void clear() {
                        if (sourceMode == Fuseable.ASYNC) {
                            qs.clear();
                            return;
                        }

                        // use guard on the queue instance as the best way to ensure
there is no racing on draining
                        // the call to this method must be done only during the ASYNC
fusion so all the callers will be waiting
                        // this should not be performance costly with the assumption
the cancel is rare operation
                        if (DISCARD_GUARD.getAndIncrement(this) != 0) {
                            return;
                        }

                        int missed = 1;

                        for (; ; ) {
                            Operators.onDiscardQueueWithClear(qs,
actual.currentContext(), null);

                            int dg = discardGuard;
```

```
                        if (missed == dg) {
                              missed = DISCARD_GUARD.addAndGet(this, -missed);
                              if (missed == 0) {
                                    break;
                              }
                        }
                        else {
                              missed = dg;
                        }
                  }
            }

            @Override
            public boolean isEmpty() {
                  return qs == null || qs.isEmpty();
            }

            @Override
            @Nullable
            public T poll() {
                  return qs == null ? null : qs.poll();
            }

            @Override
            public int requestFusion(int requestedMode) {
                  if (sourceMode != Fuseable.NONE && (requestedMode &
Fuseable.ASYNC) != 0) {
                        outputFused = true;
                        return Fuseable.ASYNC;
                  }
                  return Fuseable.NONE;
            }

            @Override
            public int size() {
                  return qs == null ? 0 : qs.size();
            }

            @Override
            public CoreSubscriber<? super T> actual() {
                  return actual;
            }

            @Override
            @Nullable
            public Object scanUnsafe(Attr key) {
                  if (key == Attr.REQUESTED_FROM_DOWNSTREAM) return requested;
                  if (key == Attr.PARENT) return s;
                  if (key == Attr.CANCELLED) return cancelled;
                  if (key == Attr.TERMINATED) return done;
                  if (key == Attr.ERROR) return error;
                  if (key == Attr.DELAY_ERROR) return delayError;
                  if (key == Attr.RUN_ON) return worker;
                  if (key == Attr.RUN_STYLE) return Attr.RunStyle.ASYNC;
```

```
                  return InnerOperator.super.scanUnsafe(key);
            }
      }
}



package reactor.core.publisher;

import java.util.ArrayList;
import java.util.concurrent.atomic.LongAdder;
import java.util.stream.Stream;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;
import reactor.core.Fuseable;
import reactor.test.StepVerifier;
import reactor.test.subscriber.AssertSubscriber;

import static org.assertj.core.api.Assertions.assertThat;

public class FluxPrefetchTest {
      private static final int sourceSize = 1000;

      private static final Object[] nonFuseableSource   =
                  new Object[]{Flux.range(1, sourceSize).hide(), Fuseable.NONE};
      private static final Object[] syncFuseableSource  =
                  new Object[]{Flux.range(1, sourceSize), Fuseable.SYNC};
      private static final Object[] asyncFuseableSource =
                  new Object[]{Flux.range(1, sourceSize).onBackpressureBuffer(),
                              Fuseable.ASYNC};

      private static Stream<Integer> requestedModes() {
            return Stream.of(Fuseable.NONE, Fuseable.ASYNC, Fuseable.ANY);
      }

      private static Stream<Object[]> requestedModesWithPrefetchModes() {
            return requestedModes().flatMap(requestMode -> Stream.of(true,
false)

.map(prefetchMode -> new Object[]{

requestMode,

prefetchMode}));
      }

      private static Stream<Object[]> sources() {
            return Stream.of(nonFuseableSource, syncFuseableSource,
asyncFuseableSource);
      }

      // Fusions Tests
      @DisplayName("Prefetch value from Non-Fused upstream")
```

```
	@ParameterizedTest(name = "Prefetch value from Non-Fused upstream with
downstream with fusion={0} and Prefetch Mode={1}")
	@MethodSource("requestedModesWithPrefetchModes")
	public void prefetchValuesFromNonFuseableUpstream(int requestedMode,
			boolean prefetchMode) {
		Flux<Integer> nonFuseableSource = Flux.range(1, sourceSize)
												.hide();

		StepVerifier.FirstStep<Integer> firstStep =

	StepVerifier.create(nonFuseableSource.prefetch(prefetchMode));

		StepVerifier.Step<Integer> step;
		if (requestedMode != Fuseable.NONE) {
			step = firstStep.expectFusion(requestedMode, Fuseable.ASYNC);
		}
		else {
			step = firstStep.expectSubscription();
		}

		step.thenAwait()
		    .consumeSubscriptionWith(s ->
assertThat(((FluxPrefetch.PrefetchSubscriber<?>) s).sourceMode).isEqualTo(
				Fuseable.NONE))

		    .expectNextCount(sourceSize)
		    .verifyComplete();
	}

	@DisplayName("Prefetch Value from Async-Fused upstream")
	@ParameterizedTest(name = "Prefetch value from Async-Fused upstream with
downstream with fusion={0} and Prefetch Mode={1}")
	@MethodSource("requestedModesWithPrefetchModes")
	public void prefetchValuesFromAsyncFuseableUpstream(int requestedMode,
			boolean prefetchMode) {
		Flux<Integer> asyncFuseableSource = Flux.range(1, sourceSize)
												.onBackpressureBuffer();

		StepVerifier.FirstStep<Integer> firstStep =

	StepVerifier.create(asyncFuseableSource.prefetch(prefetchMode));

		StepVerifier.Step<Integer> step;
		if (requestedMode != Fuseable.NONE) {
			step = firstStep.expectFusion(requestedMode, Fuseable.ASYNC);
		}
		else {
			step = firstStep.expectSubscription();
		}

		step.thenAwait()
		    .consumeSubscriptionWith(s ->
assertThat(((FluxPrefetch.PrefetchSubscriber<?>) s).sourceMode).isEqualTo(
				Fuseable.ASYNC))
```

```
                .expectNextCount(sourceSize)
                .verifyComplete();
        }

        @DisplayName("Prefetch Value from Sync-Fused upstream")
        @ParameterizedTest(name = "Prefetch Value from Sync-Fused upstream with
downstream with fusion={0} and Prefetch Mode={1}")
        @MethodSource("requestedModesWithPrefetchModes")
        public void prefetchValuesFromSyncFuseableUpstream(int requestedMode,
                    boolean prefetchMode) {
              Flux<Integer> syncFuseableSource = Flux.range(1, sourceSize);

              StepVerifier.FirstStep<Integer> firstStep =

        StepVerifier.create(syncFuseableSource.prefetch(prefetchMode));

              StepVerifier.Step<Integer> step;
              if (requestedMode != Fuseable.NONE) {
                    step = firstStep.expectFusion(requestedMode,
                                (requestedMode & Fuseable.SYNC) != 0 ?
Fuseable.SYNC : Fuseable.ASYNC)

                                      .thenAwait()
                                      .consumeSubscriptionWith(s ->
assertThat(((FluxPrefetch.PrefetchSubscriber<?>) s).sourceMode).isEqualTo(
                                                  (requestedMode & Fuseable.SYNC) !=
0 ? Fuseable.SYNC :
                                                              Fuseable.NONE));
              }
              else {
                    step = firstStep.expectSubscription()
                                      .thenAwait()
                                      .consumeSubscriptionWith(s ->
assertThat(((FluxPrefetch.PrefetchSubscriber<?>) s).sourceMode).isEqualTo(
                                                  Fuseable.SYNC));
              }

              step.expectNextCount(sourceSize)
                  .verifyComplete();
        }

        // Backpressure Tests
        @DisplayName("Check backpressure from different sources")
        @ParameterizedTest(name = "Prefetch value from {0}")
        @MethodSource("sources")
        public void backpressureFromDifferentSourcesTypes(Flux<Integer> source) {
              AssertSubscriber<Integer> ts = AssertSubscriber.create(0);

              source.prefetch()
                    .subscribe(ts);

              ts.assertNoEvents();

              int step = 250;
              int count = 0;
```

```
            while (count < sourceSize) {
                  ts.request(step);
                  count += step;
                  ts.assertValueCount(count);
            }

            ts.assertComplete();
      }

      // Prefetch mode Tests
      @ParameterizedTest
      @MethodSource("requestedModes")
      public void immediatePrefetchInEagerPrefetchMode(int requestedMode) {
            int prefetch = 256;
            ArrayList<Long> requests = new ArrayList<>();

            Flux<Integer> nonFuseableSource = Flux.range(1, sourceSize)
                                                  .hide()
                                                  .doOnRequest(requests::add);

            StepVerifier.FirstStep<Integer> firstStep =
                        StepVerifier.create(nonFuseableSource.prefetch(prefetch,
false), 0);

            StepVerifier.Step<Integer> step;
            if (requestedMode != Fuseable.NONE) {
                  step = firstStep.expectFusion(requestedMode, requestedMode &
Fuseable.ASYNC);
            }
            else {
                  step = firstStep.expectSubscription();
            }

            step.then(() -> assertThat(requests).hasSize(1)
                                                .containsExactly((long)
prefetch))
                .thenRequest(Long.MAX_VALUE)
                .expectNextCount(sourceSize)
                .verifyComplete();
      }

      @ParameterizedTest
      @MethodSource("requestedModes")
      public void delayedPrefetchInLazyPrefetchMode(int requestedMode) {
            LongAdder requestCount = new LongAdder();

            Flux<Integer> nonFuseableSource = Flux.range(1, sourceSize)
                                                  .hide()
                                                  .doOnRequest((r) ->
requestCount.increment());

            StepVerifier.FirstStep<Integer> firstStep =
                        StepVerifier.create(nonFuseableSource.prefetch(true),
0);
```

```
            StepVerifier.Step<Integer> step;
            if (requestedMode != Fuseable.NONE) {
                  step = firstStep.expectFusion(requestedMode, requestedMode &
Fuseable.ASYNC);
            }
            else {
                  step = firstStep.expectSubscription();
            }

            step.then(() -> assertThat(requestCount.longValue()).isEqualTo(0))
                .thenRequest(Long.MAX_VALUE)
                .expectNextCount(sourceSize)
                .verifyComplete();
      }
}
```

1.2 Демостраціний проект дистанційного замовлення

```
package delivery.app.user.repository.model;

import java.time.LocalDateTime;

import com.fasterxml.jackson.annotation.JsonInclude;
import delivery.app.user.dto.ActionType;
import delivery.app.user.dto.Point;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.Id;
import org.springframework.data.annotation.Transient;
import org.springframework.data.relational.core.mapping.Table;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Table("ACTIONS")
public class Action {

  @Id
  String id;

  String orderId;

  ActionType type;

  @CreatedDate
  LocalDateTime createdAt;

  @Transient
  Point coordinate;
```

```
}



package delivery.app.user.repository;

import delivery.app.user.repository.model.Action;

import org.springframework.data.repository.reactive.ReactiveCrudRepository;

public interface ActionRepository extends ReactiveCrudRepository<Action,
String> {

}



package delivery.app.user.controller;

import delivery.app.user.AuthenticationService;
import delivery.app.user.dto.Authority;
import delivery.app.user.dto.UsernameAndPassword;
import lombok.AllArgsConstructor;
import reactor.core.publisher.Mono;

import java.util.Collection;

import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.Payload;
import org.springframework.stereotype.Controller;

@Controller
@MessageMapping("api")
@AllArgsConstructor
public class AuthenticationController {

  final AuthenticationService authenticationService;

  @MessageMapping("authenticate")
  public Mono<Collection<Authority>> authenticate(@Payload UsernameAndPassword
userAndPassword) {
    return authenticationService.authenticate(userAndPassword.getUsername(),
userAndPassword.getPassword());
  }
}



package delivery.app.common;

import
delivery.app.common.security.rsocket.requester.JWTMetadataRSocketConnectorConfi
gurer;
import io.rsocket.core.RSocketServer;
```

```
import io.rsocket.transport.netty.server.TcpServerTransport;
import
org.springframework.boot.autoconfigure.rsocket.RSocketMessageHandlerCustomizer;
import org.springframework.boot.rsocket.server.RSocketServerCustomizer;
import
org.springframework.messaging.rsocket.annotation.support.RSocketMessageHandler;
import
org.springframework.security.messaging.handler.invocation.reactive.Authenticati
onPrincipalArgumentResolver;
import reactor.netty.http.server.HttpServer;

import org.springframework.beans.factory.ObjectProvider;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Scope;
import org.springframework.messaging.rsocket.RSocketConnectorConfigurer;
import org.springframework.messaging.rsocket.RSocketRequester;
import org.springframework.messaging.rsocket.RSocketStrategies;

@Configuration
public class BaseConfiguration {

  @Bean
  public JWTMetadataRSocketConnectorConfigurer
clientInternalJWTRequestCustomizer() {
    return new JWTMetadataRSocketConnectorConfigurer();
  }

  @Bean
  @Scope("prototype")
  @ConditionalOnClass({RSocketRequester.class, io.rsocket.RSocket.class,
HttpServer.class,
      TcpServerTransport.class})
  public RSocketRequester.Builder rSocketRequesterBuilder(RSocketStrategies
strategies,
      ObjectProvider<RSocketConnectorConfigurer> rSocketConnectorConfigurers) {
    RSocketRequester.Builder builder = RSocketRequester.builder()
        .rsocketStrategies(strategies);

    rSocketConnectorConfigurers.forEach(builder::rsocketConnector);

    return builder;
  }

  @Bean
  public RSocketMessageHandlerCustomizer rSocketMessageHandlerCustomizer() {
    return messageHandler -> messageHandler.getArgumentResolverConfigurer()
        .addCustomResolver(new AuthenticationPrincipalArgumentResolver());
  }
}



package delivery.app.cart.repository.model;
```

```
import java.util.Collection;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;
import lombok.Data;
import lombok.RequiredArgsConstructor;

@Data
@RequiredArgsConstructor
public class Cart {
  final String id;
  final ConcurrentMap<String, Item> items = new ConcurrentHashMap<>();

  public boolean hasProduct(String productId) {
    return items.containsKey(productId);
  }

  public void update(Item itemUpdate) {
    items.compute(itemUpdate.getProductId(), (key, item) -> {
      if (item != null) {
        final int nextQuantity = item.getQuantity() + itemUpdate.getQuantity();

        if (nextQuantity > 0) {
          return new Item(key, nextQuantity);
        }

        return null;
      }

      return itemUpdate;
    });
  }

  public Collection<Item> items() {
    return items.values();
  }
}
```

```
package delivery.app.cart.controller;

import delivery.app.user.CartService;
import delivery.app.user.dto.Cart;
import delivery.app.user.dto.Item;
import lombok.AllArgsConstructor;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.Payload;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.stereotype.Controller;
import reactor.core.publisher.Mono;

@Controller
@MessageMapping("api.cart")
```

```
@AllArgsConstructor
public class CartController {

  final CartService cartService;

  @MessageMapping("")
  public Cart get(@AuthenticationPrincipal String user) {
    return cartService.get(user);
  }

  @MessageMapping("patch")
  public Mono<Void> patch(@Payload Item item, @AuthenticationPrincipal String
user) {
    return cartService.update(user, item);
  }
}
```

```
package delivery.app.cart.repository;

import delivery.app.cart.repository.model.Cart;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;
import org.springframework.lang.Nullable;
import org.springframework.stereotype.Service;

@Service
public class CartRepository {

  final ConcurrentMap<String, Cart> carts = new ConcurrentHashMap<>();

  public Cart findOrCreate(String cartId) {
    return this.carts.computeIfAbsent(cartId, Cart::new);
  }

  @Nullable
  public Cart removeIfPresent(String cartId) {
    return this.carts.remove(cartId);
  }
}
```

```
package delivery.app.cart;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CartServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(CartServiceApplication.class, args);
  }
```

```
}
```

```
package delivery.app.catalog.controller;

import delivery.app.catalog.CatalogService;
import delivery.app.catalog.dto.Product;
import java.util.Collection;
import java.util.List;
import lombok.AllArgsConstructor;
import org.springframework.messaging.handler.annotation.DestinationVariable;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.stereotype.Controller;
import reactor.core.publisher.Mono;

@Controller
@MessageMapping("api.products")
@AllArgsConstructor
public class CatalogController {

  final CatalogService catalogService;

  @MessageMapping("")
  public Mono<Collection<Product>> list() {
    return catalogService.findAll();
  }

  @MessageMapping("{productId}")
  public Mono<Product> find(@DestinationVariable("productId") String productId)
{
    return catalogService.find(productId);
  }

  @MessageMapping("{productId}.exist")
  public Mono<Void> exist(@DestinationVariable("productId") String productId) {
    return catalogService.exist(productId);
  }
}
```

```
package delivery.app.common.security.rsocket;

import reactor.core.publisher.Mono;

import org.springframework.context.annotation.Bean;
import
org.springframework.security.authentication.ReactiveAuthenticationManager;
import org.springframework.security.config.annotation.rsocket.RSocketSecurity;
import
org.springframework.security.rsocket.authentication.AuthenticationPayloadInterc
eptor;
import
org.springframework.security.rsocket.core.PayloadSocketAcceptorInterceptor;
```

```
public abstract class CommonRSocketSecurityConfigurerAdapter {

  @Bean
  public PayloadSocketAcceptorInterceptor rsocketInterceptor(RSocketSecurity
rsocket, ReactiveAuthenticationManager manager) {
    AuthenticationPayloadInterceptor interceptor = new
AuthenticationPayloadInterceptor(manager);
    interceptor.setAuthenticationConverter(new
SimpleJWTAuthenticationConverter());

    rsocket.addPayloadInterceptor(interceptor);

    return rsocket.build();
  }

  protected abstract void doConfigure(RSocketSecurity rsocket) throws
Exception;

  @Bean
  public ReactiveAuthenticationManager noOpsAuthManager() {
    return Mono::just;
  }
}



package delivery.app.user.service;

import delivery.app.user.AuthenticationService;
import delivery.app.user.dto.Authority;
import delivery.app.user.repository.UserRepository;
import lombok.AllArgsConstructor;
import reactor.core.publisher.Mono;
import reactor.core.scheduler.Schedulers;

import java.util.Arrays;
import java.util.Collection;
import java.util.stream.Collectors;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

@Service
@AllArgsConstructor
public class DefaultAuthenticationService implements AuthenticationService {

  final PasswordEncoder passwordEncoder;
  final UserRepository userRepository;

  @Override
  public Mono<Collection<Authority>> authenticate(String username, CharSequence
password) {
    return userRepository.findByName(username)
                         .publishOn(Schedulers.boundedElastic())
```

```
                          .<Collection<Authority>>handle((user, sink) -> {
                            if (user != null &&
passwordEncoder.matches(password, user.getEncodedPassword())) {

sink.next(Arrays.stream(user.getAuthorities().split(",")).map(Authority::new)
                                          .collect(Collectors.toSet()));
                              return;
                            }

                            sink.error(new BadCredentialsException("Given
username or password are incorrect"));
                          })
                          .switchIfEmpty(Mono.error(() -> new
BadCredentialsException("Given username or password are incorrect")));
  }
}



package delivery.app.catalog.service;

import delivery.app.catalog.CatalogService;
import delivery.app.catalog.dto.Product;
import delivery.app.catalog.repository.ProductRepository;
import java.util.Collection;
import java.util.NoSuchElementException;
import java.util.stream.Collectors;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
import reactor.core.publisher.Mono;

@Service
@AllArgsConstructor
public class DefaultCatalogService implements CatalogService {

  final ProductRepository productRepository;

  @Override
  public Mono<Product> find(String productId) {
    return productRepository.findById(productId)
        .map(model -> new Product(model.getId(), model.getName(),
model.getDescription(),
            model.getImgLink(), model.getPrice(), model.getCurrency(),
model.isAvailable()))
        .switchIfEmpty(Mono.error(
            () -> new NoSuchElementException("The product with the given id
does not exist")));
  }

  @Override
  public Mono<Collection<Product>> findAll() {
    // TODO:
    return productRepository.findAll()
        .map(model -> new Product(model.getId(), model.getName(),
model.getDescription(),
```

```
            model.getImgLink(), model.getPrice(), model.getCurrency(),
model.isAvailable()))
        .collect(Collectors.toList());
  }

  @Override
  public Mono<Void> exist(String productId) {
    return productRepository.existsById(productId)
        .handle(((exist, sink) -> {
          if (exist) {
            sink.complete();
          } else {
            sink.error(new NoSuchElementException("The product with the given
id does not exist"));
          }
        }));
  }

  @Override
  public Mono<Void> update(String productId, Product product) {
    return productRepository.save(
        new delivery.app.catalog.repository.model.Product(productId,
product.getName(),
            product.getDescription(), product.getImgLink(), product.getPrice(),
            product.getCurrency(), product.isAvailable()))
        .then(Mono.empty());
  }
}
```

```
package delivery.app.user.service;

import delivery.app.user.UserService;
import delivery.app.user.dto.User;
import delivery.app.user.repository.UserRepository;
import java.util.Collection;
import java.util.stream.Collectors;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Service;
import reactor.core.publisher.Mono;

@Service
@AllArgsConstructor
@Slf4j
public class DefaultUserService implements UserService {

  final UserRepository userRepository;

  @Override
  @PreAuthorize("hasRole('ROLE_ADMIN')")
  public Mono<User> find(String username) {
```

```
    return userRepository.findByName(username)
        .map(
            userModel -> new User(userModel.getName(), userModel.getEmail(),
userModel.getAddress(),
                userModel.getPhone()));
  }

  @Override
  @PreAuthorize("hasRole('ROLE_ADMIN')")
  public Mono<Collection<User>> findAll() {
    log.info("Searching for all users");
    return userRepository.findAll()
        .map(model -> new User(model.getName(), model.getEmail(),
model.getAddress(),
            model.getPhone()))
        .collect(Collectors.toList());
  }
}
```

```
package delivery.app.gateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class GatewayServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(GatewayServiceApplication.class, args);
  }
}
```

```
package delivery.app.cart.repository.model;

import lombok.Value;

@Value
public class Item {

  String productId;
  int quantity;
}
```

```
package delivery.app.common.security.rsocket.requester;

import io.rsocket.Payload;
import io.rsocket.RSocket;
import io.rsocket.core.RSocketConnector;
import io.rsocket.plugins.RSocketInterceptor;
```

```
import io.rsocket.util.RSocketProxy;
import org.reactivestreams.Publisher;
import org.springframework.security.core.context.SecurityContext;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

import org.springframework.messaging.rsocket.RSocketConnectorConfigurer;
import org.springframework.security.core.context.ReactiveSecurityContextHolder;
import org.springframework.security.core.context.SecurityContextHolder;

import static
delivery.app.common.security.rsocket.requester.JWTSecurityHeaders.addJWTBearerT
oken;

public class JWTMetadataRSocketConnectorConfigurer implements
RSocketConnectorConfigurer {

  @Override
  public void configure(RSocketConnector connector) {
    connector
        .interceptors(ir -> ir.forRequester(new
SecurityMetadataPopulatingRSocketInterceptor()));
  }

  static class SecurityMetadataPopulatingRSocketInterceptor implements
RSocketInterceptor {

    @Override
    public RSocket apply(RSocket socket) {
      return new SecurityMetadataPopulatingRSocketProxy(socket);
    }
  }

  static class SecurityMetadataPopulatingRSocketProxy extends RSocketProxy {

    public SecurityMetadataPopulatingRSocketProxy(RSocket socket) {
      super(socket);
    }

    @Override
    public Mono<Void> fireAndForget(Payload payload) {
      return ReactiveSecurityContextHolder.getContext()
          .defaultIfEmpty(SecurityContextHolder.createEmptyContext())
          .flatMap(
              sc -> super.fireAndForget(addJWTBearerToken(payload,
sc.getAuthentication())));
    }

    @Override
    public Mono<Payload> requestResponse(Payload payload) {
      return ReactiveSecurityContextHolder.getContext()
          .defaultIfEmpty(SecurityContextHolder.createEmptyContext())
          .flatMap(sc -> super
              .requestResponse(addJWTBearerToken(payload,
sc.getAuthentication())));
```

```
    }

    @Override
    public Flux<Payload> requestStream(Payload payload) {
      return ReactiveSecurityContextHolder.getContext()
          .defaultIfEmpty(SecurityContextHolder.createEmptyContext())
          .flatMapMany(
              sc -> super.requestStream(addJWTBearerToken(payload,
sc.getAuthentication())));
    }

    @Override
    public Flux<Payload> requestChannel(Publisher<Payload> payloads) {
      return ReactiveSecurityContextHolder.getContext()
          .materialize()
          .flatMapMany(signal -> {
            final SecurityContext sc = signal.get();
            if (sc != null && sc.getAuthentication() != null &&
sc.getAuthentication()
                .isAuthenticated()) {
              return Flux.from(payloads).switchOnFirst((firstSignal, wholeFlux)
-> {
                Payload payload = firstSignal.get();

                if (payload != null) {
                  return super.requestChannel(wholeFlux.skip(1)
                      .startWith(addJWTBearerToken(payload,
sc.getAuthentication())));
                }

                return wholeFlux;
              });
            }

            return super.requestChannel(payloads);
          });
    }
  }
}



package delivery.app.common.security.rsocket.requester;

import java.util.UUID;
import java.util.stream.Collectors;

import com.auth0.jwt.JWT;
import com.auth0.jwt.algorithms.Algorithm;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.ByteBufAllocator;
import io.netty.buffer.CompositeByteBuf;
import io.rsocket.Payload;
import io.rsocket.metadata.AuthMetadataCodec;
import io.rsocket.metadata.CompositeMetadataCodec;
```

```
import io.rsocket.metadata.WellKnownMimeType;
import io.rsocket.util.ByteBufPayload;

import org.springframework.security.core.Authentication;

class JWTSecurityHeaders {

  static Payload addJWTBearerToken(Payload payload, Authentication
authentication) {
    if (authentication != null && authentication.isAuthenticated()) {
      try {
        final String token = JWT.create()
                                .withIssuer("Gateway Service")
                                .withJWTId(UUID.randomUUID()
                                               .toString())
                                .withClaim("authorities",
                                        authentication.getAuthorities()
                                                      .stream()
                                                      .map(String::valueOf)

.collect(Collectors.toList()))
                                .withSubject(authentication.getPrincipal()
                                                           .toString())
                                .sign(Algorithm.none());

        ByteBuf metadata = payload.sliceMetadata()
                                  .retain();
        ByteBufAllocator byteBufAllocator = metadata.alloc();

        CompositeByteBuf bufs = byteBufAllocator.compositeBuffer();
        bufs.addComponent(true, metadata);

        ByteBuf authMetadataPayload = AuthMetadataCodec.encodeBearerMetadata(
                byteBufAllocator,
                token.toCharArray());

        CompositeMetadataCodec.encodeAndAddMetadata(bufs,
                byteBufAllocator,
                WellKnownMimeType.MESSAGE_RSOCKET_AUTHENTICATION,
                authMetadataPayload);

        return ByteBufPayload.create(payload.sliceData()
                                            .retain(), bufs);
      }
      finally {
        payload.release();
      }
    }

    return payload;
  }
}
```

```
package delivery.app.catalog.repository.model;

import java.math.BigDecimal;
import java.util.Currency;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import org.springframework.data.annotation.Id;
import org.springframework.data.relational.core.mapping.Table;

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Table("products")
public class Product {

  @Id
  String id;

  String     name;
  String     description;
  String     imgLink;
  BigDecimal price;
  Currency   currency;
  boolean    available;
}



package delivery.app.catalog.repository;

import delivery.app.catalog.repository.model.Product;

import org.springframework.data.repository.reactive.ReactiveCrudRepository;

public interface ProductRepository extends ReactiveCrudRepository<Product,
String> {

}




  public RemoteAuthenticationManager(AuthenticationService
authenticationService) {
    this.authenticationService = authenticationService;
  }

  @Override
  public Mono<Authentication> authenticate(Authentication authentication)
          throws AuthenticationException {
```

```
    if (authentication instanceof AnonymousAuthenticationToken) {
      return Mono.just(authentication);
    }

    return authenticationService
            .authenticate(authentication.getName(),
authentication.getCredentials().toString())
            .map(authorities -> {
              final UsernamePasswordAuthenticationToken token =
                      new
UsernamePasswordAuthenticationToken(authentication.getPrincipal(),
                              authentication.getCredentials(),
                              authorities.stream()
                                         .map(authority -> new
SimpleGrantedAuthority(
                                                 authority.getName()))
                                         .collect(Collectors.toSet()));

              token.eraseCredentials();

              if (authentication instanceof CredentialsContainer) {
                ((CredentialsContainer) authentication).eraseCredentials();
              }

              return token;
            });
  }
}



package delivery.app.gateway.controller;

import java.util.HashMap;
import java.util.Map;

import org.springframework.http.HttpMethod;
import org.springframework.http.MediaType;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

import org.springframework.core.io.buffer.DataBuffer;
import org.springframework.http.HttpEntity;
import org.springframework.http.ResponseEntity;
import org.springframework.messaging.rsocket.RSocketRequester;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class RoutingController {

  static final Map<String, Integer> serviceToBaseUrlMap = new HashMap<>();

  static {
```

```
    serviceToBaseUrlMap.put("users", 8081);
    serviceToBaseUrlMap.put("cart", 8082);
    serviceToBaseUrlMap.put("products", 8083);
    serviceToBaseUrlMap.put("orders", 8084);
    serviceToBaseUrlMap.put("payments", 8085);
  }

  final Map<String, RSocketRequester> rsocketRequestersMap;

  public RoutingController(RSocketRequester.Builder rsocketRequesterBuilder) {
    rsocketRequestersMap = new HashMap<>();
    serviceToBaseUrlMap.forEach((service, port) ->
            rsocketRequestersMap.put(service,
rsocketRequesterBuilder.tcp("localhost", port)));
  }

  @RequestMapping(value = "/api/{servicePath}")
  public ResponseEntity<Mono<DataBuffer>> handle(@PathVariable("servicePath")
String servicePath,
          HttpEntity<Mono<DataBuffer>> httpEntity, HttpMethod httpMethod) {

    return rsocketRequestersMap
            .keySet()
            .stream()
            .filter(servicePath::startsWith)
            .findFirst()
            .map(rsocketRequestersMap::get)
            .map(requester -> requester.route("api.{servicePath}" + (httpMethod
== HttpMethod.PATCH ? ".patch" : ""), servicePath.replace('/', '.'))
                                    .data(httpEntity.getBody() == null ?
Mono.empty() : httpEntity.getBody())
                                    .retrieveMono(DataBuffer.class))
            .map(body -> ResponseEntity.ok().body(body))
            .orElse(ResponseEntity.notFound().build());
  }
}



package delivery.app.catalog.configuration;

import
delivery.app.common.security.rsocket.CommonRSocketSecurityConfigurerAdapter;

import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.method.configuration.EnableReact
iveMethodSecurity;
import
org.springframework.security.config.annotation.rsocket.EnableRSocketSecurity;
import org.springframework.security.config.annotation.rsocket.RSocketSecurity;

@Configuration
@EnableRSocketSecurity
@EnableReactiveMethodSecurity
```

```
public class SecurityConfiguration extends
CommonRSocketSecurityConfigurerAdapter {

  @Override
  protected void doConfigure(RSocketSecurity rsocket) throws Exception {
    rsocket.authorizePayload(authorize ->
            authorize.route("api.products.*").permitAll()
                    .anyExchange().permitAll()
                    .anyRequest().authenticated());
  }
}
```

```
package delivery.app.common.security.rsocket;

import com.auth0.jwt.JWT;
import com.auth0.jwt.interfaces.DecodedJWT;
import io.netty.buffer.ByteBuf;
import io.rsocket.metadata.CompositeMetadata;
import reactor.core.publisher.Mono;

import java.nio.charset.StandardCharsets;
import java.util.stream.Collectors;

import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken
;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.rsocket.api.PayloadExchange;
import
org.springframework.security.rsocket.authentication.PayloadExchangeAuthenticati
onConverter;
import org.springframework.security.rsocket.metadata.BearerTokenMetadata;

public class SimpleJWTAuthenticationConverter implements
PayloadExchangeAuthenticationConverter {

  private static final String BEARER_MIME_TYPE_VALUE =
          BearerTokenMetadata.BEARER_AUTHENTICATION_MIME_TYPE.toString();

  @Override
  public Mono<Authentication> convert(PayloadExchange exchange) {
    ByteBuf metadata = exchange.getPayload().metadata();
    CompositeMetadata compositeMetadata = new CompositeMetadata(metadata,
false);

    for (CompositeMetadata.Entry entry : compositeMetadata) {
      if (BEARER_MIME_TYPE_VALUE.equals(entry.getMimeType())) {
        ByteBuf content = entry.getContent();
        String token = content.toString(StandardCharsets.UTF_8);
        final DecodedJWT decodedJWT = JWT.decode(token);
```

```
        return Mono.just(new
UsernamePasswordAuthenticationToken(decodedJWT.getSubject(),
                null,
                decodedJWT.getClaim("authorities")
                          .asList(String.class)
                          .stream()
                          .map(SimpleGrantedAuthority::new)
                          .collect(Collectors.toList())));
      }
    }
    return Mono.empty();
  }
}
```

```
package delivery.app.user.controller;

import java.util.UUID;

import delivery.app.user.OrdersService;
import delivery.app.user.dto.Action;
import delivery.app.user.dto.OrderId;
import lombok.AllArgsConstructor;
import reactor.core.publisher.Flux;

import org.springframework.http.MediaType;
import org.springframework.http.codec.ServerSentEvent;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
@AllArgsConstructor
public class TrackingController {

  final OrdersService trackingService;

  @PostMapping("/api/tracking")
  public Flux<ServerSentEvent<Action>> subscribe(@RequestBody OrderId order) {
    return trackingService.subscribe(order.getOrderId())
                          .map(a -> ServerSentEvent.<Action>builder()
                                                   .id(UUID.randomUUID()
                                                           .toString())
                                                   .data(a)
                                                   .build());
  }

  @RequestMapping(path = "/internal/tracking", method = RequestMethod.POST,
consumes = MediaType.APPLICATION_JSON_VALUE)
  @ResponseBody
  public void update(@RequestBody Action action) {
```

```
    trackingService.update(action);
  }
}
```

```
package delivery.app.user.repository.model;

import java.time.LocalDateTime;
import java.util.Arrays;
import java.util.Set;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.Id;
import org.springframework.data.relational.core.mapping.Table;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Table("USERS")
public class User {

  @Id
  String id;

  String name;
  String encodedPassword;
  String email;
  String phone;
  String address;

  @CreatedDate
  LocalDateTime createdAt;
  String        authorities;
}
```

```
package delivery.app.user.repository;

import delivery.app.user.repository.model.User;
import reactor.core.publisher.Mono;

import org.springframework.data.repository.reactive.ReactiveCrudRepository;

public interface UserRepository extends ReactiveCrudRepository<User, String> {

  Mono<User> findByName(String username);
}
```

```
package delivery.app.user;

import java.time.LocalDateTime;
import java.util.Arrays;
import java.util.UUID;

import delivery.app.user.repository.UserRepository;
import delivery.app.user.repository.model.User;
import io.r2dbc.spi.ConnectionFactory;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;
import org.springframework.r2dbc.connection.init.ConnectionFactoryInitializer;
import org.springframework.r2dbc.connection.init.ResourceDatabasePopulator;

@SpringBootApplication
public class UserServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(UserServiceApplication.class, args);
  }

  @Bean
  ConnectionFactoryInitializer initializer(ConnectionFactory connectionFactory)
{
    ConnectionFactoryInitializer initializer = new
ConnectionFactoryInitializer();
    initializer.setConnectionFactory(connectionFactory);
    initializer.setDatabasePopulator(new ResourceDatabasePopulator(new
ClassPathResource("sql/schema.sql"), new ClassPathResource("sql/data.sql")));

    return initializer;
  }


  @Bean
  public CommandLineRunner demo(UserRepository userRepository) {
    return (args) -> {
      userRepository.saveAll(
              Arrays.asList(
                      new User(UUID.randomUUID().toString(), "user",
"$2a$06$fJhpqOTtOsY6MpxpnwjPnO97TsrQsoc.C.DNtWJyu4yQHR/9PkiSK","user@example.co
m", "+123456789001", "hello world, land, hello, 012345", LocalDateTime.now(),
"ROLE_USER"),
                      new User(UUID.randomUUID().toString(), "admin",
"$2a$06$fJhpqOTtOsY6MpxpnwjPnO97TsrQsoc.C.DNtWJyu4yQHR/9PkiSK","admin@example.c
om", "+123456789001", "hello world, land, hello, 012345", LocalDateTime.now(),
"ROLE_USER,ROLE_ADMIN"))
      );
    };
```

```
  }
}
```

```
package delivery.app.gateway.security;

import delivery.app.user.AuthenticationService;
import org.springframework.context.annotation.Bean;
import org.springframework.http.HttpMethod;
import
org.springframework.security.config.annotation.method.configuration.EnableReact
iveMethodSecurity;
import
org.springframework.security.config.annotation.web.reactive.EnableWebFluxSecuri
ty;
import org.springframework.security.config.web.server.ServerHttpSecurity;
import org.springframework.security.web.server.SecurityWebFilterChain;

@EnableWebFluxSecurity
@EnableReactiveMethodSecurity
public class WebSecurityConfiguration{

  @Bean
  public SecurityWebFilterChain securitygWebFilterChain(ServerHttpSecurity
http, RemoteAuthenticationManager manager) {
    return http.authenticationManager(manager)
                .authorizeExchange()
                .pathMatchers("/login").permitAll()
                .pathMatchers("/css/**", "/index").permitAll()
                .pathMatchers(HttpMethod.GET, "/api/products",
"/api/products/*").permitAll()
                .anyExchange().authenticated()
                .and()
                .formLogin(spec -> spec.loginPage("/login"))
                .build();
  }

  @Bean
  public RemoteAuthenticationManager
remoteAuthenticationManager(AuthenticationService authenticationService) {
    return new RemoteAuthenticationManager(authenticationService);
  }
}
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2021 р.

УПРАВЛІННЯ ПОТОКАМИ ДАНИХ У БІЛІОТЕЦІ PROJECT-REACTOR

Графічний матеріал

КПІ.ІП-7120.045490.06.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ О.А. Халус

Нормоконтроль:

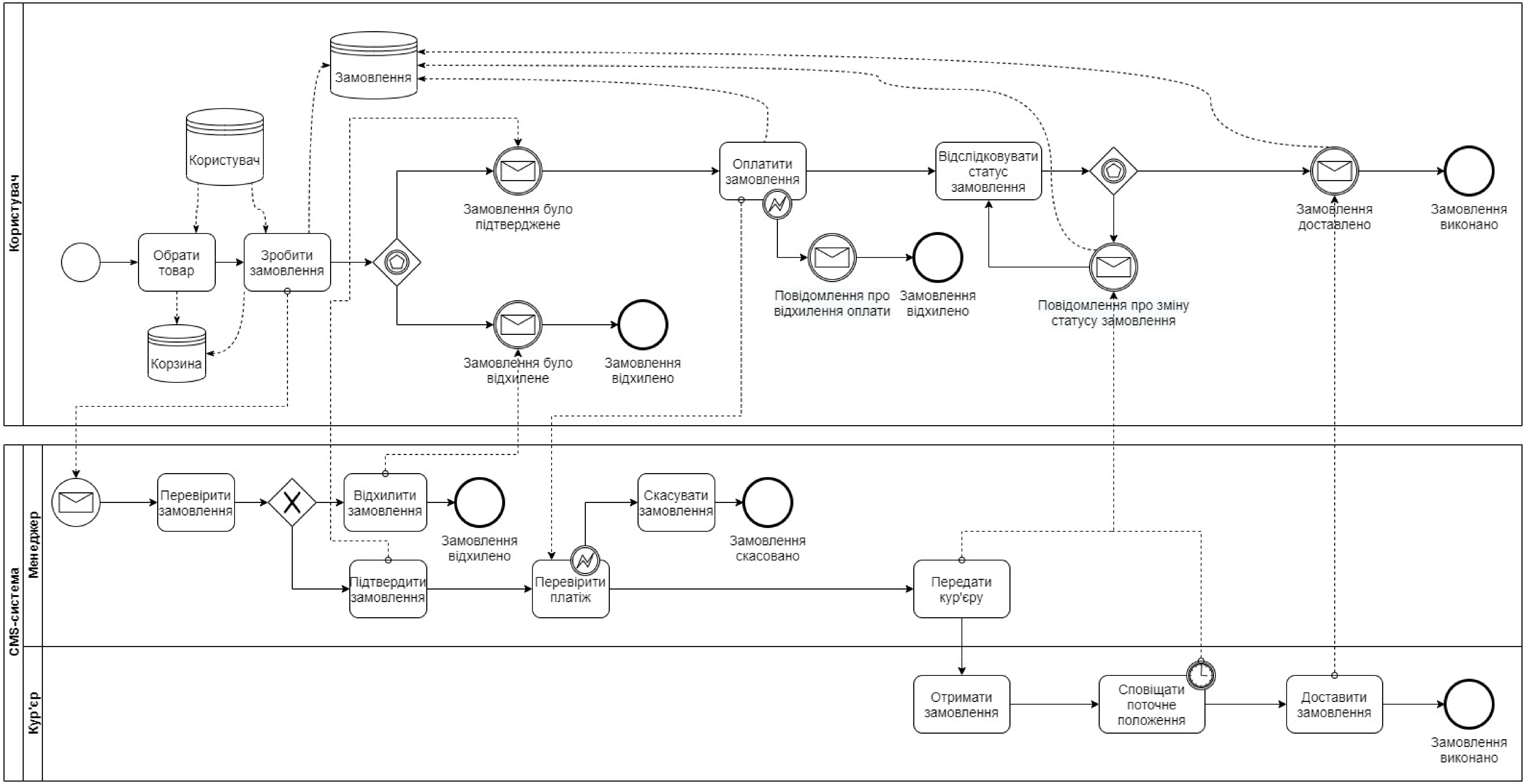
_____ К.І. Ліщук

Виконавець:

_____Д.І. Михайлов

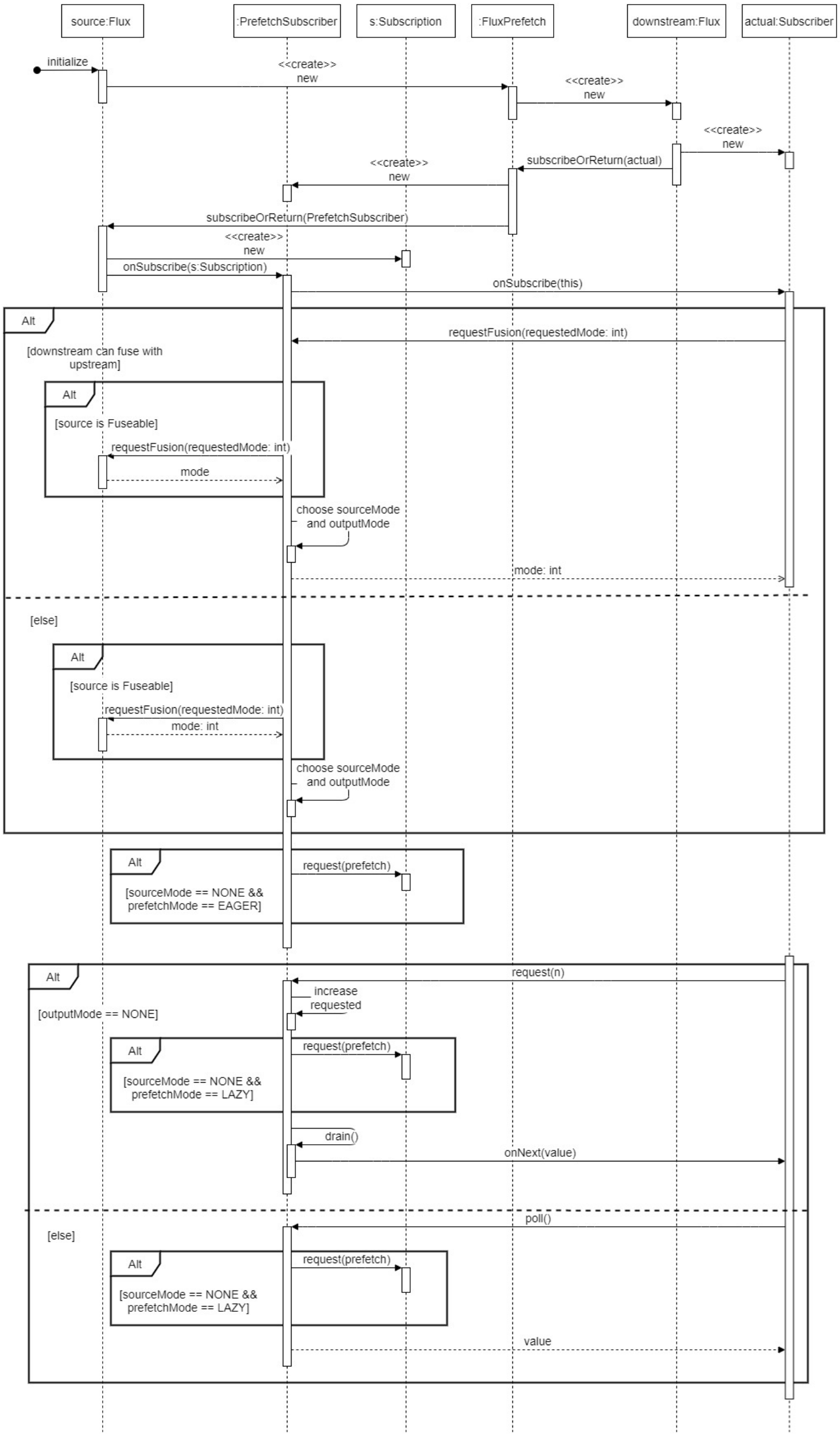
Київ – 2021 року

КПІ.ІП-7120.045490.06.99.СС

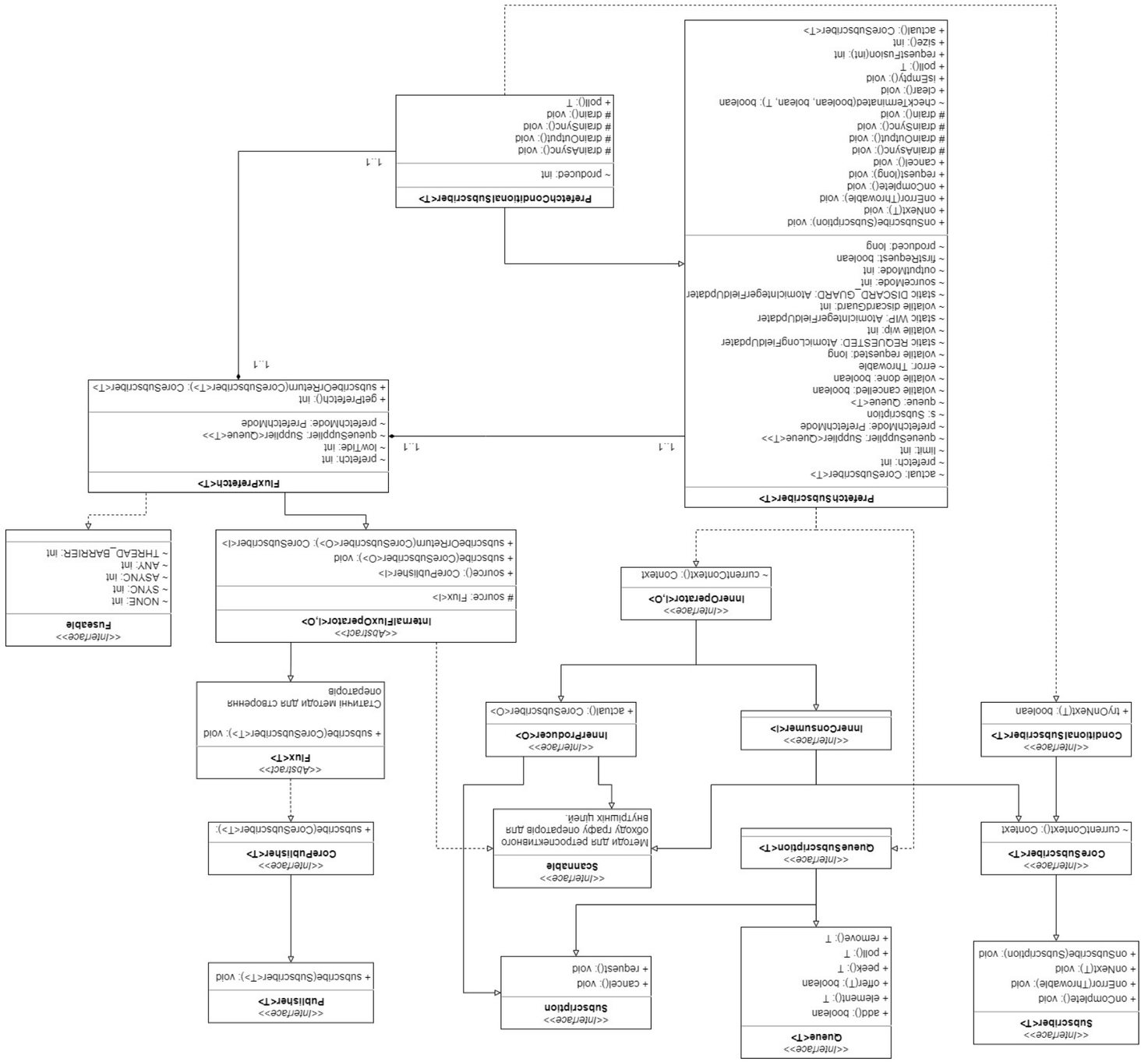


					КПІ.ІП-7120.045490.06.99.СС			
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна бізнес процесів	Літера	Маса	Масштаб
Розробив		Михайлов Д.І.						
Перевірив		Халус О.А.				Аркуш 1		Аркушів 1
Т. кон.								
					Управління потоками даних у бібліотеці Project-Reactor	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-71		
Н. кон.		Ліщук К.І.						
Затвердив		Халус О.А.						

КПІ.ІП-7120.045490.06.99.СС



					КПІ.ІП-7120.045490.06.99.СС	
					Схема структурна послідовності	Лит. / Арк. 1 / Аркушів 1
Зм.	Арк.	№ докум.	Підп.	Дата		Аркуш / Аркушів
Розроб.		Михайлов Д.І			Управління потоками даних у бібліотеці Project-Reactor	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-71
Перев.		Халус О.А.				
Т. Кон.						
Н. Кон.		Ліщук К.І.				
Затв.		Халус О,А.				



					КПІ.ІП-7120.045490.06.99.СС
Зм.	Арк.	№ докумeнта	Пiдпис	Дата	
Розробив		Михайлов Д.І.			Схема структура класів програмного забезпечення для Project-Reactor
Перевірив		Харус О.А.			
Т. контр.					
					КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-71
Н. контр.		Плщук К.І.			Управління потоками даних у бібліотеці Project-Reactor
Затвердив		Харус О.А.			
					Літера / Маса / Масштаб
					Аркуш 1 / Аркушів 1