การพัฒนาการทดสอบอัตโนมัติสำหรับระบบถอนกองทุน และซิงค์คอนเนคเตอร์แอปพลิเคชัน

DEVELOPMENT OF THE AUTOMATION TESTING SUITE FOR THE REDEMPTION AND SINK CONNECTOR SYSTEMS

กฤษฎา ชิตนุกูล

ลภัสรดา พงศ์พันธ์พฤทธิ์

สหกิจศึกษานี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรบัณฑิต

(วิทยาการคอมพิวเตอร์)

ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2565

DEVELOPMENT OF THE AUTOMATION TESTING SUITE FOR THE REDEMPTION AND SINK CONNECTOR SYSTEMS

KRITSADA CHITNUKOOL
LAPATLADA PHONPHANPARET

A COOPERATIVE EDUCATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE (COMPUTER SCIENCE)
DEPARTMENT OF COMPUTER SCIENCE, SCHOOL OF SCIENCE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
ACADEMIC YEAR 2022

หัวข้อสหกิจศึกษา	การพัฒนาการทดสอบอัตโนมัติสำหรับระบบถอนกองทุนและซิงค์คอนเนคเตอร์แอปพลิเคชัน Development of the automation testing suite for the redemption and sink connector systems
ชื่อนักศึกษา	นายกฤษฎา ชิตนุกูล รหัสนักศึกษา 62050127 นางสาวลภัสรดา พงศ์พันธ์พฤทธิ์ รหัสนักศึกษา 62050219
ปริญญา	วิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชา	วิทยาการคอมพิวเตอร์
ปีการศึกษา	2565
อาจารย์ที่ปรึกษา	ผศ.ดร.อนันตพร หรรษคุณาฒัย

คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (สจล.) อนุมัติให้สหกิจศึกษานี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์) ประจำปีการศึกษา 2565

คณะกรรมการสอบ	ลายมือชื่อ
ดร.วิชญะ ต่อวงศ์ไพชยนต์ กรรมการ	
ผศ.ดร.อนันตพร หรรษคุณาฒัย อาจารย์ที่ปรึกษา	

หัวข้อสหกิจศึกษา	การพัฒนาการทดสอบอัตโนมัติสำหรับระบบถอนกองทุนและซิงค์คอนเนคเตอร์แอปพลิเคชัน Development of the automation testing suite for the redemption and sink connector systems
ชื่อนักศึกษา	นายกฤษฎา ชิตนุกูล รหัสนักศึกษา 62050127 นางสาวลภัสรดา พงศ์พันธ์พฤทธิ์ รหัสนักศึกษา 62050219
ปริญญา	วิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชา	วิทยาการคอมพิวเตอร์
คณะ	วิทยาศาสตร์
มหาวิทยาลัย	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (สจล.)
ปีการศึกษา	2565
อาจารย์ที่ปรึกษา	ผศ.ดร.อนันตพร หรรษคุณาฒัย

บทคัดย่อ

สหกิจศึกษาครั้งนี้ได้มีโอกาสร่วมทำงานกับเอสเอสแอนด์ซี เทคโนโลยี จำกัดซึ่งเป็นบริษัทผู้ให้บริการด้านการเงินและการลงทุนสำหรับอุตสาหกรรมบริการทางการเงิน อีกทั้งยังเป็นเจ้าของและการพัฒนาเทคโนโลยีสำหรับบัญชีหลักทรัพย์และมีการวิเคราะห์ด้านประสิทธิภาพและความเสี่ยง โดยทางบริษัทมีโปรเจคไมโครเซอร์วิส (Mircroservices) สำหรับการซื้อขายกองทุนและในโปรเจคนี้ถูกสร้างขึ้นด้วยเฟรมเวิร์ค Spring boot และทดสอบโดยใช้เครื่องมือ Cucumber และ Junit อีกทั้งยังใช้แนวคิด CI/CD ในการทดสอบ โดยในการพัฒนาระบบนั้นควรมีการทดสอบอัตโนมัติเพื่อทำให้ระบบมีความสมบูรณ์มากยิ่งขึ้น มีขอบเขตการทดสอบอยู่ที่ระบบถอนกองทุนและซิงค์คอนเนคเตอร์แอปพลิเคชัน ทดสอบโดยการจำลองกรณีต่าง ๆ ในเซอร์วิส ผลของการทดสอบนั้นจากการทดสอบอัตโนมัติได้ว่าระบบทำงานได้ตามที่คาดหวังไว้ทุกกรณี

คำสำคัญ : การทดสอบอัตโนมัติ, ซิงค์คอนเนคเตอร์แอปพลิเคชัน, ไมโครเซอร์วิส, ระบบการถอนกองทุน

Title	Development of the automation testing suite for the redemption and sink connector systems
Students	Mr. Kritsada Chitnukool Student ID 62050127 Miss Lapatlada Phonphanparet Student ID 62050219
Degree	Bachelor of Science (Computer Science)
Department	Computer Science
Faculty	Science
University	King Mongkut's Institute of Technology Ladkrabang (KMITL)
Academic Year	2022
Advisor	Asst.Prof.Dr. Anantaporn Hanskunatai

Abstract

In this internship, we had the opportunity to work with SS&C Technologies, a financial and investment services company. They are a provider of financial services for the financial services industry and develop technology for securities accounts with performance analysis and risk assessment. We worked on microservices that can redemption and exchange. This project was developed using the Spring Boot framework. The testing for this project was done using Cucumber and JUnit as testing tools, and it followed the CI/CD approach for testing. The focus of the system development was on automated testing to ensure a high level of system completeness. The testing scope included redemption service and sink connector application. The tests were conducted by simulating various scenarios within the services. The results of the automated testing showed that the system performed as expected in all cases.

Keywords : Automate Testing, Sink Connector Application, Microservices, Redemption Service

กิตติกรรมประกาศ

สหกิจศึกษาโครงการการพัฒนาการทดสอบอัตโนมัติสำหรับระบบถอนกองทุนและซิงค์คอนเนคเตอร์แอปพลิเคชัน สามารถดำเนินการจนสำเร็จลุล่วงด้วยดี เนื่องจากได้รับความช่วยเหลือและสนับสนุนจากบุคคลหลายท่าน ซึ่งทางผู้จัดทำขอขอบพระคุณผู้มีพระคุณทุกท่านดังนี้

ขอขอบพระคุณ ผศ.ดร.อนันตพร หรรษคุณาฒัย อาจารย์ที่ปรึกษา และ ดร.วิชญะ ต่อวงศ์ไพชยนต์ กรรมการ ที่เสียสละเวลาในการตรวจสอบเนื้อหาเพื่อแก้ไข ให้คำปรึกษา และให้การช่วยเหลือตลอดการทำสหกิจศึกษาครั้งนี้

ขอขอบพระคุณพี่ ๆ ทุกคนจากทางบริษัท SS&C ที่คอยให้คำแนะนำ และความช่วยเหลือในทุก ๆ ปัญหาที่เกิดขึ้นตลอดการทำสหกิจศึกษาครั้งนี้

ขอขอบพระคุณ บิดา มารดา และสมาชิกในครอบครัวของผู้จัดทำ ที่คอยสนับสนุน ช่วยเหลือและให้กำลังใจตลอดการทำสหกิจศึกษาครั้งนี้

สุดท้ายนี้ผู้จัดทำหวังเป็นอย่างยิ่งว่าสหกิจศึกษาเรื่องนี้จะเป็นประโยชน์แก่ผู้ที่สนใจ และสามารถนำไปพัฒนาต่อยอดในอนาคตได้

กฤษฎา ชิตนุกูล
ลภัสรดา พงศ์พันธ์พฤทธิ์

สารบัญ

หน้า

บทคัดยาษาไทย.....ก

บทคัดย่อภาษาอังกฤษ.....ข

กิตติกรรมประกาศ.....ค

สารบัญ.....ง

บทที่ 1 บทนำ.....1

1.1 ความเป็นมาและความสำคัญของปัญหา.....1

1.2 วัตถุประสงค์.....1

1.3 ขอบเขต.....1

1.4 ประโยชน์ที่คาดว่าจะได้รับ.....2

บทที่ 2 เทคโนโลยีและทฤษฎีที่เกี่ยวข้อง.....3

2.1 เทคโนโลยีที่เกี่ยวข้อง.....3

2.1.1. Kafka.....3

2.1.2. Spring Boot.....4

2.1.3. Postman.....4

2.1.4. PostgreSQL.....5

2.1.5. Cucumber.....6

2.1.6. Junit.....7

2.1.7. H2.....8

2.1.8. Git Extension.....8

2.2 ทฤษฎีที่เกี่ยวข้อง.....12

2.2.1. การทดสอบซอฟต์แวร์ (Software testing).....12

2.2.2. Continuous Integration/Continuous Delivery (CI/CD).....14

บทที่ 3 ขั้นตอนการดำเนินงาน....17
3.1 แผนภาพแสดงขั้นตอนการทำงานของระบบ (System Flow Diagram)....17
3.2 แผนภาพแสดงการทำงานของผู้ทดสอบ (Use Case Diagram)....17
3.3 แผนภาพกิจกรรม (Activity Diagram)....21
3.3.1. Activity Diagram ของการทดสอบระบบถอนกองทุนกรณีที่ถูกต้องและสามารถถอนกองทุนได้....21
3.3.2. Activity Diagram ของการทดสอบระบบถอนกองทุนกรณีที่ไม่ถูกต้องและไม่สามารถถอนกองทุนได้....23
3.3.3. Activity Diagram ของการทดสอบซิงค์คอนเนคเตอร์....25
บทที่ 4 ผลการดำเนินงาน....27
4.1 การถอนกองทุน....27
4.2 ซิงค์คอนเนคเตอร์....45
บทที่ 5 สรุปผลการดำเนินงานและข้อเสนอแนะ....58
5.1 สรุปผลการดำเนินงาน....58
5.2 ข้อเสนอแนะ....58
เอกสารอ้างอิง....58

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เอสเอสแอนด์ซี เทคโนโลยี จำกัดมีโปรเจคในการสร้างไมโครเซอร์วิส (Microservices) ใหม่สำหรับการดำเนินการทางการเงินในระบบซื้อ-ขายกองทุนและมีการส่งและรับข้อความผ่าน Kafka ซึ่งไมโครเซอร์วิส คือ เทคนิคในการออกแบบซอฟต์แวร์รูปแบบหนึ่งที่จะทำให้แต่ละระบบมีขนาดเล็กและสามารถนำไปใช้งาน ปรับขนาดของโปรเจคได้อย่างอิสระ (แต่ละไมโครเซอร์วิสไม่จำเป็นต้องเชื่อมโยงกันทั้งหมด) อีกทั้งแต่ละไมโครเซอร์วิสสามารถเขียนโดยคนละภาษาได้ ส่วน Kafka คือ ระบบที่ใช้สำหรับกระจายการส่งข้อมูลแบบทันทีระหว่างแอปพลิเคชันและเซิร์ฟเวอร์ โดยธุรกรรมทางการเงินในโปรเจคประกอบไปด้วยการถอนกองทุนและการย้ายเงินระหว่างกองทุน ซึ่งการถอนกองทุนจะต้องมีเงื่อนไขต่าง ๆ เช่น ต้องมีการตรวจสอบว่าบัญชีที่ต้องการจะถอนมีจริงไหม ตรวจสอบว่าเงินในบัญชีที่ต้องการจะถอนมีจำนวนเงินเพียงพอหรือไม่ เช่นเดียวกับการย้ายเงินระหว่างกองทุนต้องมีเงื่อนไข เช่น ต้องมีการตรวจสอบว่ากองทุนต้นทางมีจริงหรือไม่ กองทุนปลายทางมีจริงหรือไม่ โดยระบบการกองทุนนั้นจะมีส่วนซิงค์คอนเนคเตอร์ซึ่งเป็นส่วนที่อ่านข้อมูลจาก Kafka และบันทึกลงในฐานข้อมูลของระบบการถอนกองทุน

การพัฒนาไมโครเซอร์วิสนี้จะสมบูรณ์จะต้องประกอบไปด้วยเซอร์วิสที่ใช้งานได้ ตัวทดสอบของเซอร์วิสนั้น ๆ และการทดสอบซอฟต์แวร์ตั้งแต่ต้นจนจบ โดยตัวทดสอบของเซอร์วิสนั้นในตอนเริ่มต้นของโปรเจคมีการทดสอบแบบแมนนวล เนื่องจากต้องส่งมอบงานเพื่อให้ตรงตามกำหนดและตามข้อกำหนด เมื่อมีการพัฒนาระบบได้ตรงตามข้อกำหนดในระดับหนึ่งแล้ว จึงต้องการการทดสอบแบบอัตโนมัติเพื่อลดระยะเวลาในการทดสอบ โดยการทดสอบแบบอัตโนมัติ คือ การทดสอบที่สามารถทำงานได้ด้วยตนเอง

1.2 วัตถุประสงค์

1. พัฒนาการทดสอบแบบอัตโนมัติเพื่อให้ระบบซื้อ-ขายกองทุนมีความสมบูรณ์มากขึ้น
2. ลดขั้นตอนและระยะเวลาในการพัฒนาระบบ

1.3 ขอบเขต

พัฒนาการทดสอบแบบอัตโนมัติในส่วนระบบการถอนกองทุนและซิงค์คอนเนคเตอร์

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. ประโยชน์ต่อตนเอง
   1.1. เรียนรู้ เข้าใจเครื่องมือต่าง ๆ ที่ใช้ในโครงงานและสามารถประยุกต์ใช้ในการทำงานจริงได้
   1.2. ฝึกฝนทักษะการทำงานร่วมกันเป็นทีม
   1.3. เรียนรู้การทำงานแบบ Agile/Scrum
   1.4. เรียนรู้ทักษะการเขียนโปรแกรม
2. ประโยชน์ต่อบริษัท
   2.1. ลดขั้นตอนในการพัฒนาระบบ เนื่องจากมีการทดสอบแบบอัตโนมัติเข้ามาช่วย
   2.2. เพื่อลดปัญหาและเวลาในขั้นตอนการส่งต่อระบบการทดสอบแบบอัตโนมัติจะรับรองว่าเซอร์วิสที่ผ่านการทดสอบแล้วจะไม่ส่งผลกระทบต่อเซอร์วิสอื่น ๆ
   2.3. สามารถนำข้อผิดพลาดในการทดสอบระบบไปพัฒนาระบบให้มีประสิทธิภาพมากขึ้น

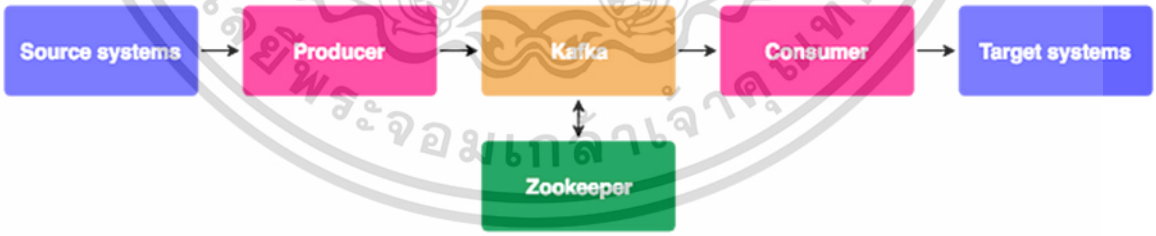
บทที่ 2
เทคโนโลยีและทฤษฎีที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงเทคโนโลยีและทฤษฎีที่เกี่ยวข้อง โดยเทคโนโลยีที่เกี่ยวข้องในการพัฒนาระบบงาน คือ Kafka, Spring Boot, Postman, PostgreSQL, Cucumber, Junit, H2 และ Git Extension สุดท้ายจะกล่าวถึงทฤษฎีที่เกี่ยวข้องในการพัฒนาระบบงาน คือ การทดสอบซอฟต์แวร์ (Software testing) และ CI/CD

2.1 เทคโนโลยีที่เกี่ยวข้อง

2.1.1. Kafka

Apache Kafka คือ แพลตฟอร์มการรับและส่งข้อมูลหรือข้อความระหว่างคอมพิวเตอร์หลาย ๆ เครื่องที่กระจายอยู่ต่างที่กัน โดยมักส่งข้อมูลแบบไม่ต้องรอให้งานอื่น ๆ เสร็จก่อนที่จะทำงานอื่น ๆ ต่อไป หรือก็คือสามารถทำงานพร้อมกันได้หลาย ๆ กิจกรรมโดยไม่จำเป็นต้องรอกัน (Asynchronous) ที่ต้องการให้แพลตฟอร์มผลิตธุรกรรมได้มากในช่วงเวลาหนึ่ง (High-throughput) และมีความเร็วในการตอบสนองต่ำ (Low-latency) สำหรับจัดการกับข้อมูลที่ถูกสร้างขึ้นในระยะเวลานั้นได้อย่างมีประสิทธิภาพ โดยสามารถจัดการกับข้อมูลได้อย่างรวดเร็ว (Real-time Processing) มีการเก็บข้อมูลเพื่อให้ทนต่อความผิดพลาด (Fault-tolerance Durable) และส่งข้อมูลจากผู้ส่ง (Publisher หรือ Producer) ไปยังผู้รับ (Subscriber หรือ Consumer) โดยสามารถส่งให้ผู้รับ (Subscriber) กี่ตัวก็ได้



รูปที่ 2.1 ระบบนิเวศน์ (Ecosystem) ของ Kafka
ที่มา : https://phayao.medium.com/สร้าง-microservices-ด้วย-spring-boot-spring-cloud-b3ea6f4a8b92

จากรูปที่ 2.1 เป็นระบบนิเวศน์ของ Kafka หรือหลักการทำงานคร่าว ๆ ของ Kafka นั่นเอง โดยจะแบ่งออกเป็นขาเข้าและขาออก

ขั้นตอนการทำงานของขาเข้าจะเริ่มจากแหล่งที่มาของข้อมูล (Source systems) อาจจะเป็นฐานข้อมูลหรือบันทึกข้อมูลต่าง ๆ ที่เกิดขึ้นในระบบคอมพิวเตอร์หรือแอปพลิเคชันต่าง ๆ (Logging) จากนั้นผู้ส่ง (Producer) จะทำการดึงข้อมูลจากแหล่งที่มาของข้อมูลนำเข้าไปยัง Kafka

ขั้นตอนการทำงานของขาออกจะเริ่มจากมีระบบเป้าหมาย (Target systems) ที่ต้องการข้อมูล จากนั้นผู้รับ (Consumer) จะทำการดึงข้อมูลมาจาก Kafka ส่งต่อไปยังระบบเป้าหมาย โดยข้อมูลใน Kafka ก็ยังคงอยู่

สุดท้ายคือ ผู้จัดการ (Zookeeper) ทำหน้าที่จัดการว่าควรไปอ่านที่ไหนแล้วหลังจากอ่านแล้วควรไปอ่านข้อมูลที่ไหนต่อ โดยตั้งแต่ Kafka เวอร์ชัน 4.0 จะไม่จำเป็นต้องใช้ Zookeeper ในการใช้ Kafka

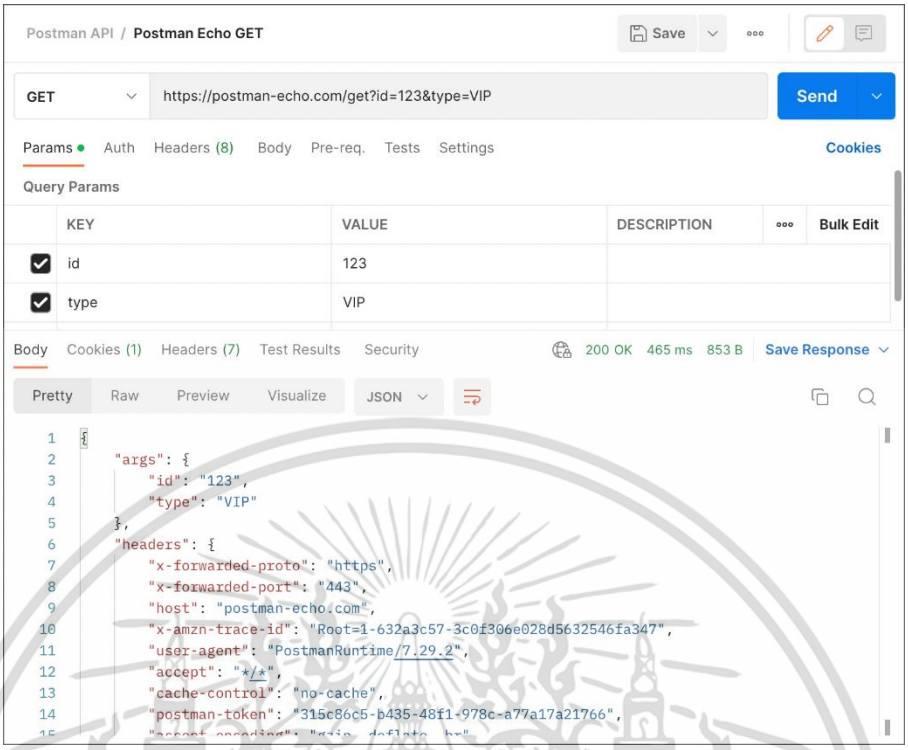
2.1.2. **Spring Boot**

Spring เป็นเฟรมเวิร์คหนึ่งที่สามารถทำให้สร้างเว็บแอปพลิเคชันหรือเว็บเซอร์วิสได้ง่ายขึ้น เนื่องจากมีจาวาเว็บเซิร์ฟเวอร์ที่สร้างมากับ Spring Boot โดยมีพอร์ตเริ่มต้นคือ 8080

เนื่องจาก Spring เป็นแพลตฟอร์มที่รวมหลาย ๆ เฟรมเวิร์คเข้าด้วยกัน ทำให้รองรับการพัฒนาได้หลายด้าน เช่น เว็บเฟรมเวิร์ค, เฟรมเวิร์คสำหรับการทดสอบทำให้มีความยุ่งยากในการตั้งค่าใช้งานอย่างมาก ดังนั้นผู้พัฒนา Spring จึงพัฒนา Spring Boot ขึ้นเพื่อแก้ปัญหาเรื่องความซับซ้อนให้เป็นอัตโนมัติมากยิ่งขึ้น ทำให้ผู้พัฒนาไม่จำเป็นต้องตั้งค่าทุกอย่างด้วยตัวเอง

2.1.3. **Postman**

Postman คือ เครื่องมือสำหรับทดสอบเซอร์วิส (API Service) โดย API นั้นย่อมากจาก Application Programming Interface โดย Postman สามารถจำลองส่วนของข้อมูลที่ถูกส่งมาพร้อมกับคำขอ (Request Body) หรือส่วนที่แจ้งรายละเอียดเกี่ยวกับคำขอนั้น (Header) เพื่อทดสอบ API ว่าทำงานได้อย่างถูกต้องหรือไม่ อีกทั้งยังสามารถบันทึกเป็นคอลเลกชันที่ใช้ในการทดสอบ ทำให้สามารถเรียกใช้คำขอที่ต้องการได้ตามตัวอย่างหน้าโปรแกรมรูปที่ 2.2



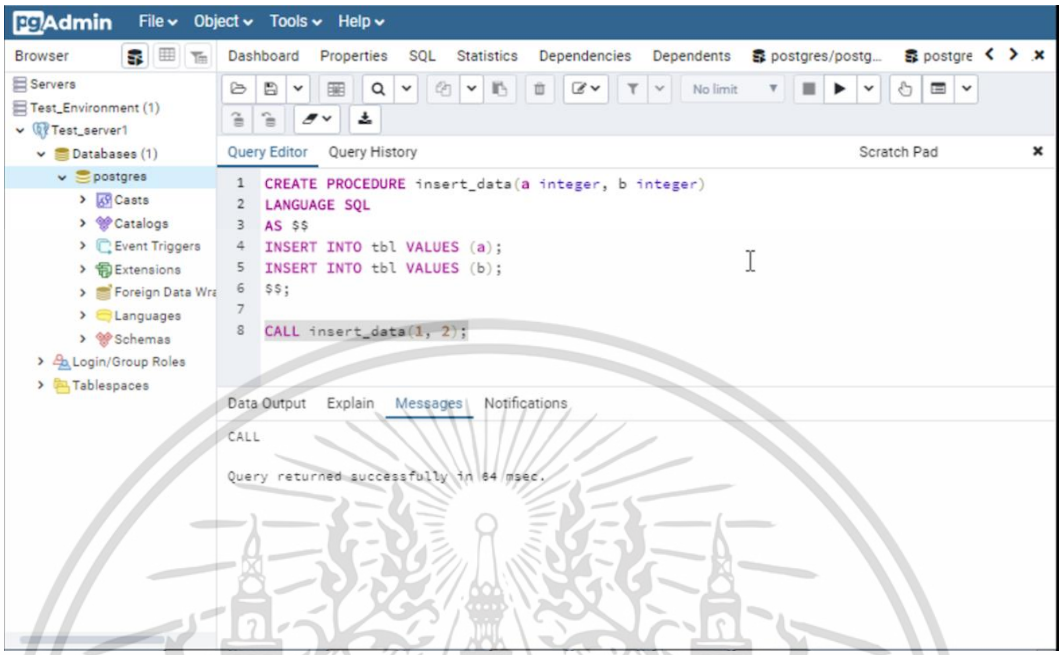
รูปที่ 2.2 ตัวอย่างหน้าโปรแกรมของ Postman

ที่มา : https://learning.postman.com/docs/sending-requests/requests/

โดย Postman สามารถจำลองการส่งข้อมูลจากฐานข้อมูลมาแสดงผลหรือที่เรียกว่า POST และสามารถจำลองการรับหรือดึงข้อมูลจากฐานข้อมูลมาแสดงผลหรือที่เรียกว่า GET

2.1.4. PostgreSQL

PostgreSQL เป็นระบบฐานข้อมูล SQL เชิงวัตถุสัมพันธ์ (Object-Relational) ที่มีประสิทธิภาพและยืดหยุ่น ซึ่ง PostgreSQL ได้รับความนิยมในด้านความน่าเชื่อถือ, ความทนทานและมีประสิทธิภาพสูง โดย PostgreSQL มีความยืดหยุ่นทั้งในด้านปริมาณข้อมูลที่สามารถจัดเก็บและจัดการได้ รวมไปถึงด้านการรองรับจำนวนผู้ใช้งานที่ใช้งานพร้อมกัน อีกทั้งยังสามารถทำงานได้หลายระบบปฏิบัติการ เช่น Linux, Unix และ Windows และเป็นระบบฐานข้อมูลที่ทันสมัยที่สุดของ Open Source ที่สามารถนำไปใช้งานได้โดยไม่มีค่าใช้จ่ายตามตัวอย่างหน้าโปรแกรมของเครื่องมือ PostgreSQL ที่ชื่อว่า pgAdmin ตามรูปที่ 2.3



รูปที่ 2.3 ตัวอย่างหน้าโปรแกรมของเครื่องมือ PostgresSQL

ที่มา : https://www.enterprisedb.com/postgres-tutorials/pgadmin-comparable-tool-plsql-developer-postgresql

2.1.5. **Cucumber**

Cucumber คือ เครื่องมือที่ใช้สำหรับออกแบบและทำการทดสอบโปรแกรมอัตโนมัติ ที่ใช้สร้างสคริปต์ที่ใช้ทดสอบระบบแบบ BDD (Behavior Driven Development) คือเขียนอยู่ในรูปแบบภาษาที่เข้าใจง่ายสำหรับทุกคน หลีกเลี่ยงคำศัพท์เทคนิคให้มากที่สุดเพื่อจะให้เหมาะกับการนำเสนอแก่ผู้ที่ไม่ค่อยเข้าใจด้านเทคนิคเมื่อมาอ่านก็สามารถเข้าใจได้โดยง่าย เนื่องจากในโครงสร้างของโค้ดที่ใช้ทดสอบนั้นจะใช้ Gherkin Syntax เป็นภาษาสำหรับบรรยายความต้องการของลูกค้าและเกณฑ์การตรวจสอบว่าระบบหรือฟีเจอร์ที่พัฒนาต้องปฏิบัติตามเงื่อนไขและความต้องการที่กำหนดไว้หรือไม่ ซึ่งเป็นส่วนหนึ่งของ User Story หรือความต้องการของลูกค้า (Acceptance Criteria) ในรูปแบบที่คล้ายภาษามนุษย์ซึ่งช่วยให้เข้าใจคำขอต่าง ๆ ได้ง่าย ทั้งในเชิงธุรกิจและเทคนิคโดยมีองค์ประกอบดังนี้

1. Scenario: อธิบายสถานการณ์ของการใช้ระบบ
2. Given: เป็นขั้นตอนที่ที่ต้องเกิดขึ้นเพื่อนำระบบไปอยู่ในสถานะที่ต้องการ ก่อนที่ผู้ใช้งานจะมีการกระทำใดๆ กับระบบ ส่วนของ Given สามารถมีหลายข้อได้และเชื่อมกันด้วย And

3. When: ตั้งเงื่อนไขเมื่อผู้ใช้กระทำบางสิ่งกับระบบตามเงื่อนไขที่กำหนด When สามารถมีหลายข้อได้และเชื่อมกันด้วย And
4. Then: ผลลัพธ์ที่เกิดขึ้น มักจะกล่าวถึงสิ่งที่เราต้องตรวจสอบ เช่น ข้อความ, ผลลัพธ์ เป็นต้น และยังรวมถึงระบบภายนอกที่เกี่ยวข้องด้วย เช่น หากมีการส่งข้อมูลไปยังระบบอื่น เป็นต้น

และ Cucumber ประกอบด้วย 2 ส่วนหลักๆ คือ

1. Feature File: เป็นไฟล์ที่จะเขียนในเชิงธุรกิจหรือจากความต้องการของผู้ใช้โดยใช้ภาษา Gherkin
2. Step Definitions: การที่ Feature File จะทำงานได้นั้นจะต้องอาศัย Step Definition เป็นตัวกำหนดการทำงานของสคริปต์ซึ่งในแต่ละขั้นตอนนั้นจะเขียนเป็นภาษาโปรแกรมมิ่งจึงต้องอาศัยคนที่เข้าใจโค้ดสร้างสคริปต์ขึ้นมาหรือการแปลงขั้นตอนการทดสอบใน Feature File ให้กลายเป็นโค้ด

2.1.6. **Junit**

เป็นไลบรารีสำหรับการเขียนการทดสอบฟังก์ชัน (Unit Test) ทำให้มีโครงสร้างในการเขียนการทดสอบฟังก์ชันที่ดี โดยมีการอธิบายว่าในแต่ละกรณีของการทดสอบ (Test Case) นั้นทำอะไร เพื่อลดเวลาในการอ่านหรือทำความเข้าใจว่าต้องการทดสอบอะไร อีกทั้งยังเพื่อให้ในแต่ละกรณีของการทดสอบทำการทดสอบส่วนใดส่วนหนึ่งเท่านั้น

โดยหลักการของการเขียนการทดสอบฟังก์ชันที่ดีโดยอาศัยไลบราลี Junit คือ 1 กรณีของการทดสอบต่อ 1 Method และหากในหลาย ๆ กรณีการทดสอบมีการใช้โค้ดที่ซ้ำกัน ผู้พัฒนาสามารถลดโค้ดในส่วนที่ซ้ำกันได้ด้วยการติดตั้งข้อมูลเริ่มต้นด้วยการใช้ @Before ได้ดังรูปที่ 2.4

```
1   FizzBuzz fizzBuzz;
2
3   @Before
4   public void setUp() {
5           FizzBuzz = new FizzBuzz();
6   }
7
8   @Test
9   public void sayOne() {
10          String result = fizzBuzz.say(1);
11          assertEquals("1", actual);
12  }
13
14  @Test
15  public void sayTwo() {
16          String result = fizzBuzz.say(2);
17          assertEquals("2", actual);
18  }
```

รูปที่ 2.4 ตัวอย่างการใช้ @Before

ที่มา : https://www.somkiat.cc/junit-good-struture/

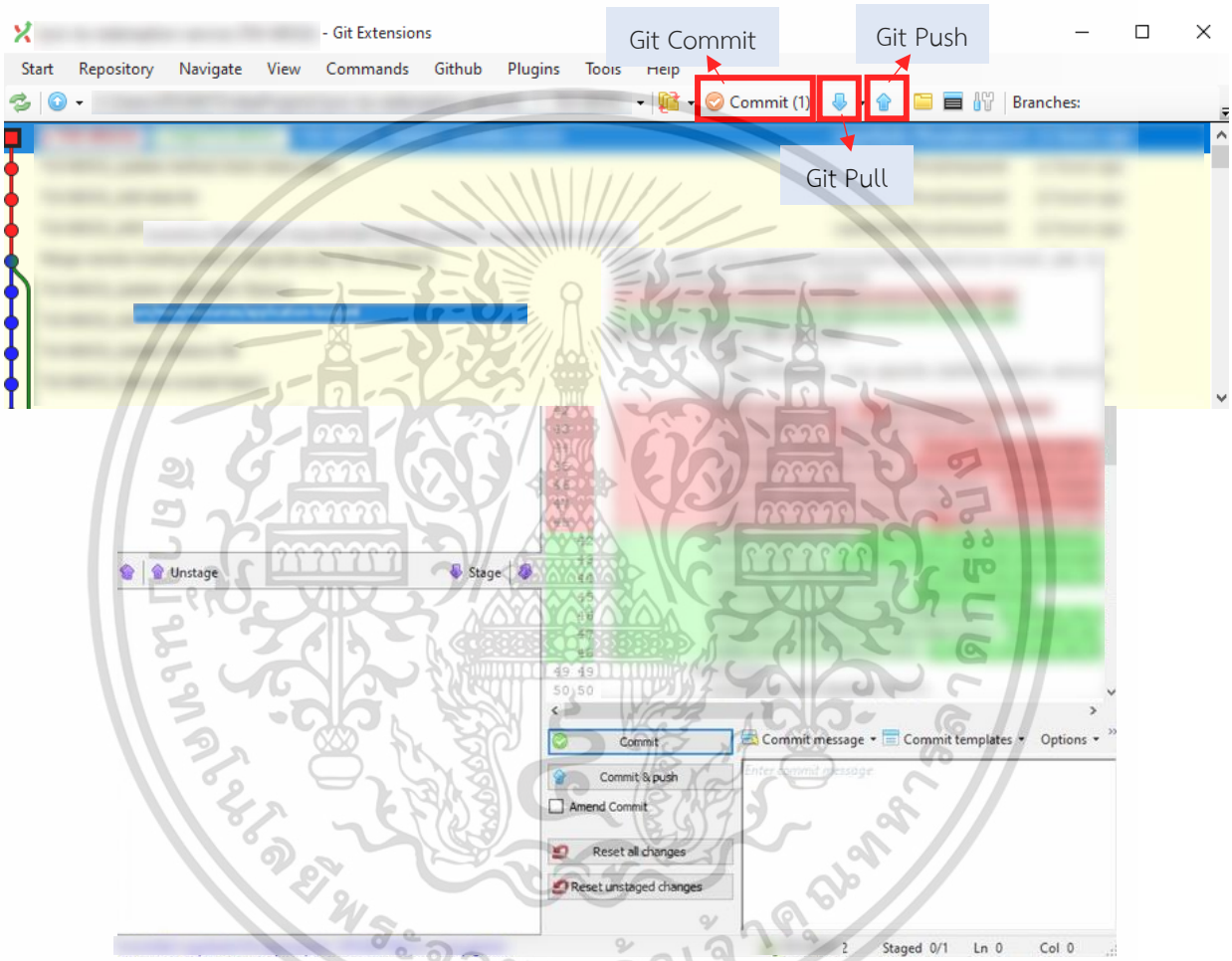
2.1.7. **H2**

เป็นซอฟต์แวร์ฐานข้อมูลเชิงสัมพันธ์ (Relational Database) ที่ Spring Boot เลือกใช้เป็นค่าตั้งต้นรองรับการใช้งานภาษา SQL ในการติดต่อกับฐานข้อมูล รองรับ JDBC API รองรับการใช้งานทั้งแบบเซิร์ฟเวอร์และแบบแยกส่วนรองรับการใช้งานแบบโหลดฐานข้อมูลทั้งหมดไว้ในหน่วยความจำ (In-Memory Database) มีแอปพลิเคชันที่ใช้จัดการฐานข้อมูลที่สามารถทำงานผ่านบราวเซอร์ได้ ในการทำงาน H2 จะโหลดฐานข้อมูลทั้งหมดไว้ในหน่วยความจำนั้น ข้อมูลในระหว่างที่เราใช้งานแอปพลิเคชันจะเก็บอยู่ในหน่วยความจำทั้งหมด และจะหายไปเมื่อปิดแอปพลิเคชัน

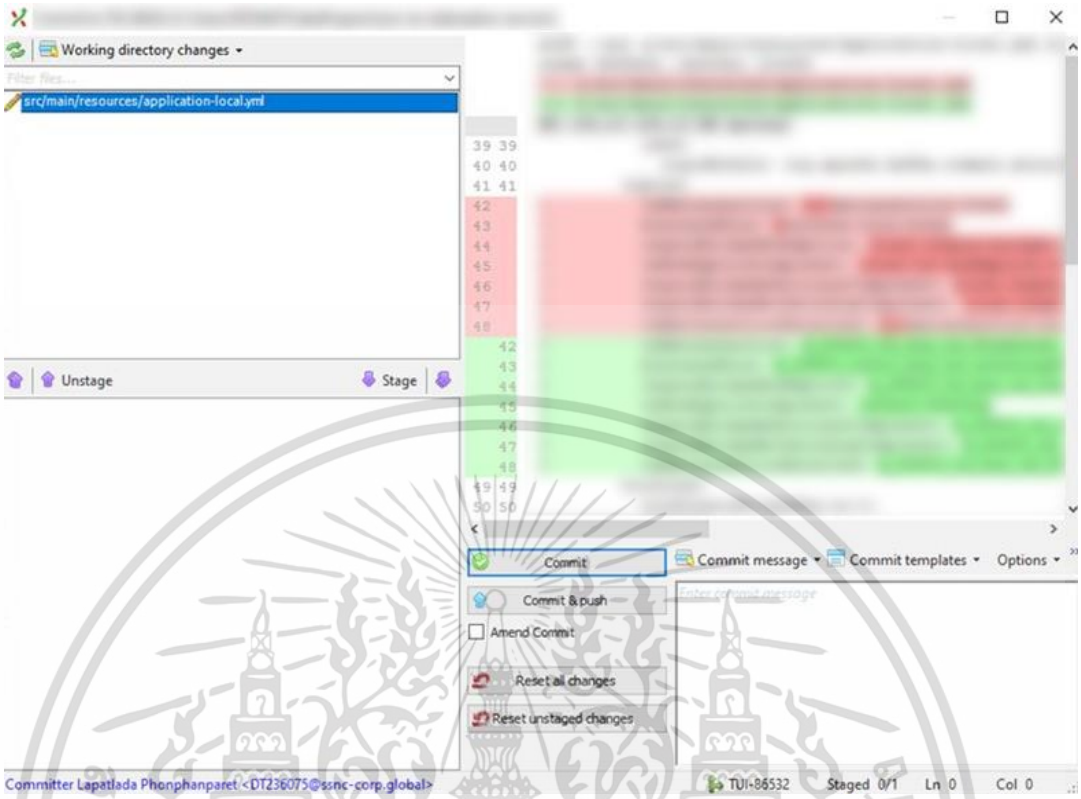
2.1.8. **Git Extension**

เป็นเครื่องมือที่ช่วยจัดการเวอร์ชันของโค้ดโดยจะเก็บประวัติว่าไฟล์แต่ละไฟล์ถูกสร้าง/ลบและแก้ไขโดยใคร เมื่อไหร่และอย่างไร ทำให้เราสามารถติดตามการเปลี่ยนแปลงของโค้ดได้ตลอด หรือหากต้องการย้อนกลับไปใช้เวอร์ชันเก่าของโค้ดเราก็สามารถทำได้เช่นกัน

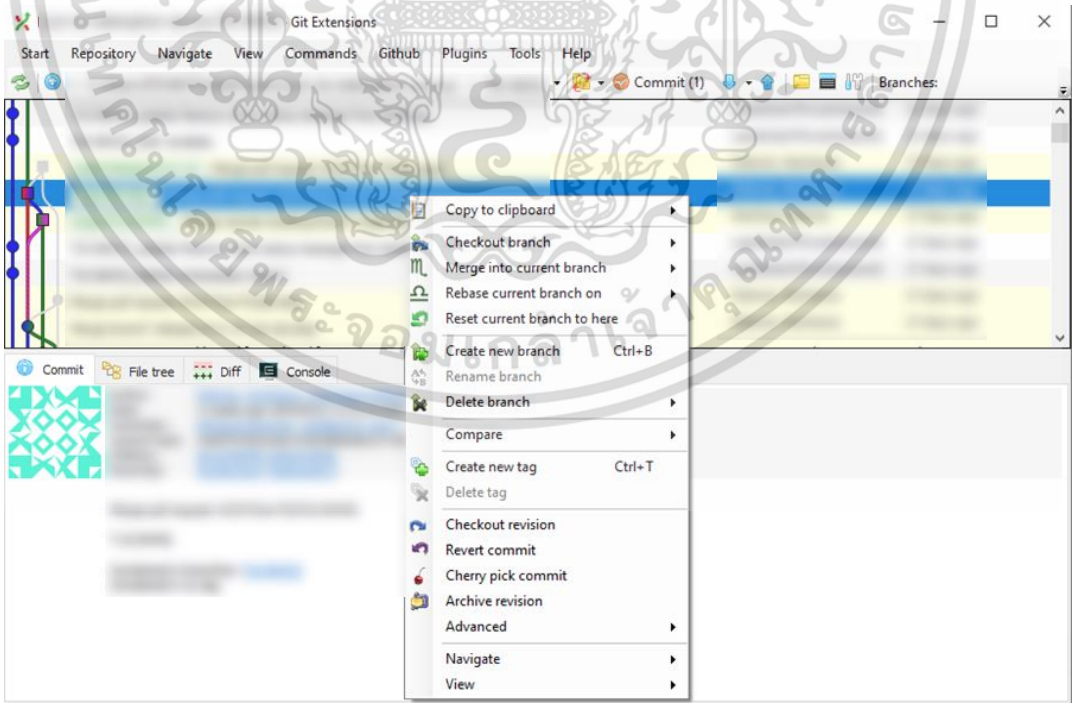
การใช้งาน Git Extension จะต้องทำการโคลนรีโพสิทอรี (Clone Repository) ที่ต้องการจาก GitHub มายังเครื่องและเมื่อทำการสร้าง/ลบ/แก้ไขโค้ดในรีโพสิทอรีที่ต้องการเรียบร้อยและต้องการอัพเดตโค้ดเวอร์ชันใหม่ในสำเนาของโค้ด (Branch) ที่ต้องการ Git Extension ก็จะมีฟังก์ชันพื้นฐานต่าง ๆ เช่น



รูปที่ 2.5 ตัวอย่างหน้าต่างโปรแกรม Git Extension ก่อนสั่ง Git Commit



รูปที่ 2.6 ตัวอย่างหน้าต่างโปรแกรม Git Extension ก่อนสั่ง Git Commit



รูปที่ 2.7 ตัวอย่างหน้าต่างโปรแกรม Git Extension เมื่อต้องการสั่ง Git Merge

จากรูปที่ 2.5 จะเป็นตัวอย่างหน้าต่างโปรแกรม Git Extension โดยใน Git Extension จะมีคำสั่งพื้นฐานต่าง ๆ ดังนี้

1. Git Pull เป็นการดึงไฟล์ที่มีการเปลี่ยนแปลงมาอัพเดตในเครื่อง
2. Git Commit ก่อนที่จะมีการสั่ง Git Commit จะต้องมีการเพิ่มไฟล์ที่ต้องการจะเปลี่ยนแปลงเข้าไปใน Stage ตามรูป 2.6 เมื่อสั่ง Git Commit จะเป็นการยืนยันไฟล์ที่อยู่ใน Stage นั้น ๆ
3. Git Push เป็นการส่งไฟล์ที่ Commit เข้าสู่รีโพสิทอรี
4. Git Merge เป็นการรวม 2 Branch เข้าด้วยกัน สามารถทำได้โดยการคลิกขวาที่ Branch หลัก ตามรูป 2.7 และเลือก Merge into Current Branch

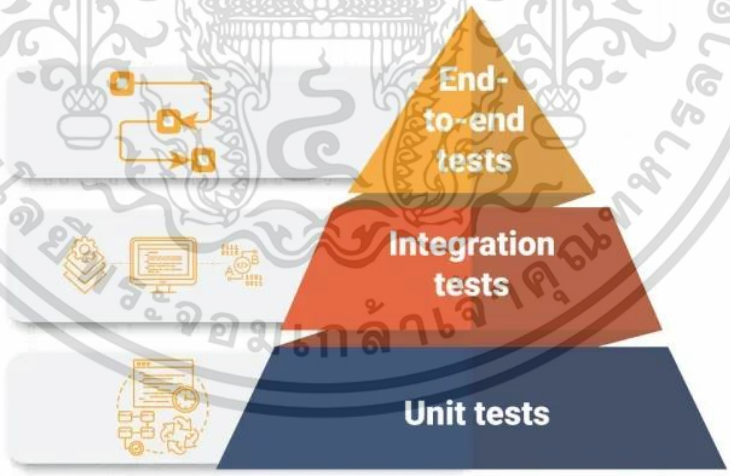
ตารางที่ 2.1 ตารางแสดงการใช้งานเครื่องมือ

Tool	การนำไปใช้งาน
Kafka	ระบบการถอนกองทุนและซิงค์คอนเนคเตอร์มีการส่งและรับข้อความ (Message) หรือข้อมูลกับเซอร์วิสต่าง ๆ โดยอาศัย Kafka โดยในการทดสอบเซอร์วิสการถอนกองทุนและซิงค์คอนเนคเตอร์นั้นจะทำการจะลองข้อความที่จะส่งไปยัง Kafka Topic และรอรับข้อความที่ตอบกลับมาจาก Kafka Topic จากนั้นทำการตรวจสอบว่าข้อความที่ได้รับมา (Actual Result) นั้นตรงกับที่ควรจะเป็น (Expect Result) หรือไม่
Spring Boot	ระบบการถอนกองทุนและซิงค์คอนเนคเตอร์รวมถึงเซอร์วิสต่าง ๆ ที่เกี่ยวข้องสร้างด้วย Spring Boot ดังนั้นก่อนที่จะสร้างสคริปต์ในการทดสอบแอปพลิเอชันนั้นจะต้องทำความเข้าใจ Spring Boot เพื่อเข้าใจถึงลำดับการทำงานของเซอร์วิสที่ทดสอบทั้งสอง
Postman	ใช้สำหรับการจำลองและทดสอบการส่ง API เพื่อรอรับผลลัพธ์ที่ควรได้ (Expect Result) จากการตอบกลับจาก Kafka Topic จากนั้นนำผลลัพธ์ที่ได้ไปตรวจสอบในการทดสอบเซอร์วิสว่าข้อความที่ได้จากการทดสอบ (Actual Result) นั้นตรงกับผลลัพธ์ที่ได้จากการทดสอบการส่ง API (Expect Result) หรือไม่
PostgreSQL	ใช้สำหรับการทดสอบ ในระบบการถอนกองทุนเมื่อทุกกระบวนการของแอปพลิเคชันเสร็จสิ้นแล้วจะส่งผลลัพธ์ไปยังฐานข้อมูล PostgreSQL การจะทำการ

	ทดสอบระบบได้จะต้องทำการดึงข้อมูลที่คาดว่าจะได้รับ (Expect Result) มาเพื่อทำการเปรียบเทียบกับผลที่ได้จากการทดสอบ (Actual Result)
Cucumber	ใช้สำหรับการเขียนสคริปต์การทดสอบ
Junit	ใช้ไลบรารีประกอบการเขียนสคริปต์การทดสอบ เช่น การติดตั้งเครื่องมือพื้นฐานในการทดสอบ การแยกส่วนสคริปต์การทดสอบโดยอาศัยเครื่องมือของไลบรารี Junit
H2	ใช้ในการสร้างฐานข้อมูลในการทดสอบ โดยจะถูกสร้างขึ้นเมื่อแอปพลิเคชันเริ่มทำงาน เมื่อการทดสอบเสร็จสิ้น ฐานข้อมูลที่ถูกสร้างโดย H2 จะถูกลบทิ้งไป จะใช้ในส่วนการทดสอบซิงค์คอนเนคเตอร์จำลองการบันทึกลงฐานข้อมูล
Git Extension	เป็นเครื่องมือในการทำงานร่วมกับสมาชิกในทีม โดยเมื่อมีการเปลี่ยนแปลงโค้ดในแอปพลิเคชัน หากต้องการอัพเดตความเปลี่ยนแปลงกับสมาชิกในทีมก็จะใช้ Git Extension ในการอัพเดตความเปลี่ยนแปลง

2.2 ทฤษฎีที่เกี่ยวข้อง

2.2.1. **การทดสอบซอฟต์แวร์ (Software testing)**



รูปที่ 2.8 Testing Pyramid

ที่มา : https://www.headspin.io/blog/the-testing-pyramid-simplified-for-one-and-all

จากรูปที่ 2.8 แสดงให้เห็นถึงพีรามิดการทดสอบเป็นคอนเซ็ปต์ในการทดสอบซอฟต์แวร์ โดยจะแบ่งการทดสอบออกเป็น 3 ระดับ ดังนี้

1. การทดสอบฟังก์ชัน (Unit Tests)

เป็นการทดสอบฟังก์ชันอย่างมีขอบเขตในระดับ Unit หรือหน่วยที่เล็กที่สุด เป็นการทดสอบแบบเดี่ยว ต้องไม่มีพ่วงการทำงานของส่วนอื่น ๆ เข้ามาเกี่ยวข้องกับการทดสอบ เช่น ฐานข้อมูลหรือเซอร์วิสต่าง ๆ เช่น หากผู้พัฒนานั้นสร้าง Class ที่ประกอบด้วย Method ต่าง ๆ การทดสอบฟังก์ชันนั้นคือการทดสอบ Method เหล่านั้นว่าสามารถทำงานได้อย่างถูกต้องหรือไม่

ข้อดีของการทดสอบแบบการทดสอบฟังก์ชันคือ ทำให้การพัฒนาโปรแกรมทำได้เร็วขึ้น เนื่องจากผู้พัฒนาสามารถรู้ได้แน่ชัดว่าหน่วยใดทำงานผิดพลาด

ตัวอย่างของการทดสอบฟังก์ชัน คือ สมมุติว่าในระบบมีฟังก์ชันหนึ่งที่ทำการตรวจสอบว่าเป็นเลขคี่หรือไม่ โดยฟังก์ชันนี้ไม่ได้เกี่ยวข้องกับฟังก์ชันอื่นหรือมีการเชื่อมต่อกับเน็ตเวิร์คหรือฐานข้อมูลอื่น ๆ ฟังก์ชันนี้จะทำการรับค่าตัวเลข หากเป็นเลขคี่จะคาดหวังว่าจะได้รับค่า True เป็นการตอบกลับ แต่หากเลขที่ส่งเข้าไปเป็นเลขคู่จะคาดหวังว่าจะได้รับค่า False เป็นการตอบกลับ การทดสอบฟังก์ชันนั้นจะทำการทดสอบคำสั่งต่าง ๆ ภายในฟังก์ชันว่าทำงานได้ถูกต้องตรงตามที่คาดหวังไว้หรือไม่ในทุก ๆ กรณีที่เป็นไปได้

2. การทดสอบระบบโดยรวม (Integration Tests)

เป็นการทดสอบฟังก์ชันนั้น ๆ ทั้งฟังก์ชันโดยรวมถึงเซอร์วิสนั้นควรจะเชื่อมต่อเน็ตเวิร์ค, ฐานข้อมูลหรือเซอร์วิสอื่น ๆ ที่เซอร์วิสของเราควรทำได้ โดยการทดสอบในระดับนี้แสดงให้เห็นว่าระบบที่ทดสอบสามารถทำงานร่วมกันได้อย่างถูกต้อง

ข้อดีของการทดสอบระบบโดยรวม คือ เป็นรูปแบบการทดสอบที่ใกล้เคียงกับการทำงานจริง ๆ ของระบบ ทำให้มั่นใจได้ในระดับหนึ่งว่าระบบทำงานได้อย่างถูกต้อง

ตัวอย่างของการทดสอบระบบโดยรวม คือ สมมุติว่าในระบบมีฟังก์ชันหนึ่งที่มีการเชื่อมต่อกับฐานข้อมูล ฟังก์ชันนี้จะทำการบันทึกข้อมูลลงในฐานข้อมูล การทดสอบระบบโดยรวมนั้นจะทำการดึงข้อมูลจากฐานข้อมูลมาเพื่อตรวจสอบว่าข้อมูลที่ฟังก์ชันนี้บันทึกลงในฐานข้อมูลนั้นเป็นไปตามที่คาดหวังหรือไม่ในทุก ๆ กรณีที่เป็นไปได้

3. การทดสอบระบบตั้งแต่ต้นจนจบ (End-to-end Testing)

เป็นการทดสอบที่ใช้เวลาในการรันนานที่สุด เนื่องจากการทดสอบในระดับนี้จะทำการทดสอบเหมือนคนใช้งานจริง ๆ โดยจะใช้ทรัพยากรและข้อมูลจำลองเหมือนผู้ใช้เป็นคนใช้งานจริง ๆ โดยการทดสอบแบบนี้มีข้อดี คือ ผู้ใช้สามารถเห็นและรู้ว่าตัวเองต้องการจะทำอะไร และอยากได้อะไรกลับมา เพราะผู้ใช้ไม่รู้เบื้องหลังว่าเราใช้อะไรในการพัฒนาระบบบ้างผู้ใช้รู้เพียงว่ากดปุ่ม A แล้วจะต้องได้ A การทดสอบระบบตั้งแต่ต้นจนจบจะสะดวกมากขึ้นหากมีการทำการทดสอบส่วนต่าง ๆ ของโปรแกรมแบบอัตโนมัติ

ตัวอย่างของการทดสอบระบบตั้งแต่ต้นจนจบ คือ สมมุติว่าระบบที่ต้องการจะทดสอบเป็นระบบที่ให้ผู้ใช้สมัครสมาชิกและเข้าสู่ระบบ การทดสอบระบบตั้งแต่ต้นจนจบจะทำการทดสอบเหมือนผู้ใช้ใช้งานระบบจริง ๆ โดยจะเริ่มจากการสมัครสมาชิก จากนั้นเข้าสู่ระบบ โดยจะต้องทดสอบในทุก ๆ กรณีที่เป็นไปได้ เช่น ในการทดสอบระบบสมัครสมาชิกจะต้องมีการระบุชื่อผู้ใช้และรหัสผ่าน จะต้องจำลองชื่อผู้ใช้และรหัสผ่านในทุก ๆ กรณีที่เป็นไปได้ ทั้งในกรณีที่จะสามารถสมัครสมาชิกได้และไม่สามารถสมัครสมาชิกได้ ตัวอย่างกรณีที่ไม่สามารถสมัครสมาชิกได้ เช่น ชื่อผู้ใช้มีอักษรพิเศษ, รหัสผ่านไม่ปลอดภัยมากพอ เป็นต้น

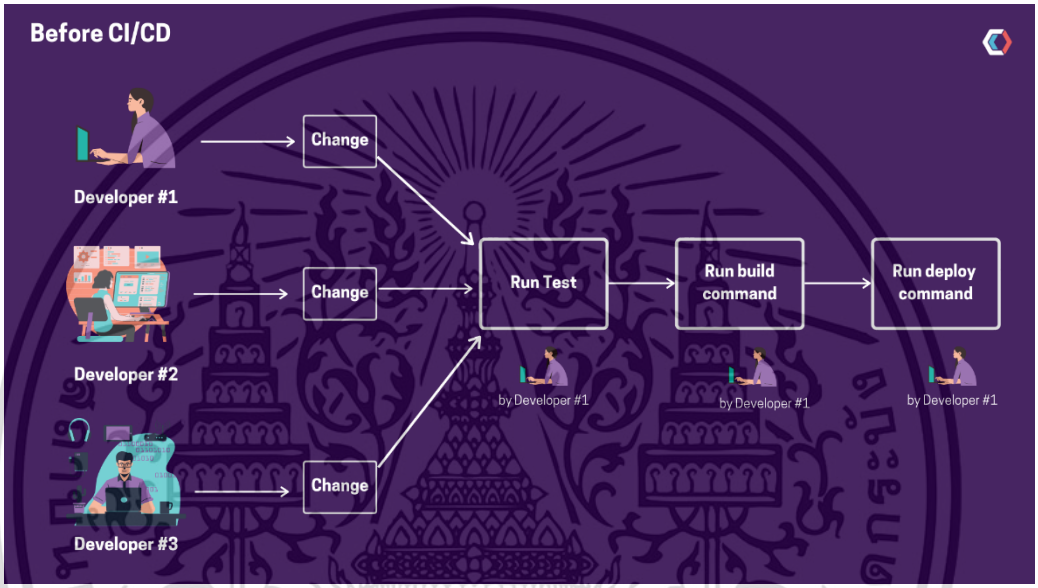
2.2.2. **Continuous Integration/Continuous Delivery (CI/CD)**

CI/CD คือ กระบวนการที่จะมาช่วยในการพัฒนาซอฟต์แวร์ให้มีประสิทธิภาพมากขึ้น ทั้งระยะเวลาการพัฒนาและคุณภาพของซอฟต์แวร์

CI ย่อมาจาก Continuous Integration คือ กระบวนการที่จัดการโค้ดต้นฉบับ (Source Code) ของเราให้ผ่านกระบวนการการทดสอบ, การสร้าง (Building) เพื่อให้แน่ใจว่าโค้ดต้นฉบับสามารถใช้งานได้จริง ไม่มีข้อผิดพลาด มีความพร้อมที่จะถูกใช้งานและส่งมอบให้กับลูกค้า

CD ย่อมาจาก Continuous Delivery หรือ Continuous Deployment กระบวนการที่ช่วยเหลือให้เราสามารถนำซอฟต์แวร์ที่พัฒนาและทดสอบแล้วไปใช้งานในระบบจริงหรือการนำโปรแกรมขึ้นไปอยู่บนเซิร์ฟเวอร์ได้อย่างมีประสิทธิภาพ โดยการนำโค้ดต้นฉบับที่ผ่านการสร้างและทดสอบมาแล้ว ซึ่งอาจอยู่ในรูปแบบที่แตกต่างกัน ให้จัดการนำโปรแกรมขึ้นไปอยู่บนเซิร์ฟเวอร์ตามที่เราต้องการและสามารถใช้งานได้อย่างถูกต้อง โดยกระบวนการ CD มี 2 ประเภท

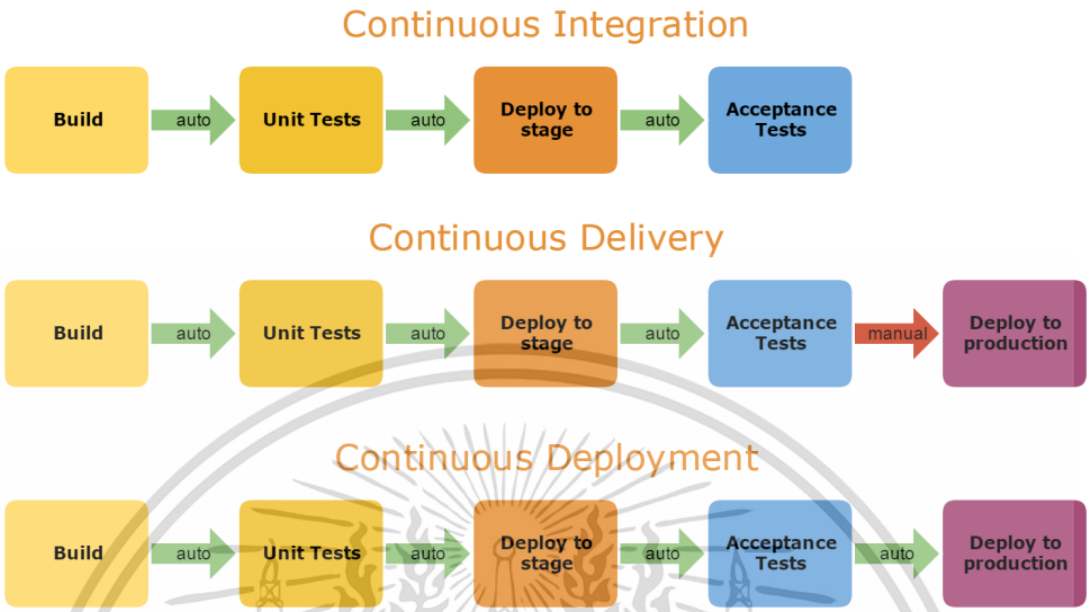
1. Continuous Delivery คือ การส่งมอบซอฟต์แวร์ที่ใช้วิธีการแมนนวลในการนำโปรแกรมขึ้นบนเซิร์ฟเวอร์หลังจากผ่านขั้นตอน CI มาเรียบร้อยแล้ว โดยจะต้องมีการอนุมัติจากผู้รับรอง ซึ่งในที่นี้อาจหมายถึงผู้จัดการในการตรวจสอบก่อนนำโปรแกรมขึ้นไปอยู่บนเซิร์ฟเวอร์
2. Continuous Deployment คือ การส่งมอบซอฟต์แวร์ในรูปแบบอัตโนมัติหลังจากผ่านขั้นตอน CI มาเรียบร้อยแล้ว จะทำการนำโปรแกรมขึ้นไปอยู่บนเซิร์ฟเวอร์ทันทีโดยไม่ต้องรอการอนุมัติ



รูปที่ 2.9 ขั้นตอนการทำงานก่อนมี CI/CD

ที่มา : https://www.codium.co/blogs/33-CICD

จากรูปที่ 2.9 จะเห็นได้ว่าขั้นตอนการทำงานก่อนจะมีการใช้ CI/CD เมื่อโปรแกรมเมอร์ในทีมแต่ละคนมีการเปลี่ยนแปลงหรืออัพเดตโค้ดเข้ามา จะต้องมีโปรแกรมเมอร์ 1 คนทำหน้าที่จัดการรันคำสั่งต่าง ๆ แบบแมนนวล ซึ่งในแต่ละวันโปรแกรมเมอร์มีการเปลี่ยนแปลงหรืออัพเดตโค้ดมากกว่า 1 ครั้ง โปรแกรมเมอร์คนที่ทำหน้าที่รวบรวมโค้ดของทุกคนจะต้องทำกระบวนการนั้นซ้ำซ้อน ส่งผลให้โปรแกรมเมอร์ไม่มีความสุขในการทำงาน เนื่องจากเป็นงานที่น่าเบื่อ อีกทั้งยังเป็นการลดทอนศักยภาพของโปรแกรมเมอร์นั้น ๆ อีกด้วย



รูปที่ 2.10 ขั้นตอนการทำงานของ CI/CD

ที่มา : https://clm-consulting.com/services/it-transformations/

จากรูปที่ 2.10 ขั้นตอนการทำงานของ Continuous Integration เริ่มต้นจากขั้นตอนการ Build นั้นคือการรวบรวมโค้ดที่ได้จากการพัฒนาของผู้พัฒนาแต่ละคนในทีมให้เป็นชิ้นเดียวกัน จากนั้นจะเข้าสู่ขั้นตอนทดสอบฟังก์ชันโดยอัตโนมัติ โดยขั้นตอนนี้จะเป็นขั้นตอนการทดสอบด้วยสคริปต์ทดสอบ เพื่อตรวจสอบว่าโค้ดแต่ละส่วนไม่มีความผิดพลาด ก่อนจะเข้าสู่ขั้นตอน Deploy to Stage หรือการนำระบบหรือซอฟต์แวร์ที่ถูกสร้างขึ้นไปอยู่ในสถานะของการทดสอบ เพื่อทดสอบความเสถียรและความพร้อมใช้งานก่อนนำไปใช้งานจริงโดยอัตโนมัติ จากนั้นจะเข้าสู่ขั้นตอน Acceptance Tests หรือเป็นการทดสอบระบบหรือซอฟต์แวร์ที่เน้นทดสอบว่าระบบสามารถทำงานได้ตามความต้องการของลูกค้าหรือไม่ ซึ่งจะแตกต่างจากการทดสอบปกติ คือ ผลของการทดสอบจะขึ้นอยู่กับผู้ใช้งานเป็นหลักซึ่งจะลดโอกาสในการเกิดข้อผิดพลาดของระบบ

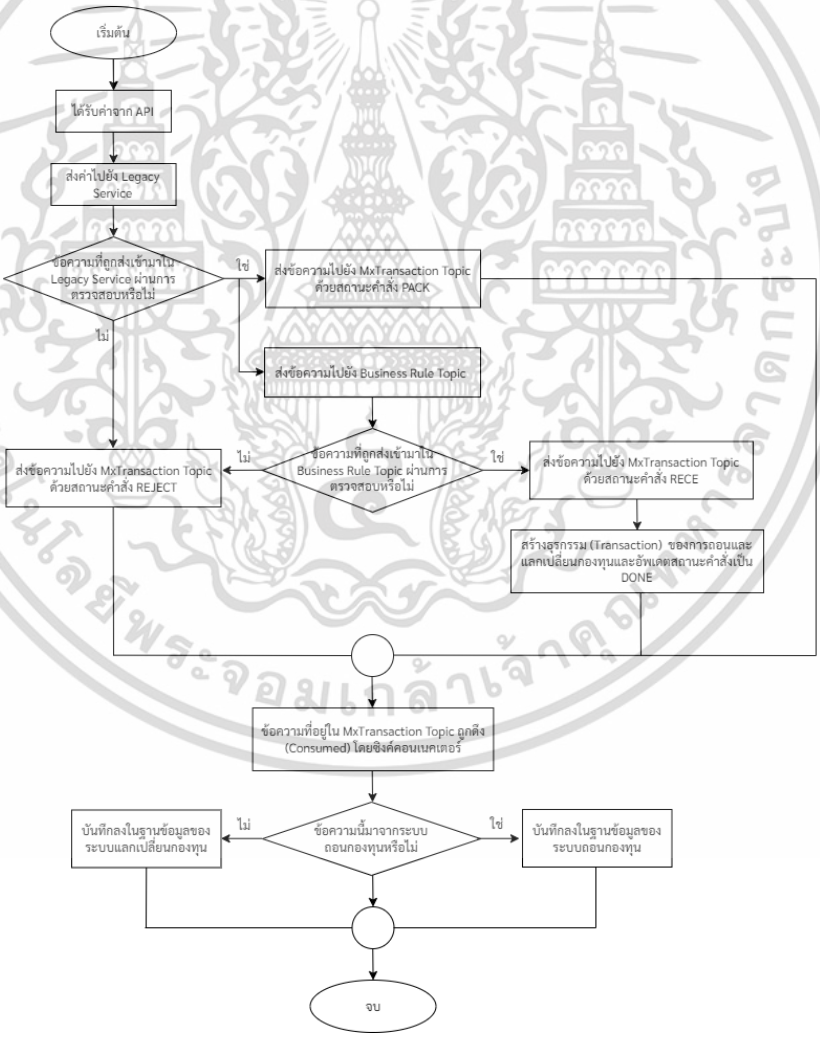
ขั้นตอนการทำงานของ Continuous Delivery จะเกิดขึ้นหลัง Acceptance Tests โดยจะทำการ Deploy to Production คือ การนำเอาระบบหรือซอฟต์แวร์ที่พัฒนาเรียบร้อยแล้วไปใช้งานจริง โดยจะทำการอัพโหลดโค้ดหรือไฟล์ของโปรแกรมไปยังเซิร์ฟเวอร์ที่จะนำไปใช้งานจริง โดยเป็นขั้นตอนสำคัญในการเปิดตัวและใช้งานแอปพลิเคชันในฐานะของการใช้งานจริงที่ผู้ใช้สามารถเข้าถึงและใช้งานได้จริง ๆ

บทที่ 3
ขั้นตอนการดำเนินงาน

ในบทนี้จะกล่าวถึงขั้นตอนการดำเนินงานการทดสอบ โดยจะประกอบด้วยแผนภาพแสดงขั้นตอนการทำงานของระบบ แผนภาพแสดงการทำงานของผู้ทดสอบและแผนภาพกิจกรรม

3.1 แผนภาพแสดงขั้นตอนการทำงานของระบบ (System Flow Diagram)

System Flow Diagram นี้จะแสดงขั้นตอนการทำงานของไมโครเซอร์วิสในขอบเขตของการทดสอบ ซึ่งประกอบด้วยการทำงานของระบบถอนกองทุนและซิงค์คอนเนคเตอร์และระบบอื่น ๆ ที่เกี่ยวข้องกับการทดสอบเท่านั้น



รูปที่ 3.1 System Flow Diagram ของไมโครเซอร์วิส

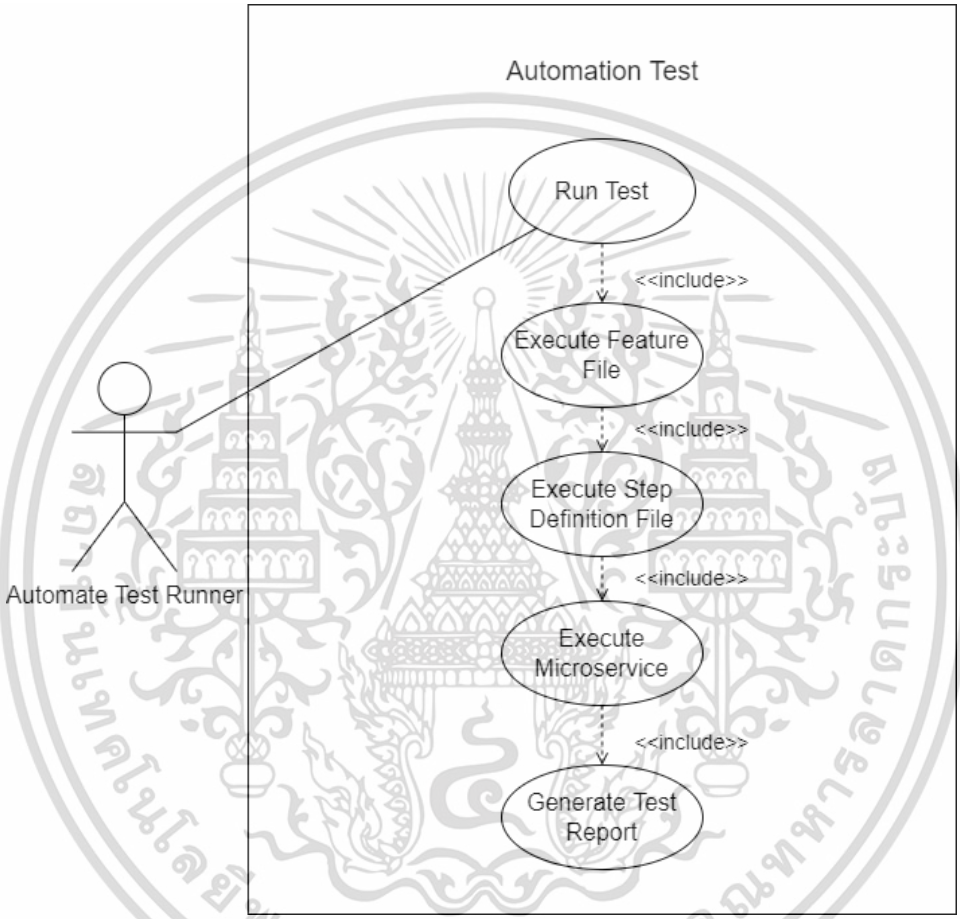
จากรูปที่ 3.1 กระบวนการทำงานระบบถอนกองทุน คือ เมื่อมีผู้ใช้ส่งข้อมูลผ่าน Rest API มาที่ระบบถอนกองทุนแล้วระบบถอนกองทุนจะทำการส่งข้อความไปหา Legacy Service

เมื่อระบบถอนกองทุนส่งข้อความไปยัง Legacy Service แล้วก็จะทำการตรวจสอบข้อความที่ได้รับ ถ้าตรวจสอบผ่าน สถานะของคำสั่งของข้อความนี้จะเป็น PACK แต่ถ้าหากตรวจสอบไม่ผ่านสถานะของคำสั่งจะเป็น null และจะมีคำอธิบายว่าตรวจสอบไม่ผ่านจากสาเหตุใด ถ้าหากสถานะของคำสั่งเป็น PACK จะถูกส่งไปยัง Business Rule Topic เพื่อตรวจสอบเงื่อนไขทางธุรกิจและจะสร้างข้อความเพื่อไปรอในคิวเพื่อรอให้ระบบที่รอรับข้อความ (Subscribe) อยู่ เช่น ซิงค์คอนเนคเตอร์ นำข้อความไปใช้จากนั้นซิงค์คอนเนคเตอร์จะนำข้อความนั้นมาบันทึกไว้ในฐานข้อมูลถ้าหากข้อความนั้นเป็นกรณีที่ผ่านเงื่อนไขทางธุรกิจจะได้สถานะคำสั่งเป็น DONE แต่ถ้าหากเป็นกรณีที่ไม่ผ่านเงื่อนไขทางธุรกิจจะได้สถานะของคำสั่งเป็น null และจะมีคำอธิบายว่าไม่ผ่านเงื่อนไขทางธุรกิจจากสาเหตุใด

กระบวนการทำงานของซิงค์คอนเนคเตอร์ คือ เมื่อมีข้อความจากระบบ, Business Rule และ TA ซิงค์คอนเนคเตอร์ส่งเข้ามาที่ Kafka Topic จากนั้นซิงค์คอนเนคเตอร์จะทำการอ่านข้อความที่อยู่ใน Topic เพื่อตรวจสอบว่าข้อความนั้นเป็นข้อความของระบบถอนกองทุนหรือระบบแลกเปลี่ยนกองทุนหรือจาก Business Rule เมื่อตรวจสอบแล้วก็จะทำการส่งข้อความนั้นไปจัดเก็บในฐานข้อมูลของระบบนั้น เช่น หากข้อความนั้นมาจากระบบถอนกองทุนจะถูกจัดเก็บในฐานข้อมูลของระบบถอนกองทุน

3.2 แผนภาพแสดงการทำงานของผู้ทดสอบ (Use Case Diagram)

Use Case Diagram นี้แสดงให้เห็นถึงขั้นตอนการทำการทดสอบอัตโนมัติของการถอนกองทุนและซิงค์คอนเนคเตอร์แอปพลิเคชันซึ่งแสดงความสัมพันธ์ของผู้ทดสอบระบบกับการทดสอบอัตโนมัติ



รูปที่ 3.2 Use Case Diagram ของการทดสอบอัตโนมัติ

จากรูปที่ 3.2 กระบวนการทำงานของการทดสอบแบบอัตโนมัติ คือ เมื่อผู้ทดสอบระบบต้องการทดสอบระบบก็จะทำการรันตัวทดสอบขึ้นมา จากนั้นตัวทดสอบก็จะเข้าสู่ขั้นตอน Execute Feature File ซึ่งเป็นหนึ่งในเครื่องมือ Cucumber Feature File จะเป็นตัวกำหนดขั้นตอนการทำงานของการทดสอบ โดยจะถูกเขียนด้วยภาษาที่มนุษย์เข้าใจได้ง่าย จากนั้นเมื่อเข้าในแต่ละขั้นตอนของ Feature File จะเรียกคำสั่งจาก Step Definition (Execute Step Definition File) ซึ่งจะเป็นการกำหนดว่าขั้นตอนใน Feature File นั้นทำงานอะไรบ้างในภาษาโปรแกรมมิ่ง เมื่อมีการเริ่มการทดสอบก็จะเรียกไมโครเซอร์วิส

ขึ้นมาเพื่อทดสอบ และสุดท้ายก็จะออกรายงานของการทดสอบว่ามีทั้งหมดกี่กรณีและมีกี่กรณีที่ผ่านหรือไม่ผ่าน

ตารางที่ 3.1 Use Case Description Run Test

Use Case ID	U01
Use Case Name	Run Test
Actor	Automate Test Runner
Description	เพื่อให้ผู้ทดสอบระบบเริ่มการทดสอบ
Level	Primary Use Case
Pre-Condition	-
Post-Condition	การทดสอบมีการเรียก Feature File เพื่อเริ่มการทดสอบ
Included Use Case	-
Extend Use Case	-
Main Flows	1. ผู้ทดสอบระบบเริ่มการทดสอบด้วยการรันโค้ดการทดสอบ 2. ฟังก์ชันการทดสอบจะเริ่มต้นการทำงาน

ตารางที่ 3.2 Use Case Description Execute Feature File

Use Case ID	U02
Use Case Name	Execute Feature File
Actor	Automate Test Runner
Description	เพื่อให้เริ่มต้นการทำงานในแต่ละขั้นตอนของ Feature File
Level	Primary Use Case
Pre-Condition	Automate Test Runner เริ่มการทดสอบระบบ จำลองข้อความที่คาดหวังว่าจะได้รับ
Post-Condition	มีการเรียก Step Definition ตามขั้นตอนใน Feature File
Included Use Case	Run Test
Extend Use Case	-

Main Flows	1. ขั้นตอนการทดสอบการถอนกองทุน 1.1 ตรวจสอบว่าข้อความที่ถูกต้อง คาดหวังว่าจะผ่านการตรวจสอบจากทั้ง Legacy Service และ Business Rule Topic มีสถานะของคำสั่งเป็น DONE 1.2 ตรวจสอบว่าข้อความที่ไม่ถูกต้อง คาดหวังว่าจะไม่ผ่านการตรวจสอบจาก Legacy Service มีสถานะของคำสั่งเป็น REJECT และคำอธิบายเหตุผลที่ไม่ผ่านการตรวจสอบตรงกับที่คาดหวังไว้ 2. ขั้นตอนการทดสอบซิงค์คอนเนคเตอร์ 2.1 ตรวจสอบว่าข้อความที่มาจากระบบการถอนกองทุนและระบบการขายกองทุนนั้นถูกจัดเก็บอยู่ในฐานข้อมูลที่ถูกต้องหรือไม่และมีสถานะของคำสั่งตรงตามที่คาดหวังไว้หรือไม่ 2.2 ตรวจสอบว่าข้อความที่มาจาก Business Rule Topic นั้นถูกจัดเก็บอยู่ในฐานข้อมูลที่ถูกต้องหรือไม่และมีสถานะของคำสั่งตรงตามที่คาดหวังไว้หรือไม่ 2.3 ตรวจสอบว่าข้อความที่มาจาก TA ซิงค์คอนเนคเตอร์นั้นถูกจัดเก็บอยู่ในฐานข้อมูลที่ถูกต้องหรือไม่และมีสถานะของคำสั่งตรงตามที่คาดหวังไว้หรือไม่

ตารางที่ 3.3 Use Case Description Execute Step Definition

Use Case ID	U03
Use Case Name	Execute Step Definition
Actor	Automate Test Runner
Description	เพื่อให้การทดสอบเริ่มต้นตามขั้นตอนของ Feature File
Level	Primary Use Case
Pre-Condition	ขั้นตอนใน Automate Test Runner ต้องมีใน Feature File
Post-Condition	ทดสอบตามขั้นตอนใน Step Definition
Included Use Case	Execute Feature File
Extend Use Case	-
Main Flows	1. เมื่อ Feature File สั่งคำสั่ง Step Definition จะถูกเรียกขึ้นมา

	2. เริ่มการทดสอบตามคำสั่งโปรแกรมมิ่งใน Step Definition

ตารางที่ 3.4 Use Case Description Execute Microservices

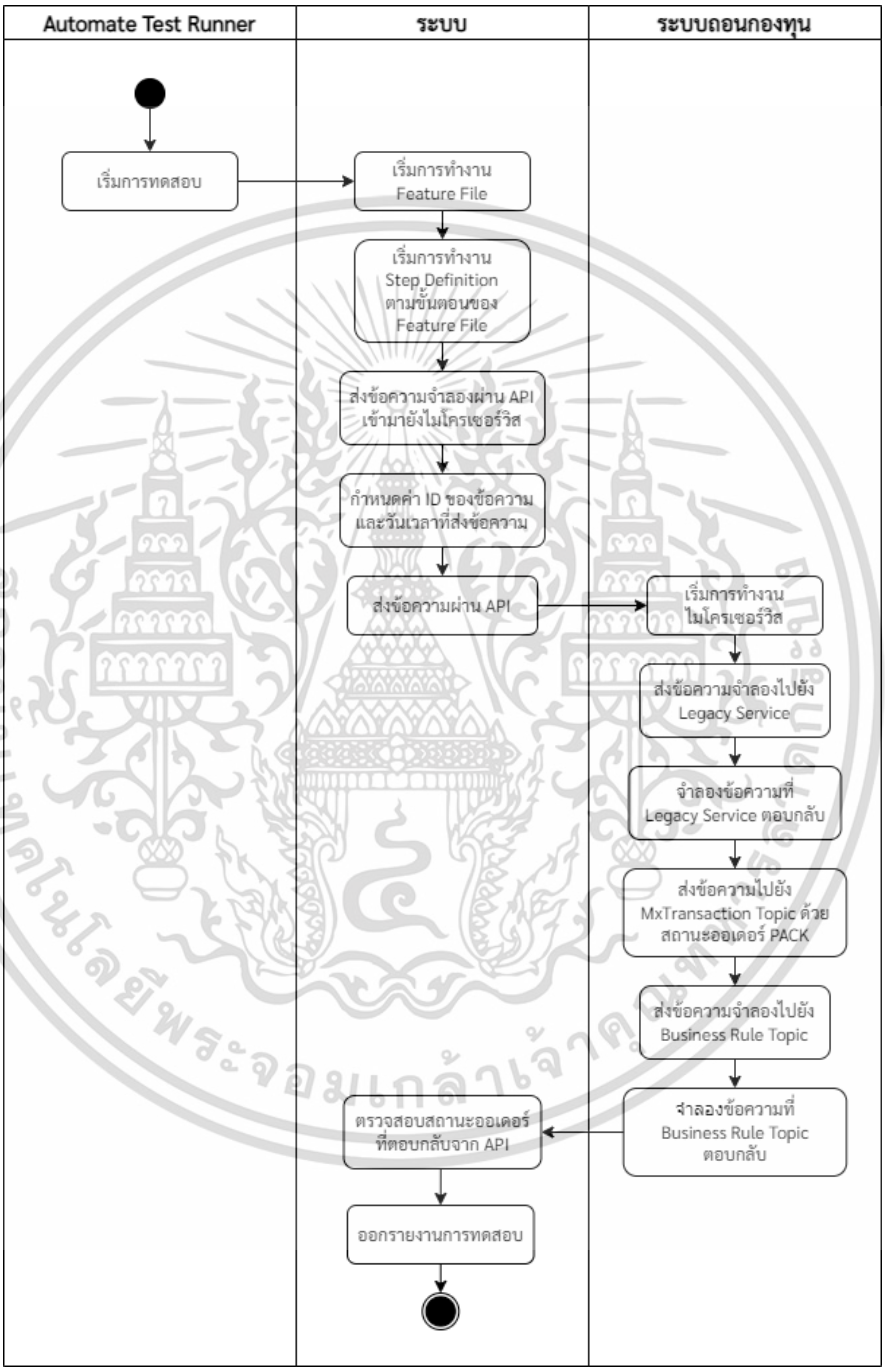
Use Case ID	U04
Use Case Name	Execute Microservices
Actor	Automate Test Runner
Description	เพื่อเรียกไมโครเซอร์วิสเพื่อทดสอบ
Level	Primary Use Case
Pre-Condition	มีการทดสอบตามขั้นตอนใน Step Definition ในไมโครเซอร์วิสทำการฟิกค่าเพื่อให้การทดสอบไม่คลาดเคลื่อน
Post-Condition	ทดสอบเสร็จสิ้น
Included Use Case	Execute Step Definition
Extend Use Case	-
Main Flows	1. เรียกไมโครเซอร์วิสขึ้นมา 2. ทำการทดสอบตาม Step Definition

ตารางที่ 3.5 Use Case Description Generate Test Report

Use Case ID	U05
Use Case Name	Generate Test Report
Actor	Automate Test Runner
Description	เพื่อออกรายงานผลการทดสอบ
Level	Primary Use Case
Pre-Condition	การทดสอบเสร็จสิ้น
Post-Condition	ได้รับรายงานการทดสอบ
Included Use Case	Execute Microservices
Extend Use Case	-
Main Flows	1. เมื่อสิ้นสุดการทดสอบจึงออกรายงานว่ามีกี่กรณีที่ผ่านการทดสอบและกี่กรณีที่ไม่ผ่านการทดสอบ

3.3 แผนภาพกิจกรรม (Activity Diagram)

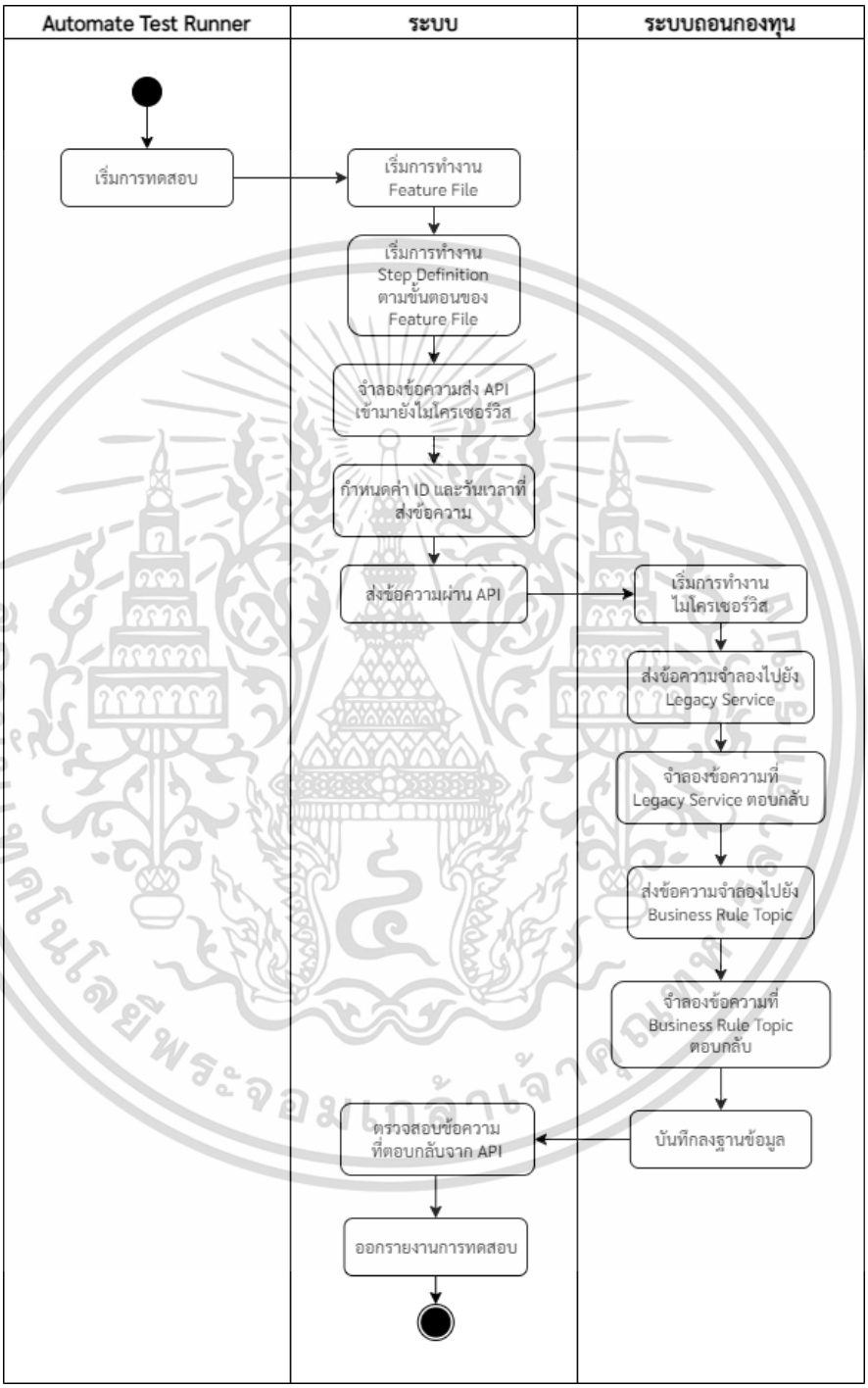
3.3.1. Activity Diagram ของการทดสอบระบบถอนกองทุนกรณีที่ถูกต้องและสามารถถอนกองทุนได้



รูปที่ 3.3 Activity Diagram ของการทดสอบระบบถอนกองทุนกรณีที่ถูกต้องและสามารถถอนกองทุนได้

จากรูปที่ 3.3 เป็นแผนภาพกิจกรรมของการทดสอบระบบถอนกองทุนที่เป็นกรณีที่ข้อความถูกต้องและสามารถถอนกองทุนได้ เริ่มจากการที่ผู้ทดสอบระบบ (Automate Test Runner) กดเริ่มการทดสอบ จากนั้นจะเริ่มการทำงาน Feature File ที่ถูกเขียนโดยภาษาที่มนุษย์สามารถเข้าใจได้และเริ่มการทำงาน Step Definition ที่ถูกเขียนด้วยภาษาโปรแกรมมิ่งตามขั้นตอนของ Feature File จากนั้นระบบจะส่งข้อความที่จำลองไว้ผ่าน API เข้าไปยังไมโครเซอร์วิส โดยจะมีการกำหนดค่า ID ของข้อความและวันเวลาที่ส่งข้อความ เพื่อให้การทดสอบไม่คลาดเคลื่อน สามารถตรวจสอบการตอบกลับจากระบบได้อย่างถูกต้อง ต่อมาระบบจะส่งข้อความผ่าน API และไมโครเซอร์วิสจะเริ่มทำงาน จากนั้นระบบถอนกองทุนจะส่งข้อความจำลองไปยัง Legacy Service และจำลองข้อความที่ Legacy ตอบกลับ เมื่อเสร็จสิ้นระบบถอนกองทุนจะส่งข้อความไปยัง MxTransaction Topic ด้วยสถานะออเดอร์ PACK จากนั้นจะส่งข้อความจำลองไปยัง Business Rule Topic และจำลองข้อความที่ Business Rule Topic ตอบกลับมาและสุดท้ายตรวจสอบสถานะออเดอร์ที่ตอบกลับมาจาก API ว่าเป็น DONE หรือไม่ ถ้าหากการทำงานของระบบถอนกองทุนถูกต้องสถานะของออเดอร์ที่ตอบกลับมาจาก API จะต้องเป็น DONE และเมื่อการทดสอบเสร็จสิ้นจะออกรายงานผลการทดสอบว่าการทำงานของระบบถอนกองทุนขั้นตอนไหนที่ทำงานผิดพลาด

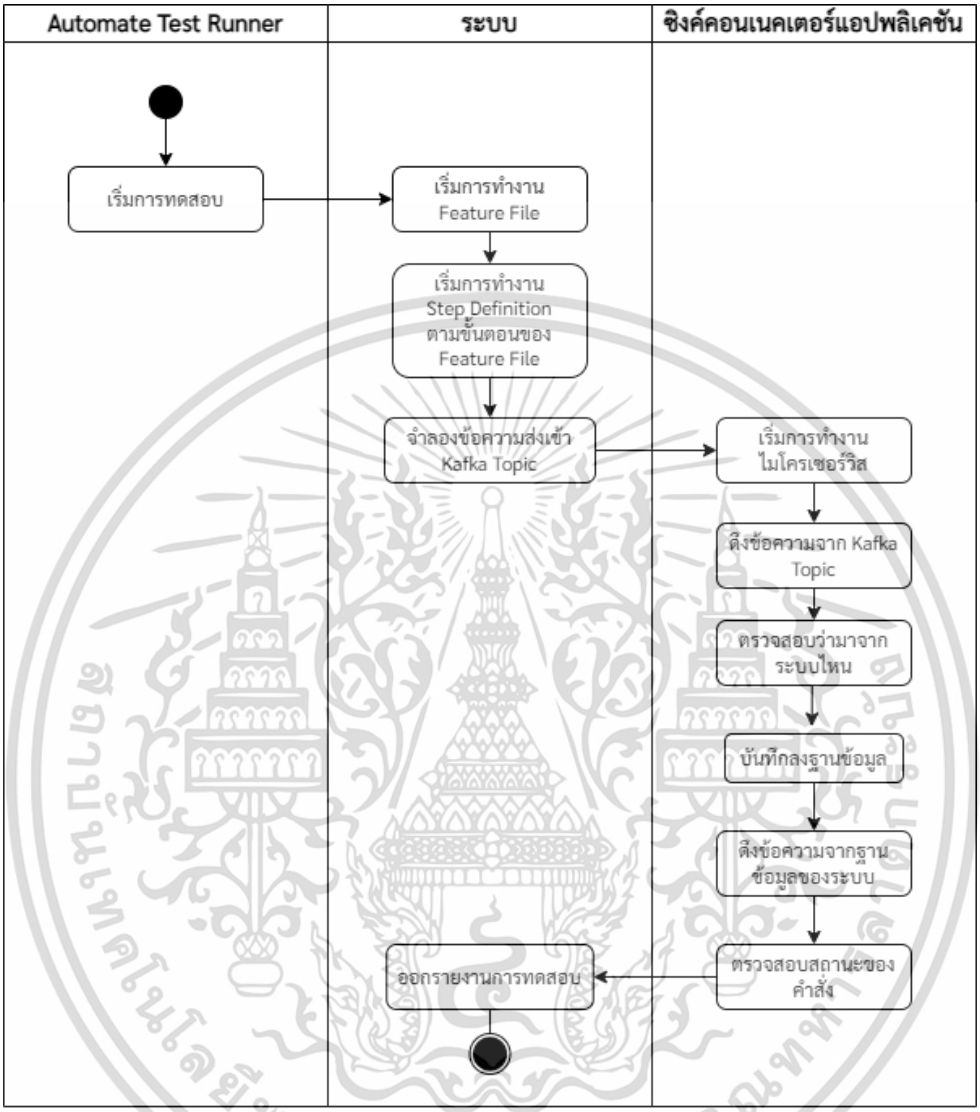
3.3.2. Activity Diagram ของการทดสอบระบบถอนกองทุนกรณีที่ไม่ถูกต้องและไม่สามารถถอนกองทุนได้



รูปที่ 3.4 Activity Diagram ของการทดสอบระบบถอนกองทุนกรณีที่ไม่ถูกต้องและไม่สามารถถอนกองทุนได้

จากรูปที่ 3.4 เป็นแผนภาพกิจกรรมของการทดสอบระบบถอนกองทุนที่เป็นกรณีที่ข้อความไม่ถูกต้องและไม่สามารถถอนกองทุนได้ เริ่มจากการที่ผู้ทดสอบระบบ (Automate Test Runner) กดเริ่มการทดสอบ จากนั้นจะเริ่มการทำงาน Feature File ที่ถูกเขียนโดยภาษาที่มนุษย์สามารถเข้าใจได้และเริ่มการทำงาน Step Definition ที่ถูกเขียนด้วยภาษาโปรแกรมมิ่งตามขั้นตอนของ Feature File จากนั้นระบบจะส่งข้อความที่จำลองไว้ผ่าน API เข้าไปยังไมโครเซอร์วิส โดยจะมีการกำหนดค่า ID ของข้อความและวันเวลาที่ส่งข้อความ เพื่อให้การทดสอบไม่คลาดเคลื่อน สามารถตรวจสอบการตอบกลับจากระบบได้อย่างถูกต้อง ต่อมาระบบจะส่งข้อความผ่าน API และไมโครเซอร์วิสจะเริ่มทำงาน จากนั้นระบบถอนกองทุนจะส่งข้อความจำลองไปยัง Legacy Service และจำลองข้อความที่ Legacy ตอบกลับ จากนั้นจะส่งข้อความจำลองไปยัง Business Rule Topic และจำลองข้อความที่ Business Rule Topic ตอบกลับมาและบันทึกข้อความนั้นลงในฐานข้อมูล ต่อมาตรวจสอบสถานะออเดอร์ที่ตอบกลับมาจาก API ว่าเป็น REJECT หรือไม่ ถ้าหากการทำงานของระบบถอนกองทุนถูกต้องสถานะของออเดอร์ที่ตอบกลับมาจาก API จะต้องเป็น REJECT และมีคำอธิบายเหตุผลความไม่ถูกต้องของข้อความนั้น ๆ ตามกรณีและเมื่อการทดสอบเสร็จสิ้นจะออกรายงานผลการทดสอบว่าการทำงานของระบบถอนกองทุนขั้นตอนไหนที่ทำงานผิดพลาด

3.3.3. Activity Diagram ของการทดสอบซิงค์คอนเนคเตอร์



รูปที่ 3.5 Activity Diagram ของการทดสอบซิงค์คอนเนคเตอร์

จากรูปที่ 3.5 เป็นแผนภาพของการทดสอบซิงค์คอนเนคเตอร์ เริ่มจากการที่ผู้ทดสอบระบบ (Automate Test Runner) กดเริ่มการทดสอบ จากนั้นจะเริ่มการทำงาน Feature File ที่ถูกเขียนโดยภาษาที่มนุษย์สามารถเข้าใจได้และเริ่มการทำงาน Step Definition ที่ถูกเขียนด้วยภาษาโปรแกรมมิ่งตามขั้นตอนของ Feature File จากนั้นจะทำการจำลองข้อความส่งเข้า Kafka Topic จากนั้นจะเริ่มการทำงานไมโครเซอร์วิส ต่อมาดึงข้อความจาก Kafka Topic และตรวจสอบว่ามาจากระบบไหน ระบบถอนกองทุนหรือระบบแลกเปลี่ยนกองทุนหรือ Business Rule จากนั้นบันทึกลงฐานข้อมูลลงตามระบบที่

ข้อความนั้นส่งมา เช่น หากข้อความนั้น ๆ ถูกส่งมาจากระบบถอนกองทุน ข้อความนั้นจะถูกบันทึกลงไปในฐานข้อมูลของระบบถอนกองทุน จากนั้นตรวจสอบสถานะของคำสั่งว่าเป็นไปตามที่ต้องการหรือไม่ เช่น หากเป็นกรณีที่สถานะของคำสั่งจะต้องเป็น DONE ก็จะตรวจสอบว่าสถานะของคำสั่งที่ดึงออกมาจากฐานข้อมูลนั้นเป็น DONE หรือไม่และเมื่อการทดสอบเสร็จสิ้นจะออกรายงานผลการทดสอบว่าการทำงานของระบบถอนกองทุนขั้นตอนไหนที่ทำงานผิดพลาด

บทที่ 4
ผลการดำเนินงาน

ในบทนี้จะกล่าวถึงการพัฒนาการทดสอบอัตโนมัติของระบบไมโครเซอร์วิสโดยในส่วนแรกจะเป็นการพัฒนาการทดสอบอัตโนมัติของระบบถอนกองทุน ถัดมาเป็นส่วนของการพัฒนาการทดสอบอัตโนมัติของซิงค์คอนเนคเตอร์

โดยจะใช้เครื่องมือในการสร้างการทดสอบอัตโนมัติด้วย Cucumber ในการทำงานของ Cucumber มีวิธีการทำงานตามบทที่ 3 หัวข้อที่ 3.2 โดยในการดำเนินงานนี้ประกอบไปด้วยไฟล์ที่ใช้ในการสร้างการทดสอบอัตโนมัติ 2 ไฟล์นั่นคือ 1. Feature File ซึ่งใช้ภาษา Gherkin ในการกำหนดสคริปต์การทดสอบในแต่ละกรณีทดสอบ และ 2. Step Definition ซึ่งใช้ภาษา Java สั่งการการทำงานฟังก์ชันต่าง ๆ ภายในโปรแกรม ให้ทำการทดสอบตามสคริปต์การทดสอบในการทดสอบ 2 ไมโครเซอร์วิสดังนี้

1. การถอนกองทุน
2. ซิงค์คอนเนคเตอร์

4.1 การถอนกองทุน

การทดสอบระบบถอนกองทุนเป็นการทดสอบที่จำลองข้อมูลที่ส่งไปยังเซอร์วิสหรือ Topic อื่น ๆ และจำลองการตอบกลับของข้อมูลจากเซอร์วิสหรือ Topic นั้น ๆ เพื่อตรวจสอบสถานะของคำสั่งว่าถูกต้องหรือไม่ จึงแบ่งกรณีทดสอบได้ดังตารางที่ 4.1

ตารางที่ 4.1 กรณีทดสอบระบบการถอนกองทุน

ในการถอนกองทุนสามารถทำได้ 2 แบบ คือ การส่งเช็คไปยังบ้านของลูกค้าและการโอนเงินเข้าบัญชีธนาคาร

Test Case Id	Test Case	Expected Result
1	ส่งเช็คที่อยู่ลูกค้าที่ไม่ใช่ที่อยู่ที่ลงทะเบียนไว้ - จัดเตรียมข้อมูลที่มีที่อยู่ของลูกค้าส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic	- Legacy Service และ Business Rule ตรวจสอบผ่าน - สถานะของคำสั่งเป็น DONE

	- ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API	
2	โอนเงินเข้าบัญชีธนาคารของลูกค้า - จัดเตรียมข้อมูลที่มีบัญชีธนาคารของลูกค้าส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API	- Legacy Service และ Business Rule ตรวจสอบผ่าน - สถานะของคำสั่งเป็น DONE
3	โอนเงินเข้าบัญชีธนาคารที่ไม่ใช่ของลูกค้า - จัดเตรียมข้อมูลที่มีบัญชีธนาคารและข้อมูลของเจ้าของบัญชีส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API	- Legacy Service และ Business Rule ตรวจสอบผ่าน - สถานะของคำสั่งเป็น DONE
4	ส่งเช็คด้วยที่อยู่ลูกค้าแบบมีภาษีและ Withholding - จัดเตรียมข้อมูลที่มีที่อยู่ของลูกค้าและอัตราภาษีส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API	- Legacy Service และ Business Rule ตรวจสอบผ่าน - สถานะของคำสั่งเป็น DONE
5	โอนเงินเข้าธนาคารลูกค้าพร้อม Tax lot - จัดเตรียมข้อมูลที่มีบัญชีธนาคารของลูกค้าและ Tax lot ส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API	- Legacy Service และ Business Rule ตรวจสอบผ่าน - สถานะของคำสั่งเป็น DONE
6	เลขที่บัญชีของกองทุนที่ต้องการจะถอนไม่ถูกต้อง	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ

	- จัดเตรียมข้อมูลที่มีเลขที่บัญชีของกองทุนลูกค้าส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	
7	คำแนะนำในการถอนกองทุนไม่ถูกต้อง - จัดเตรียมข้อมูลที่มีคำแนะนำในการถอนกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ
8	ข้อมูลของกองทุนที่ต้องการจะถอนไม่ถูกต้อง - จัดเตรียมข้อมูลที่มีข้อมูลกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ
9	จำนวนเงินที่ต้องการจะถอนจากกองทุนไม่ถูกต้อง - จัดเตรียมข้อมูลที่มีข้อมูลการถอนกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ
10	ภาษีในการถอนกองทุนไม่ถูกต้อง	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT

	- จัดเตรียมข้อมูลที่มีที่ภาษีในการถอนกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ
11	ผู้ใช้ไม่มีสิทธิ์ถอนกองทุนที่ต้องการจะถอน - จัดเตรียมข้อมูลที่มีเลขบัญชีของกองทุนและประเภทของลูกค้าส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ
12	ไม่มีหน่วยหรือส่วนแบ่งของกองทุนที่สามารถถอนกองทุนได้ - จัดเตรียมข้อมูลที่มีหน่วยของกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ
13	หน่วยที่สามารถถอนกองทุนได้ - จัดเตรียมข้อมูลที่มีหน่วยของกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ

14	ไม่มี cdsc (Contingent Deferred Sales Charge) ซึ่งวิธีการเรียกเก็บค่าธรรมเนียมที่มักใช้ในกองทุนรูปแบบหนึ่ง - จัดเตรียมข้อมูลที่มี cdsc ของกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ
15	ต้องไม่มีประเภทของการถอนกองทุนสำหรับบัญชีที่ไม่มีผู้ดูแลหรือผู้ปกครองทรัพย์สินของผู้เป็นเจ้าของบัญชีอื่น ๆ - จัดเตรียมข้อมูลที่มีประเภทการถอนกองทุนและผู้ดูแลกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ
16	ประเภทของการถอนกองทุนต้องไม่เป็นช่องว่างสำหรับบัญชีที่มีผู้ดูแลหรือผู้ปกครองทรัพย์สินของผู้เป็นเจ้าของบัญชีอื่น ๆ - จัดเตรียมข้อมูลที่มีประเภทการถอนกองทุนและผู้ดูแลบัญชีส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ
17	ประเภทของจำนวนเงินกับข้อมูลใน cdsc ไม่ตรงกัน	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT

	- จัดเตรียมข้อมูลที่ cdsc ลูกค้าส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ

เมื่อกำหนดกรณีทดสอบได้แล้วก็จะนำกรณีทดสอบเหล่านี้ไปสร้างการทดสอบแบบอัตโนมัติขึ้นมาได้ดังนี้

1. Feature File

เป็นไฟล์ที่ใช้ในการกำหนดสคริปต์การทดสอบ โดยในระบบการถอนกองทุนจะมีสคริปต์การทดสอบตามกรณีทดสอบดังนี้

Test Case 1-5 : ข้อมูลการถอนกองทุนที่ผ่านการตรวจสอบจาก Legacy Service, Business Rule และสถานะของคำสั่งเป็น DONE โดยในการอธิบายจะเป็นตัวอย่าง 1 ใน Test Case เนื่องจากในทั้ง 5 Test Case นั้นใช้วิธีการทดสอบเดียวกัน ต่างเพียงข้อมูลที่ใช้ในการทดสอบ

จะมีการเก็บข้อมูลการถอนกองทุนผ่านไฟล์ request/success-NFID-ACH-FIFO-OVERRIDE.json, เก็บข้อมูลจำลองในการส่งไปยัง Legacy Service ผ่านไฟล์ fund-accout/success-NFID-ACH-FIFO-OVERRIDE.json และ additional-info/success-NFID-ACH-FIFO-OVERRIDE.json และข้อมูลที่ Legacy Service ตอบกลับมา response/success-NFID-ACH-FIFO-OVERRIDE.json และ response/success-NFID-ACH-FIFO-OVERRIDE.json จำลองข้อมูลในการส่งไปยัง Business Rule ผ่านไฟล์ iso/success-NFID-ACH-FIFO-OVERRIDE.json และจำลองข้อมูลข้อมูลที่ Business Rule ผ่านไฟล์ iso/success-NFID-ACH-FIFO-OVERRIDE.json

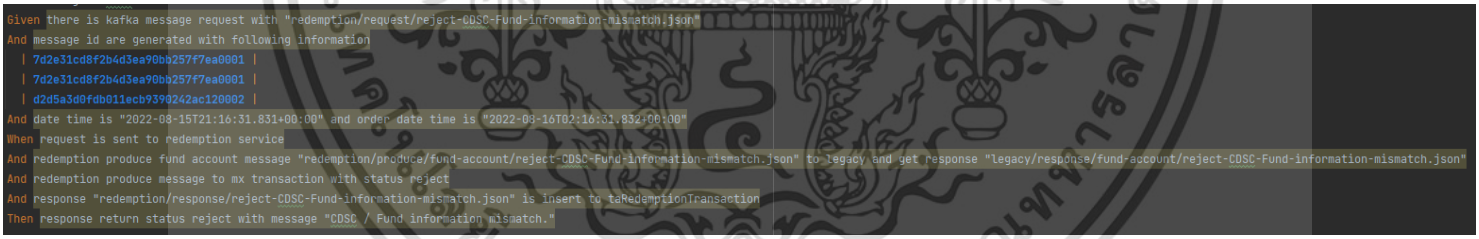
```
Given there is kafka message request with "redemption/request/success-NFID-ACH-CBS-FIFO-OVERRIDE.json"
And message id are generated with following information
  | 7d2e31cd8f2b4d3ea90bb257f7ea0001 |
  | 7d2e31cd8f2b4d3ea90bb257f7ea0001 |
  | d2d5a3d0fdb011ecb9390242ac120002 |
  | 7d2e31cd8f2b4d3ea90bb257f7ea0002 |
And date time is "2022-08-15T21:16:31.831+00:00" and order date time is "2022-08-16T02:16:31.832+00:00"
When request is sent to redemption service
And redemption produce fund account message "redemption/produce/fund-account/success-NFID-ACH-CBS-FIFO-OVERRIDE.json" to legacy and get response "legacy/response/fund-account/success-NFID-ACH-CBS-FIFO-OVERRIDE.json"
And redemption produce additional info message "redemption/produce/additional-info/success-NFID-ACH-CBS-FIFO-OVERRIDE.json" to legacy and get response "legacy/response/additional-info/success-NFID-ACH-CBS-FIFO-OVERRIDE.json"
And redemption produce message to mx transaction with status PACK
And redemption produce message "redemption/produce/iso/success-NFID-ACH-CBS-FIFO-OVERRIDE.json" to business rule and get response "redemption/produce/done/success-NFID-ACH-CBS-FIFO-OVERRIDE.json"
Then response return status DONE
```

รูปที่ 4.1 สคริปต์การทดสอบของ Test Case 1-5

จากรูปที่ 4.1 สคริปต์การทดสอบนี้จะเรียกไฟล์ที่เก็บข้อมูลกางถอนกองทุน จากนั้นกำหนดค่า ID และวันเวลาของข้อความ จากนั้นส่งข้อมูลเข้าไมโครเซอร์วิสผ่าน API ต่อมาส่งข้อมูลที่เรียกจากไฟล์ไปยัง Legacy Service และจำลองข้อมูลที่ Legacy Service ตอบกลับโดยการเรียกจากไฟล์ที่เก็บข้อมูล ส่งข้อมูลไปยัง MxTransaction ด้วยสถานะของคำสั่งเป็น “PACK” ส่งข้อมูลไปยัง Business Rule โดยการเรียกไฟล์ที่เก็บข้อมูลและจำลองข้อมูลที่ Business Rule โดยการเรียกจากไฟล์ที่เก็บข้อมูลและสุดท้ายข้อความที่ตอบกลับมายัง API ต้องมีสถานะของคำสั่งเป็น “DONE”

Test Case 6-17 : ข้อมูลการถอนกองทุนที่ไม่ผ่านการตรวจสอบจาก Legacy Service, Business Rule จากนั้นบันทึกลงในฐานข้อมูล โดยสถานะของคำสั่งเป็น REJECT และมีคำอธิบายเหตุผลในการ REJECT โดยในการอธิบายจะเป็นตัวอย่าง 1 ใน Test Case เนื่องจากในทั้ง 12 Test Case นั้นใช้วิธีการทดสอบเดียวกัน ต่างเพียงข้อมูลที่ใช้ในการทดสอบ

จะมีการเก็บข้อมูลการถอนกองทุนผ่านไฟล์ request/reject-CDSC-Fund-information-mismatch.json เก็บข้อมูลจำลองในการส่งไปยัง Legacy Service ผ่านไฟล์ fund-accout/ reject-CDSC-Fund-information-mismatch.json และข้อมูลที่ Legacy Service ตอบกลับมา response/ reject-CDSC-Fund-information-mismatch.json จำลองข้อมูลที่บันทึกลงฐานข้อมูลผ่านไฟล์ response/reject-CDSC-Fund-information-mismatch.json



รูปที่ 4.2 สคริปต์การทดสอบของ Test Case 6-17

จากรูปที่ 4.2 สคริปต์การทดสอบนี้จะเรียกไฟล์ที่เก็บข้อมูลกางถอนกองทุน จากนั้นกำหนดค่า ID และวันเวลาของข้อความ จากนั้นส่งข้อมูลเข้าไมโครเซอร์วิสผ่าน API ต่อมาส่งข้อมูลที่เรียกจากไฟล์ไปยัง Legacy Service และจำลองข้อมูลที่ Legacy Service ตอบกลับโดยการเรียกจากไฟล์ที่เก็บข้อมูล ส่งข้อมูลไปยัง MxTransaction ด้วยสถานะของคำสั่งเป็น “REJECT” บันทึกข้อมูลที่ตอบกลับในฐานข้อมูล และสุดท้ายข้อความที่ตอบกลับมายัง API ต้องมีสถานะของคำสั่งเป็น “REJECT” และคำอธิบายในการ REJECT นั้นตรงกับกรณีที่ทดสอบอยู่

2. Step Definition

เป็นไฟล์ที่ใช้ในการสั่งการการทำงานฟังก์ชันต่าง ๆ ภายในโปรแกรมที่ต้องการทดสอบ ซึ่งแต่ละเมธอดในไฟล์นี้จะถูกเรียกตามชื่อของสคริปต์การทดสอบใน Feature File โดยแต่ละสคริปต์ทำการเรียกเมธอดดังนี้

ตารางที่ 4.2 เมธอดที่ถูกเรียกใช้งานในแต่ละสคริปต์การทดสอบระบบการถอนกองทุน

Script ID	ชื่อสคริปต์การทดสอบใน Feature File	ชื่อเมธอดที่ถูกเรียกใช้งานใน Step Definition
1	Given there is Kafka message request with “...”	@Given(“there is kafka message request with {string}”)
2	And message id are generated with following information |...|	@And(“message id are generated with following information”)
3	And date time is “...” and order date time is “...”	@And(“date time is {string} and order date time is {string}”)
4	When request is sent to redemption sevice	@When(“request is sent to redemption service”)
5	And redemption produce fund account message “...” to legacy and get response “...”	@And(“redemption produce fund account message {string} to legacy and get response {string}”)
6	And redemption produce additional info message “...” to legacy and get response “...”	@And(“redemption produce additional info message {string} to legacy and get response {string}”)
7	And redemption produce message to mx transaction with status PACK	@And(“redemption produce message to mx transaction with status PACK”)
8	And redemption produce message to mx transaction with status reject	@And(“redemption produce message to mx transaction with status reject”)

9	And redemption produce message “...” to business rule and get response “...”	@And(“redemption produce message {string} to business rule and get response {string}”)
10	And response “...” is insert to taRedemptionTransaction	@And(“response {string} is insert to taRedemptionTransaction”)
11	Then response return status DONE	@Then(“response return status DONE”)
12	Then response return status reject with message “...”	@Then(“response return status reject with message {string}”)

โดยการทำงานของแต่ละเมธอดที่ถูกเรียกใช้งานมีดังนี้

Script ID 1 : @Given(“there is kafka message request with {string}”)

```
@Given( "there is kafka message request with {string}" )
public void there_is_kafka_message_with (String pathToFile) throws IOException, URISyntaxException
{
    requestBody = convertJsonToObj( pathToFile, Redemption.class );
}
```

รูปที่ 4.3 เมธอดที่ถูกเรียกใช้งานของ Script ID 1 สำหรับการเตรียมข้อมูล

Script ID 2 : @And(“message id are generated with following information”)

```
@And( "message id are generated with following information" )
public void message_id_are_generated_with_following_information (List<String> dataList)
{
    when( kafkaMsgUtil.genUUIDWithoutDash() ).thenAnswer( AdditionalAnswers.returnsElementsOf(dataList) );
}
```

รูปที่ 4.4 เมธอดที่ถูกเรียกใช้งานของ Script ID 2 สำหรับการเตรียมข้อมูล

Script ID 3 : @And(“date time is {string} and order date time is {string}”)

```
@And( "date time is {string} and order date time is {string}")
public void date_time_is_and_order_date_time_is ( String DtTm, String OrdrDtTm ) throws DatatypeConfigurationException
{
    XMLGregorianCalendar DtTmObj = DatatypeFactory.newInstance()
            .newXMLGregorianCalendar(DtTm);
    when( isoUtil.createDateTime( timeZoneId )).thenReturn( DtTmObj );

    XMLGregorianCalendar OrdrDtTmObj = DatatypeFactory.newInstance()
            .newXMLGregorianCalendar(OrdrDtTm);
    when( isoUtil.buildOrderDateTime( timeZoneId,  tradeEntryTimestamp: null )).thenReturn( OrdrDtTmObj );

}
```

รูปที่ 4.5 เมธอดที่ถูกเรียกใช้งานของ Script ID 3 สำหรับการเตรียมข้อมูล

จากรูปที่ 4.3, 4.4 และ 4.5 เป็นเมธอดที่ใช้ในการจัดเตรียมข้อมูล โดยรูปที่ 4.3 เป็นการเตรียมข้อมูลให้ตรงตามรูปแบบข้อมูลที่จะส่งเข้ามายังไมโครเซอร์วิสผ่าน API หากข้อมูลไม่ตรงกับรูปแบบข้อมูลจะไม่สามารถส่งข้อมูลเข้าไมโครเซอร์วิสได้ รูปที่ 4.4 เป็นการกำหนด ID ของข้อความให้เป็นค่าเดียวกัน และรูปที่ 4.5 เป็นการกำหนดวันที่เวลาที่ใช้ในการถอนกองทุน สาเหตุที่ต้องทำการกำหนดค่าต่าง ๆ ที่กล่าวมานั้นเพื่อใช้ในการทดสอบ เนื่องจากค่าเหล่านี้มีการเปลี่ยนแปลงตลอดเวลา หากไม่กำหนดไว้จะยากต่อการทดสอบ

Script ID 4 : @When(“request is sent to redemption service”)

```
@When( "request is sent to redemption service" )
public void request_is_sent_to_redemption_service ()
{
    sentRequest();
    subscribe = this.responseFlux.subscribe( statusResponse -> response = statusResponse );
}
```

รูปที่ 4.6 เมธอดที่ถูกเรียกใช้งานของ Script ID 4

จากรูปที่ 4.6 เมธอดนี้ทำหน้าที่ส่งข้อมูลที่ได้จัดเตรียมไว้ไปยังไมโครเซอร์วิสผ่าน API และเพื่อเริ่มการทำงานของไมโครเซอร์วิสอีกด้วย

Script ID 5 : @And(“redemption produce fund account message {string} to legacy and get response {string}”)

```
@And( "redemption produce fund account message {string} to legacy and get response {string}" )
public void redemption_produce_fund_account_message_to_legacy_and_get_response ( String pathToFileExpect, String pathToFileResponse ) throws IOException, URISyntaxException
{
    LegacyBridgeRequest expected = convertJsonToObj( pathToFileExpect, LegacyBridgeRequest.class );
    Message<byte[]> result = output.receive( timeout: 10000, legacyBridgeRedemptionTopic);
    LegacyBridgeRequest resultObj = objectMapper.readValue( result.getPayload(), LegacyBridgeRequest.class);
    assertEquals( expected.toString(), resultObj.toString() );

    String messageKey = "7d2e31cd8f2b4d3ea90bb257f7ea0001";
    FundAccountListResponse response = convertJsonToObj( pathToFileResponse, FundAccountListResponse.class );
    sendEvent( buildMessage( response, messageKey ) );
}
```

รูปที่ 4.7 เมธอดที่ถูกเรียกใช้งานของ Script ID 5

Script ID 6 : @And(“redemption produce additional info message {string} to legacy and get response {string}”)

```
@And( "redemption produce additional info message {string} to legacy and get response {string}" )
public void redemption_produce_additional_info_message_to_legacy_and_get_response ( String pathToFileExpect, String pathToFileResponse ) throws IOException, URISyntaxException
{
    AdditionalInfoAPIRequest expected = convertJsonToObj( pathToFileExpect, AdditionalInfoAPIRequest.class );
    Message<byte[]> result = output.receive( timeout: 10000, legacyBridgeAdditionalComponentTopic);
    AdditionalInfoAPIRequest resultObj = objectMapper.readValue( result.getPayload(), AdditionalInfoAPIRequest.class);
    assertEquals( expected.toString(), resultObj.toString() );

    String messageKey = "7d2e31cd8f2b4d3ea90bb257f7ea0002";
    AdditionalInfoAPIResponse response = convertJsonToObj( pathToFileResponse, AdditionalInfoAPIResponse.class );
    sendEvent( buildMessage( response, messageKey ) );
}
```

รูปที่ 4.8 เมธอดที่ถูกเรียกใช้งานของ Script ID 6

จากรูปที่ 4.7 และ 4.8 เมธอดนี้ทำหน้าที่ส่งข้อมูลจำลองไปยัง Legacy Service จากนั้นจำลองข้อมูลที่ Legacy Service ตอบกลับมา เนื่องจากในระบบถอนกองทุนจะมีการส่งไปหา Legacy Service 2 รอบเพื่อให้ครบรอบการทำงานของไมโครเซอร์วิส ในการทดสอบจึงต้องส่งข้อมูล 2 รอบด้วยเช่นกัน แต่หากกรณีที่ไม่ผ่านการตรวจสอบจาก Legacy Service จะทำการส่งข้อมูลไปยัง Legacy Service เพียง 1 รอบเท่านั้นตามรูปที่ 4.7

Script ID 7 : @And(“redemption produce message to mx transaction with status PACK”)

```
@And( "redemption produce message to mx transaction with status PACK" )
public void redemption_produce_message_to_mx_transaction_with_status_PACK () throws IOException
{
    Message<byte[]> resultMsg = output.receive( timeout: 10000, transactionTopic);
    MxSetr004AndMxSetr016 result = objectMapper.readValue( resultMsg.getPayload(), MxSetr004AndMxSetr016.class );

    assertEquals( OrderStatus4Code.PACK,
            result.getOrdrInstrStsRpt() List<OrderInstructionStatusReportV04>
                    .get( 0 ) OrderInstructionStatusReportV04
                    .getStsRpt() Status24Choice
                    .getIndvOrdrDtlsRpt() List<IndividualOrderStatusAndReason7>
                    .get( 0 ) IndividualOrderStatusAndReason7
                    .getOrdrSts() OrderStatus5Choice
                    .getSts() );
}
```

รูปที่ 4.9 เมธอดที่ถูกเรียกใช้งานของ Script ID 7

จากรูปที่ 4.9 เมธอดนี้จะทำงานก็ต่อเมื่อข้อมูลผ่านการตรวจสอบจาก Legacy Service โดยจะทำหน้าที่ส่งข้อมูลไปยัง MxTransaction Topic ด้วยสถานะของคำสั่ง “PACK”

Script ID 8 : @And(“redemption produce message to mx transaction with status reject”)

```
@And( "redemption produce message to mx transaction with status reject" )
public void redemption_produce_message_to_mx_transaction_with_status_reject () throws IOException
{
    Message<byte[]> resultMsg = output.receive( timeout: 10000, transactionTopic);
    MxSetr004AndMxSetr016 result = objectMapper.readValue( resultMsg.getPayload(), MxSetr004AndMxSetr016.class );

    assertNull( result.getOrdrInstrStsRpt() List<OrderInstructionStatusReportV04>
                    .get( 0 ) OrderInstructionStatusReportV04
                    .getStsRpt() Status24Choice
                    .getIndvOrdrDtlsRpt() List<IndividualOrderStatusAndReason7>
                    .get( 0 ) IndividualOrderStatusAndReason7
                    .getOrdrSts() OrderStatus5Choice
                    .getSts() );
}
```

รูปที่ 4.10 เมธอดที่ถูกเรียกใช้งานของ Script ID 8

จากรูปที่ 4.10 เมธอดนี้จะทำงานก็ต่อเมื่อข้อมูลไม่ผ่านการตรวจสอบจาก Legacy Service โดยจะทำหน้าที่ส่งข้อมูลไปยัง MxTransaction Topic ด้วยสถานะของคำสั่ง “REJECT”

Script ID 9 : @And(“redemption produce message {string} to business rule and get response {string}”)

```
@And( "redemption produce message {string} to business rule and get response {string}" )
public void redemption_produce_message_to_business_rule_get_response (String expectedFile, String pathToFile) throws IOException, URISyntaxException, JSONException
{
    MxSetr004AndMxSetr016 expectedObj =  convertJsonToObj( expectedFile, MxSetr004AndMxSetr016.class );
    String expectedString = objectMapper.writeValueAsString( expectedObj );

    Message<byte[]> resultMsg = output.receive( timeout: 10000, businessRuleTopic );
    MxSetr004AndMxSetr016 result = objectMapper.readValue( resultMsg.getPayload(), MxSetr004AndMxSetr016.class );
    String actualString = objectMapper.writeValueAsString( result );

    JSONAssert.assertEquals( new JSONObject( expectedString ), new JSONObject( actualString ), strict: true );

    taRedemptionTransactionRepository.save( convertJsonToObj( pathToFile, TaRedemptionTransaction.class ) );
}
```

รูปที่ 4.11 เมธอดที่ถูกเรียกใช้งานของ Script ID 9

จากรูปที่ 4.11 เมธอดนี้จะถูกเรียกใช้งานเมื่อข้อมูลผ่านและไม่ผ่านการตรวจสอบจาก Legacy Service โดยเมธอดนี้จะทำหน้าที่ส่งข้อมูลไปยัง Business Rule จากนั้นจำลองข้อมูลที่ Business Rule จะตอบกลับมา

Script ID 10 : @And(“response {string} is insert to taRedemptionTransaction”)

```
@And( "response {string} is insert to taRedemptionTransaction" )
public void response_is_insert_to_taRedemptionTransaction ( String pathToFile ) throws IOException, URISyntaxException
{
    var msg :TaRedemptionTransaction  = convertJsonToObj( pathToFile, TaRedemptionTransaction.class );
    taRedemptionTransactionRepository.save( msg );
}
```

รูปที่ 4.12 เมธอดที่ถูกเรียกใช้งานของ Script ID 10

จากรูปที่ 4.12 เมธอดนี้ถูกเรียกใช่งานเมื่อไม่ผ่านการตรวจสอบจาก Legacy Service โดยเมธอดนี้จะส่งข้อมูลไปบันทึกลงฐานข้อมูล

Script ID 11 : @Then(“response return status DONE”)

```
@Then( "response return status DONE" )
public void response_return_status_DONE () throws InterruptedException
{
    checkSubscribeStatus();
    assertEquals( OrderStatus4Code.DONE, response.getOrderStatus().getStatus() );
}
```

รูปที่ 4.13 เมธอดที่ถูกเรียกใช้งานของ Script ID 11

จากรูปที่ 4.13 เมธอดนี้จะถูกเรียกใช้งานเมื่อผ่านการตรวจสอบและข้อมูลถูกส่งไปยัง Business Rule Topic แล้ว โดยเมธอดนี้จะทำการตรวจสอบว่าสถานะของออเดอร์นั้นเป็น “DONE” หรือไม่

Script ID 12 : @Then(“response return status reject with message {string}”)

```
@Then( "response return status reject with message {string}" )
public void response_return_status_reject_with_message ( String RejectMessage ) throws InterruptedException
{
    checkSubscribeStatus();
    assertEquals( RejectMessage, response.getOrderStatus().getReject() );
}
```

รูปที่ 4.14 เมธอดที่ถูกเรียกใช้งานของ Script ID 12

จากรูปที่ 4.14 เมธอดนี้จะถูกเรียกใช้งานเมื่อไม่ผ่านการตรวจสอบและข้อมูลถูกบันทึกลงในฐานข้อมูลแล้ว โดยเมธอดนี้จะทำการตรวจสอบว่าสถานะของคำสั่งนั้นเป็น “REJECT” หรือไม่และคำอธิบายของการ REJECT นั้นตรงกับกรณีที่ทดสอบหรือไม่

ตารางที่ 4.3 ผลการทดสอบของกรณีทดสอบระบบถอนกองทุนในการทดสอบอัตโนมัติ

Test Case Id	Test Case	Expected Result	Test Result
1	ส่งเช็คที่อยู่ลูกค้าที่ไม่ใช่ที่อยู่ที่ลงทะเบียนไว้ - จัดเตรียมข้อมูลที่มีที่อยู่ของลูกค้าส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API	- Legacy Service และ Business Rule ตรวจสอบผ่าน - สถานะของคำสั่งเป็น DONE	ผ่าน

2	โอนเงินเข้าบัญชีธนาคารของลูกค้า - จัดเตรียมข้อมูลที่มีบัญชีธนาคารของลูกค้าส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API	- Legacy Service และ Business Rule ตรวจสอบผ่าน - สถานะของคำสั่งเป็น DONE	ผ่าน
3	โอนเงินเข้าบัญชีธนาคารที่ไม่ใช่ของลูกค้า - จัดเตรียมข้อมูลที่มีบัญชีธนาคารและข้อมูลของเจ้าของบัญชีส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API	- Legacy Service และ Business Rule ตรวจสอบผ่าน - สถานะของคำสั่งเป็น DONE	ผ่าน
4	ส่งเช็คด้วยที่อยู่ลูกค้าแบบมีภาษีและ Withholding - จัดเตรียมข้อมูลที่มีที่อยู่ของลูกค้าและอัตราภาษีส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API	- Legacy Service และ Business Rule ตรวจสอบผ่าน - สถานะของคำสั่งเป็น DONE	ผ่าน
5	โอนเงินเข้าธนาคารลูกค้าพร้อม Tax lot - จัดเตรียมข้อมูลที่มีบัญชีธนาคารของลูกค้าและ Tax lot ส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API	- Legacy Service และ Business Rule ตรวจสอบผ่าน - สถานะของคำสั่งเป็น DONE	ผ่าน
6	เลขที่บัญชีของกองทุนที่ต้องการจะถอนไม่ถูกต้อง - จัดเตรียมข้อมูลที่มีเลขที่บัญชีของกองทุนลูกค้าส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT	ผ่าน

	- ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ	
7	คำแนะนำในการถอนกองทุนไม่ถูกต้อง - จัดเตรียมข้อมูลที่มีคำแนะนำในการถอนกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ	ผ่าน
8	ข้อมูลของกองทุนที่ต้องการจะถอนไม่ถูกต้อง - จัดเตรียมข้อมูลที่มีข้อมูลกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ	ผ่าน
9	จำนวนเงินที่ต้องการจะถอนจากกองทุนไม่ถูกต้อง - จัดเตรียมข้อมูลที่มีข้อมูลการถอนกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ	ผ่าน
10	ภาษีในการถอนกองทุนไม่ถูกต้อง - จัดเตรียมข้อมูลที่มีที่ภาษีในการถอนกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT	ผ่าน

	- ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ	
11	ผู้ใช้ไม่มีสิทธิ์ถอนกองทุนที่ต้องการจะถอน - จัดเตรียมข้อมูลที่มีเลขบัญชีของกองทุนและประเภทของลูกค้าส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ	ผ่าน
12	ไม่มีหน่วยหรือส่วนแบ่งของกองทุนที่สามารถถอนกองทุนได้ - จัดเตรียมข้อมูลที่มีหน่วยของกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ	ผ่าน
13	หน่วยที่สามารถถอนกองทุนได้ - จัดเตรียมข้อมูลที่มีหน่วยของกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ	ผ่าน
14	ไม่มี cdsc (Contingent Deferred Sales Charge) ซึ่งวิธีการเรียกเก็บค่าธรรมเนียมที่มักใช้ในกองทุนรูปแบบหนึ่ง - จัดเตรียมข้อมูลที่มี cdsc ของกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ	ผ่าน

	- ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล		
15	ต้องไม่มีประเภทของการถอนกองทุนสำหรับบัญชีที่ไม่มีผู้ดูแลหรือผู้ปกครองทรัพย์สินของผู้เป็นเจ้าของบัญชีอื่น ๆ - จัดเตรียมข้อมูลที่มีประเภทการถอนกองทุนและผู้ดูแลกองทุนส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ	ผ่าน
16	ประเภทของการถอนกองทุนต้องไม่เป็นช่องว่างสำหรับบัญชีที่มีผู้ดูแลหรือผู้ปกครองทรัพย์สินของผู้เป็นเจ้าของบัญชีอื่น ๆ - จัดเตรียมข้อมูลที่มีประเภทการถอนกองทุนและผู้ดูแลบัญชีส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ	ผ่าน
17	ประเภทของจำนวนเงินกับข้อมูลใน cdsc ไม่ตรงกัน - จัดเตรียมข้อมูลที่ cdsc ลูกค้าส่งไปยัง Legacy Service และ Business Rule ในการตรวจสอบข้อมูล - ส่งข้อความไปยัง MxTransaction Topic - ข้อมูลถูกส่งเข้าไมโครเซอร์วิสผ่าน API - บันทึกข้อมูลลงในฐานข้อมูล	- Legacy Service ตรวจสอบไม่ผ่าน - สถานะของคำสั่งเป็น REJECT - สถานะของการ REJECT ตรงกับกรณีที่ทดสอบ	ผ่าน

4.2 ซิงค์คอนเนคเตอร์

การทดสอบซิงค์คอนเนคเตอร์จะเป็นการทดสอบการจัดเก็บข้อมูลธุรกรรมของการถอนกองทุนและการแลกเปลี่ยนกองทุน และตรวจสอบว่าระบบสามารถแยกแยะประเภทของธุรกรรมและจัดเก็บในฐานข้อมูลของธุรกรรมประเภทนั้น ๆ ได้ถูกต้องหรือไม่ จึงแบ่งกรณีทดสอบได้ดังตารางที่ 4.4

ตารางที่ 4.4 กรณีทดสอบซิงค์คอนเนคเตอร์

Test Case ID	Test Case	Expected Result
1	ข้อมูลการถอนกองทุนที่ผ่านการตรวจสอบจาก Legacy Service - จัดเตรียมข้อมูลของการถอนกองทุนที่ Legacy Service ตรวจสอบผ่าน - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการถอนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น PACK
2	ข้อมูลการแลกเปลี่ยนกองทุนที่ผ่านการตรวจสอบจาก Legacy Service - จัดเตรียมข้อมูลของการแลกเปลี่ยนกองทุนที่ Legacy Service ตรวจสอบผ่าน - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการแลกเปลี่ยนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น PACK
3	ข้อมูลการถอนกองทุนที่ไม่ผ่านการตรวจสอบจาก Legacy Service หรือ Business Rule - จัดเตรียมข้อมูลของการถอนกองทุนที่ Legacy Service ตรวจสอบไม่ผ่าน - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการถอนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น REJECT
4	ข้อมูลการแลกเปลี่ยนกองทุนที่ไม่ผ่านการตรวจสอบจาก Legacy Service หรือ Business Rule - จัดเตรียมข้อมูลของการแลกเปลี่ยนกองทุนที่ Legacy Service ตรวจสอบไม่ผ่าน - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการแลกเปลี่ยนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น REJECT

5	ข้อมูลการถอนกองทุนที่ผ่านการตรวจสอบจาก Business Rule - จัดเตรียมข้อมูลของการถอนกองทุนที่ Business Rule ตรวจสอบผ่าน - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการถอนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น RECE
6	ข้อมูลการแลกเปลี่ยนกองทุนที่ผ่านการตรวจสอบจาก Business Rule - จัดเตรียมข้อมูลของการแลกเปลี่ยนกองทุนที่ Business Rule ตรวจสอบผ่าน - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการแลกเปลี่ยนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น RECE
7	ข้อมูลการถอนกองทุนที่ถูกส่งมาจาก TA Sink Connector - จัดเตรียมข้อมูลของการถอนกองทุนที่ส่งมาจาก TA Sink Connector - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการถอนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น DONE
8	ข้อมูลการแลกเปลี่ยนกองทุนที่ถูกส่งมาจาก TA Sink Connector - จัดเตรียมข้อมูลของการแลกเปลี่ยนกองทุนที่ส่งมาจาก TA Sink Connector - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการแลกเปลี่ยนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น DONE

เมื่อกำหนดกรณีทดสอบได้แล้วก็จะนำกรณีทดสอบเหล่านี้ไปสร้างการทดสอบแบบอัตโนมัติขึ้นมาได้ดังนี้

1. Feature File

เป็นไฟล์ที่ใช้ในการกำหนดสคริปต์การทดสอบ โดยในซิงค์คอนเนคเตอร์จะมีสคริปต์ การทดสอบตามกรณีทดสอบดังนี้

Test Case ID 1, 3, 5, 7 : ข้อมูลการถอนกองทุนที่ผ่านการตรวจสอบจาก Legacy Service, Business Rule, TA Sink Connector และข้อมูลการถอนกองทุนที่ไม่ผ่านการตรวจสอบจาก Legacy Service หรือ Business Rule.

จะมีการเก็บข้อมูลการถอนกองทุนที่ตรวจสอบจาก Legacy Service ผ่านในไฟล์ชื่อ redemption-pack.json, เก็บข้อมูลการถอนกองทุนที่ตรวจสอบจาก Business Rule ผ่านในไฟล์ชื่อ redemption-

rece.json, เก็บข้อมูลการถอนกองทุนที่ถูกส่งมาจาก TA Sink Connector ในไฟล์ชื่อ redemption-done.json และเก็บข้อมูลการถอนกองทุนที่ตรวจสอบจาก Legacy Service หรือ Business Rule ไม่ผ่านในไฟล์ชื่อ redemption-reject.json

```
Scenario: Incoming Redemption message from Service
  Given There is a Kafka message from file "request/redemption-pack.json"
  When This record has been processed
  Then This record should be in ta_redemption_transaction table
  And Order status should be PACK

Scenario: Incoming Redemption message from Rule
  Given There is a Kafka message from file "request/redemption-rece.json"
  When This record has been processed
  Then This record should be in ta_redemption_transaction table
  And Order status should be RECE

Scenario: Incoming Redemption message from TA Sink connector
  Given There is a Kafka message from file "request/redemption-done.json"
  When This record has been processed
  Then This record should be in ta_redemption_transaction table
  And Order status should be DONE

Scenario: Incoming Redemption message that validate failed from Service
  Given There is a Kafka message from file "request/redemption-reject.json"
  When This record has been processed
  Then This record should be in ta_redemption_transaction table
  And Order status should be REJECT
```

รูปที่ 4.15 สคริปต์การทดสอบของ Test Case ID 1, 5, 7, 3 ตามลำดับ

จากรูปที่ 4.15 สคริปต์การทดสอบนี้จะทำการเรียกไฟล์ที่เก็บข้อมูลการถอนกองทุนแต่ละไฟล์ตามกรณีทดสอบ แล้วนำข้อมูลไปจัดเตรียมแล้วส่งไปยัง Kafka Topic จากนั้นจะทำการตรวจสอบว่าข้อมูลการถอนกองทุนนี้ถูกเก็บในฐานข้อมูลของการถอนกองทุนหรือไม่ และจะมีการตรวจสอบสถานะคำสั่งตามกรณีทดสอบ โดยหากเป็นข้อมูลที่ผ่านการตรวจสอบจาก Legacy Service สถานะคำสั่งต้องเป็น “PACK”, หากเป็นข้อมูลที่ผ่านการตรวจสอบจาก Business Rule สถานะคำสั่งต้องเป็น “RECE”, หากเป็นข้อมูลที่ถูกส่งมาจาก TA Sink Connector สถานะคำสั่งต้องเป็น “DONE” และหากเป็นข้อมูลที่ไม่ผ่านการตรวจสอบจาก Legacy Service และ Business Rule สถานะคำสั่งต้องเป็น “REJECT”

Test Case ID 2, 4, 6, 8 : ข้อมูลการแลกเปลี่ยนกองทุนที่ผ่านการตรวจสอบจาก Legacy Service, Business Rule, TA Sink Connect และข้อมูลการแลกเปลี่ยนกองทุนที่ไม่ผ่านการตรวจสอบจาก Legacy Service หรือ Business Rule

จะมีการเก็บข้อมูลการแลกเปลี่ยนกองทุนที่ตรวจสอบจาก Legacy Service ผ่านในไฟล์ชื่อ exchange-pack.json, เก็บข้อมูลการแลกเปลี่ยนกองทุนที่ตรวจสอบจาก Business Rule ผ่านในไฟล์ชื่อ exchange-rece.json, เก็บข้อมูลการแลกเปลี่ยนกองทุนที่ถูกส่งมาจาก TA Sink Connector ในไฟล์ชื่อ exchange-done.json และเก็บข้อมูลการแลกเปลี่ยนกองทุนที่ตรวจสอบจาก Legacy Service หรือ Business Rule ไม่ผ่านในไฟล์ชื่อ exchange-reject.json

รูปที่ 4.16 สคริปต์การทดสอบของ Test Case ID 2, 6, 8, 4 ตามลำดับ

จากรูปที่ 4.16 สคริปต์การทดสอบนี้จะทำการเรียกไฟล์ที่เก็บข้อมูลการแลกเปลี่ยนกองทุนแต่ละไฟล์ตามกรณีทดสอบ แล้วนำข้อมูลไปจัดเตรียมแล้วส่งไปยัง Kafka Topic จากนั้นจะทำการตรวจสอบว่าข้อมูลการแลกเปลี่ยนกองทุนนี้ถูกเก็บในฐานข้อมูลของการแลกเปลี่ยนกองทุนหรือไม่ และจะมีการตรวจสอบสถานะคำสั่งตามกรณีทดสอบ โดยหากเป็นข้อมูลที่ผ่านการตรวจสอบจาก Legacy Service

สถานะคำสั่งต้องเป็น “PACK”, หากเป็นข้อมูลที่ผ่านการตรวจสอบจาก Business Rule สถานะคำสั่งต้องเป็น “RECE”, หากเป็นข้อมูลที่ถูกส่งมาจาก TA Sink Connector สถานะคำสั่งต้องเป็น “DONE” และหากเป็นข้อมูลที่ไม่ผ่านการตรวจสอบจาก Legacy Service และ Business Rule สถานะคำสั่งต้องเป็น “REJECT”

2. Step Definition

เป็นไฟล์ที่ใช้ในการสั่งการการทำงานฟังก์ชันต่าง ๆ ภายในโปรแกรมที่ต้องการทดสอบ ซึ่งแต่ละเมธอดในไฟล์นี้จะถูกเรียกตามชื่อของสคริปต์การทดสอบใน Feature File โดยแต่ละสคริปต์ทำการเรียกเมธอดดังนี้

ตารางที่ 4.5 เมธอดที่ถูกเรียกใช้งานในแต่ละสคริปต์การทดสอบซิงค์คอนเนคเตอร์

Script ID	ชื่อสคริปต์การทดสอบใน Feature File	ชื่อเมธอดที่ถูกเรียกใช้งานใน Step Definiton
1	Given There is a Kafka message from file “...”	@Given(“There is a Kafka message from file {string}”)
2	When This record has been processed	@When(“This record has been processed”)
3	Then This record should be in ta_redemption_transaction table	@Then(“This record should be in ta_redemption_transaction table”)
4	Then This record should be in ta_exchange_transaction table	@Then(“This record should be in ta_exchange_transaction table”)
5	And Order status should be PACK	@And(“Order status should be PACK”)
6	And Order status should be RECE	@And(“Order status should be RECE”)
7	And Order status should be DONE	@And(“Order status should be DONE”)
8	And Order status should be REJECT	@And(“Order status should be REJECT”)

โดยการทำงานของแต่ละเมธอดที่ถูกเรียกใช้งานมีดังนี้

Script ID 1 : @Given(“There is a Kafka message from file {string}”)

```
@Given("There is a Kafka message from file {string}")
public void There_is_a_Kafka_message_from_Intelligence_service_from_file(String pathToFile) throws IOException, URISyntaxException
{
    String key = "\""+kafkaMsgUtil.genUUIDwithoutDash()+"\"";
    MsgKeyForQuery = key.replace( target: "\"", replacement: "");
    redeemMsg = convertJsonToObj( pathToFile, MxSetr004AndMxSetr016.class);
    payload = buildMessage(redeemMsg, key);
}
```

รูปที่ 4.17 เมธอดที่ถูกเรียกใช้งานของ Script ID 1 สำหรับข้อมูลการถอนกองทุน

```
@Given("There is a Kafka message from file {string}")
public void There_is_a_Kafka_message_from_Intelligence_service_from_file(String pathToFile) throws IOException, URISyntaxException
{
    String key = "\""+kafkaMsgUtil.genUUIDwithoutDash()+"\"";
    MsgKeyForQuery = key.replace( target: "\"", replacement: "");
    exchangeMsg = convertJsonToObj( pathToFile, MxSetr004AndMxSetr013.class);
    payload = buildMessage(exchangeMsg, key);
}
```

รูปที่ 4.18 เมธอดที่ถูกเรียกใช้งานของ Script ID 2 สำหรับข้อมูลการแลกเปลี่ยนกองทุน

จากรูปที่ 4.17 และ 4.18 จะเห็นได้ว่ามี 2 เมธอดสำหรับประเภทของข้อมูลที่ต่างกัน ซึ่งในที่นี่ไม่ใช่เมธอดที่ซ้อนทับกันในไฟล์ Step Definition เดียวกัน แต่เป็นเมธอดที่อยู่ในไฟล์ Step Definition คนละไฟล์ ซึ่งแยกเป็นไฟล์สำหรับรับข้อมูลการถอนกองทุน และไฟล์สำหรับรับข้อมูลการแลกเปลี่ยนกองทุน แล้วเมื่อทำการรันการทดสอบอัตโนมัติจะมีการตั้งค่าการเรียกไฟล์ Step Definition ที่ต่างกันสำหรับข้อมูลแต่ละประเภท เนื่องจากเมธอดนี้จะทำการจัดเตรียมข้อมูลให้ตรงตามรูปแบบข้อมูลที่สามารถส่งเข้าไปที่ Kafka Topic ได้ โดยในการจัดเตรียมข้อมูลนั้นจะต้องมีการจำแนกข้อมูลบางส่วนต่าง ๆ ออกมาที่ข้อมูลของการถอนกองทุนและการแลกเปลี่ยนกองทุนมีความแตกต่างกัน โดยในการทำงานของเมธอดนี้จะทำการสร้าง Key ของข้อมูลขึ้นมา แล้วทำการเก็บ Key ของข้อมูลไว้ในตัวแปร MsgKeyForQuery จากนั้นจะนำข้อมูลไปแปลงให้อยู่ใน Class แล้วทำการส่ง Class ของข้อมูลและ Key เพื่อไปจัดเตรียมข้อมูลให้ตรงตามรูปแบบข้อมูลที่สามารถส่งไปที่ Kafka Topic ได้

Script ID 2 : @When(“This record has been processed”)

```
@When("This record has been processed")
public void this_record_has_been_processed() { super.sinkToDB(payload); }
```

รูปที่ 4.19 เมธอดที่ถูกเรียกใช้งานของ Script ID 2

จากรูปที่ 4.19 เมธอดนี้ทำหน้าที่ในการส่งข้อมูลที่จัดเตรียมไว้ไปที่ Kafka Topic เพื่อ Trigger ให้ระบบเริ่มการทำงานในการนำข้อมูลมาจำแนกประเภทแล้วเก็บในฐานข้อมูลของข้อมูลประเภทนั้น ๆ

Script ID 3 : @Then(“This record should be in ta_redemption_transaction table”)

```
@Then("This record should be in ta_redemption_transaction table")
public void This_record_should_be_in_ta_redemption_transaction_table()
{
    var MsgKeyFromDB : Optional<TaRedemptionTransaction> = redemptionRepository.findById(MsgKeyForQuery);
    var Message : TaRedemptionTransaction = MsgKeyFromDB.get();

    assertEquals(MsgKeyForQuery, String.valueOf(Message.getMsgKey()));
}
```

=

รูปที่ 4.20 เมธอดที่ถูกเรียกใช้งานของ Script ID 3

จากรูปที่ 4.20 เมธอดนี้ทำงานเมื่อทำการทดสอบด้วยข้อมูลการถอนกองทุน ซึ่งทำหน้าที่ในการตรวจสอบฐานข้อมูลของการถอนกองทุน เพื่อแสดงว่าข้อมูลที่ใช้ในการทดสอบถูกเก็บในฐานข้อมูลของการถอนกองทุนหรือไม่ โดยจะทำการสั่งคิวรี่เพื่อค้นหาข้อมูลในฐานข้อมูลด้วย Key ของข้อมูลที่ถูกสร้างและเก็บไว้ในตัวแปร MsgKeyForQuery ว่ามี Key ของข้อมูลภายในฐานข้อมูลตรงกับ Key ของข้อมูลที่ถูกเก็บไว้ในตัวแปร MsgKeyForQuery หรือไม่ แล้วจะทำการเก็บข้อมูลในฐานข้อมูลที่มี Key ตรงกับ Key ที่เก็บในตัวแปร MsgKeyForQuery จากนั้นทำการเปรียบเทียบ Key ที่เก็บในตัวแปร MsgKeyForQuery กับ Key ของข้อมูลในฐานข้อมูล หากไม่ตรงกันแปลว่าตัวแปรที่ทำการเก็บข้อมูลไว้นั้นไม่มีข้อมูลให้เปรียบเทียบ เนื่องจากค้นหา Key ที่เก็บในตัวแปร MsgKeyForQuery กับ Key ของข้อมูลที่อยู่ในฐานข้อมูลไม่เจอ และจะไม่ผ่านการทดสอบ หรือหากตรงกัน จะผ่านการทดสอบ

Script ID 4 : @Then(“This record should be in ta_exchange_transaction table”)

```
@Then("This record should be in ta_exchange_transaction table")
public void This_record_should_be_in_ta_exchange_transaction_table()
{
    var MsgKeyFromDB : Optional<TaExchangeTransaction> = exchangeRepository.findById(MsgKeyForQuery);
    var Message : TaExchangeTransaction = MsgKeyFromDB.get();

    assertEquals(MsgKeyForQuery, String.valueOf(Message.getMsgKey()));
}
```

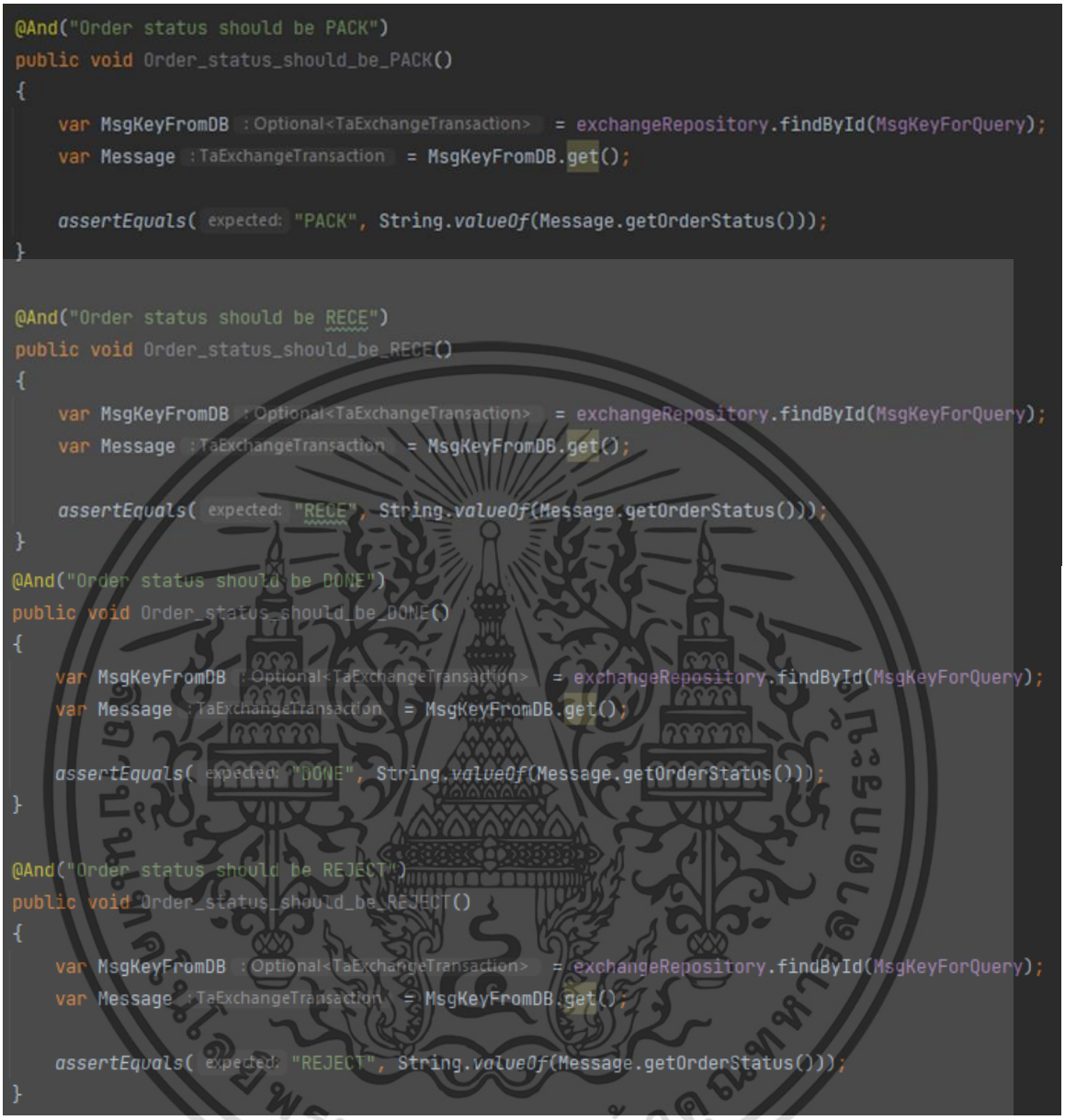
รูปที่ 4.21 เมธอดที่ถูกเรียกใช้งานของ Script ID 4

จากรูปที่ 4.21 เมธอดนี้ทำงานเมื่อทำการทดสอบด้วยข้อมูลการแลกเปลี่ยนกองทุน ซึ่งทำหน้าที่ในการตรวจสอบฐานข้อมูลของการแลกเปลี่ยนกองทุน เพื่อแสดงว่าข้อมูลที่ใช้ในการทดสอบถูกเก็บในฐานข้อมูลของการแลกเปลี่ยนกองทุนหรือไม่ โดยจะทำการสั่งคิวรี่เพื่อค้นหาข้อมูลในฐานข้อมูลด้วย Key ของข้อมูลที่ถูกสร้างและเก็บไว้ในตัวแปร MsgKeyForQuery ว่ามี Key ของข้อมูลภายในฐานข้อมูลตรงกับ Key ของข้อมูลที่ถูกเก็บไว้ในตัวแปร MsgKeyForQuery หรือไม่ แล้วจะทำการเก็บข้อมูลในฐานข้อมูลที่มี Key ตรงกับ Key ที่เก็บในตัวแปร MsgKeyForQuery จากนั้นทำการเปรียบเทียบ Key ที่เก็บในตัวแปร MsgKeyForQuery กับ Key ของข้อมูลในฐานข้อมูล หากไม่ตรงกันแปลว่าตัวแปรที่ทำการเก็บข้อมูลไว้นั้นไม่มีข้อมูลให้เปรียบเทียบ เนื่องจากค้นหา Key ที่เก็บในตัวแปร MsgKeyForQuery กับ Key ของข้อมูลที่อยู่ในฐานข้อมูลไม่เจอ และจะไม่ผ่านการทดสอบ หรือหากตรงกัน จะผ่านการทดสอบ

Script ID 5, 6, 7, 8 : @And(“Order status should be …”)

```
@And("Order status should be PACK")
public void Order_status_should_be_PACK()
{
    var MsgKeyFromDB :Optional<TaRedemptionTransaction> = redemptionRepository.findById(MsgKeyForQuery);
    var Message :TaRedemptionTransaction = MsgKeyFromDB.get();

    assertEquals( expected: "PACK", String.valueOf(Message.getOrderStatus()));
}

@And("Order status should be RECE")
public void Order_status_should_be_RECE()
{
    var MsgKeyFromDB :Optional<TaRedemptionTransaction> = redemptionRepository.findById(MsgKeyForQuery);
    var Message :TaRedemptionTransaction = MsgKeyFromDB.get();

    assertEquals( expected: "RECE", String.valueOf(Message.getOrderStatus()));
}
@And("Order status should be DONE")
public void Order_status_should_be_DONE()
{
    var MsgKeyFromDB :Optional<TaRedemptionTransaction> = redemptionRepository.findById(MsgKeyForQuery);
    var Message :TaRedemptionTransaction = MsgKeyFromDB.get();

    assertEquals( expected: "DONE", String.valueOf(Message.getOrderStatus()));
}

@And("Order status should be REJECT")
public void Order_status_should_be_REJECT()
{
    var MsgKeyFromDB :Optional<TaRedemptionTransaction> = redemptionRepository.findById(MsgKeyForQuery);
    var Message :TaRedemptionTransaction = MsgKeyFromDB.get();

    assertEquals( expected: "REJECT", String.valueOf(Message.getOrderStatus()));
}
```

รูปที่ 4.22 เมธอดที่ถูกเรียกใช้งานของ Script ID 5, 6, 7, 8 สำหรับข้อมูลการถอนกองทุน



```
@And("Order status should be PACK")
public void Order_status_should_be_PACK()
{
    var MsgKeyFromDB :Optional<TaExchangeTransaction> = exchangeRepository.findById(MsgKeyForQuery);
    var Message :TaExchangeTransaction = MsgKeyFromDB.get();

    assertEquals( expected: "PACK", String.valueOf(Message.getOrderStatus()));
}

@And("Order status should be RECE")
public void Order_status_should_be_RECE()
{
    var MsgKeyFromDB :Optional<TaExchangeTransaction> = exchangeRepository.findById(MsgKeyForQuery);
    var Message :TaExchangeTransaction = MsgKeyFromDB.get();

    assertEquals( expected: "RECE", String.valueOf(Message.getOrderStatus()));
}
@And("Order status should be DONE")
public void Order_status_should_be_DONE()
{
    var MsgKeyFromDB :Optional<TaExchangeTransaction> = exchangeRepository.findById(MsgKeyForQuery);
    var Message :TaExchangeTransaction = MsgKeyFromDB.get();

    assertEquals( expected: "DONE", String.valueOf(Message.getOrderStatus()));
}

@And("Order status should be REJECT")
public void Order_status_should_be_REJECT()
{
    var MsgKeyFromDB :Optional<TaExchangeTransaction> = exchangeRepository.findById(MsgKeyForQuery);
    var Message :TaExchangeTransaction = MsgKeyFromDB.get();

    assertEquals( expected: "REJECT", String.valueOf(Message.getOrderStatus()));
}
```

รูปที่ 4.23 เมธอดที่ถูกเรียกใช้งานของ Script ID 5, 6, 7, 8 สำหรับข้อมูลการแลกเปลี่ยนกองทุน

จากรูปที่ 4.22 เมธอดทั้งหมดในรูปนี้จะถูกเรียกใช้งานเมื่อทำการทดสอบด้วยข้อมูลการถอนกองทุน จากรูปที่ 4.23 เมธอดทั้งหมดในรูปนี้จะถูกเรียกใช้งานเมื่อทำการทดสอบด้วยข้อมูลการแลกเปลี่ยนกองทุน โดยเมธอดในทั้งสองรูปนี้จะทำหน้าที่ตรวจสอบสถานะคำสั่งในข้อมูลที่อยู่ในฐานข้อมูลของประเภทข้อมูลนั้น ๆ ขึ้นอยู่กับข้อมูลที่ใช้ทดสอบ โดยมีการทำงานคล้าย ๆ กับ Script ID 3, 4 แตกต่างตรงที่ในส่วนที่ทำการเปรียบเทียบข้อมูลจะเปรียบเทียบสถานะคำสั่งของข้อมูลกับสถานะคำสั่ง

ที่คาดหวังไว้ ซึ่งหากไม่ตรงกันแปลว่าสถานะคำสั่งไม่ตรงกันกับที่คาดหวังไว้ หรือหากตรงกันแปลว่าสถานะคำสั่งตรงกันกับที่คาดหวังไว้

ตารางที่ 4.6 ผลการทดสอบของกรณีทดสอบชิงค์คอนเนคเตอร์ในการทดสอบอัตโนมัติ

Test Case ID	Test Case	Expected Result	Test Result
1	ข้อมูลการถอนกองทุนที่ผ่านการตรวจสอบจาก Legacy Service - จัดเตรียมข้อมูลของการถอนกองทุนที่ Legacy Service ตรวจสอบผ่าน - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการถอนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น PACK	ผ่าน
2	ข้อมูลการแลกเปลี่ยนกองทุนที่ผ่านการตรวจสอบจาก Legacy Service - จัดเตรียมข้อมูลของการแลกเปลี่ยนกองทุนที่ Legacy Service ตรวจสอบผ่าน - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการแลกเปลี่ยนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น PACK	ผ่าน
3	ข้อมูลการถอนกองทุนที่ไม่ผ่านการตรวจสอบจาก Legacy Service หรือ Business Rule - จัดเตรียมข้อมูลของการถอนกองทุนที่ Legacy Service ตรวจสอบไม่ผ่าน - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการถอนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น REJECT	ผ่าน
4	ข้อมูลการแลกเปลี่ยนกองทุนที่ไม่ผ่านการตรวจสอบจาก Legacy Service หรือ Business Rule - จัดเตรียมข้อมูลของการแลกเปลี่ยนกองทุนที่ Legacy Service ตรวจสอบไม่ผ่าน - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการแลกเปลี่ยนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น REJECT	ผ่าน

5	ข้อมูลการถอนกองทุนที่ผ่านการตรวจสอบจาก Business Rule - จัดเตรียมข้อมูลของการถอนกองทุนที่ Business Rule ตรวจสอบผ่าน - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการถอนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น RECE	ผ่าน
6	ข้อมูลการแลกเปลี่ยนกองทุนที่ผ่านการตรวจสอบจาก Business Rule - จัดเตรียมข้อมูลของการแลกเปลี่ยนกองทุนที่ Business Rule ตรวจสอบผ่าน - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการแลกเปลี่ยนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น RECE	ผ่าน
7	ข้อมูลการถอนกองทุนที่ถูกส่งมาจาก TA Sink Connector - จัดเตรียมข้อมูลของการถอนกองทุนที่ส่งมาจาก TA Sink Connector - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการถอนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น DONE	ผ่าน
8	ข้อมูลการแลกเปลี่ยนกองทุนที่ถูกส่งมาจาก TA Sink Connector - จัดเตรียมข้อมูลของการแลกเปลี่ยนกองทุนที่ส่งมาจาก TA Sink Connector - ส่งข้อมูลไปยัง MxTransaction Topic	- ข้อมูลถูกจัดเก็บในฐานข้อมูลของการแลกเปลี่ยนกองทุน - ภายในข้อมูลมีสถานะคำสั่งเป็น DONE	ผ่าน

บทที่ 5
สรุปผลการดำเนินงานและข้อเสนอแนะ

5.1 สรุปผลการดำเนินงาน

ในระยะเวลา 6 เดือนที่ทางผู้จัดทำได้มีโอกาสเข้าร่วมโครงการสหกิจศึกษากับบริษัท เอสเอส แอนด์ซี เทคโนโลยี จำกัด เพื่อพัฒนาการทดสอบอัตโนมัติของระบบซื้อขายกองทุน ซึ่งได้รับมอบหมายในการพัฒนาการทดสอบอัตโนมัติส่วนระบบการถอนกองทุนและซิงค์คอนเนคเตอร์ โดยมีวัตถุประสงค์เพื่อทำให้ระบบซื้อขายกองทุนนั้นสมบูรณ์ อีกทั้งยังลดระยะเวลาในการทดสอบ โดยในการพัฒนานั้นจะพัฒนาบน Spring Boot ซึ่งเป็นเฟรมเวิร์คที่ช่วยในการสร้างเว็บแอปพลิเคชันหรือเว็บเซอร์วิสได้ง่ายขึ้น ลดปัญหาความซับซ้อนในการตั้งค่าโปรเจค อีกทั้งยังอาศัย Kafka ในการรับและส่งข้อมูลระหว่างเซอร์วิส โดยใช้ Cucumber ในการพัฒนาสคริปต์การทดสอบและอาศัยหลักการของ CI/CD ร่วมด้วย โดยในกระบวนการทำงานของการทดสอบอัตโนมัติของระบบซื้อขายกองทุน คือ การจำลองการส่งข้อมูลไปยังเซอร์วิสต่าง ๆ และเปรียบเทียบว่าข้อมูลที่ส่งไปยังเซอร์วิสและข้อมูลที่เซอร์วิสตอบกลับมานั้นถูกต้องหรือไม่ อีกทั้งยังต้องตรวจสอบว่าสถานะของคำสั่งนั้นตรงตามที่คาดหวังหรือไม่ และการจำลองข้อมูลที่มาจากเซอร์วิสต่าง ๆ และนำข้อมูลนั้นไปบันทึกว่าระบบสามารถบันทึกข้อมูลลงในฐานข้อมูลของเซอร์วิสที่ถูกต้องหรือไม่ โดยใช้ฐานข้อมูล H2 ในการจำลองฐานข้อมูลตามแบบฐานข้อมูลจริงขึ้นมาเพื่อใช้ในการทดสอบระบบอีกด้วยและเมื่อได้ทำการทดสอบในทุกกรณีแล้วพบว่าระบบทำงานตรงตามที่คาดหวังไว้ทุกกรณี ทำให้การทดสอบอัตโนมัติที่สร้างขึ้นนี้จะช่วยทำการทดสอบระบบเมื่อมีการเปลี่ยนแปลง แก้ไข หรืออัพเดตใด ๆ ในระบบได้ถูกต้องตามที่เคยทำการทดสอบแบบแมนนวลไว้อย่างมั่นใจ

5.2 ข้อเสนอแนะ

ในอนาคตควรทำการทดสอบตั้งแต่ต้นจนจบเพื่อความสมบูรณ์ของระบบ เนื่องจากการทดสอบที่ทางคณะผู้จัดทำพัฒนาเป็นการทดสอบระบบโดยรวมเท่านั้น หากมีการทดสอบตั้งแต่ต้นจนจบอาจทำให้ระบบมีความสมบูรณ์มากยิ่งขึ้น

เอกสารอ้างอิง

[1] Sabyeying, B. 2565. Intro to Apache Kafka Universe: Kafka คืออะไร ?. [Online]. https://mesodiar.com/intro-to-apache-kafka-universe-kafka-คืออะไร-45f652e8233f.

[2] Sketo, E. 2564. Apache Kafka – An Open Source Distributed event streaming platform. [Online]. https://esketchers.com/apache-kafka-an-open-source-distributed/#What_is_Kafka.

[3] Amornrattanakij, T. 2563. Spring Boot มีไว้ทำอะไร ?. [Online]. https://medium.com/@Teerawat.amo/spring-boot-มีไว้ทำอะไร-c1d84a7796d7.

[4] Marupat. 2564. Spring Boot : Spring Boot คืออะไร. [Online]. https://marupatnote.home.blog/2021/08/01/spring-boot-what-is-it/?fbclid=IwAR0lpNIlK1zWtX60DZZpTUNzo2GxK0s0wri-o4U3xYut98ULjJgmaIpXlpc.

[5] Spring Boot. 2565. Spring Boot. [Online]. https://spring.io/projects/spring-boot?fbclid=IwAR0w53ZdRFRiDLImMiPakStXw1cqAvdAU4OFU8POu_OGmj8qReQcM9_tPd4.

[6] Mindphp. 2565. วิธีการใช้งาน Postman. [Online]. https://www.mindphp.com/บทความ/31-ความรู้ทั่วไป/9164-using-postman.html?fbclid=IwAR0-bc2Nzoqjxcp5_yUUSyZXAI7nzjSRFAWZuf5F6fOjRO4q21_DxzJSK0Y.

[7] Prakobdee, W. 2561. ทำให้ JUnit กลับมายิ่งใหญ่อีกครั้ง มาใช้งาน JUnit 5 กันเถอะครับ. [Online]. https://medium.com/linedevth/ทำให้-junit-กลับมายิ่งใหญ่อีกครั้ง-มาใช้งาน-junit-5-กันเถอะครับ-10ea14dec817.

[8] Somkiat. 2557. แนะนำโครงสร้างของ unit test ที่ดี. [Online]. https://www.somkiat.cc/junit-good-struture/?fbclid=IwAR00xRIHRDHyRlOHXu3qV1fLAEj82o79ULgosClVBwEdllSg8iEvETclo_U.

[9] Singh, R. 2565. The Testing Pyramid: Simplified for One and All. [Online]. https://www.headspin.io/blog/the-testing-pyramid-simplified-for-one-and-all?fbclid=IwAR1v8LXDvYLuVUrrCK_SXvXd-MS_7qXhUrRX0ozs6C-aL591Cl3QPHWBBZI.

[10] Linux-console.net. PostgreSQL คืออะไร? PostgreSQL ทำงานอย่างไร. [Online]. https://th.linux-console.net/?p=1801#gsc.tab=0.

[11] Ae. 2560. PostgreSQL โพสต์เกรสคิวเอล คืออะไร โปรแกรมสำหรับจัดการข้อมูล. [Online]. https://www.mindphp.com/คู่มือ/73-คืออะไร/3872-what-is-postgresql.html.

[12] Pongsupankij, N. 2560. ใครๆก็เขียนเทสได้ ถ้ามี Cucumber. [Online]. https://developers.ascendcorp.com/anyone-can-test-3c2ff810ab25.

[13] Potivejkul, T. 2565. Gherkin and Cucumber ทางเลือกใหม่ในการจัดการ Automation Testing ให้เป็นระบบ. [Online]. https://www.stream.co.th/gherkin-and-cucumber-automation-testing/.

[14] Marupat. 2564. Spring Boot : ฐานข้อมูล H2. From: https://marupatnote.home.blog/2021/08/03/spring-boot-h2/ [28 กันยายน 2565]

[15] Peem Srinikorn. ทำความรู้จักกับ CI/CD services บน Google Cloud. From: https://cloud-ace.co.th/blogs/w0w7e1-ci-cd-services-google-cloud [28 กันยายน 2565]

[16] Natchaya Piyapan. มารู้จักกับ CI/CD ตัวช่วยให้งานโปรแกรมเมอร์ง่ายขึ้น. From: https://www.codium.co/blogs/33-CICD [28 กันยายน 2565]

[17] Suppawat K. CI/CD คืออะไร? ช่วยให้ Developer ทำงานง่ายขึ้นได้มากขนาดไหน?. From: https://blog.cloudhm.co.th/ci-cd/ [30 เมษายน 2566]

[18] MDSoft. ความรู้เบื้องต้นเกี่ยวกับ Acceptance testing. From: https://www.mdsoft.co.th/ความรู้/175-acceptance-testing.html [30 เมษายน 2566]

งานทะเบียนคณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

คำรับรองเล่มโครงงานพิเศษ/ปัญหาพิเศษ/สหกิจศึกษา

วันที่ 29 เดือนพฤษภาคม พ.ศ.2566

ข้าพเจ้า นายกฤษฎา ชิตนุกูล รหัสประจำตัว 62050127

นางสาวลภัสรดา พงศ์พันธ์พฤทธิ์ รหัสประจำตัว 62050219

นักศึกษาหลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชาวิทยาศาสตร์ ภาควิชาวิทยาการคอมพิวเตอร์ ขอรับรองว่าสหกิจศึกษา เรื่อง

ชื่อภาษาไทย การพัฒนาการทดสอบอัตโนมัติสำหรับระบบถอนกองทุนและซิงค์คอนเนคเตอร์แอปพลิเคชัน

ชื่อภาษาอังกฤษ Development of the automation testing suite for the redemption and sink connector systems

ปีการศึกษา 2565

เป็นผลงานวิจัยที่มิได้คัดลอกหรือละเมิดลิขสิทธิ์ของผู้อื่นและได้ผ่านการตรวจสอบความซ้ำซ้อนเรียบร้อยแล้ว และได้แนบเอกสารการตรวจสอบการลอกเลียนงานวรรณกรรมที่ตรวจสอบจากเล่มโครงงานพิเศษ/ปัญหาพิเศษ/สหกิจศึกษาฉบับสมบูรณ์แล้ว

โปรแกรมอักขราวิสุทธิ์ 0.33%

ลงชื่อ.....กฤษฎา..... ลงชื่อ.....ลภัสรดา.....

(กฤษฎา ชิตนุกูล) (ลภัสรดา พงศ์พันธ์พฤทธิ์)

นักศึกษา นักศึกษา

ข้าพเจ้าผศ.ดร.อนันตพร หรรษคุณาฒัย อาจารย์ที่ปรึกษาสหกิจศึกษา ได้ตรวจสอบสหกิจศึกษาของนักศึกษาข้างต้นแล้ว ขอรับรองว่าเป็นผลงานวิจัยของนักศึกษาจริงและมีเนื้อหาสมบูรณ์ จึงลงชื่อไว้เป็นหลักฐาน

ลงชื่อ.....

(ผศ.ดร.อนันตพร หรรษคุณาฒัย)

อาจารย์ที่ปรึกษา