ไมโครเซอร์วิสจำลองการทำงานของ Calastone API และ HiTrust Integration

ณัฐฑริกา ทองแถม

สหกิจศึกษานี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
ปริญญาวิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2565

Mock Calastone Service and HiTrust Integration

Nuttharika Thongtham

A COOPERATIVE EDUCATION SUBMITTED

IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR

THE DEGREE OF BACHELOR OF SCIENCE (COMPUTER SCIENCE)

DEPARTMENT OF COMPUTER SCIENCE, FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2022

หัวข้อสหกิจศึกษา	ไมโครเซอร์วิสจำลองการทำงานของ Calastone API และ HiTrust Integration MOCK CALASTONE SERVICE AND HITRUST INTEGRATION
ชื่อนักศึกษา	นางสาว ณัฐฑริกา ทองแถม รหัสนักศึกษา 62050154
ปริญญา	วิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชา	วิทยาการคอมพิวเตอร์
ปีการศึกษา	2565
อาจารย์ที่ปรึกษา	ดร. วิชญะ ต่อวงศ์ไพชยนต์

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

คณะกรรมการสอบ	ลายมือชื่อ
อ.สันธนะ อู่อุดมยิ่ง ประธานกรรมการ	
ดร. วิชญะ ต่อวงศ์ไพชยนต์ กรรมการและอาจารย์ที่ปรึกษา	

ลิขสิทธิ์ของคณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

หัวข้อสหกิจศึกษา	ไมโครเซอร์วิสจำลองการทำงานของ Calastone API และ HiTrust Integration MOCK CALASTONE SERVICE AND HITRUST INTEGRATION
ชื่อนักศึกษา	นางสาว ณัฐฑริกา ทองแถม รหัสนักศึกษา 62050154
ปริญญา	วิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชา	วิทยาการคอมพิวเตอร์
ปีการศึกษา	2565
อาจารย์ที่ปรึกษา	ดร. วิชญะ ต่อวงศ์ไพชยนต์

บทคัดย่อ

โปรเจคในโครงการสหกิจศึกษาครั้งนี้มีวัตถุประสงค์เพื่อพัฒนาไมโครเซอร์วิสที่จำลองพฤติกรรมของ Calastone API และ HiTrust Integration สำหรับบริษัท SS&C Technology โดยมีเป้าหมายคือเพื่อลดต้นทุนในการทดสอบระบบ Calastone ร่วมกับไมโครเซอร์วิสที่เป็นผลิตภัณฑ์ของบริษัท ตลอดจนแก้ปัญหาการใช้ HiTrust Integration ที่ไม่สามารถใช้ร่วมกับการทดสอบบน US Environment ได้ กระบวนการพัฒนาไมโครเซอร์วิสชิ้นนี้ถูกพัฒนาขึ้นบน Spring Boot Framework และใช้ฐานข้อมูล H2 รวมถึง Kafka และ JUnit สำหรับการทดสอบ เทคโนโลยีเหล่านี้ถูกนำมาใช้เพื่อจำลองการทำงานของ Calastone API และ HiTrust Integration ทำให้สามารถทดสอบระบบได้อย่างมีประสิทธิภาพและคุ้มค่า นอกจากนี้ การใช้ Spring Boot และฐานข้อมูล H2 ทำให้ไมโครเซอร์วิสชิ้นนี้สามารถรวมเข้ากับโครงสร้างพื้นฐานที่มีอยู่ของบริษัทได้อย่างง่ายดาย กระบวนการเริ่มต้นเมื่อ Intelligence Service ได้รับข้อมูลและส่งมายัง ไมโครเซอร์วิสจำลองการทำงานของ Calastone API จากนั้นจึงส่งไปยัง HiTrust Integration เพื่อส่งต่อไปยัง Event Messaging Service และทำการทดสอบแบบครบวงจรการทำงาน

คำสำคัญ : Spring Boot Framework, ไมโครเซอร์วิส, Kafka, ฐานข้อมูล H2

Title	MOCK CALASTONE SERVICE AND HITRUST INTEGRATION
Student	Miss Nuttharika Thongtham Student ID 62050154
Department	Computer Science
School	Science
University	King Mongkut's Institute of Technology Ladkrabang (KMITL)
Academic Year	2022
Advisor	Dr.Witchaya Towongpaichayont

Abstract

The project aimed to develop a simulation of the Calastone API and HiTrust Integration for the SS&C Technology company. The goal was to reduce costs associated with testing the Calastone system and to integrate it with Intelligence Service and Event Messaging Service, as well as to solve the problem of HiTrust not being able to be used on US Environment. The project development was created on Spring Boot Framework and used H2 Database as well as Kafka and JUnit for testing. These technologies were utilized to simulate the functionality of Calastone API and HiTrust Integration, allowing for efficient and cost-effective test of the systems. Additionally, the use of Spring Boot and H2 Database enabled the project to be easily integrated into the existing infrastructure of the company. The process began when the Intelligence Service received data and passed it through the simulated Calastone API, which then sent it to HiTrust Integration. The benefits of the project include cost savings and the ability to use HiTrust on US Environment.

Keyboard : Spring Boot Framework, Microsoft, Kafka, H2 Database

กิตติกรรมประกาศ

การจัดทำสหกิจศึกษาครั้งนี้บรรลุเป้าหมายและสำเร็จลุล่วงไปได้ด้วยดีโดยได้รับความช่วยเหลือและการสนับสนุนจากบุคลากรหลายท่าน ผู้จัดทำขอขอบพระคุณบุคคลต่าง ๆ ที่คอยให้ความช่วยเหลือและสละเวลามาให้คำแนะนำ คำปรึกษามาโดยตลอด อันได้แก่

ครอบครัวอันได้แก่บิดาและมารดาที่คอยสนับสนุนการศึกษามาโดยตลอด รวมไปจนถึงปัจจัยด้านอื่น ๆ และเป็นกำลังใจสำคัญ

ดร. วิชญะ ต่อวงศ์ไพชยนต์ และ อ.สันธนะ อู่อุดมยิ่ง ที่สละเวลาอันมีค่าคอยให้คำแนะนำและคำปรึกษาตลอดช่วงระยะเวลาการจัดทำสหกิจครั้งนี้จนบรรลุผลสำเร็จไปได้ด้วยดี

บริษัท SS&C Technologies หัวหน้าทีมและพี่เลี้ยงทุกท่าน ที่โอกาสและประสบการณ์ในการเป็นนักศึกษาฝึกงานตำแหน่ง Java Developer รวมไปถึงความรู้ในการทำงานร่วมกับผู้อื่น การทำงานในองค์กร ความรู้ทางด้านวิทยาการคอมพิวเตอร์ตั้งแต่พื้นฐานของการสร้าง Software ไปจนถึงการ Deploy ใช้งานจริง ทำให้ผลงานที่เกิดขึ้นสำเร็จและปฏิบัติงานได้อย่างราบรื่น

คณาจารย์ภาควิชาวิทยาการคอมพิวเตอร์และสาขาอื่น ๆ ที่อบรบสั่งสอน ให้ความรู้ ความสามารถแนวคิด ทั้งการทำงานและการใช้ชีวิตในอนาคต ตลอดระยะเวลา 4 ปีที่ได้ศึกษา ณ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง รวมไปถึงเพื่อน ๆ ที่คอยเป็นกำลังใจและให้ความช่วยเหลือมาโดยตลอด

สุดท้ายนี้ ยังมีบุคคลท่านอื่น ๆ ที่ไม่ได้กล่าวถึงไว้ ณ ที่นี้ ขอขอบพระคุณทุกท่าน ที่ได้ สละเวลาส่วนตนมาเพื่อช่วยเหลือ ให้คำปรึกษา ให้การสนับสนุน และเป็นกำลังใจ ในการจัดทำ สหกิจศึกษาฉบับนี้ให้สัมฤทธิ์ผลด้วยดีทุกประการ

ณัฐฑริกา ทองแถม

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญรูป

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เนื่องในฐานะผู้นำทางด้านเทคโนโลยีและบริการด้านการเงิน บริษัท SS&C Technologies ตั้งเป้าจะพัฒนาและส่งมอบบริการที่ดียิ่งขึ้นเพื่อตอบรับความต้องการของลูกค้า ซึ่ง SS&C Lyric นั้นเป็นแพลตฟอร์ม Front-to-Back ไร้รอยต่อที่ถูกสร้างขึ้นบนสถาปัตยกรรม Cloud-Native โดยเป็นระบบสำหรับการเก็บบันทึกและเปลี่ยนแปลงข้อมูลอย่างครบวงจรเพื่ออำนวยความสะดวกให้ Asset Managers, Financial Advisors, Broker-Dealers และ Retirement Providers โดย SS&C Lyric นั้นประกอบไปด้วยเทคโนโลยีหลายอย่าง ได้แก่ Microservice AI/Machine Learning และเทคโนโลยีการเก็บข้อมูลแบบกระจายซึ่งมีโครงสร้างมาจากระบบพื้นฐานของ SS&C's Global technology และสถาปัตยกรรมคอมพิวเตอร์แบบ Cloud-Native โดย SS&C Lyric มีการทำงานแบบเป็นวงจรอันประกอบไปด้วย Microservices หลายตัว ได้แก่ Calastone, Intelligence Service, Event Messaging Service และ HiTrust หากขาดส่วนประกอบใดไป การทำงานจะไม่สามารถดำเนินได้อย่างมีประสิทธิภาพ โดยกระบวนการทำงานจะเริ่มต้นขึ้นเมื่อ Intelligence Service ได้รับข้อมูล Order จาก Calastone จากนั้น Intelligence Service จะนำ Order ที่ได้ไปประมวลผลและส่งต่อข้อมูลไปยัง HiTrust เมื่อ HiTrust ได้รับข้อมูลจาก Intelligence Service ก็จะทำการประมวลผลและส่ง Kafka Message ไปยัง Kafka Topic ที่ Event Messaging Service เชื่อมต่ออยู่ และเมื่อครบกระบวนการทำงาน Intelligence Service ก็จะส่งข้อมูลเพื่ออัปเดตสถานะ Order ที่ Calastone ส่งมาตั้งแต่ขั้นแรก โดย Microservices ที่อยู่ในความรับผิดชอบของทีม Phantom อันเป็นทีมที่ผู้จัดทำได้ร่วมงานด้วยนั้นได้แก่ Intelligence Service และ Event Messaging Service ซึ่ง Microservices ทั้งสองที่กล่าวไปนั้นถูกพัฒนาอยู่บน US Environment Private Cloud ของบริษัท SS&C

ในกระบวนการการพัฒนา Software นั้นจำเป็นจะต้องมีการทดสอบตั้งแต่บนเครื่อง Local Computer ของ Developer ว่า Software ที่พัฒนาขึ้นมานั้นสามารถใช้งานได้ ก่อนจะนำไปทดสอบบน Platform ที่สูงขึ้น เช่น Development Platform ซึ่งอยู่บน Private Cloud ของบริษัท SS&C จนไปถึงการทดสอบบน UAT Platform (User Acceptance Platform) ที่ต้องให้ตัวแทนของ User มาทดลองใช้ โดยเมื่อนำ Intelligence Service มาทดลองใช้งานบน Development Platform ใน US Environment พบปัญหาว่า Intelligence Service ไม่สามารถเริ่มการทำงานบน Development Platform ได้ เนื่องจากไม่มีข้อมูล Order จาก Calastone เข้ามา ซึ่ง Calastone นั้นเป็นบริการการเงินภายนอกที่

ไม่ได้อยู่ในความดูแลของบริษัท SS&C และการจะนำ Calastone เข้ามาทดลองใช้งานจริงมีค่าใช้จ่ายจำนวนมาก ในขณะเดียวกัน HiTrust ซึ่งเป็นอีกส่วนประกอบสำคัญนั้นไม่สามารถใช้งานได้บน US Environment

โดยสองปัญหาที่กล่าวไว้ ณ ข้างต้น ทำให้ไม่สามารถทดสอบการทำงานของ Intelligence Service และ Event Messaging Service แบบครบวงจรได้ ทีม Phantom จึงริเริ่มโปรเจค Mock Calastone Service อันเป็นการจำลองพฤติกรรมการทำงานของ Calastone และ HiTrust โดยมีผู้จัดทำเป็นผู้รับผิดชอบ

1.2 วัตถุประสงค์ของงานวิจัย

1. เพื่อพัฒนาระบบไมโครเซอร์วิสจำลองพฤติกรรมการทำงานของ Calastone และ HiTrust
2. เพื่อเรียนรู้วิธีการทำงานของสถาปัตยกรรมคอมพิวเตอร์ร่วมสมัย เช่น Microservice, CQRS, Event Sourcing และ Cloud-native โดยเป็นเทคโนโลยีที่ใช้งานร่วมกันเพื่อสร้างเป็นระบบสากลที่รองรับการเปลี่ยนแปลงอันรวดเร็วของธุรกิจทางการเงิน
3. เพื่อเรียนรู้วิธีการทำงานของ Simple Rule-based AI ซึ่งมีส่วนสำคัญในระบบยุคปัจจุบัน

1.3 ขอบเขตของงานวิจัย

1. การทำงานใน Agile/Scrum Team ในระยะ Sprint 2 สัปดาห์ระหว่างการสหกิจ ตามกระบวนการพัฒนาซอฟต์แวร์พื้นฐานของ SS&C ได้แก่
   1.1. ทำงานที่ได้รับมอบหมายและติดตามงานผ่าน JIRA
   1.2. Code Version Control, Code Branching Model, และ Code Review ผ่าน Github
   1.3. ตรวจสอบคุณภาพของ Code ด้วย SONAR และ CodeQL
   1.4. การพัฒนาและ Deploy ระบบโดยใช้ Github Workflow และ Kubernetes บน Private Cloud ของ SS&C
2. การทำงานในฐานะ Java/Angular Developer
   2.1. เพื่อจำลองการทำงานของ Calastone และ HiTrust เพื่อใช้งานร่วมกับ Intelligence Service และ Event Messaging Service
   2.2. เพื่อเพิ่มประสิทธิภาพแก่ Intelligence Service หรือ Rule Engine
   2.3. เพื่อพัฒนา Microservices ของระบบ Lyric อย่างน้อยหนึ่งระบบหรือมากกว่า
   2.4. เพื่อช่วยเหลือทางด้านการศึกษาและวิจัยนวัตกรรม

2.5. รับผิดชอบการทำ Testing Automation สำหรับ Code ที่เขียน

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. ประโยชน์ต่อตนเอง
   1.1. ได้รับประสบการณ์และฝึกฝนทักษะการทำงานร่วมกันเป็นทีม
   1.2. ได้รับความรู้และประสบการณ์ในการทำงานจริง
   1.3. ได้เรียนรู้และใช้งานเครื่องมือ เทคโนโลยีต่าง ๆ ที่ทันสมัยและถูกใช้การทำงานจริง
   1.4. ได้เรียนรู้การทำงานแบบ Agile/Scrum Team
   1.5. ได้ฝึกทักษะการสื่อสารกับผู้อื่น
   1.6. ได้รับความรู้และประสบการณ์ในการเขียนโปรแกรม
   1.7. สามารถวิเคราะห์และแก้ไขปัญหาได้อย่างเป็นระบบ
2. ประโยชน์ต่อองค์กร
   2.1. ได้เพิ่มประสิทธิภาพของระบบซึ่งเป็นผลิตภัณฑ์ขององค์กร
   2.2. ได้พัฒนา Microservice ของระบบซึ่งเป็นผลิตภัณฑ์ขององค์กร

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

เพื่อการดำเนินการที่มีประสิทธิภาพและความเข้าใจของผู้อ่าน ในบทนี้จะกล่าวถึงเทคโนโลยีและทฤษฎีที่เกี่ยวข้อง โดยเทคโนโลยีที่ใช้ในการพัฒนาระบบนี้ได้แก่ Spring Boot, Kafka, Postman, Junit, H2, Git และ IntelliJ IDEA และในส่วนสุดท้ายของบทจะกล่าวถึงทฤษฎีที่เกี่ยวข้องในการพัฒนาระบบอันประกอบไปด้วย Microservice, RESTful API, Multithreaded, Unit Testing และ CI / CD

2.1 เทคโนโลยีที่เกี่ยวข้อง

2.1.1. Spring Boot

รูปที่ 2.1 สัญลักษณ์ของ Spring Boot

ที่มา : https://spring.io/

Spring เป็น Framework ที่อำนวยความสะดวกให้ Developer สามารถพัฒนาและสร้างสรรค์ Web Application หรือ Web Service โดยภาษา Java ได้ง่าย เนื่องด้วยมี Java Web Server ติดตั้งมากับ Spring Boot ที่ Port Default 8080 นอกจากนี้ Spring ยังเป็น Platform ที่รวบรวมอีกหลากหลาย Framework อันช่วยอำนวยความสะดวกให้ผู้ใช้งานเอาไว้ทำให้ Spring นั้นรองรับการพัฒนางานได้หลายด้าน เช่น Web Framework, Testing Framework รวมไปถึงลดความยุ่งยากในการตั้งค่าการใช้งานและเพิ่มความยืดหยุ่นในการพัฒนาระบบของ Developer

2.1.2.Kafka

รูปที่ 2.2 สัญลักษณ์ของ Kafka

ที่มา : https://kafka.apache.org/

Apache Kafka หรือ Kafka คือ Open-Source Distributed Event-Streaming Platform ที่ถูกพัฒนาให้รองรับการนำเข้าและประมวลผลข้อมูลแบบเรียลไทม์ได้อย่างมีประสิทธิภาพ การทำงานของ Kafka จะเริ่มต้นจากผู้ส่ง (Producer) ส่งข้อมูลไปยังผู้รับ (Consumer) ผ่าน Messaging Queue นอกจากนี้ Kafka ยังเป็น Streaming Platform ที่สามารถจัดการกับข้อมูลได้อย่างรวดเร็ว และมีการเก็บข้อมูลที่คงทนต่อความผิดพลาดอีกด้วย

2.1.3. H2 Database

รูปที่ 2.3 สัญลักษณ์ของ H2 Database

ที่มา : https://www.h2database.com/

H2 Database เป็นซอฟต์แวร์ฐานข้อมูลเชิงสัมพันธ์ (Relational Database) ที่ Spring Boot ใช้เป็นค่าเริ่มต้นเพื่อรองรับการใช้งานภาษา SQL ในการติดต่อกับฐานข้อมูล รวมไปถึงรองรับการใช้งานร่วมกับ JDBC (Java Database Connectivity) ซึ่งเป็นไลบรารี่ใน Java ที่ใช้สำหรับติดต่อกับฐานข้อมูลเชิงสัมพันธ์ H2 นั้นรองรับการใช้งานทั้งแบบ Server และแบบ Embedded โดยทั้งสองแบบที่กล่าวมาข้างต้นนั้นจะมีการใช้งานแบบโหลดฐานข้อมูลทั้งหมดไว้ในหน่วยความจำ (In-Memory Database) ในขณะที่แอปพลิเคชันยังทำงาน H2 จะโหลดฐานข้อมูลเก็บไว้ในหน่วยความจำทั้งหมด โดย H2 จะหยุดทำงานและลบข้อมูลให้หายไปเมื่อปิดแอปพลิเคชัน

2.1.4.Postman

รูปที่ 2.4 สัญลักษณ์ของ Postman

ที่มา : https://www.postman.com/

Postman คือเครื่องมือสำหรับการพัฒนาและทดสอบ API โดย Postman สามารถจำลองการส่ง Request ไปยัง API ที่ต้องการเพื่อทดสอบว่า API ดังกล่าวทำงานได้อย่างถูกต้องหรือไม่ ผู้ใช้สามารถสร้างและกำหนดข้อมูลต่าง ๆ ของ Request อันประกอบไปด้วย Request Body, Data, Header ได้ตามต้องการ รวมถึงสามารถเก็บรบรวม Request ที่ใช้ในการทดสอบบันทึกไว้เป็น Collection เพื่อให้สามารถเรียกใช้งาน Request ที่บันทึกไว้ได้อีกด้วย

2.1.5. Junit

รูปที่ 2.5 สัญลักษณ์ของ Junit

ที่มา : https://junit.org/junit4/

Junit เป็นไลบรารีสำหรับการเขียน Unit Test ในภาษา Java ซึ่งช่วยอำนวยความสะดวกแก่ Developer ในการทำ Automated Test กับแอปพลิเคชันที่สร้างขึ้นมา รวมไปถึงช่วยในการทำ Regression Test เมื่อมีการปรับปรุงหรือเปลี่ยนแปลง code ในอนาคต

Junit นั้นมีโครงสร้างในการเขียนที่เข้าใจง่ายและไม่ซับซ้อน มีการแยก Test Case แต่ละ Case ด้วย Annotation @Test รวมไปถึงสามารถลดการทำงานในกรณีที่หลาย ๆ Test Case มีการทำงานที่ซ้ำซ้อนกันโดยการใช้ Annotation @Before หรือ @After กับ Code ที่ซ้ำซ้อนกันส่วนนั้น

2.1.6.IntelliJ IDEA

รูปที่ 2.6 สัญลักษณ์ของ IntelliJ IDEA

ที่มา : https://www.jetbrains.com/company/brand/

IntelliJ IDEA เป็น IDE (Integrated Development Environment) หรือสภาพแวดล้อมสำหรับการพัฒนา Software แบบเบ็ดเสร็จ โดย IntelliJ IDEA ถูกสร้างขึ้นโดย JetBrains เพื่อเป็นโปรแกรมประยุกต์ซอฟต์แวร์ที่ช่วยให้ Developer พัฒนา Software ได้อย่างมีประสิทธิภาพ รวมถึงช่วยอำนวยความสะดวกในการทำงานของ Developer ด้วยการผสมผสานความสามารถต่าง ๆ เข้าด้วยกัน เช่น การแก้ไข การสร้าง การคอมไพล์ การทดสอบ การจัด Package ของ Software การ Reformat Code ทำให้ Developer ไม่ต้องจัดการทุกอย่างด้วยตัวเองและสร้างสรรค์ระบบได้สะดวกมากยิ่งขึ้น

2.1.7. Git

รูปที่ 2.7 สัญลักษณ์ของ Git

ที่มา : https://github.com/gitextensions

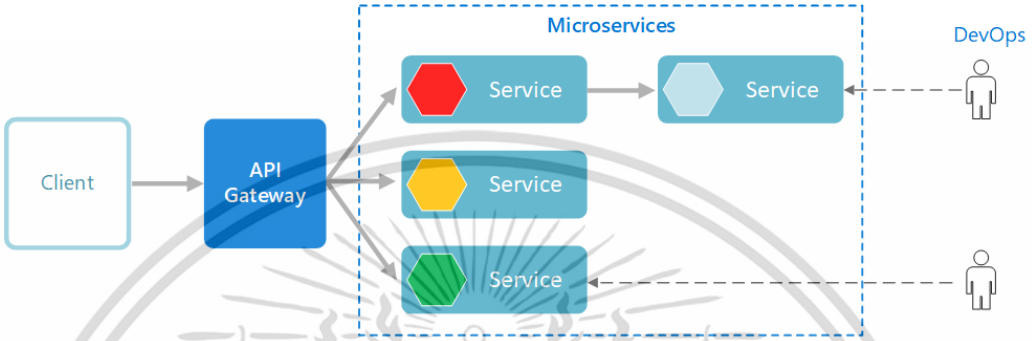
Git คือเครื่องมือที่ช่วยจัดการ Version ของ Code โดยจะเก็บประวัติว่าไฟล์แต่ละไฟล์นั้นถูกสร้าง ลบ หรือแก้ไขโดยใคร เมื่อใดและอย่างไรเอาไว้ ทำให้สามารถติดตามการเปลี่ยนแปลงของ Code ในแต่ละ Version ได้ รวมไปถึงกรณีที่ต้องการย้อนกลับไปใช้ Code ใน Version เก่า Git ก็สามารถทำได้เช่นกัน

การนำ Git มาช่วยในการจัดการ Version นั้นจะต้องทำการ Clone Repository ของงานที่ต้องการมายังเครื่องคอมพิวเตอร์ Local ก่อน จากนั้นจึงจะสามารถทำการสร้าง แก้ไข หรือลบไฟล์ต่าง ๆ

ในงานนั้น ๆ โดย Git จะเก็บบันทึกการเปลี่ยนแปลงของไฟล์เมื่อมีการใช้คำสั่ง Git Commit และหากต้องการจะอัปเดตการเปลี่ยนแปลงไปยัง Remote Repository ก็สามารถทำได้โดยการใช้คำสั่ง Git Push

2.2 ทฤษฎีที่เกี่ยวข้อง

2.2.1. Microservice



รูปที่ 2.8 แผนผังแสดงรูปแบบสถาปัตยกรรมของ Microservice

ที่มา : https://www.jetbrains.com/company/brand/

Microservice หรือ Microservice Architecture คือสถาปัตยกรรมการออกแบบและพัฒนา Software ให้อยู่ในรูปแบบที่มีขนาดเล็ก สามารถ Deploy รวมไปถึง Scale ได้อย่างอิสระและมีหน้าที่การทำงานที่ชัดเจน โดยแบ่งหน้าที่ต่าง ๆ ในระบบออกเป็น Service ย่อยที่ทำงานร่วมกัน ซึ่งแต่ละ Service นั้นมีอิสระต่อกัน ไม่จำเป็นต้องเชื่อมโยงกันทั้งหมด หาก Service ใดขัดข้องก็สามารถปรับปรุงแก้ไขเฉพาะส่วนนั้น ๆ ได้โดยไม่กระทบต่อการทำงานโดยรวม นอกจากนี้ Microservice สามารถพัฒนาโดยใช้คนละภาษากันได้ ทำให้สะดวกต่อการพัฒนาและบำรุงรักษาตามที่ Developer ถนัด

2.2.2.RESTful API

API หรือ Application Programming Interface เป็นส่วนต่อประสานโปรแกรมประยุกต์ โดย API นั้นจะเหมือนสัญญาที่กำหนดกฎเกณฑ์ที่ต้องปฏิบัติตามในการสื่อสารกับระบบซอฟต์แวร์อื่น ซึ่งสัญญานี้จะกำหนดวิธีที่ทั้งสองสื่อสารกันโดยใช้คำขอ (Request) และการตอบกลับ (Response) API นั้นถูกแบ่งออกเป็นสองฝั่งหลัก ๆ ด้วยกันได้แก่ฝั่ง Client และฝั่ง Server โดยฝั่งที่เป็นฝ่ายส่งคำขอนั้นจะเรียกว่า Client และฝั่งที่ส่งการตอบกลับจะเรียกว่าฝั่ง Server โดย API นั้นทำหน้าที่เสมือนเป็น Gateway เชื่อมระหว่าง Client และทรัพยากรต่าง ๆ จากฝั่ง Server โดย API นั้นมีการทำงานในหลากหลายรูปแบบแล้วแต่ว่าจะใช้สถาปัตยกรรม Software ใดมาออกแบบ โดยการเลือกสถาปัตยกรรมที่เหมาะสมนั้นขึ้นอยู่กับความเหมาะสม เวลา และสาเหตุที่สร้าง API ตัวนั้นขึ้นมา

REST หรือ Representational State Transfer เป็นหนึ่งในสถาปัตยกรรม Software ที่กำหนดเงื่อนไขว่า API ควรทำงานอย่างไร โดย REST นั้นจะใช้ประโยชน์จากเทคโนโลยี Web Protocol เพื่อสร้าง Web Service

RESTful API คือ API ที่ใช้รูปแบบสถาปัตยกรรมแบบ REST โดย RESTful นั้นอนุญาตให้ Client มีการส่ง Request เพื่อเข้าถึงทรัพยากรของฝั่ง Server ผ่านชุดคำสั่งที่กำหนดไว้ล่วงหน้า โดยการโต้ตอบพื้นฐานของระบบที่ใช้ REST นั้นจะอยู่ในรูปแบบของ Hypertext Transfer Protocol (HTTP) โดยฝั่ง Client จะทำการสร้าง Request และส่งคำขอไปยัง URI (Uniform Resource Identifier หรือข้อมูลที่ใช้ในการระบุตัวตนของทรัพยากรต่าง ๆ) ที่กำหนดและรับ Response จากฝั่ง Server กลับมาเป็น Payload ในรูปแบบ HTML, JSON, XML หรือในรูปแบบอื่น ๆ

HTTP Protocol ที่ถูกใช้ใน RESTful API นั้นเป็น Protocol ที่สะดวกและเป็นที่นิยม โดย HTTP Method ที่สำคัญใน RESTful API นั้นมีอยู่ 4 Method ด้วยกัน ได้แก่

1. **GET** ใช้ในการร้องขอข้อมูลจากฝั่ง Server
2. **POST** ใช้ในการสร้างข้อมูลในฝั่ง Server
3. **PUT** ใช้ในการอัปเดตข้อมูลเดิมที่มีอยู่แล้วหรือสร้างข้อมูลใหม่ในฝั่ง Server
4. **DELETE** ใช้ในการลบข้อมูลในฝั่ง Server

และ HTTP Response Status Code ที่คอยระบุสถานะของ Response จาก Request ที่ส่งไปนั้น ๆ ว่ามีสถานะและรายละเอียดเป็นอย่างไร โดยจะแบ่งออกเป็น 5 หมวดหมู่ด้วยกัน ได้แก่

1. **1XX** (informational)

   Response ที่อยู่ในหมวดหมู่ 1XX นั้นแสดงว่า Request ที่ Client ส่งไปยังฝั่ง Server นั้นถูกรับไปประมวลผลแล้วและสามารถทำงานไปยัง Process ถัดไป ตัวอย่างที่สำคัญของหมวดหมู่นี้ได้แก่

   - 100 : Continue - Server ได้รับ Request แล้วและ Client สามารถส่ง Request ต่อไปได้
   - 102 : Checkpoint – Server กำลังประมวลผล

2. **2XX** (Successful)

   Response ที่อยู่ในหมวดหมู่ 2XX นั้นแสดงว่า Request ที่ Client ส่งไปยังฝั่ง Server นั้นถูกประมวลผลเรียบร้อยและไม่มีข้อผิดพลาดใด ๆ ตัวอย่างที่สำคัญของหมวดหมู่นี้ได้แก่

   - 200 : OK - การส่ง Request ไปยังฝั่ง Server สำเร็จ
   - 201 : Created – Create ข้อมูลลง Database สำเร็จ

- 202 : Accepted - ฝั่ง Server ได้รับ Request แล้วแต่ยังประมวลผลไม่เสร็จ

3. **3XX** (Redirection)

   Response ที่อยู่ในหมวดหมู่ 3XX นั้นแสดงว่า Request ที่ Client ส่งไปยังฝั่ง Server นั้นถูก Redirect ไปประมวลผลที่อื่นเพื่อทำให้ Process สำเร็จ ตัวอย่างที่สำคัญของหมวดหมู่นี้ได้แก่

   - 301 : Moved Permanently – Request นี้ย้ายไป URL อื่นถาวร
   - 302 : Found – Request นี้ย้ายไป URL อื่นชั่วคราว
   - 304 : Not Modified – Request เรียกนี้ยังมีเนื้อหายังไม่ได้แก้ไขตั้งแต่การเรียกครั้งล่าสุด 4xx (Client Error)

4. **4XX** (Client Error)

   Response ที่อยู่ในหมวดหมู่ 4XX นั้นจะเกิดขึ้นเมื่อ Request ที่ Client ส่งไปยังฝั่ง Server นั้นมีข้อผิดพลาด ไม่ว่าจะเป็น Body Request ผิด, URL ผิด หรือ Syntax ผิด โดยตัวอย่างที่สำคัญของหมวดหมู่นี้ได้แก่

   - 400 : Bad Request - Body Request หรือ Syntax ที่ส่งมานั้นไม่ถูกต้อง
   - 401 : Unauthorized – Client ยังไม่ได้ระบุตัวตนหรือ Request ไม่มี Header
   - 403 : Forbidden – Client ระบุตัวตนแล้วแต่ไม่มีสิทธิ์ในการเข้าถึงเนื้อหาส่วนนี้
   - 404 : Not Found – ไม่พบข้อมูลในฝั่ง Server หรือไม่พบ URL ที่ Client ระบุมา
   - 405 : Method Not Allowed – Method ที่ใช้ใน Request ไม่ถูกต้อง
   - 422 : Unprocessable Entity – Request ที่ส่งเข้ามานั้นมีรูปแบบที่ถูกต้อง แต่ข้อมูลที่ส่งเข้ามาไม่ถูกต้องหรือไม่ครบตามความต้องการของ Server

5. **5XX** (Server Error)

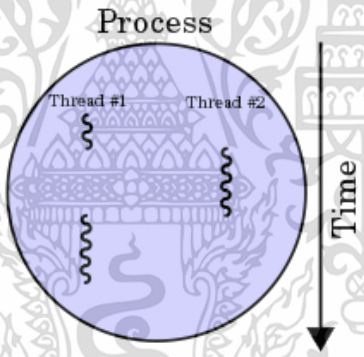
   Response ที่อยู่ในหมวดหมู่ 5XX นั้นแสดงว่า Server กำลังมีปัญหาบางอย่าง ตัวอย่างที่สำคัญของหมวดหมู่นี้ได้แก่

   - 500 : Internal Sever Error - มีข้อผิดพลาดบางอย่างภายใน server โดยไม่ทราบสาเหตุ
   - 502 : Bad Gateway - มีปัญหาในการรับส่งข้อมูลกันระหว่าง Server
   - 503 : Service Unavailable – Server มีการใช้งานเกินพิกัดหรือกำลังอยู่ระหว่างการปรับปรุง server

2.2.3. Multithreading

Thread คือหน่วยย่อยของ Process หรือในอีกความหมาย Thread คือการประมวลผลย่อยของคอมพิวเตอร์ โดยการทำงานของ Process นั้นแบ่งออกเป็น 2 ประเภทด้วยกัน ได้แก่การทำงานแบบ Singlethread หรือใน 1 Process จะมี Thread ทำงานอยู่ 1 Thread และการทำงานอีกแบบได้แก่ Multithread ซึ่งใน 1 Process จะมีการทำงานอยู่มากกว่า 1 Thread

Multithreaded จึงเป็นการประมวลผลหลายงานพร้อมกัน เช่น หน้าเว็บหนึ่งที่แสดงข้อมูลและดาวน์โหลดข้อมูลไปพร้อม ๆ กัน Thread ที่ทำหน้าที่ประมวลผลการแสดงข้อมูลและดาวน์โหลดข้อมูลก็จะถูกแยกออกเป็นคนละ Thread แต่ทั้ง 2 Thread นั้นทำงานไปพร้อม ๆ กันใน 1 Processer นอกจากนี้การทำงานแบบ Multithread ยังสามารถกำหนดเวลาก่อนหลังในการเริ่มทำงานของแต่ละ Thread ได้ด้วย เช่น หน้าเว็บหนึ่งที่ต้องการให้ทำการดาวน์โหลดข้อมูลก่อน 10 วินาทีจึงจะเริ่มทำการแสดงผลข้อมูลก็สามารถทำได้โดยการกำหนดระยะเวลา Delay ของ Thread ที่ประมวลผลการแสดงข้อมูล โดยระยะเวลาการ Delay จะเริ่มนับตั้งแต่ Processor นั้น ๆ เริ่มทำงาน



รูปที่ 2.9 แผนภาพจำลองการทำงานแบบ Multithread

ที่มา : https://en.wikipedia.org/wiki/Multithreading

2.2.4.Unit Testing

Unit Tests เป็นหนึ่งในระดับการทำ Software Testing โดย Unit Testing นั้นเป็นการทดสอบการทำงานของระบบในแต่ละหน่วย (Unit) ซึ่งคำว่าหน่วยหรือ Unit นั้นหมายถึงส่วนของการทำงานที่เล็กที่สุดเท่าที่จะทำการทดสอบได้ โดยในการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming) หน่วยที่ทดสอบอาจจะเป็น Method Function หรือ Module ก็ได้ ซึ่ง Unit Testing ในแต่ละครั้งนั้นส่วนใหญ่แล้วจะมี Developer เป็นผู้ดำเนินการ

จุดประสงค์ในการทำ Unit Testing นั้นก็เพื่อตรวจสอบว่ากระบวนทำงานของ Module หรือ Unit ดังกล่าวว่าเป็นไปตามที่ออกแบบไว้หรือไม่ โดย Unit Testing ที่ดีนั้นจะต้องมี Test Case ครอบคลุมการทำงานที่สำคัญในทุก Unit ต้องทำงานได้เร็วและน่าเชื่อถือ อีกข้อสำคัญคือ Test Case ทุกเคสจะต้องเป็นอิสระต่อกัน และใน 1 Test Case นั้นควรจะทดสอบการทำงานเพียง 1 อย่างต่อ 1 เทสเคสเท่านั้น

การทำ Unit Testing นั้นค้นหาและลดข้อผิดพลาดที่มีโอกาสก่อให้เกิดปัญหาลง รวมถึงทำให้การพัฒนาระบบทำได้รวดเร็วและสะดวกขึ้นในระยะยาวที่ต้องมีการทดสอบซ้ำ ๆ เพราะหากมีการปรับปรุงหรือแก้ไข Code หากมีการทำ Unit Testing ไว้ในระบบ ข้อผิดพลาดที่เกิดจากการปรับปรุงหรือแก้ไข้ Code ก็จะถูกดักจับได้ทันที

2.2.5. CI / CD

CI / CD คือกระบวนการที่จะมาช่วยในการพัฒนา Software ให้มีประสิทธิภาพมากยิ่งขึ้นทั้งในแง่ของระยะเวลาการพัฒนาและคุณภาพของ Software

CI หรือ Continuous Integration คือกระบวนการที่ช่วยดูแลจัดการ Source Code ผ่านกระบวนการต่าง ๆ อย่าง Testing และ Building เพื่อให้แน่ใจว่า Source Code ดังกล่าวสามารถใช้งานได้จริง ไม่มีข้อผิดพลาด โดย Testing นั้นจะเป็นการทำ Automated Testing และ Building นั้นจะเป็นการทดสอบว่า Source Code ดังกล่าวสามารถ Build ขึ้นมาและใช้งานจริงได้

CD คือกระบวนการที่ช่วยให้ Developer สามารถ Deploy Software ของตนได้อย่างมีประสิทธิภาพ โดยจะนำ Source Code ที่ผ่านกระบวนการ CI มาแล้วมาจัดการ Deploy ขึ้นไปอยู่บน Server ตามที่กำหนดและใช้งานได้อย่างถูกต้อง โดย Source Code ที่ผ่านกระบวนการ CI มาแล้วนั้นมีได้ในหลายรูปแบบด้วยกัน เช่น JAR File, Static File หรือ Container Image โดย CD จะช่วยอำนวยความสะดวกในการนำ Source Code เหล่านั้นขึ้นไป Deploy บน Server ให้โดยที่ Developer ไม่ต้องจัดการเอง กระบวนการ CD มีด้วยกัน 2 ประเภท ได้แก่

1. Continuous Delivery คือการส่งมอบ Software ที่ใช้วิธีการ Manual ในการ Deploy หลังจากผ่านขั้นตอน CI มาเรียบร้อยแล้ว โดยจะต้องมีการอนุมัติจากผู้รับรอง ซึ่งในที่นี่อาจหมายถึง Manager ทั้งนี้เพื่อทำการตรวจสอบก่อน Deploy ขึ้นสู่ Server โดย Continuous Delivery มักจะใช้กับการส่งมอบ Software ขึ้นบน High-Level Platform หรือ Platform ที่มีความสำคัญ ต้องการความถูกต้องและความละเอียดแม่นยำสูง เช่น UAT (User Acceptance) Platform หรือ Production ที่เป็นการส่งมอบ Software ขึ้นใช้งานจริง

2. Continuous Deployment คือการส่งมอบ Software แบบอัตโนมัติ โดยหลังจากผ่านขั้นตอน CI มาเรียบร้อยแล้วก็จะทำการ Deploy ขึ้นสู่ Server ทันที ไม่ต้องรอการอนุมัติหรือต้องผ่านการตรวจสอบแบบ Manual มักใช้กับการส่งมอบ Software ขึ้นบน Low-Level Platform หรือ Platform ที่ไม่ได้มีความอ่อนไหวมากนัก เช่น Development Platform ที่เป็น Platform การทำงานของ Developer ที่อยู่บน Server

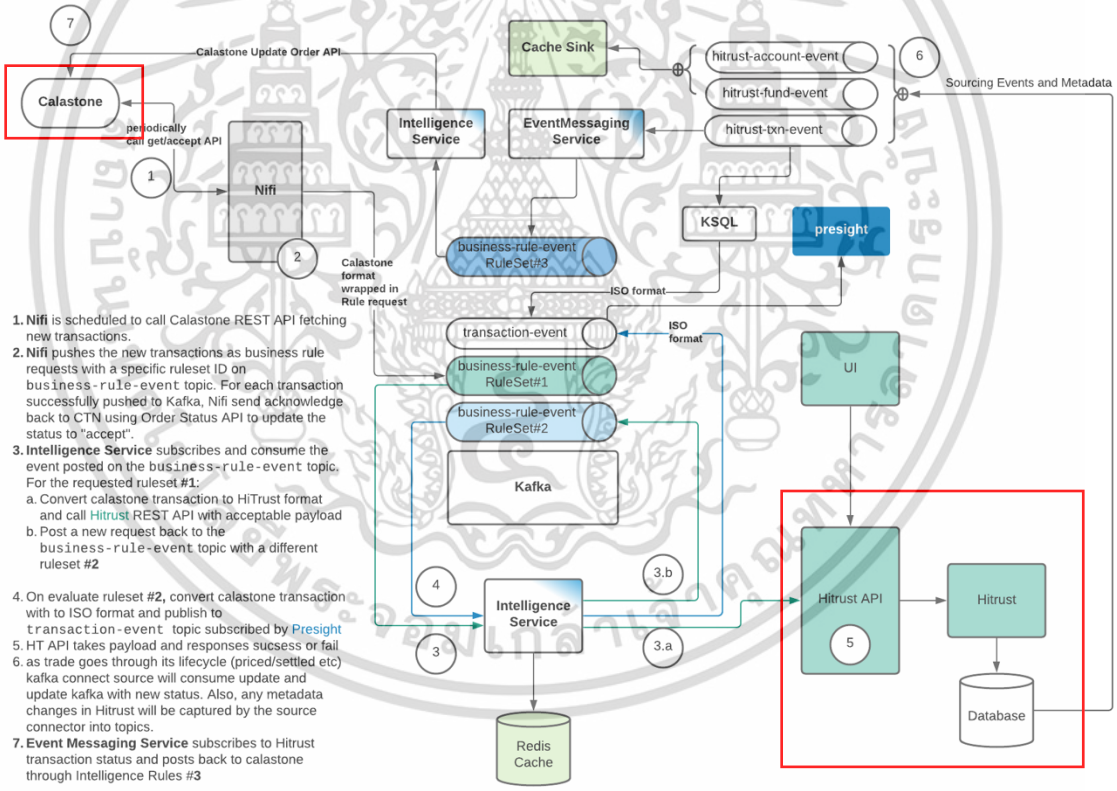
บทที่ 3
วิธีการดำเนินงานวิจัย

ในบทนี้จะกล่าวถึง Microservice จำลองพฤติกรรมการทำงานของ Calastone และ HiTrust โดยมีกระบวนการดำเนินงานเป็นไปตามขั้นตอนดังนี้

3.1 Requirement

สร้าง Microservice ที่จำลองการทำงานของ Calastone APIs และ Hitrust Integration ที่ใช้ในโปรเจค เพื่อนำไปทดลองใช้แทนที่ API จริงในการทำ full-loop testing

3.1.1.System Architecture



รูปที่ 3.1 แผนภาพแสดง System Architecture ของการทำงาน

และ HiTrust อันเป็นระบบภายนอก โดยการทำงานจะเริ่มต้นเมื่อ Calastone Service ทำการส่ง Order มายัง NiFi และ NiFi รับ Order เหล่านั้นเพื่อเริ่มการทำงาน ณ จุดนี้ Order ถูกสร้างขึ้นมาใหม่นั้นจะมี

Order Status เป็น “Submitted” จากนั้นจะทำการ execute Rule ที่กำหนดผ่าน Rule Engine ใน Intelligence Service โดยกระบวนการนี้ถูกกำหนดไว้ให้ทำงานทุก ๆ 1 นาทีและส่ง Request กลับไปยัง Calastone เพื่ออัปเดต Order Status เป็น “Accepted” เมื่อเสร็จสิ้นกระบวนการ Order จะถูกส่งต่อไปยัง Hitrust และ Hitrust จะทำการส่ง Response ของ Order นั้นกลับมาและส่งข้อมูลราคาไปยัง Kafka Topic ที่ถูกกำหนดเอาไว้เพื่อให้ Event Messaging Service สามารถรับและส่งข้อมูลกลับไปยัง Intelligence Service ได้ หลังจากได้รับ Response และ Event จาก Event Messaging Service แล้ว Intelligence Service จะทำการเรียกใช้ Confirm API ของ Calastone Service เพื่ออัปเดต Order Status เป็น Completed และเสร็จสิ้นกระบวนการทำงานใน 1 รอบ

โดยทั้ง Calastone Service และ HiTrust นั้นเป็นระบบที่ไม่ได้อยู่ในการควบคุม ดังนั้นการทำงานทั้งหมดที่จำเป็นจะถูกจำลองขึ้นมาในระบบ Mock ที่สร้างขึ้น โดยมีการออกแบบต่าง ๆ ดังที่จะกล่าวต่อไปในภายในบทที่ 3 นี้

3.1.2.Functional Requirement

POST Create Orders

Trigger การสร้าง Order จำลองใน In-memory Database เพื่อเริ่มกระบวนการ Testing ใน 1 Request ต้องสามารถสร้างได้หลาย Order และใน 1 Order ต้องสามารถรองรับได้หลาย Transaction

GET Get Updated Orders

ส่งข้อมูลรายการของ Order ในช่วงเวลา 5 วันล่าสุด หากใน Request มีวันและเวลาส่งไปด้วย ให้ส่งข้อมูลรายการ์ Order ภายในวันที่กำหนดจนถึง 5 วันถัดไป

GET Get Order Details

ส่งข้อมูล Order และ OrderDetails ของ OrderID ที่ต้องการ

POST Accept Orders

เปลี่ยนสถานะของ Order เป็น Accepted และอัปเดต Deal Reference ของ Order นั้น ๆ

POST Reject Orders

เปลี่ยนสถานะของ Order เป็น Rejected พร้อมอัปเดต Reject Reason

POST Confirm Orders

เปลี่ยนสถานะของ Order เป็น Confirmed และอัปเดต Details ที่ส่งมาใน Body Request

PUT Create Subscription Order

สร้าง Subscription Order จาก Body Request โดยใน request จะสามารถรับได้แค่ค่า net เท่านั้น และ Account ที่สร้าง Order ต้องไม่เป็น Blocked Account เมื่อ ทำงานสำเร็จ Mock CTN Service จะ publish events ไปยัง Kafka Topic ที่กำหนดโดยส่งในรูปแบบ JSON

PUT Create Redemption Order

สร้าง Redemption Order จาก Body Request โดยใน Request สามารถรับได้ทั้งค่า net และ units และ Account ที่สร้าง Order ต้องไม่เป็น Blocked Account เมื่อทำงานสำเร็จ Mock CTN Service จะ publish events ไปยัง Kafka Topic ที่กำหนดโดยส่งในรูปแบบ JSON

POST Authorize Transaction

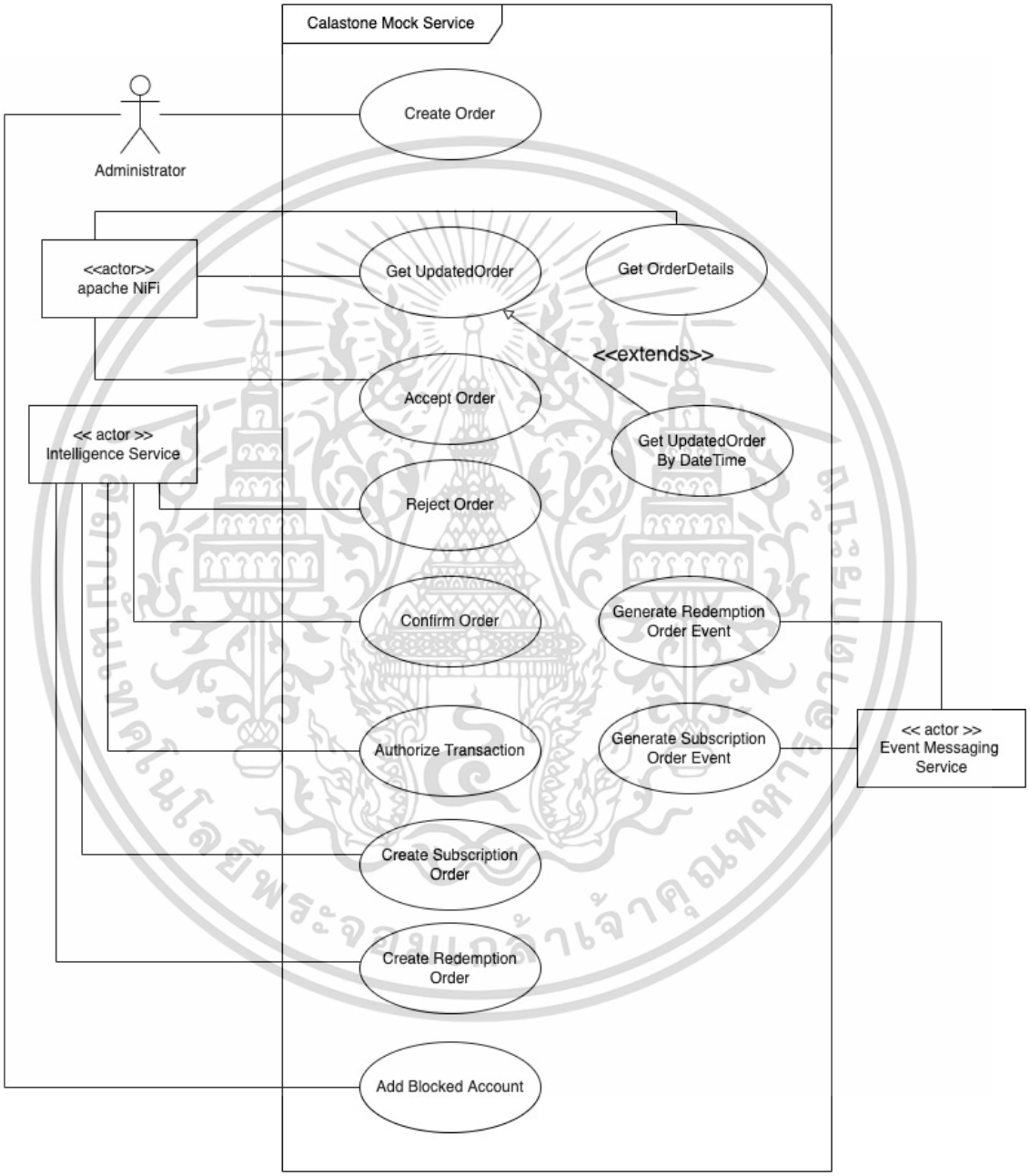
ยืนยันตัวตนของ Transaction หลังจาก Order ถูกส่งมายัง HiTrust ไม่ว่าจะเป็น Subscription Order หรือ Redemption Order โดย Response ไม่มี Body Response และต้องมี Http Status เป็น 200 OK

POST Create Blacklist

บันทึกรายการ Blocked Account ลงฐานข้อมูล

3.2 Use Case Diagram

Mock Calastone Service เป็นระบบที่จำลองการทำงานของระบบการเงินภายนอกที่ใช้งานจริงในโปรเจค โดยจะจัดทำในรูปแบบของ Microservice ซึ่งมีการออกแบบและแนวทางดังนี้



รูปที่ 3.2 Use Case Diagram ของ Calastone Mock Service

Mock Calastone Service จะเริ่มการทำงานเมื่อมีการส่ง POST Request Create Order เข้ามาในระบบ โดยจะเริ่มสร้าง Order และบันทึกข้อมูลลงใน In-memory Database จากนั้นจะเริ่มทำงานต่อเมื่อ apache NiFi ส่ง Request Get Updated Order หรือ Get OrderDetails เข้ามา Mock Calastone Service จะทำการส่ง Order กลับไปใน response และเมื่อ apache NiFi ส่ง Request Accept Order เข้ามา Mock Calastone Service จะทำการอัปเดต OrderStatus เป็น Accepted เพื่อรอให้ Intelligece Service มาเรียกใช้งาน Reject Order, Confirm Order, Authorize Transaction, Create Redemption Order, Create Subscription Order ต่อไป หลังจากเสร็จสิ้นกระบวนการนั้น ๆ แล้ว ก็จะส่ง Response กลับไปยัง Intelligence Service โดยหาก API ที่ถูกเรียกใช้นั้นเป็น Create Subscription Order หรือ Create Redemption Order หลังจากส่ง Response ไปให้ Intelligence Service แล้ว ระบบจะเริ่มการทำงานของ Generate Redemption Order Event และ Generate Subscription Order Event เพื่อส่ง Kafka Event ไปยัง Event Messaging Service ด้วย

Use Case Description

ตารางที่ 3.1 Create Order

Name	Create Order
Description	สร้าง Order จาก Request ที่ส่งเข้ามาและทำการเก็บบันทึกข้อมูล Order ไว้ใน In-Memory Database
Input	Order, Transaction
Output	OrderResponse : orderID, createdDate, updatedDate และ orderStatus
Condition	ใน 1 Request สามารถมีได้หลาย Order และใน 1 Order สามารถมีได้หลาย Transaction

ตารางที่ 3.2 Get Order Details

Name	Get Order Details
Description	ส่งข้อมูล Order และ Order Details ของ Order ID ที่ต้องการ
Input	Order ID
Output	OrderDetailsResponse :orderID, createdDate, updatedDate, orderStatus, submitDetails, validationDetails, transactions

ตารางที่ 3.3 Get Updated Order

Name	Get Updated Order
Description	ส่งข้อมูลรายการของ Order ในช่วงเวลา 5 วันล่าสุด หากใน Request มีวันและเวลาส่งไปด้วย ให้ส่งข้อมูลรายการ์ Order ภายในวันที่กำหนดจนถึง 5 วันถัดไป
Input	providedDatetime
Output	OrderResponse : orderID, createdDate, updatedDate และ orderStatus
Condition	หากใน Request มี providedDatetime ให้ส่งข้อมูลรายการ Order ภายในวันที่กำหนดจนถึง 5 วันถัดไป

ตารางที่ 3.4 Accept Order

Name	Accept Order
Description	เปลี่ยนสถานะของ Order เป็น Accepted และอัปเดต Deal Reference ของ Order นั้น ๆ
Input	OrderID, DealReference
Output	AcceptResponse : orderID, createdDate, updatedDate, orderStatus, submitDetails, validationDetails, acceptDetails (with accepted as true and dealReference from request), transactions

ตารางที่ 3.5 Reject Order

Name	Reject Order
Description	เปลี่ยนสถานะของ Order เป็น Rejected พร้อมอัปเดต Reject Reason
Input	OrderID, RejectReason
Output	AcceptResponse : orderID, createdDate, updatedDate, orderStatus, submitDetails, validationDetails, acceptDetails (with accepted as false and rejectReason), transactions

ตารางที่ 3.6 Confirm Order

Name	Confirm Order
Description	เปลี่ยนสถานะของ Order เป็น Confirmed และอัปเดต Details ที่ส่งมาใน Body Request
Input	OrderID, TransactionDetails, FxDetails
Output	ConfirmResponse : orderID, createdDate, updatedDate, orderStatus, submitDetails, validationDetails, acceptDetails (with accepted as false and rejectReason), confirmDetails, transactions
Condition	FxDetails สามารถเป็นค่าว่างได้

ตารางที่ 3.7 Create Redemption Order

Name	Create Redemption Order
Description	สร้าง Redemption Order จาก Body Request เมื่อทำงานสำเร็จจะ publish events ไปยัง Kafka Topic ที่กำหนด
Input	Transaction Request
Output	Response : Data, SystemMessages Events : messageValue
Condition	Request สามารถรับได้ทั้งค่า net และ units Account ที่สร้าง Order ต้องไม่เป็น Blocked Account

ตารางที่ 3.8 Create Subscription Order

Name	Create Subscription Order
Description	สร้าง Subscription Order จาก Body Request เมื่อทำงานสำเร็จจะ publish events ไปยัง Kafka Topic ที่กำหนด
Input	Transaction Request
Output	Response : Data, SystemMessages Events : messageValue
Condition	Request สามารถรับได้ทั้งค่า net และ units Account ที่สร้าง Order ต้องไม่เป็น Blocked Account

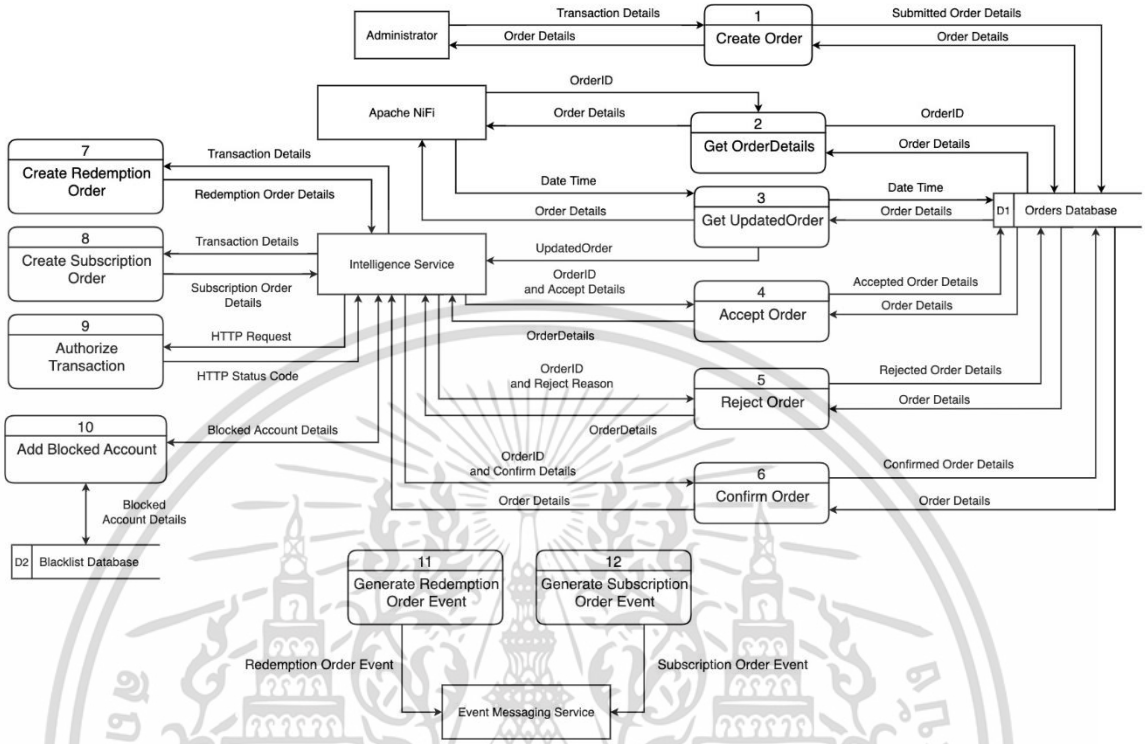
ตารางที่ 3.9 Authorize Transaction

Name	Authorize Transaction
Description	ยืนยันตัวตนของ Transaction หลังจาก Order ถูกส่งมายัง HiTrust
Input	HTTP POST Request
Output	HTTP Status 200 OK

ตารางที่ 3.10 Add Blocked Account

Name	Add Blocked Account
Description	บันทึกรายการ Blocked Account ลงฐานข้อมูล
Input	Blocked Account
Output	Added Account

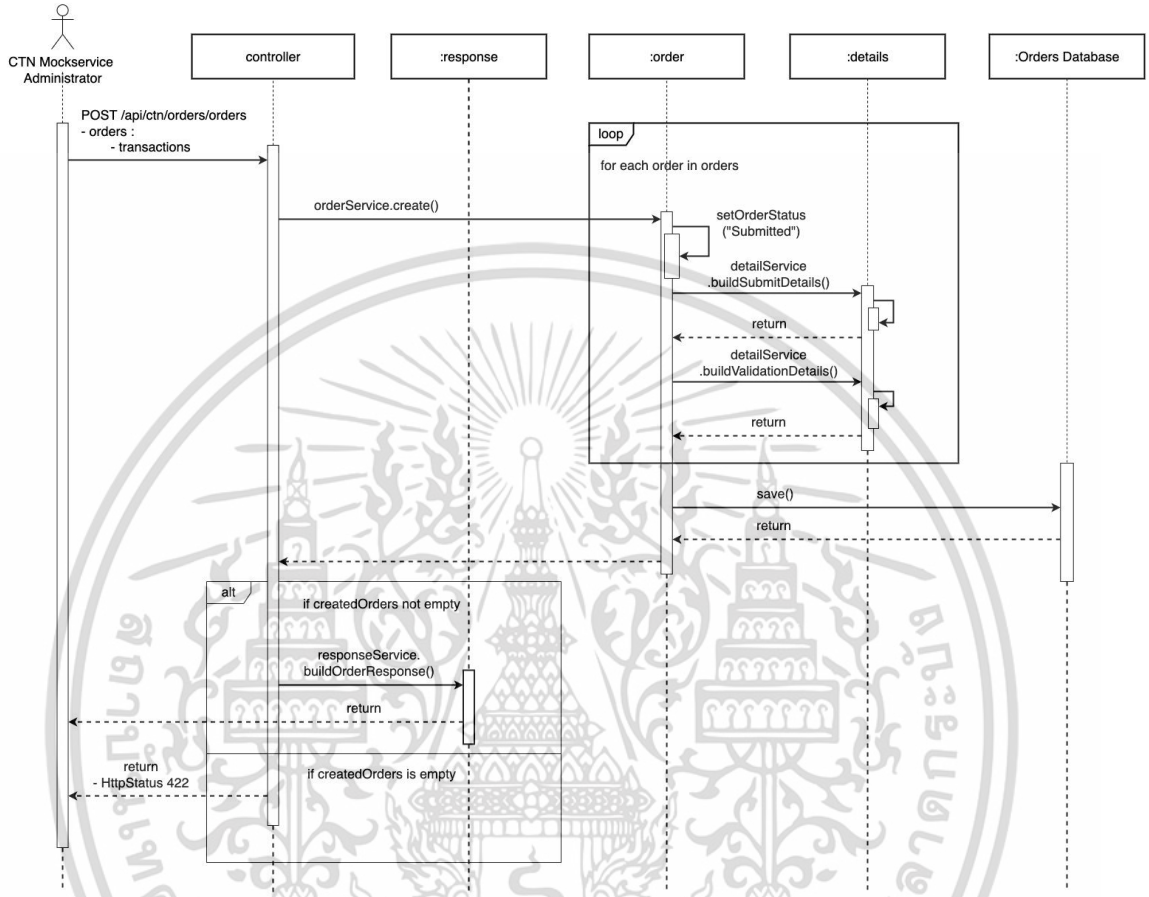
3.3 Data Flow Diagram



รูปที่ 3.3 Data Flow Diagram ของ Calastone Mock Service

3.4 Sequence Diagram

3.4.1.Create Order

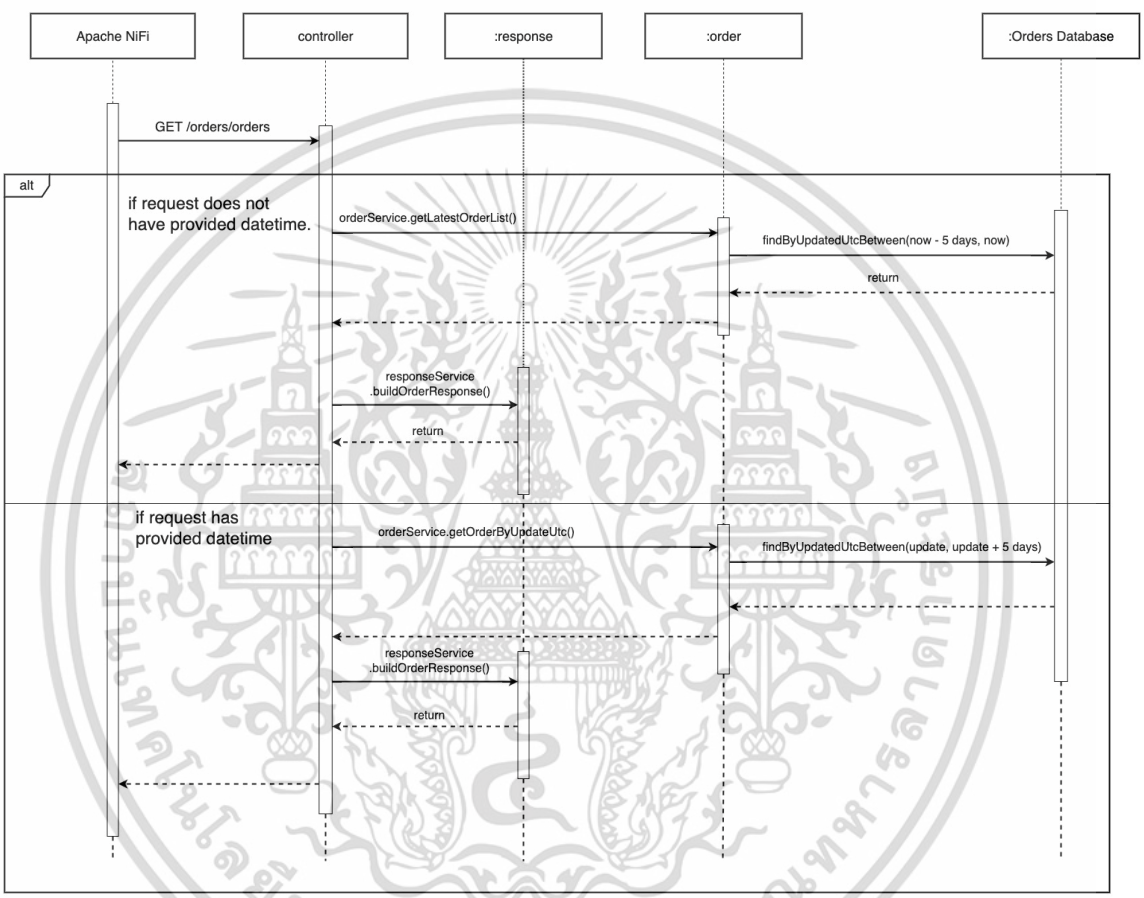


รูปที่ 3.4 Sequence Diagram ของการ Create Order

กระบวนการทำงานของ Create Order จะเริ่มต้นเมื่อ Administrator ทำการส่ง POST Request มายัง API ใน Controller จากนั้นจะทำการเรียกใช้ method create() จาก orderService พร้อมส่งรายการ orders ใน Body Request ไป เมื่อ OrderService ได้รับ orders ซึ่งเป็น List แล้ว ก็จะทำการ Loop เพื่ออัปเดตทุก order ใน List โดยกระบวนการอัปเดตจะเริ่มจากการ setOrderStatus ให้เป็น “Submitted” จากนั้นจะทำการเรียก method buildSubmitDetails และ buildValidationDetails จาก DetailService เพื่อทำการสร้าง SubmitDetails และ ValidationDetails มาอัปเดตใน order เมื่ออัปเดตทุก order ใน orders แล้ว OrderService ก็จะทำการเรียกใช้ method save(Orders) เพื่อบันทึก Orders ลง Database เมื่อ Orders Database บันทึกข้อมูลเรียบร้อยแล้วก็จะทำการ Return ค่าที่บันทึกแล้วกลับมายัง OrderService เมื่อ OrderService ได้รับค่าที่ return กลับมาแล้วก็จะ return กลับไปยัง Controller ต่อ จากนั้น Controller จะทำการตรวจสอบว่าค่าที่ส่งมานั้นไม่เป็นค่าว่าง จากนั้น

จะเรียกใช้ method buildOrderResponse() จาก responseService เพื่อทำการสร้าง Response ส่งกลับในกรณีที่การ Create Order สำเร็จ แต่หากค่าที่ OrderService ส่งกลับมานั้นเป็นค่าว่าง แสดงว่าการ Create Order นั้นไม่สำเร็จ Controller ก็จะทำการส่ง HTTP Status 422 กลับไปเป็น Response

3.4.2.Get Updated Order

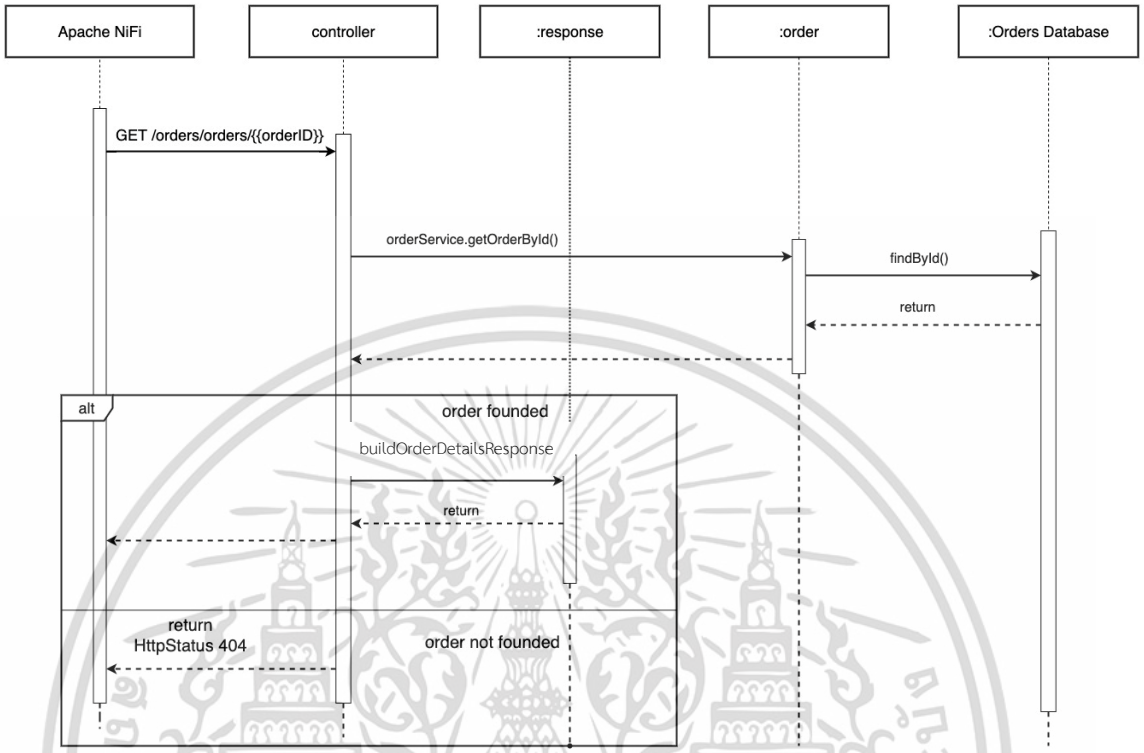


รูปที่ 3.5 Sequence Diagram ของการ Get Updated Order

กระบวนการทำงานของ Get Updated Order จะเริ่มต้นเมื่อ Apache NiFi ทำการส่ง POST Request มายัง API ใน Controller จากนั้น Controller จะทำการตรวจสอบว่า Request ที่ส่งมามีวันและเวลากำหนดมาด้วยหรือไม่ หากไม่มี Controllor ก็จะทำการเรียกใช้ method get-LatestOrderList() จาก OrderService และ OrderService จะทำการเรียกใช้ method findByUpdatedUtcBetween() เพื่อค้นหา Order ในช่วงวันเวลาที่ต้องการจาก Orders Database ในกรณีนี้ Intelligence Service ไม่ได้ส่งวันและเวลาที่ต้องการมา OrderService จะทำการกำหนดวันและเวลาที่ต้องการค้นหาเป็นช่วงเวลาตั้งวันเวลา ณ ตอนนั้นย้อนหลังกลับไป 5 วัน ในทางกลับกัน หากมี

การส่งวันและเวลาที่ต้องการมาใน Request Controller จะทำการเรียกใช้ method getOrderBy-UpdateUtc() จาก OrderService และ OrderService ก็จะทำการเรียกใช้ method findBy-UpdatedUtcBetween() เพื่อค้นหา Order ในช่วงวันเวลานับตั้งแต่ providedDate 5 วัน เมื่อค้นหาข้อมูลแล้วเสร็จ Orders Database ก็จะ return List ของ order ในช่วงวันเวลาที่ทำการค้นหากลับไปยัง OrderService และเมื่อ OrderService ได้รับค่าที่ return กลับมาแล้วก็จะ return กลับไปยัง Controller ต่อ จากนั้นจะเรียกใช้ method buildOrderResponse() เพื่อทำการสร้าง Response ของ Updated Orders กลับไปยัง Apache NiFi

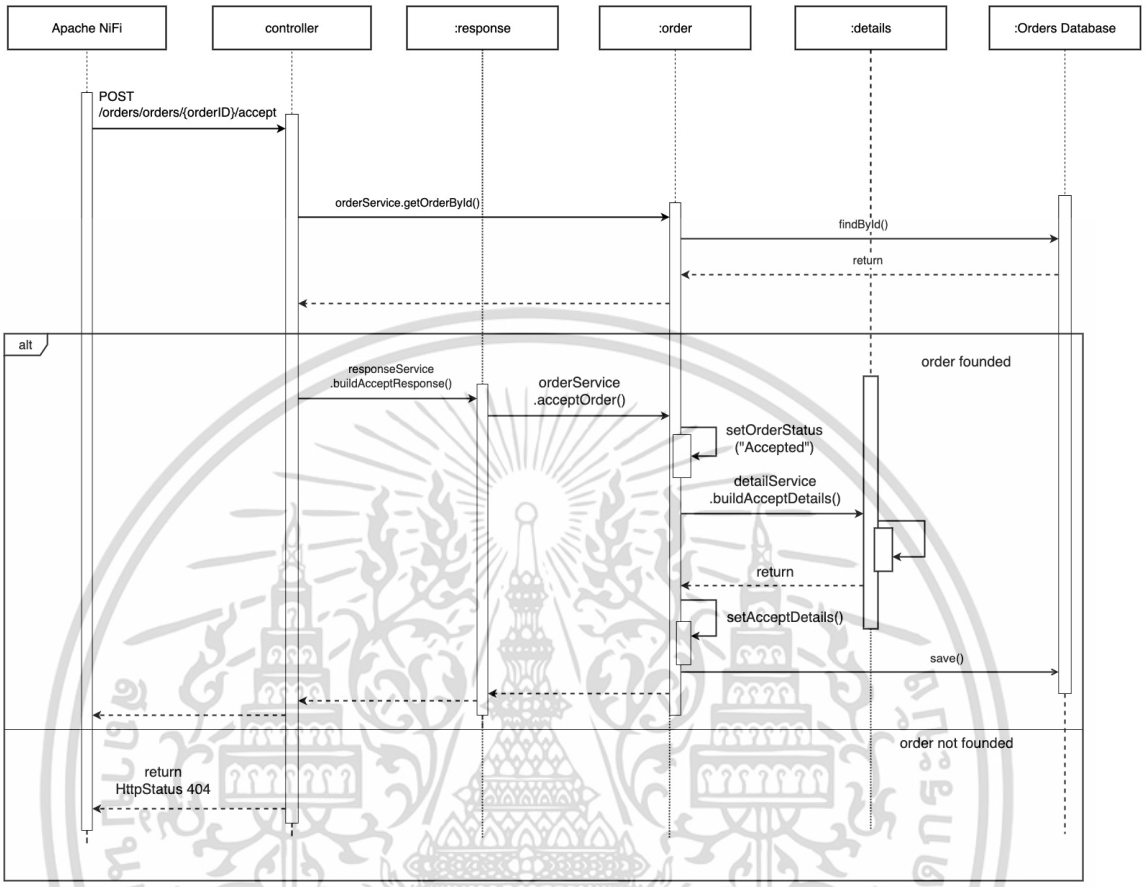
3.4.3.Get Order Details



รูปที่ 3.6 Sequence Diagram ของการ Get Order Details

กระบวนการทำงานของ Get Order Details จะเริ่มต้นเมื่อ Apache NiFi ทำการส่ง POST Request พร้อม Order ID มายัง API ใน Controller จากนั้น Controller จะทำการเรียกใช้ method getOrderById() จาก OrderService ซึ่ง OrderService ก็จะทำการเรียกใช้ method findById() เพื่อค้นหา Order ของ OrderID ที่ต้องการจาก Orders Database เมื่อค้นหาข้อมูลแล้วเสร็จ Orders Database ก็จะ return order ของ Order ID นั้นกลับมายัง OrderService จากนั้น Controller จะทำการตรวจสอบว่าค่าที่ส่งมานั้นไม่เป็นค่าว่าง หากค่าที่ส่งกลับมาเป็นค่าว่างแสดงว่าไม่มีข้อมูลของ Order ID นั้น ๆ อยู่ใน Database Controller จะทำการส่ง HTTP Status 404 Not Found กลับไปยัง Intelligence Service แต่หากค่าที่ OrderService ส่งกลับมานั้นไม่เป็นค่าว่าง แสดงว่ามีข้อมูลอยู่ใน Database Controller จะเรียกใช้ method buildOrderDetailsResponse() เพื่อทำการสร้าง Response ของ Get Order Details กลับไปยัง Apache NiFi

3.4.4.Accept Order

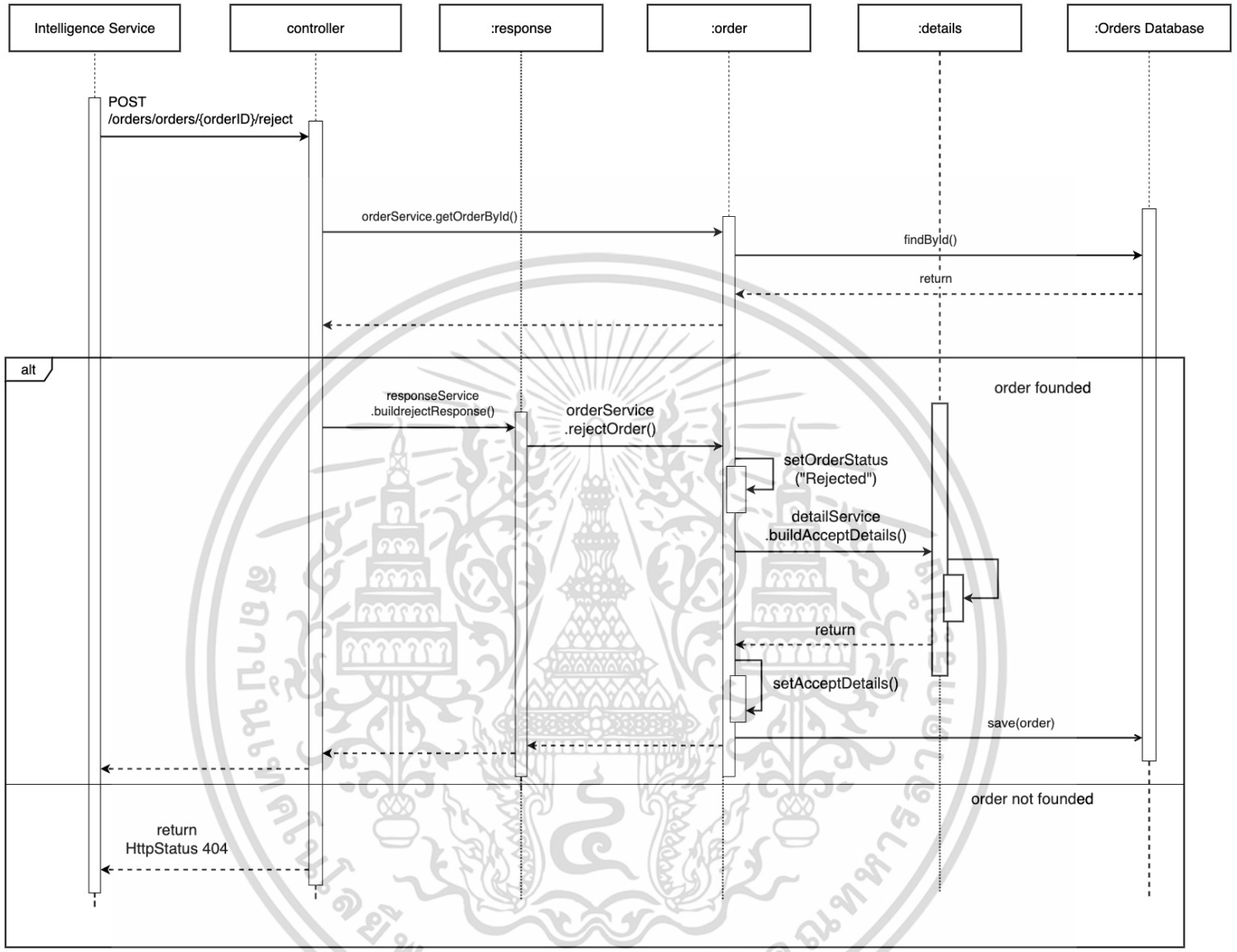


รูปที่ 3.7 Sequence Diagram ของการ Accept Order

กระบวนการทำงานของ Accept Order จะเริ่มต้นเมื่อ Apache NiFi ทำการส่ง POST Request พร้อม Order ID และ DealRefence มายัง API ใน Controller จากนั้น Controller จะทำการเรียกใช้ method getOrderById() จาก OrderService เพื่อค้นหา Order ของ OrderID ที่ต้องการ Accept Order ซึ่ง OrderService ก็จะทำการเรียกใช้ method findById() เพื่อค้นหา Order ที่ต้องการจาก Orders Database เมื่อค้นหาข้อมูลแล้วเสร็จ Orders Database ก็จะ return order ของ Order ID นั้น กลับมายัง OrderService และส่งต่อกลับไปยัง Controller จากนั้น Controller จะทำการตรวจสอบว่าค่าที่ส่งมานั้นไม่เป็นค่าว่าง หากค่าที่ส่งกลับมาเป็นค่าว่างแสดงว่าไม่มีข้อมูลของ Order ID นั้น ๆ อยู่ใน Database และไม่สามารถทำการ Accept Order ได้ Controller จะทำการส่ง HTTP Status 404 Not Found กลับไปยัง Intelligence Service แต่หากค่าที่ OrderService ส่งกลับมานั้นไม่เป็นค่าว่าง แสดงว่ามีข้อมูลอยู่ใน Database และสามารถ Accept Order ได้ Controller จะเรียกใช้ method buildAcceptResponse() เพื่อทำการสร้าง Response ประเภท Accepted จาก Accept Order

กลับไปยัง Intelligence Service โดย ResponseService ก็จะทำการเรียกใช้ method acceptOrder() จาก OrderService เพื่อเป็นการ Accept Order โดย Order Service จะอัปดเดต OrderStatus ของ Order นั้นเป็น “Accepted” แล้วเรียกใช้ buildAcceptDetails() จาก DetailService เพื่อสร้าง AcceptDetails ที่มีค่าของ DealReference จากใน Body Request บันทึกไว้ เมื่อได้รับ AcceptDetails จาก DetailService และทำการ setAcceptDetails() ที่ order แล้ว OrderService ก็จะส่ง order ที่ทำการ Accept เรียบร้อยกลับไปยัง ResponseService เพื่อสร้าง Response ของ Accept Order ให้เสร็จสิ้น เพื่อนำส่งกลับให้ Apache NiFi ต่อไป

3.4.5.Reject Order

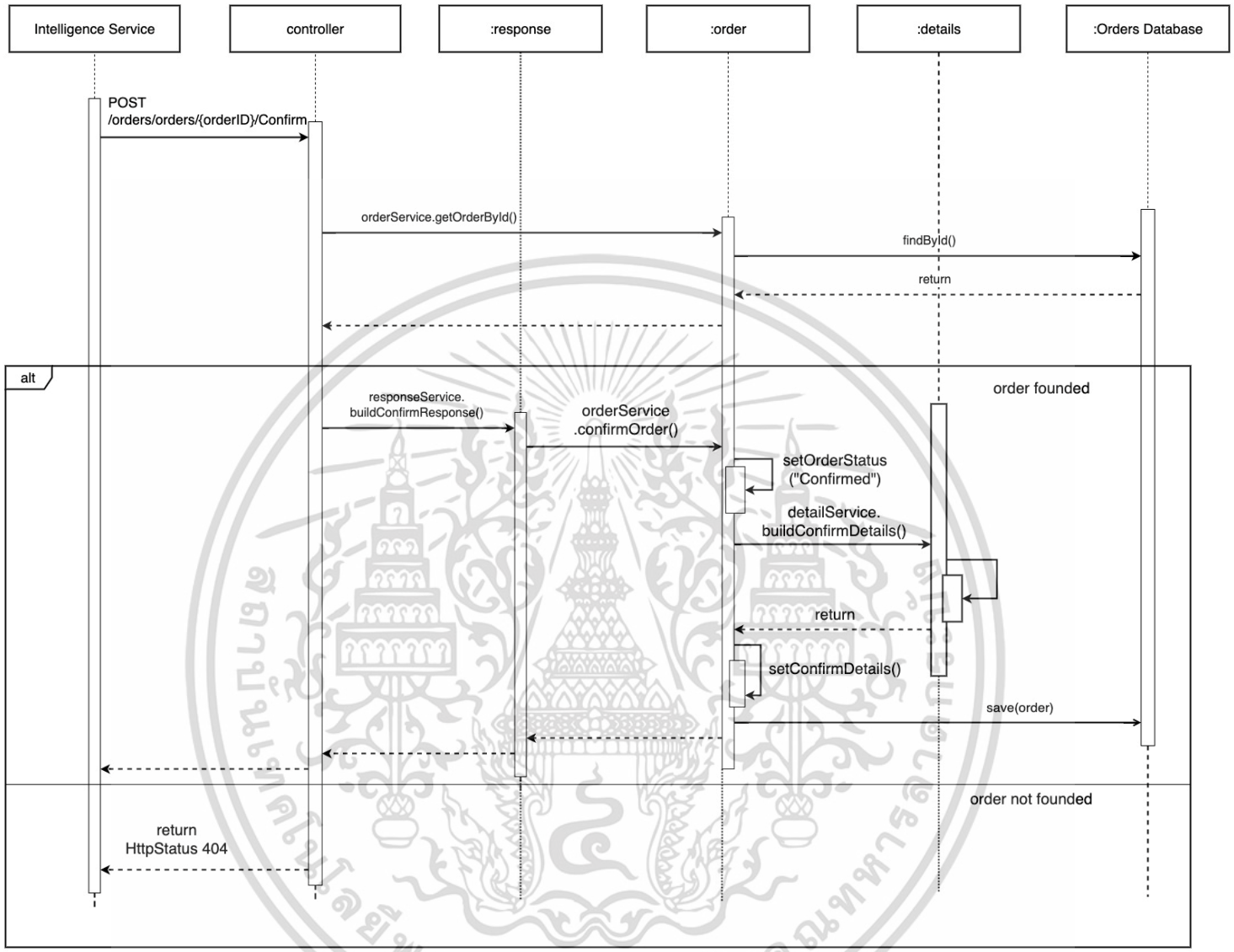


รูปที่ 3.8 Sequence Diagram ของการ Reject Order

กระบวนการทำงานของ Reject Order จะเริ่มต้นเมื่อ Intelligence Service ทำการส่ง POST Request พร้อม Order ID และ RejectReason มายัง API ใน Controller จากนั้น Controller จะทำการเรียกใช้ method getOrderById() จาก OrderService เพื่อค้นหา Order ของ OrderID ที่ต้องการ Reject Order ซึ่ง OrderService ก็จะทำการเรียกใช้ method findById() เพื่อค้นหา Order ที่ต้องการจาก Orders Database เมื่อค้นหาข้อมูลแล้วเสร็จ Orders Database ก็จะ return order ของ Order ID นั้นกลับมายัง OrderService และส่งต่อกลับไปยัง Controller จากนั้น Controller จะทำการ

ตรวจสอบว่าค่าที่ส่งมานั้นไม่เป็นค่าว่าง หากค่าที่ส่งกลับมาเป็นค่าว่างแสดงว่าไม่มีข้อมูลของ Order ID นั้น ๆ อยู่ใน Database และไม่สามารถทำการ Reject Order ได้ Controller จะทำการส่ง HTTP Status 404 Not Found กลับไปยัง Intelligence Service แต่หากค่าที่ OrderService ส่งกลับมานั้นไม่เป็นค่าว่าง แสดงว่ามีข้อมูลอยู่ใน Database และสามารถ Reject Order ได้ Controller จะเรียกใช้ method buildAcceptResponse() เพื่อทำการสร้าง Response ประเภท Rejected จาก Reject Order กลับไปยัง Intelligence Service โดย ResponseService ก็จะทำการเรียกใช้ method rejectOrder() จาก OrderService เพื่อเป็นการ Reject Order โดย Order Service จะอัปเดต OrderStatus ของ Order นั้นเป็น “Rejected” แล้วเรียกใช้ buildAcceptDetails จาก DetailService เพื่อสร้าง AcceptDetails ที่มี rejectReason จากใน Body Request บันทึกไว้ เมื่อได้รับ AcceptDetails จาก DetailService และทำการ setAcceptDetails ที่ order แล้ว OrderService ก็จะส่ง order ที่ทำการ Accept เรียบร้อยกลับไปยัง ResponseService เพื่อสร้าง Response ของ Reject Order ให้เสร็จสิ้น เพื่อนำส่งกลับให้ Intelligence Service ต่อไป

3.4.6.Confirm Order

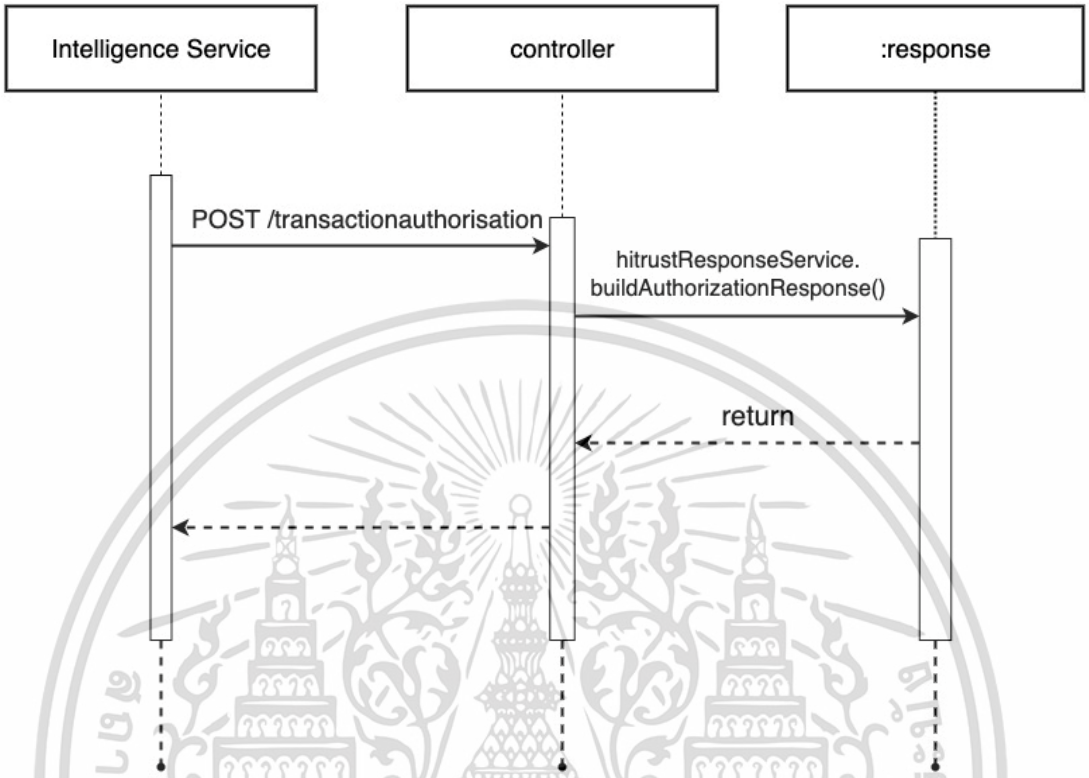


รูปที่ 3.9 Sequence Diagram ของการ Get Updated Order

กระบวนการทำงานของ Confirm Order จะเริ่มต้นเมื่อ Intelligence Service ทำการส่ง POST Request พร้อม Order ID, Transaction Details และ FxDetails มายัง API ใน Controller จากนั้น Controller จะทำการเรียกใช้ method getOrderById() จาก OrderService เพื่อค้นหา Order ของ OrderID ที่ต้องการ Confirm Order ซึ่ง OrderService ก็จะทำการเรียกใช้ method findById() เพื่อค้นหา Order ที่ต้องการจาก Orders Database เมื่อค้นหาข้อมูลแล้วเสร็จ Orders Database ก็จะ return order ของ Order ID นั้นกลับมายัง OrderService และส่งต่อกลับไปยัง Controller จากนั้น

Controller จะทำการตรวจสอบว่าค่าที่ส่งมานั้นไม่เป็นค่าว่าง หากค่าที่ส่งกลับมาเป็นค่าว่างแสดงว่าไม่มีข้อมูลของ Order ID นั้น ๆ อยู่ใน Database และไม่สามารถทำการ Confirm Order ได้ Controller จะทำการส่ง HTTP Status 404 Not Found กลับไปยัง Intelligence Service แต่หากค่าที่ OrderService ส่งกลับมานั้นไม่เป็นค่าว่าง แสดงว่ามีข้อมูลอยู่ใน Database และสามารถ Confirm Order ได้ Controller จะเรียกใช้ method buildConfirmResponse() เพื่อทำการสร้าง Response ของ Confirm Order กลับไปยัง Intelligence Service โดย ResponseService ก็จะทำการเรียกใช้ method confirmOrder() จาก OrderService เพื่อเป็นการ Confirm Order โดย Order Service จะอัปเดต OrderStatus ของ Order นั้นเป็น “Confirmed” แล้วเรียกใช้ buildConfirmDetails จาก DetailService เพื่อสร้าง ConfirmDetails ที่มี TransactionDetails และ FxDetails จากใน Body Request เมื่อได้รับ ConfirmedDetails จาก DetailService และทำการ setConfirmDetails ที่ order แล้ว OrderService ก็จะส่ง order ที่ทำการ Confirm เรียบร้อยกลับไปยัง ResponseService เพื่อสร้าง Response ของ Confirm Order ให้เสร็จสิ้น เพื่อนำส่งกลับให้ Intelligence Service ต่อไป

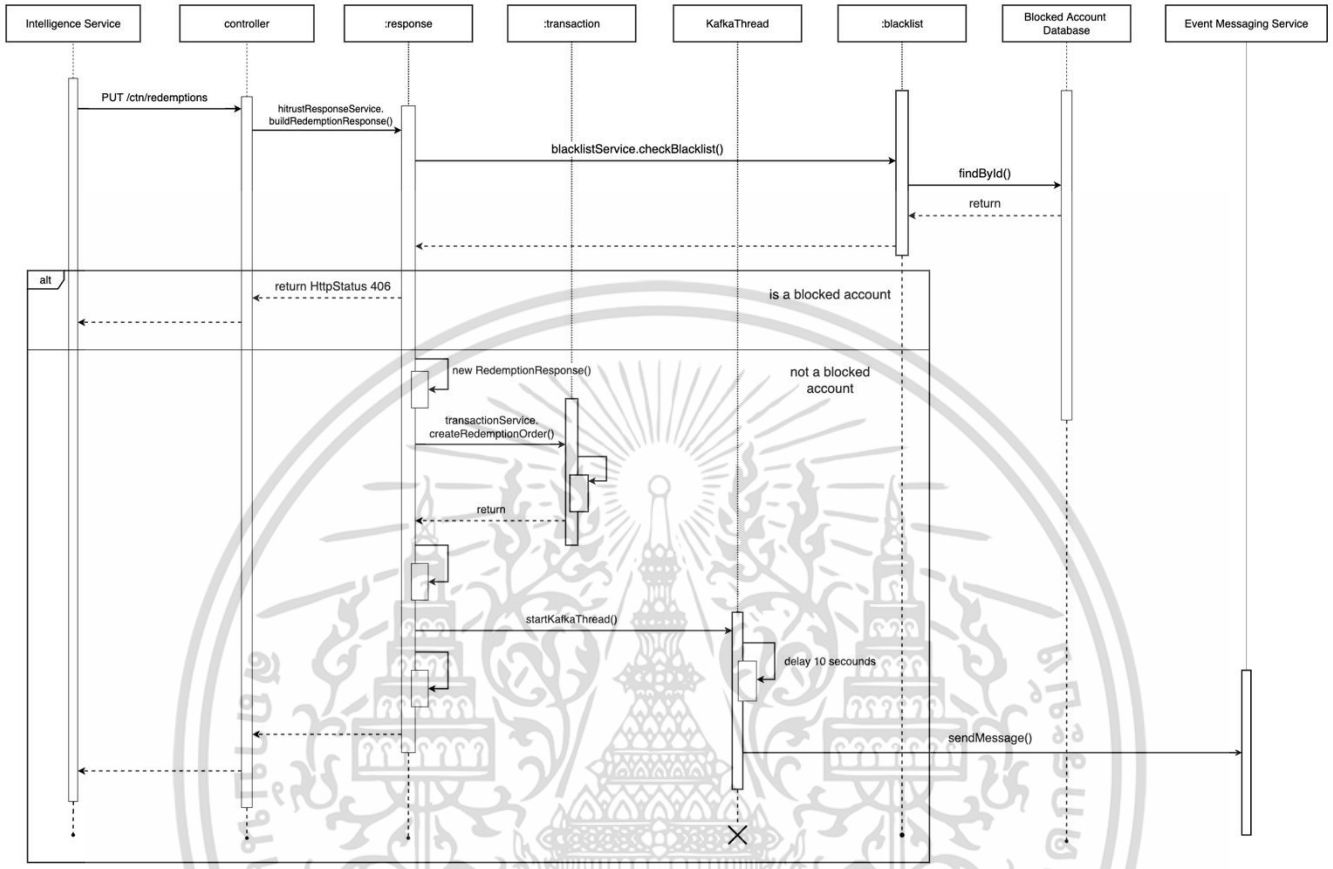
3.4.7.Authorize Transaction



รูปที่ 3.10 Sequence Diagram ของการ Authorize Transaction

กระบวนการทำงานของ Authorize Transaction จะเริ่มต้นเมื่อ Intelligence Service ทำการส่ง PUT Request มายัง API ใน Controller จากนั้นจะทำการเรียกใช้ method buildAuthorization-Response() จาก hitrustResponseService จากนั้นเมื่อ hitrustResponseService ถูกเรียกใช้แล้วก็จะทำการ return Http Status 200 กลับไปยัง Controller เพื่อส่งกลับไปให้ Intelligence Service ต่อ

3.4.8.Create Redemption Order

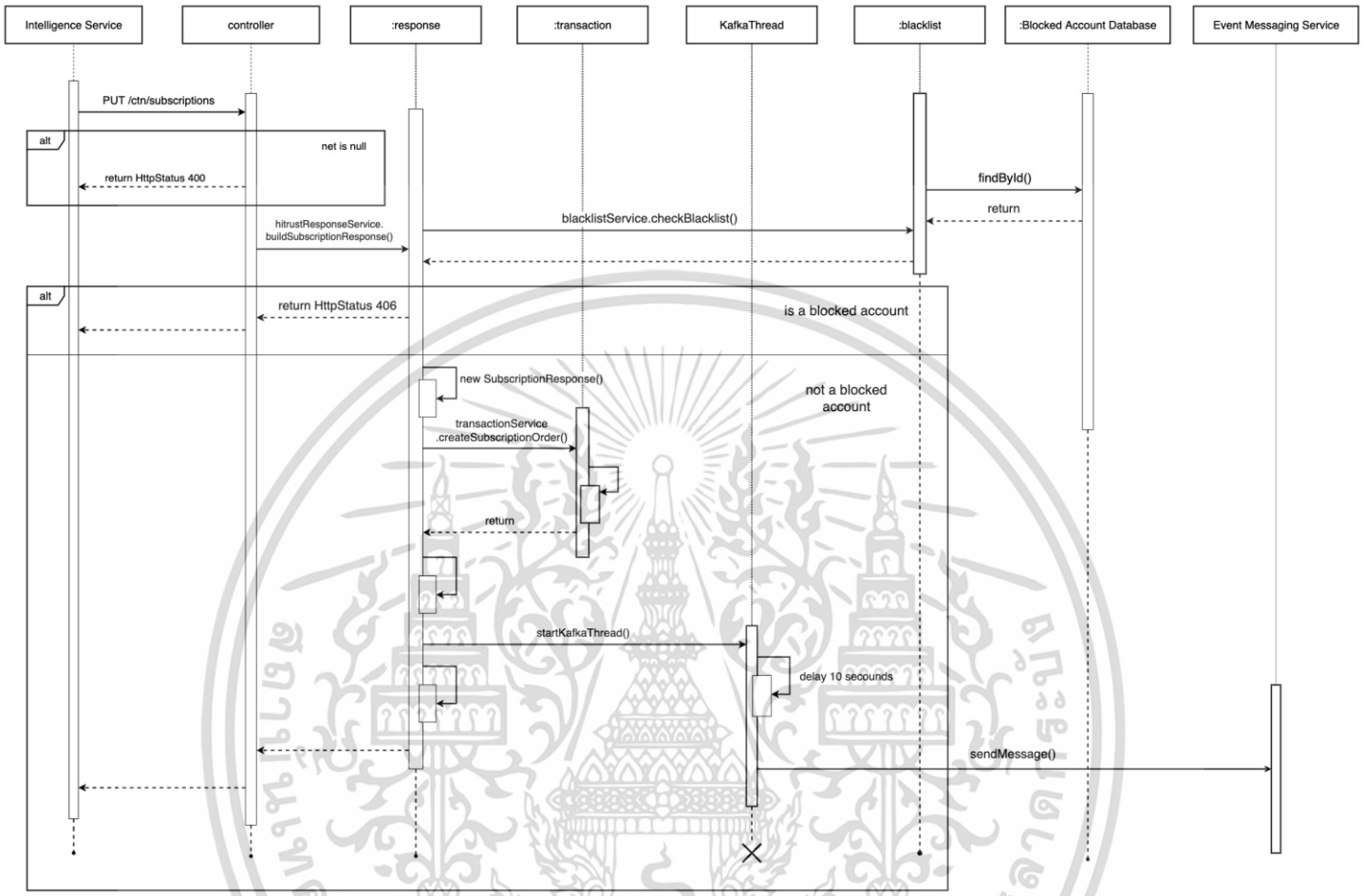


รูปที่ 3.11 Sequence Diagram ของการ Create Redemption Order

กระบวนการทำงานของ Create Redemption Order จะเริ่มต้นเมื่อ Intelligence Service ทำการส่ง POST Request มายัง API ใน Controller จากนั้นจะทำการเรียกใช้ method buildRedemptionResponse() จาก hitrustResponseService จากนั้น hitrustResponseService ก็จะทำการเรียกใช้ method checkBlacklist จาก blacklistService เพื่อทำการตรวจสอบว่า account ที่ต้องการ create order เป็น blocked account หรือไม่ หากตรวจสอบแล้วพบว่าเป็น blocked account ก็จะทำการ return Http Status 406 กลับไปให้ Intelligence Service แต่หากไม่เป็น blocked account ก็จะเข้าสู่กระบวนการ create order โดย hitrustResponseService จะสร้าง Object ของ RedemptionResponse ขึ้นมา แล้วเรียกใช้ createRedemptionOrder จาก transactionService เพื่อทำการ create object ของ redemption order จากนั้น transactionService จะ return object ที่สร้างมากลับไปให้ hitrustResponseService หลังจากได้รับค่าจาก transactionService แล้ว hitrustResponseService จะทำการเรียกใช้ method startKafkaThread ซึ่งเป็นการสร้าง Thread ของ

การทำงานขึ้นมาอีก 1 เธรด และกำหนดให้ Task ของ Thread นั้นคือการ sendMessage() ไปยัง EventMessaging Service ผ่าน kafkaTopic ที่กำหนด โดยกำหนดให้มีเวลา Delay ก่อนเริ่ม Task 10 วินาที เพื่อให้เธรดที่เป็นการประมวลผลและส่ง Response กลับไปให้ Intelligence Service นั้นทำงานเสร็จก่อน

3.4.9.Create Subscription Order

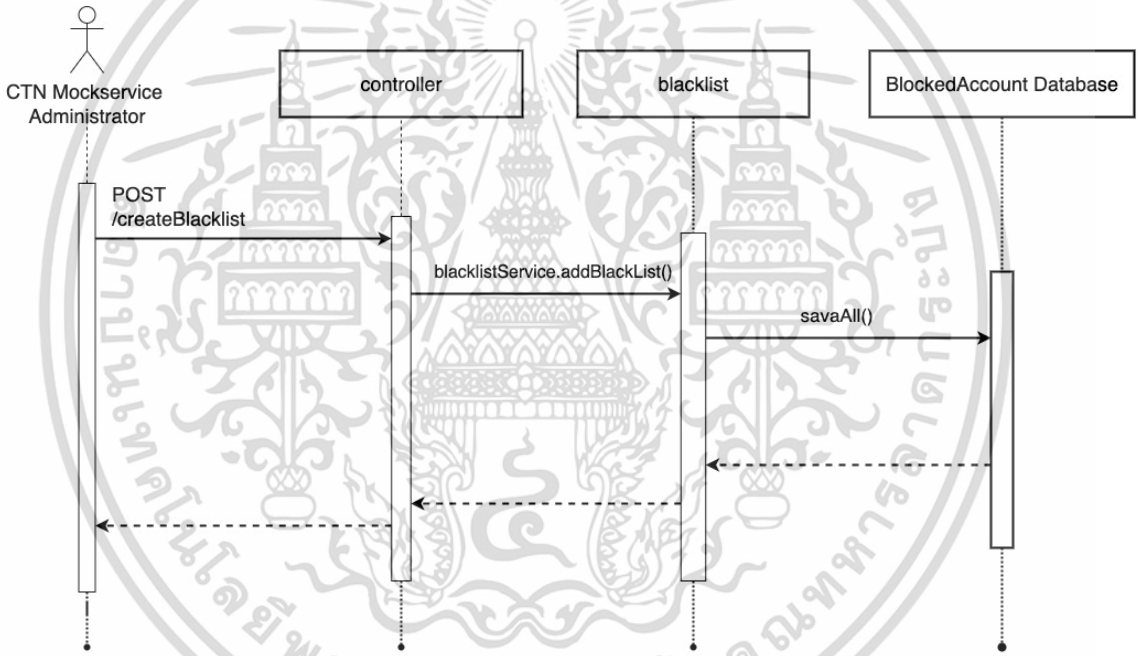


รูปที่ 3.12 Sequence Diagram ของการ Create Subscription Order

กระบวนการทำงานของ Create Subscription Order จะเริ่มต้นเมื่อ Intelligence Service ทำการส่ง POST Request มายัง API ใน Controller โดยขั้นแรก Controller จะทำการตรวจสอบก่อนว่า Body Request ที่ส่งเข้ามานั้นมีค่า net เพียงอย่างเดียวหรือไม่ หากไม่เป็นค่า net แต่เป็นอีกค่า (units) ก็จะทำการส่ง Http Status 400 กลับไปเพราะ Body Request นั้นไม่ตรงเงื่อนไข แต่หากค่าใน Body Request เป็นค่า net Conttroler ก็จากนั้นจะทำการเรียกใช้ method buildSubscriptionResponse() จาก hitrustResponseService จากนั้น hitrustResponseService ก็จะทำการเรียกใช้ method checkBlacklist จาก blacklistService เพื่อทำการตรวจสอบว่า account ที่ต้องการ create order เป็น blocked account หรือไม่ หากตรวจสอบแล้วพบว่าเป็น blocked account ก็จะทำการ return Http Status 406 กลับไปให้ Intelligence Service แต่หากไม่เป็น blocked account ก็จะเข้าสู่กระบวนการ create order โดย hitrustResponseService จะสร้าง Object ของ SubscriptionResponse ขึ้นมา

แล้วเรียกใช้ createSubscriptionOrder จาก transactionService เพื่อทำการ create object ของ redemption order จากนั้น transactionService จะ return object ที่สร้างมากลับไปให้ hitrustResponseService หลังจากได้รับค่าจาก transactionService แล้ว hitrustResponseService จะทำการเรียกใช้ method startKafkaThread ซึ่งเป็นการสร้าง Thread ของการทำงานขึ้นมาอีก 1 เธรด และกำหนดให้ Task ของ Thread นั้นคือการ sendMessage() ไปยัง EventMessaging Service ผ่าน kafkaTopic ที่กำหนด โดยกำหนดให้มีเวลา Delay ก่อนเริ่ม Task 10 วินาที เพื่อให้เธรดที่เป็นการประมวลผลและส่ง Response กลับไปให้ Intelligence Service นั้นทำงานเสร็จก่อน

3.4.10.Create Blacklist



รูปที่ 3.13 Sequence Diagram ของการ Create Blacklist

กระบวนการทำงานของ Create Blacklist จะเริ่มต้นเมื่อ Administrator ทำการส่ง POST Request มายัง API ใน Controller จากนั้นจะทำการเรียกใช้ method addBlacklist() จาก blacklistService เพื่อทำการสร้างรายการ Account ที่ต้องการใส่ใน Database จากนั้น blacklistService ก็จะทำการเรียกใช้ method saveAll() เพื่อบันทึก blocked account ลง Database เมื่อ Orders Database บันทึกข้อมูลเรียบร้อยแล้วก็จะทำการ Return ค่าที่บันทึกแล้วกลับมายัง blacklistService return ค่ากลับไปยัง Controller และ Administrator ต่อ

บทที่ 4
ผลการวิจัยและการอภิปรายผล

ในบทนี้จะกล่าวถึงการพัฒนา Mock Calastone Service ตามที่ได้รับ Requirements โดยส่วนแรกจะกล่าวถึงการพัฒนา Mock Calastone API ส่วนที่สองจะกล่าวถึงการพัฒนา HiTrust Integration ภายใน Calastone และส่วนสุดท้ายจะกล่าวถึงการทำ Unit Testing ของระบบ

4.1 การพัฒนา Mock Calastone Service

ในส่วนนี้จะกล่าวถึงการพัฒนา Mock Calastone Service ซึ่งประกอบไปด้วย 7 ส่วน ดังนี้

4.1.1. Calastone Order In-Memory Database

ในการทำงานร่วมกันระหว่าง Spring Application และ Database จำเป็นจะต้องมี Model ของข้อมูลที่จะบันทึกและนำมาใช้ระบุไว้ โดยสามารถเขียน Model ดังกล่าวในรูปแบบ Java Object เพื่อนำมาเรียกใช้ต่อในส่วนอื่น ๆ ของระบบ

```java
@Data
@Entity
@EntityListeners(AuditingEntityListener.class)
@Table(name = "orders")
@NoArgsConstructor
@AllArgsConstructor
@TypeDef(name = "json", typeClass = JsonType.class)
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "orderId", updatable = false, columnDefinition = "uuid")
    private UUID orderId;

    @Column(name = "orderRef", updatable = false, unique = true, nullable = false)
    private String orderReference = "ctnMock_" + (new Random().nextInt( bound: 1000));

    @Type(type = "json")
    @Column(columnDefinition = "json")
    private List<Transaction> transactions;

    @Column(name = "orderStatus")
    private String orderStatus;

    @Type(type = "json")
    @Column(columnDefinition = "json")
    private SubmitDetails submitDetails;
```

รูปที่ 4.1 ตัวอย่าง Model ของ Order Database

จากภาพข้างต้นบางส่วนจาก Model ของ Table ที่ชื่อว่า orders ภายใน Database โดยมี orderId (datatype : UUID) เป็น Primary Key ซึ่งตัว ID จะทำการ Generate ขึ้นมาทุกครั้งที่มี Order ใหม่

```
@Repository
public interface OrderRepository extends JpaRepository<Order, UUID> {

    7 usages  Thongtham, Nuttharika
    List<Order> findByUpdatedUtcBetween(ZonedDateTime start, ZonedDateTime end);

}
```

รูปที่ 4.2 ภาพแสดง Repository ของ Order Table

ในการทำงานร่วมกับ Database นอกจากจะต้องมี Model แล้วจำเป็นจะต้องมี Repository ที่เป็น Logic สำหรับการเข้าถึงข้อมูล เพื่อป้องกันการ Duplicate ของ Code และให้ง่ายต่อการซ่อมบำรุง โดยในงานชิ้นนี้ได้ใช้ JpaRepository จาก Spring Data JPA ซึ่งเป็น Library ที่ทาง Spring Framework มีไว้ให้อยู่แล้ว โดยใน JpaRepository ที่ Extends เข้ามานั้น มี Logic และคำสั่งพื้นฐานสำหรับบันทึกและจัดเก็บข้อมูลใน Database รวมไว้แล้ว หากต้องการ Logic หรือคำสั่งใด ๆ เพิ่มเติมก็สามารถสร้างคำสั่งใหม่ขึ้นมาได้ โดยในที่นี้ได้แก่คำสั่ง findByUpdatedUtcBetween ซึ่งเป็นคำสั่งที่จะค้นหาข้อมูลใน Table ในช่วงเวลาที่กำหนด

```
spring:


  h2:
    console:
      enabled: true
  datasource:
    url: jdbc:h2:mem:ctn
    driverClassName: org.h2.Driver
    username: sa
  jpa:
    database-platform: org.hibernate.dialect.H2Dialect
    show-sql: true
    properties:
      hibernate:
        format_sql: true
```

รูปที่ 4.3 ภาพแสดงตัวอย่างภายในไฟล์ application.yaml

นอกเหนือจากนั้น ในการทำงานร่วมกับ Database จำเป็นต้องมีการกำหนดค่าสำหรับเชื่อมต่อกับ Database ในที่นี้ใช้ H2 Database ซึ่งเป็น Database พื้นฐานที่ Spring Boot ใช้เป็นค่าเริ่มต้นเพื่อรองรับการใช้งานภาษา SQL ในการติดต่อกับฐานข้อมูล

4.1.2. Create Orders

Controllers ของ Mock Calastone Service นั้นถูกประกาศไว้พร้อมกับ Spring Annotation “@RestController” ที่ใช้สำหรับการสร้าง RESTful API ซึ่งจะทำการ Map Request ที่เข้ามาให้ไปยัง Request Handler ที่ประกาศเอาไว้รวมไปถึงการส่ง Response ด้วย

รูปที่ 4.4 ภาพแสดง Code Create Order ใน Order Controller

เมื่อมีการส่ง POST Request มาที่ URL “/orders/orders” Request จะถูก Map มายัง Handler ที่ชื่อว่า create โดยมีการกำหนด Data Structure ของ Request ที่ถูกส่งเข้ามาเอาไว้ หาก Request ดังกล่าวตรงตาม Data Structure Request นั้น ๆ ก็จะถูก Map ให้อยู่ในรูปแบบ Java Object จากนั้นจะทำการเรียกใช้ Method createOrder ที่ถูกสร้างไว้ใน Package orderService และทำการส่ง List ของ Order ที่ต้องการสร้างเข้าไป เมื่อ orderService ทำงานเสร็จก็จะส่งค่า List ของ Order ที่ถูกสร้างกลับมา จากนั้น Order Controller จะทำการตรวจสอบว่าค่าที่ถูกส่งกลับมานั้นเป็นค่าว่างหรือไม่ หากเป็นค่าว่างก็จะ Return Http Status 422 “Unprocessable Entity” พร้อมข้อความ “Create orders failed” กลับไป แต่หากค่าที่ orderService ส่งกลับมานั้นไม่เป็นค่าว่างก็จะทำการเรียกใช้ Method buildOrderResponse ใน Package responseService เพื่อทำการสร้าง Response ให้ตรงตามรูปแบบที่ต้องการสำหรับส่งกลับ

```
4 usages  Thongtham, Nuttharika
public List<Order> createOrder(List<Order> orders) {

    for (Order order : orders) {
        order.setOrderStatus("Submitted");
        order.setSubmitDetails(detailService.buildSubmitDetail(order));
        order.setValidationDetails(detailService.buildValidationDetail(order));
    }
    return orderRepository.saveAll(orders);

}
```

รูปที่ 4.5 ภาพแสดงคำสั่ง Create Order ใน OrderService

```
4 usages  Thongtham, Nuttharika
public SubmitDetails buildSubmitDetail(Order order) {

    for (Transaction transaction : order.getTransactions()) {
        if(transaction.getAccountDetails() == null){
            transaction.setAccountDetails(transactionService.buildAccount());
        }
        transaction.setFundIdentifier(transactionService.buildFundIdentifier());
    }

    SubmitDetails submitDetail = new SubmitDetails();
    submitDetail.setOrderType("CtnMockOrder");
    submitDetail.setOrderReference(order.getOrderReference());
    submitDetail.setCreatedUtc(ZonedDateTime.now().withZoneSameInstant(ZoneId.of( zoneId: "UTC")));
    submitDetail.setFundProviderIdentifier("MockCtn");
    submitDetail.setTransactions(order.getTransactions());

    return submitDetail;
}
```

รูปที่ 4.6 ภาพแสดง Code ใน Method buildSubmitDetail

```
3 usages  Thongtham, Nuttharika
public ValidationDetails buildValidationDetail(Order order) {
    return new ValidationDetails(
            valid: true,
            ZonedDateTime.now().withZoneSameInstant(ZoneId.of( zoneId: "UTC")),
            order.getOrderReference()
    );
}
```

รูปที่ 4.7 ภาพแสดง Code ใน Method buildValidationDetail

Method createOrder ภายใน Package orderService นั้น หลังจากถูกเรียกใช้และได้รับ List ของ Order ที่ต้องการสร้างแล้ว ก็จะทำการกำหนดค่าต่าง ๆ ใน Order นั้น ๆ ตามที่กำหนดไว้ โดยในส่วนนี้จะมีการเรียกใช้อีก 2 Methods จาก Package detailService เพื่อทำการสร้าง Java Object ของ

Detail ที่ต้องการ จากนั้นจึงจะทำการเรียกใช้คำสั่ง saveAll จาก Repository เพื่อทำการบันทึกข้อมูลลง Database

```
7 usages  Thongtham, Nuttharika
public List<OrderResponse> buildOrderResponse(List<Order> orderList) {
    List<OrderResponse> responses = new ArrayList<>();
    for (Order order : orderList) {
        responses.add(new OrderResponse(
                        order.getOrderId(),
                        order.getOrderReference(),
                        order.getOrderStatus(),
                        order.getCreatedUtc().withZoneSameInstant(ZoneId.of( zoneId: "UTC")),
                        order.getUpdatedUtc().withZoneSameInstant(ZoneId.of( zoneId: "UTC"))
                )
        );
    }
    return responses;
}
```

รูปที่ 4.8 ภาพแสดง Code การสร้าง Response เพื่อส่งกลับใน Method buildOrderResponse

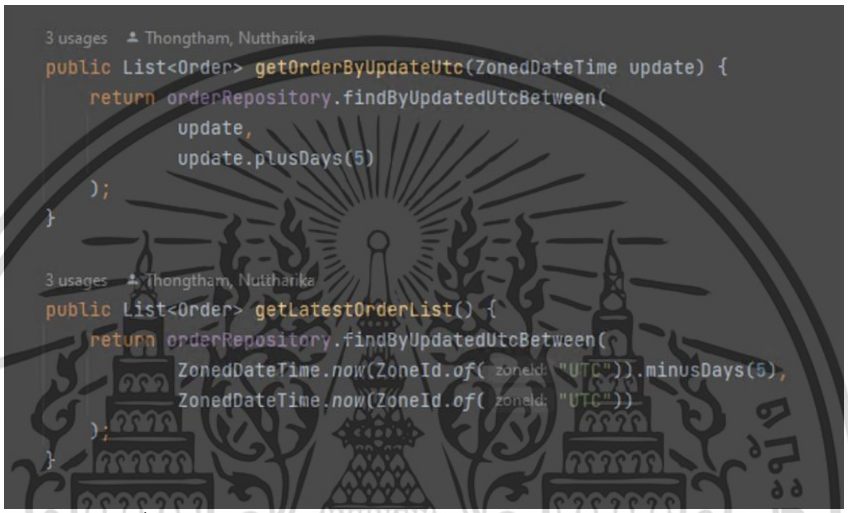
4.1.3. Get Updated Orders

```
 Thongtham, Nuttharika +1
@Operation(summary = "get a list of updated orders for the last 5 days. " +
        "if updated parameter is provided, then get last 5 days orders from that date.")
@ApiResponse(responseCode = "200", description = "Get Updated Orders Success")
@GetMapping("/orders/orders")
private List<OrderResponse> getUpdatedOrder(
        @Nullable @RequestParam String updated) {
    if (updated == null) {
        List<Order> orders = orderService.getLatestOrderList();
        return responseService.buildOrderResponse(orders);

    } else {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyyMMddHHmmssz");
        ZonedDateTime provideDate = ZonedDateTime.parse(updated, formatter);
        List<Order> orders = orderService.getOrderByUpdateUtc(provideDate);
        return responseService.buildOrderResponse(orders);

    }
}
```

รูปที่ 4.9 ภาพแสดง Get Updated Order ใน Order Controller

เมื่อมีการส่ง GET Request มาที่ URL “/orders/orders” Request จะถูก Map มายัง Handler ที่ชื่อว่า getUpdatedOrder โดยหาก Request ที่ส่งมานั้นไม่มี Parameter updated ซึ่งเป็นวันที่และเวลาที่ต้องการค้นหามาด้วย Order Controller ก็จะทำการเรียกใช้ Method getLatestOrderList เพื่อ

ค้นหาและเข้าถึงข้อมูล Order ในระยะเวลา 5 วันล่าสุดนับตั้งแต่วันที่ Request นี้ถูกสร้าง หากมีวันที่และเวลากำหนดมาด้วย Order Controller จะทำการเรียกใช้ Method getOrderByUpdateUtc พร้อมส่งวันเวลาที่กำหนดไป เมื่อได้รับ List ของ Order จาก orderService แล้ว Order Controller จะทำการเรียกใช้ Method buildOrderResponse ใน Package responseService เพื่อทำการสร้าง Response ให้ตรงตามรูปแบบที่ต้องการสำหรับส่งกลับ



รูปที่ 4.10 ภาพแสดง Method getOrderByUpdatedUtc

Method getOrderByUpdatedUtc ภายใน Package orderService นั้น หลังจากถูกเรียกใช้และได้รับวันเวลาที่ต้องกาค้นหาแล้ว ก็จะทำการเรียกใช้คำสั่ง findByUpdatedUtcBetween จาก Repository เพื่อทำการค้นหาข้อมูลจาก Table โดยเป็นการค้นหา Order ที่ถูกสร้างตั้งแต่วันที่กำหนดไปจนถึง 5 วันถัดไปของวันนั้น ๆ

Method getLatestOrderList ภายใน Package orderService จะทำการเรียกใช้คำสั่ง findByUpdatedUtcBetween จาก Repository เพื่อทำการค้นหาข้อมูลจาก Table เช่นเดียวกัน แต่เป็นการค้นหา Order ที่ถูกสร้างในช่วงระยะวันและเวลานั้นย้อนกลับไปถึง 5 วันก่อนหน้า

4.1.4.Get Order Details

เมื่อมีการส่ง GET Request มาที่ URL “/orders/orders/{orderId}” Request จะถูก Map มายัง Handler ที่ชื่อว่า getOrderDetails โดย Request ที่ส่งมานั้นจะต้องมี orderId ส่งมาด้วย จากนั้นก็จะทำการเรียกใช้ Method getOrderById เพื่อค้นหาและเข้าถึงข้อมูล Order ของ OrderId นั้น ๆ เมื่อ orderService ทำงานเสร็จก็จะส่งค่า Order ที่พบกลับมา จากนั้น Order Controller จะทำการตรวจสอบว่าค่าที่ถูกส่งกลับมานั้นเป็นค่าว่างหรือไม่ หากเป็นค่าว่างก็จะ Return Http Status 404 “Not

Found” พร้อมข้อความ “Order {orderId} Not Found” กลับไป แต่หากค่าที่ orderService ส่งกลับมานั้นไม่เป็นค่าว่างก็จะทำการเรียกใช้ Method buildOrderDetailResponse ใน Package responseService เพื่อทำการสร้าง Response ให้ตรงตามรูปแบบที่ต้องการสำหรับส่งกลับ

รูปที่ 4.11 ภาพแสดง getOrderDetails ภายใน Order Controller

รูปที่ 4.12 ภาพแสดง Method getOrderById ภายใน orderService

รูปที่ 4.13 ภาพแสดง Method buildOrderDetailsResponse ภายใน responseService

4.1.5. Accept Orders

เมื่อมีการส่ง POST Request มาที่ URL “/orders/orders/{orderId}/accept” Request จะถูก Map มายัง Handler ที่ชื่อว่า accept โดย Request ที่ส่งมานั้นจะต้องมี UUID และ AcceptRequest ส่งมาด้วย จากนั้นก็จะทำการเรียกใช้ Method getOrderById เพื่อค้นหาและเข้าถึงข้อมูล Order ของ OrderId นั้น ๆ เมื่อ orderService ทำงานเสร็จก็จะส่งค่า Order ที่พบกลับมา จากนั้น Order Controller จะทำการตรวจสอบว่าค่าที่ถูกส่งกลับมานั้นเป็นค่าว่างหรือไม่ หากเป็นค่าว่างก็จะ Return Http Status 404 “Not Found” พร้อมข้อความ “Order {orderId} Not Found” กลับไป แต่หากค่าที่ orderService ส่งกลับมานั้นไม่เป็นค่าว่างก็จะทำการเรียกใช้ Method buildAcceptResponse ใน Package responseService เพื่อทำการสร้าง Response ให้ตรงตามรูปแบบที่ต้องการสำหรับส่งกลับ โดยจะมีการเรียก Method acceptOrder ใน Package orderService เพื่อทำการสร้าง Detail และอัปเดตข้อมูล Order ใน Database

```
Thongtham, Nuttharika
@Operation(summary = "accept the created order of the order ID provided.")
@ApiResponse(responseCode = "200", description = "accept order completed.")
@ApiResponse(responseCode = "404", description = "order of provided order ID is not found.", content = @Content)
@PostMapping("/orders/orders/{orderId}/accept")
private AcceptResponse accept(
        @NotNull @PathVariable UUID orderId,
        @NotNull @RequestBody AcceptRequest request
) {
    Optional<Order> order = orderService.getOrderById(orderId);
    if (order.isPresent()) {
        return responseService.buildAcceptResponse(
                orderService.acceptOrder(order.get(), request.getDealReference()));
    } else {
        throw new ResponseStatusException(
                HttpStatus.NOT_FOUND, "Order " + orderId + " Not Found");
    }
}
```

รูปที่ 4.14 ภาพแสดง accept ภายใน Order Controller

```
3 usages   Thongtham, Nuttharika
public Order acceptOrder(Order order, String dealReference) {
    order.setOrderStatus("Accepted");
    order.setAcceptDetails(detailService.buildAcceptDetail(order, dealReference));
    return orderRepository.save(order);
}
```

รูปที่ 4.15 ภาพแสดง Method acceptOrder ภายใน orderService

Method acceptOrder จะทำการกำหนดค่า OrderStatus ของ Order นั้น ๆ ให้เป็น “Accepted” และทำการเรียกใช้ Method buildAcceptDetail ใน Package detailService จากนั้นจะทำการบันทึกลง Database ผ่านคำสั่งใน Repository

```
3 usages  Thongtham, Nuttharika
public AcceptDetails buildAcceptDetail(Order order, String dealReference) {
    AcceptDetails acceptDetails = new AcceptDetails();
    acceptDetails.setAccepted(true);
    acceptDetails.setCreatedUtc(ZonedDateTime.now().withZoneSameInstant(ZoneId.of( zoneId: "UTC")));
    acceptDetails.setOrderReference(order.getOrderReference());
    acceptDetails.setDealReference(dealReference);
    return acceptDetails;
}
```

รูปที่ 4.16 ภาพแสดง Method buildAcceptDetail ภายใน detailService

```
5 usages  Thongtham, Nuttharika
public AcceptResponse buildAcceptResponse(Order order) {
    return new AcceptResponse(
            order.getOrderId(),
            order.getOrderStatus(),
            order.getCreatedUtc().withZoneSameInstant(ZoneId.of( zoneId: "UTC")),
            order.getUpdatedUtc().withZoneSameInstant(ZoneId.of( zoneId: "UTC")),
            order.getSubmitDetails(),
            order.getValidationDetails(),
            order.getAcceptDetails()
    );
}
```

รูปที่ 4.17 ภาพแสดง Method buildAcceptResponse ภายใน responseService

4.1.6.Reject Orders

```
Thongtham, Nuttharika
@Operation(summary = "reject the created order of order ID provide with reject reason.")
@ApiResponse(responseCode = "200", description = "reject order completed.")
@ApiResponse(responseCode = "404", description = "order of provided order ID is not found.", content = @Content)
@PostMapping("/orders/orders/{orderId}/reject")
private AcceptResponse reject(
        @NotNull @PathVariable UUID orderId,
        @NotNull @RequestBody RejectRequest request
) {
    Optional<Order> order = orderService.getOrderById(orderId);

    if (order.isPresent()) {
        return responseService.buildAcceptResponse(
                orderService.rejectOrder(order.get(), request.getRejectReason()));
    } else {
        throw new ResponseStatusException(
                HttpStatus.NOT_FOUND, "Order " + orderId + " Not Found");
    }
}
```

รูปที่ 4.18 ภาพแสดง reject ภายใน Order Controller

เมื่อมีการส่ง POST Request มาที่ URL “/orders/orders/{orderId}/reject” Request จะถูก Map มายัง Handler ที่ชื่อว่า reject โดย Request ที่ส่งมานั้นจะต้องมี UUID และ RejectRequest ส่ง

มาด้วย จากนั้นก็จะทำการเรียกใช้ Method getOrderById เพื่อค้นหาและเข้าถึงข้อมูล Order ของ OrderId นั้น ๆ เมื่อ orderService ทำงานเสร็จก็จะส่งค่า Order ที่พบกลับมา จากนั้น Order Controller จะทำการตรวจสอบว่าค่าที่ถูกส่งกลับมานั้นเป็นค่าว่างหรือไม่ หากเป็นค่าว่างก็จะ Return Http Status 404 “Not Found” พร้อมข้อความ “Order {orderId} Not Found” กลับไป แต่หากค่าที่ orderService ส่งกลับมานั้นไม่เป็นค่าว่างก็จะทำการเรียกใช้ Method buildAcceptResponse ใน Package responseService เพื่อทำการสร้าง Response ให้ตรงตามรูปแบบที่ต้องการสำหรับส่งกลับ โดยจะมีการเรียก Method rejectOrder ใน Package orderService เพื่อทำการสร้าง Detail และอัปเดตข้อมูล Order ใน Database

```
3 usages  Thongtham, Nuttharika
public Order rejectOrder(Order order, String rejectReason) {
    order.setOrderStatus("Rejected");
    order.setAcceptDetails(detailService.buildRejectDetail(order, rejectReason));
    return orderRepository.save(order);
}
```

รูปที่ 4.19 ภาพแสดง Method rejectOrder ภายใน orderService

```
3 usages  Thongtham, Nuttharika
public AcceptDetails buildRejectDetail(Order order, String rejectReason) {
    AcceptDetails acceptDetails = new AcceptDetails();
    acceptDetails.setAccepted(false);
    acceptDetails.setCreatedUtc(ZonedDateTime.now().withZoneSameInstant(ZoneId.of( zoneId: "UTC")));
    acceptDetails.setOrderReference(order.getOrderReference());
    acceptDetails.setRejectReason(rejectReason);
    return acceptDetails;
}
```

รูปที่ 4.20 ภาพแสดง Method buildRejectDetail ภายใน detailService

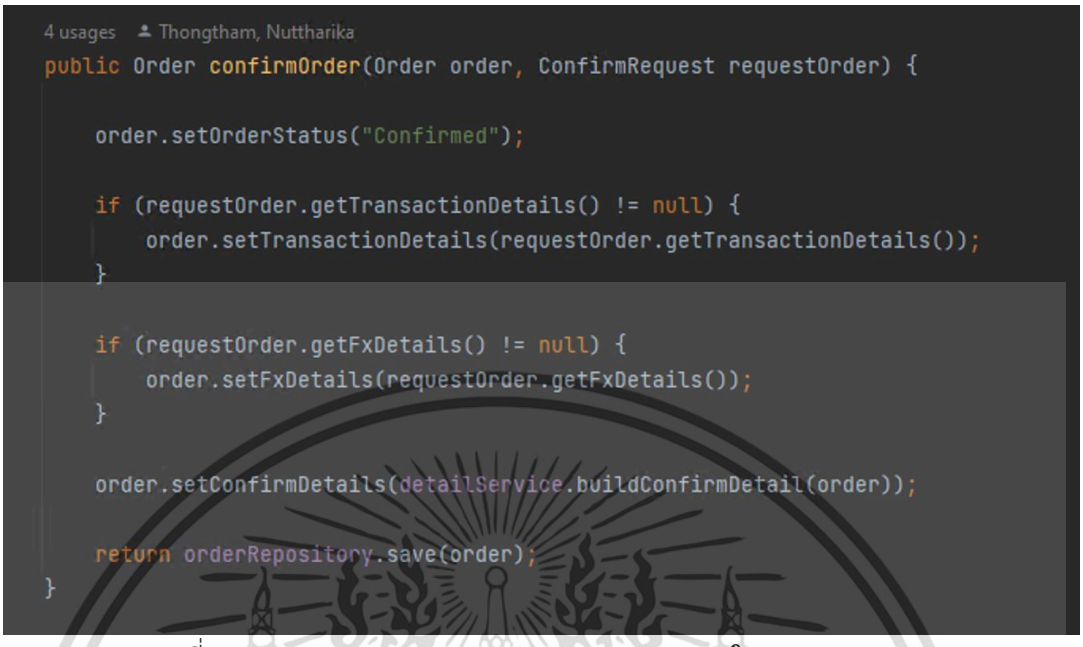
Method rejectOrder จะทำการกำหนดค่า OrderStatus ของ Order นั้น ๆ ให้เป็น “Rejected”, และทำการเรียกใช้ Method buildRejectDetail ใน Package detailService จากนั้นจะทำการบันทึกลง Database ผ่านคำสั่งใน Repository

4.1.7. Confirm Orders

```
Thongtham, Nuttharika
@Operation(summary = "confirm the order of order ID provided with confirm details.")
@ApiResponse(responseCode = "200", description = "confirm order completed.")
@ApiResponse(responseCode = "404", description = "order of provided order ID is not found.", content = @Content)
@PostMapping("/orders/orders/{orderId}/confirm")
private ConfirmResponse confirm(
        @NotNull @PathVariable UUID orderId,
        @NotNull @RequestBody ConfirmRequest request
) {
    Optional<Order> order = orderService.getOrderById(orderId);

    if (order.isPresent()) {
        return responseService.buildConfirmResponse(
                orderService.confirmOrder(order.get(), request));
    } else {
        throw new ResponseStatusException(
                HttpStatus.NOT_FOUND, "Order " + orderId + " Not Found");
    }
}
```

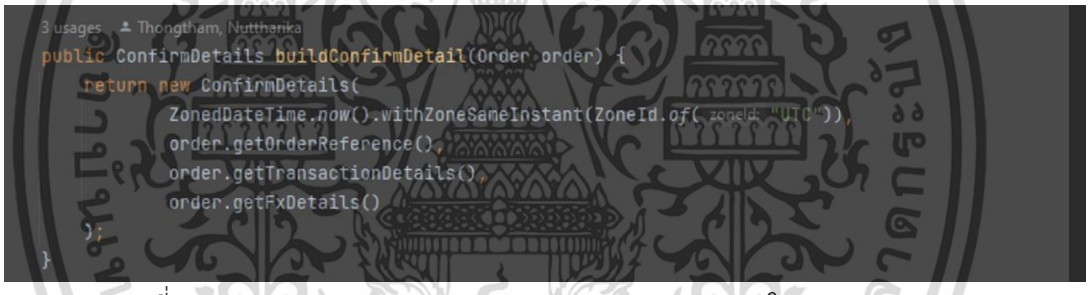
รูปที่ 4.21 ภาพแสดง confirm ภายใน Order Controller

เมื่อมีการส่ง POST Request มาที่ URL “/orders/orders/{orderId}/confirm” Request จะถูก Map มายัง Handler ที่ชื่อว่า reject โดย Request ที่ส่งมานั้นจะต้องมี UUID และ ConfirmRequest ส่งมาด้วย จากนั้นก็จะทำการเรียกใช้ Method getOrderById เพื่อค้นหาและเข้าถึงข้อมูล Order ของ OrderId นั้น ๆ เมื่อ orderService ทำงานเสร็จก็จะส่งค่า Order ที่พบกลับมา จากนั้น Order Controller จะทำการตรวจสอบว่าค่าที่ถูกส่งกลับมานั้นเป็นค่าว่างหรือไม่ หากเป็นค่าว่างก็จะ Return Http Status 404 “Not Found” พร้อมข้อความ “Order {orderId} Not Found” กลับไป แต่หากค่าที่ orderService ส่งกลับมานั้นไม่เป็นค่าว่างก็จะทำการเรียกใช้ Method buildConfirmResponse ใน Package responseService เพื่อทำการสร้าง Response ให้ตรงตามรูปแบบที่ต้องการสำหรับส่งกลับ โดยจะมีการเรียก Method rejectOrder ใน Package orderService เพื่อทำการสร้าง Detail และอัปเดตข้อมูล Order ใน Database



```
4 usages  Thongtham, Nuttharika
public Order confirmOrder(Order order, ConfirmRequest requestOrder) {

    order.setOrderStatus("Confirmed");

    if (requestOrder.getTransactionDetails() != null) {
        order.setTransactionDetails(requestOrder.getTransactionDetails());
    }

    if (requestOrder.getFxDetails() != null) {
        order.setFxDetails(requestOrder.getFxDetails());
    }

    order.setConfirmDetails(detailService.buildConfirmDetail(order));

    return orderRepository.save(order);
}
```

รูปที่ 4.22 ภาพแสดง Method confirmOrder ภายใน orderService



```
3 usages  Thongtham, Nuttharika
public ConfirmDetails buildConfirmDetail(Order order) {
    return new ConfirmDetails(
            ZonedDateTime.now().withZoneSameInstant(ZoneId.of( zoneId: "UTC")),
            order.getOrderReference(),
            order.getTransactionDetails(),
            order.getFxDetails()
    );
}
```

รูปที่ 4.23 ภาพแสดง Method buildConfirmDetail ภายใน detailService

Method confirmOrder จะทำการกำหนดค่า OrderStatus ของ Order นั้น ๆ ให้เป็น “Confirmed”, อัปเดตข้อมูล TransactionDetails และ FxDetail ที่ได้จาก ConfirmRequest จากนั้นทำการเรียกใช้ Method buildConfirmDetail ใน Package detailService แล้วทำการบันทึกลง Database ผ่านคำสั่งใน Repository

4.2 การพัฒนา HiTrust Integration

4.2.1. HiTrust Blacklist In-Memory Database

```
30 usages  Thongtham, Nuttharika
@Data
@Entity
@Table(name = "blacklists")
@NoArgsConstructor
@AllArgsConstructor
public class BlockedAccount {

    @Id
    private String portfolioNumber;

}
```

รูปที่ 4.24 ตัวอย่าง Model ของ Blacklists Database

จากภาพข้างต้นคือ Model ของ Table ที่ชื่อว่า blacklists ภายใน Database โดยมี portfolioNumber (datatype : String) เป็น Primary Key

4.2.2.Kafka Producer

```yaml
stream:
  kafka:
    binder:
      brokers: [redacted]
      configuration:
        security:
          protocol: SASL_SSL
        ssl:
          truststore:
            location: ${kafka.ssl.truststore.location}
            type: JKS
          endpoint:
            identification:
              algorithm: ""
        sasl:
          mechanism: SCRAM-SHA-512
          kerberos:
            service:
              name: ${spring.application.name}
      jaas:
        loginModule: org.apache.kafka.common.security.scram.ScramLoginModule
cloud:
  vault:
    fail-fast: true
    host: [redacted]
    scheme: https
    port: 443
    authentication: APPROLE
    config.lifecycle:
      enabled: false
    session.lifecycle:
      enabled: false
    consul:
      enabled: false
    generic:
      enabled: true
```

รูปที่ 4.25 ภาพแสดงตัวอย่างการกำหนดการเชื่อมต่อกับ Kafka ภายในไฟล์ application.yaml

4.2.3. Create Subscription Order

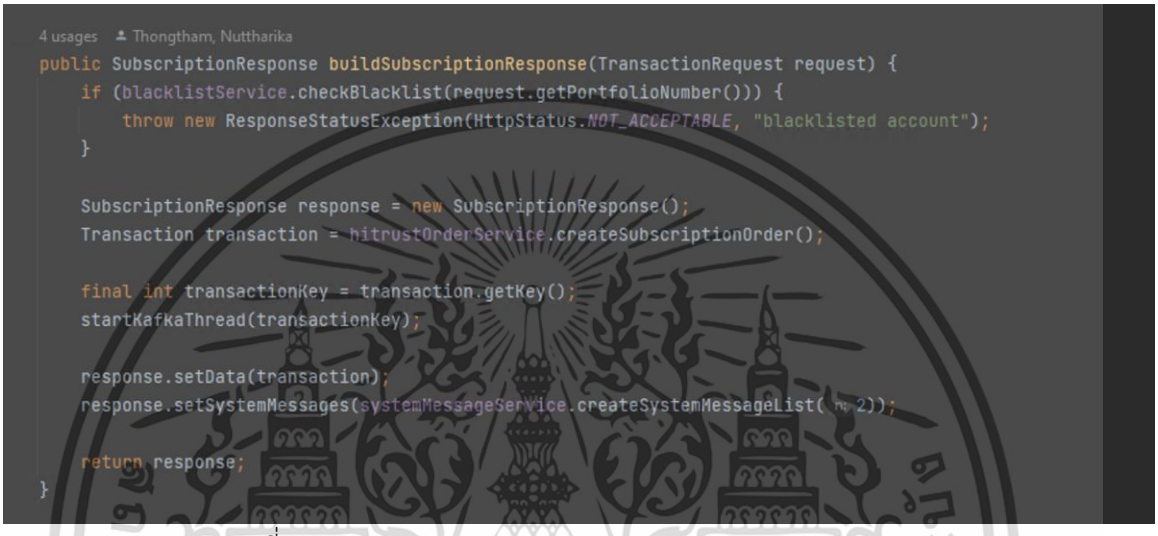
Controllers ของ Mock HiTrust Service นั้นถูกประกาศไว้พร้อมกับ Spring Annotation “@RestController” ที่ใช้สำหรับการสร้าง RESTful API ซึ่งจะทำการ Map Request ที่เข้ามาให้ไปยัง Request Handler ที่ประกาศเอาไว้รวมไปถึงการส่ง Response ด้วย

```java
Thongtham, Nutthanka
@Operation(summary = "create subscription order from HiTrust body request.")
@ApiResponse(responseCode = "200", description = "Create subscription order success")
@PutMapping("/ctn/subscriptions")
private SubscriptionResponse createSubscriptionOrder(@RequestBody @NotNull TransactionRequest request) {
    if (request.getNet() == null) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, " <net> not found in request body");
    }
    return hitrustResponseService.buildSubscriptionResponse(request);
}
```

รูปที่ 4.26 ภาพแสดง createSubscriptionOrder ภายใน HiTrust Order Controller

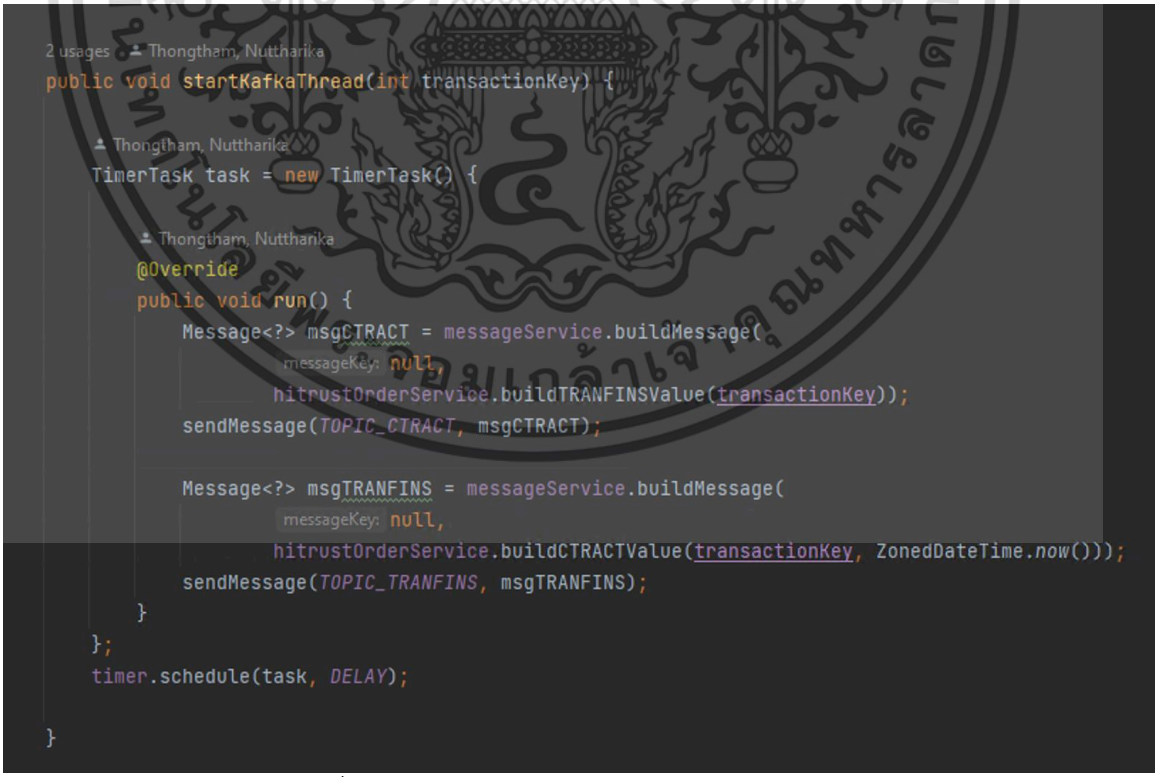
เมื่อมีการส่ง PUT Request มาที่ URL “/ctn/subscriptions” Request จะถูก Map มายัง Handler ที่ชื่อว่า createSubscriptionOrder โดยมีการกำหนด Data Structure ของ Request ที่ถูกส่งเข้ามาเอาไว้ หาก Request ดังกล่าวตรงตาม Data Structure Request นั้น ๆ ก็จะถูก Map ให้อยู่ในรูปแบบ Java Object จากนั้นจะทำการตรวจสอบว่า Request ที่ส่งมามีค่า Net หรือไม่ หากเป็นค่าว่าง

ก็จะ Return Http Status 400 “Bad Request” พร้อมข้อความ “<net> not found in body request” กลับไป แต่หากมีค่า Net ก็จะทำการเรียกใช้ Method buildSubscriptionOrder ใน Package hitrustResponseService เพื่อทำการสร้าง Response ให้ตรงตามรูปแบบที่ต้องการสำหรับส่งกลับ



```
4 usages  Thongtham, Nuttharika
public SubscriptionResponse buildSubscriptionResponse(TransactionRequest request) {
    if (blacklistService.checkBlacklist(request.getPortfolioNumber())) {
        throw new ResponseStatusException(HttpStatus.NOT_ACCEPTABLE, "blacklisted account");
    }

    SubscriptionResponse response = new SubscriptionResponse();
    Transaction transaction = hitrustOrderService.createSubscriptionOrder();

    final int transactionKey = transaction.getKey();
    startKafkaThread(transactionKey);

    response.setData(transaction);
    response.setSystemMessages(systemMessageService.createSystemMessageList( n: 2));

    return response;
}
```

รูปที่ 4.27 ภาพแสดง Method buildSubscriptionResponse



```
2 usages  Thongtham, Nuttharika
public void startKafkaThread(int transactionKey) {

    Thongtham, Nuttharika
    TimerTask task = new TimerTask() {

        Thongtham, Nuttharika
        @Override
        public void run() {
            Message<?> msgCTRACT = messageService.buildMessage(
                    messageKey: null,
                    hitrustOrderService.buildTRANFINSValue(transactionKey));
            sendMessage(TOPIC_CTRACT, msgCTRACT);

            Message<?> msgTRANFINS = messageService.buildMessage(
                    messageKey: null,
                    hitrustOrderService.buildCTRACTValue(transactionKey, ZonedDateTime.now()));
            sendMessage(TOPIC_TRANFINS, msgTRANFINS);
        }
    };
    timer.schedule(task, DELAY);

}
```

รูปที่ 4.28 ภาพแสดง Method startKafkaThread

Method buidSubscriptionResponse ภายใน hitrustResponseService นั้น เริ่มแรกจะทำการตรวจสอบ PortfolioNumber ของ Request นั้น ๆ ก่อนว่าอยู่ใน Blacklist หรือไม่ หากอยู่ใน Blacklist ก็จะทำการ Return HTTP Status 406 “Not Acceptable” กลับไป แต่หากไม่อยู่ในรายการ Blacklist ก็จะเริ่มการทำงานต่อ โดยสร้าง Object ของ Subscription Response ขึ้นมา แล้วทำการเรียกใช้ Method createSubscriptionOrder ใน Package hitrsutOrderservice เพื่อทำการสร้าง Transaction ของ Order นั้น ๆ จากนั้นเมื่อ Method startKafkaThread ถูกเรียกใช้ การทำงานจะถูกแบ่งออกเป็น 2 Thread, Thread หนึ่งถูกมอบหมายให้รับผิดชอบการส่ง Message ไปยัง Kafka Topic ที่กำหนด โดย Thread นี้จะเริ่มทำงานก็ต่อเมื่อเวลาผ่านไป 10 มิลลิวินาที ส่วนอีก Thread จะมีหน้าที่ในการส่ง Response กลับไปยัง Controller

```
5 usages   Thongtham, Nuttharika
public Transaction createSubscriptionOrder() {
    Transaction transaction = new Transaction();
    transaction.setNumber("000" + new Random().nextInt( bound: 100000));
    transaction.setKey(new Random().nextInt( bound: 100000));
    transaction.setStatus("New");
    return transaction;
}
```

รูปที่ 4.29 ภาพแสดง Method createSubscriptionOrder ภายใน hitrustOrderService

```
2 usages   Thongtham, Nuttharika
public SystemMessage createSystemMessage(){
    SystemMessage systemMessage = new SystemMessage();
    systemMessage.setCode(12345);
    systemMessage.setText("(12345) Text");
    systemMessage.setType("type");
    return systemMessage;
}

5 usages   Thongtham, Nuttharika
public List<SystemMessage> createSystemMessageList(int n){
    List<SystemMessage> systemMessages = new ArrayList<>();
    for(int i=0 ; i<n ; i++){
        systemMessages.add(createSystemMessage());
    }
    return systemMessages;
}
```

รูปที่ 4.30 ภาพแสดง Method createSystemMessage

4.2.4. Create Redemption Order

```
Thongtham, Nuttharika
@Operation(summary = "create redemption order from HiTrust body request.")
@ApiResponse(responseCode = "200", description = "Create redemption order success")
@PutMapping("/ctn/redemptions")
private RedemptionResponse createRedemptionOrder(@RequestBody @NotNull TransactionRequest request) {
    return hitrustResponseService.buildRedemptionResponse(request);
}
```

รูปที่ 4.31 ภาพแสดง createRedemptionOrder ภายใน HiTrust Order Controller

เมื่อมีการส่ง PUT Request มาที่ URL “/ctn/redemptions” Request จะถูก Map มายัง Handler ที่ชื่อว่า createRedemptionOrder โดยมีการกำหนด Data Structure ของ Request ที่ถูกส่งเข้ามาเอาไว้ หาก Request ดังกล่าวตรงตาม Data Structure Request นั้น ๆ ก็จะถูก Map ให้อยู่ในรูปแบบ Java Object จากนั้น ก็จะทำการเรียกใช้ Method buildRedemptionOrder ในPackage hitrustResponseService เพื่อทำการสร้าง Response ให้ตรงตามรูปแบบที่ต้องการสำหรับส่งกลับ

```
4 usages  Thongtham, Nuttharika
public RedemptionResponse buildRedemptionResponse(TransactionRequest request) {

    if (blacklistService.checkBlacklist(request.getPortfolioNumber())) {
        throw new ResponseStatusException(HttpStatus.NOT_ACCEPTABLE, "blacklisted account");
    }

    RedemptionResponse response = new RedemptionResponse();
    Transaction transaction = hitrustOrderService.createRedemptionOrder();

    final int transactionKey = transaction.getKey();
    startKafkaThread(transactionKey);

    response.setData(transaction);
    response.setSystemMessages(systemMessageService.createSystemMessageList( n: 2));

    return response;

}
```

รูปที่ 4.32 ภาพแสดง buildRedemptionResponse ภายใน hitrustResponseService

```
3 usages  Thongtham, Nuttharika
public Transaction createRedemptionOrder() {
    Transaction transaction = new Transaction();
    transaction.setNumber("000" + new Random().nextInt( bound: 100000));
    transaction.setKey(new Random().nextInt( bound: 100000));
    transaction.setStatus("Authorised");
    return transaction;
}
```

รูปที่ 4.33 ภาพแสดง createRedemptionOrder ภายใน hitrustOrderService

Method buidRedemptionResponse ภายใน hitrustResponseService นั้น เริ่มแรกจะทำการตรวจสอบ PortfolioNumber ของ Request นั้น ๆ ก่อนว่าอยู่ใน Blacklist หรือไม่ หากอยู่ใน Blacklist ก็จะทำการ Return HTTP Status 406 “Not Acceptable” กลับไป แต่หากไม่อยู่ในรายการ Blacklist ก็จะเริ่มการทำงานต่อ โดยสร้าง Object ของ Redemption Response ขึ้นมา แล้วทำการเรียกใช้ Method createRedemptionOrder ใน Package hitrsutOrderservice เพื่อทำการสร้าง Transaction ของ Order นั้น ๆ จากนั้นเมื่อ Method startKafkaThread ถูกเรียกใช้ การทำงานจะถูกแบ่งออกเป็น 2 Thread, Thread หนึ่งถูกมอบหมายให้รับผิดชอบการส่ง Message ไปยัง Kafka Topic ที่กำหนด โดย Thread นี้จะเริ่มทำงานก็ต่อเมื่อเวลาผ่านไป 10 มิลลิวินาที ส่วนอีก Thread จะมีหน้าที่ในการส่ง Response กลับไปยัง Controller

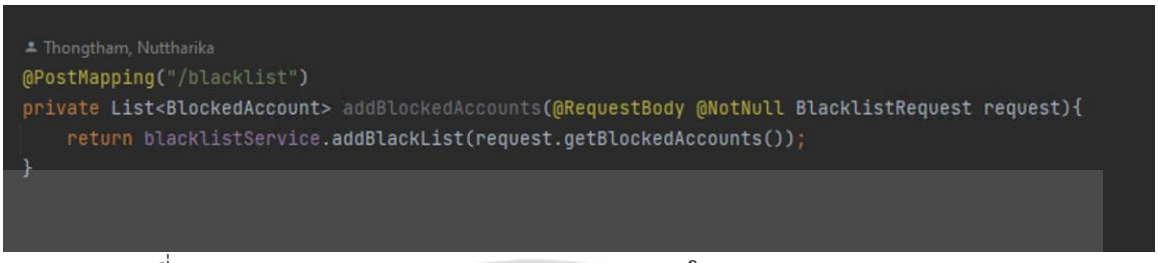
4.2.5. Authorize Transaction

```
Thongtham, Nuttharika
@Operation(summary = "authorize transaction after it sent to HiTrust successfully.")
@ApiResponse(responseCode = "200", description = "Authorise transaction success")
@PostMapping("/transactionauthorisation")
private AuthorizationResponse transactionAuth(@RequestBody @NotNull TransactionAuthRequest request) {
    return hitrustResponseService.buildAuthorizationResponse();
}
```

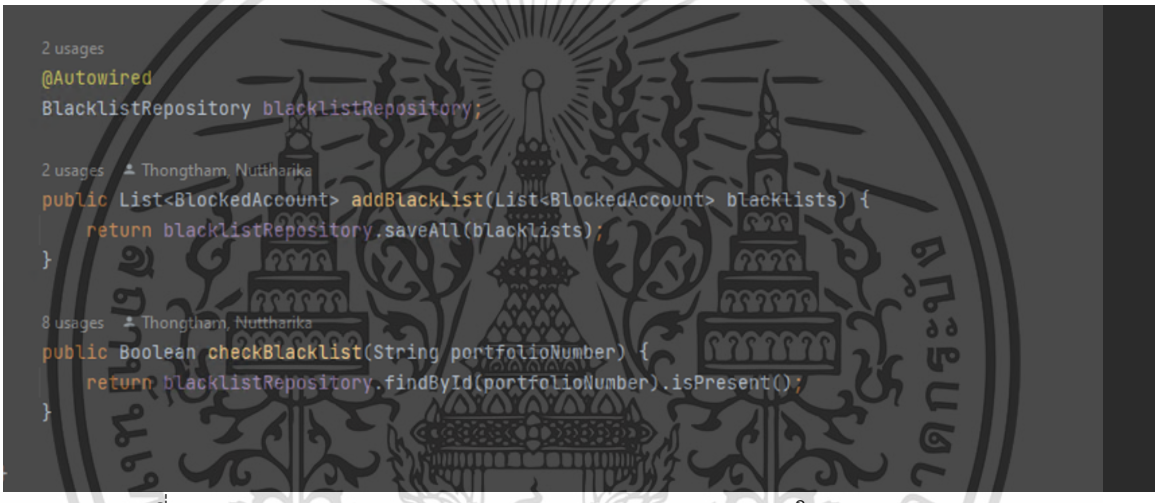
รูปที่ 4.34 ภาพแสดง transactionAuth ภายใน HiTrust Order Controller

เมื่อมีการส่ง POST Request มาที่ URL “/transactionautherisation” Request จะถูก Map มายัง Handler ที่ชื่อว่า transactionAuth โดย Authorize Transaction เป็นเพียง Method ที่รับ Request ตาม Data Structure ใน TransactionAuthRequest และ Return Http Status 200 กลับไปเท่านั้น

4.2.6. Create Blacklist



รูปที่ 4.35 ภาพแสดง addBlockedAccounts ภายใน HiTrust Order Controller



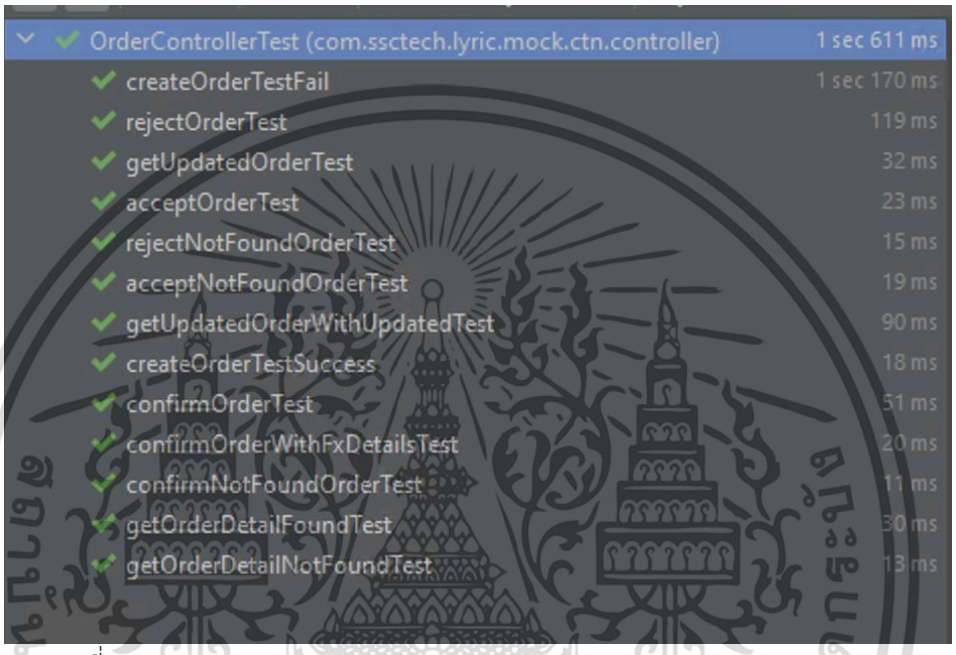
รูปที่ 4.36 ภาพแสดง Method addBlockedAccounts ภายใน Blacklist Service

เมื่อมีการส่ง POST Request มาที่ URL “/blacklists” Request จะถูก Map มายัง Handler ที่ชื่อว่า addBlockedAccounts โดยจะต้องที Request Body ที่มี Data Structure ตรงกับที่กำหนดไว้และไม่เป็นค่าว่าง จากนั้นก็จะทำการเรียก Method addBlacklist ใน Package blacklistService โดยภายใน addBlacklist ก็จะทำการเรียกใช้คำสั่ง saveAll จาก repository อีกครั้งเพื่อทำการบันทึกรายการ Blacklist ที่ต้องการ

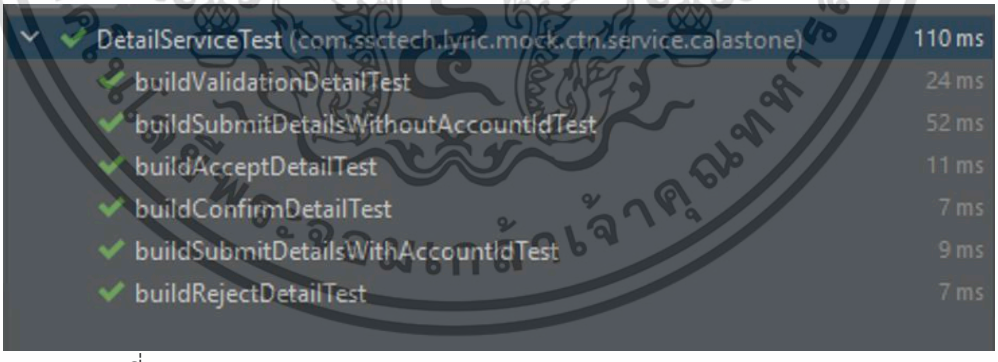
4.3 การทำ Unit Testing

4.3.1. รายการ Testcase และผลการทดสอบของ Mock Calastone Service

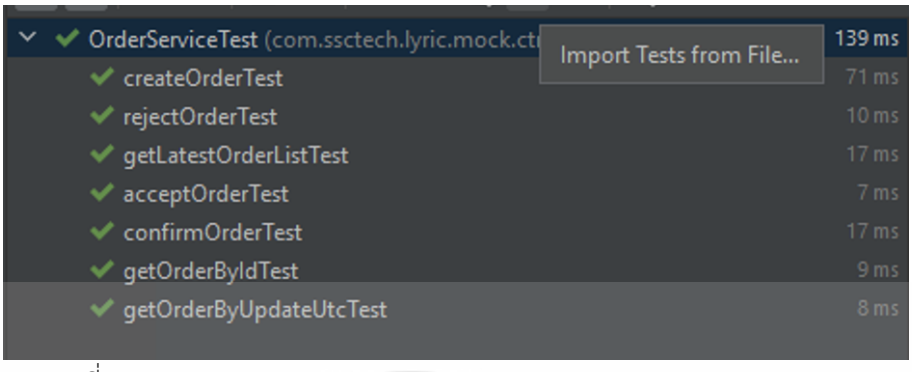
Mock Calastone Service ได้ทำการทดสอบแบบ Unit Testing โดยใช้ Junit ในการทดสอบ มีรายการ Testcase และผลดังนี้



รูปที่ 4.37 ภาพแสดง Testcase ของ Order Controller และผลการทดสอบ



รูปที่ 4.38 ภาพแสดง Testcase ของ Detail Service และผลการทดสอบ



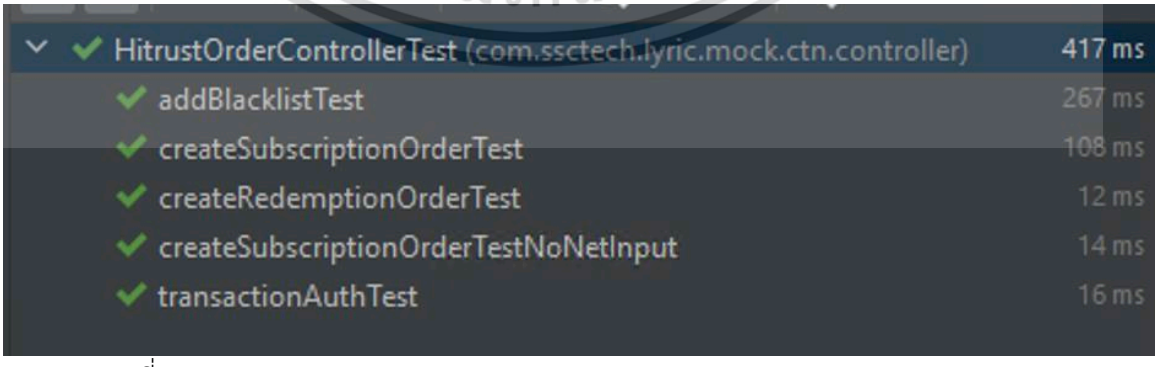
รูปที่ 4.39 ภาพแสดง Testcase ของ Order Service และผลการทดสอบ

รูปที่ 4.40 ภาพแสดง Testcase ของ Response Service และผลการทดสอบ

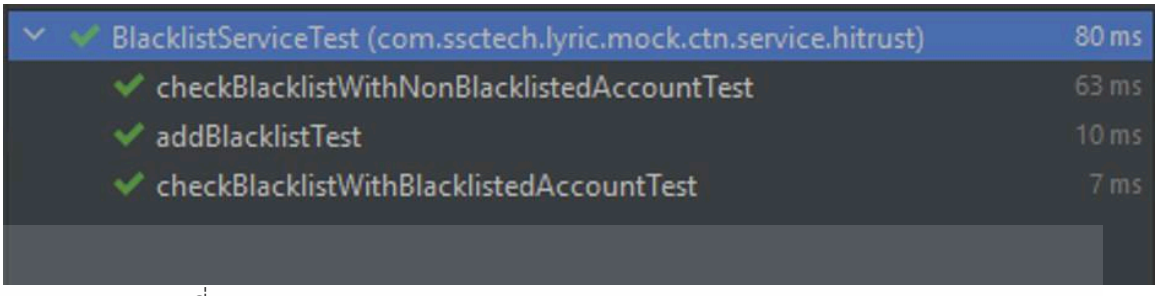
รูปที่ 4.41 ภาพแสดง Testcase ของ Transaction Service และผลการทดสอบ

4.3.2. รายการ Testcase และผลการทดสอบของ HiTrust Integration

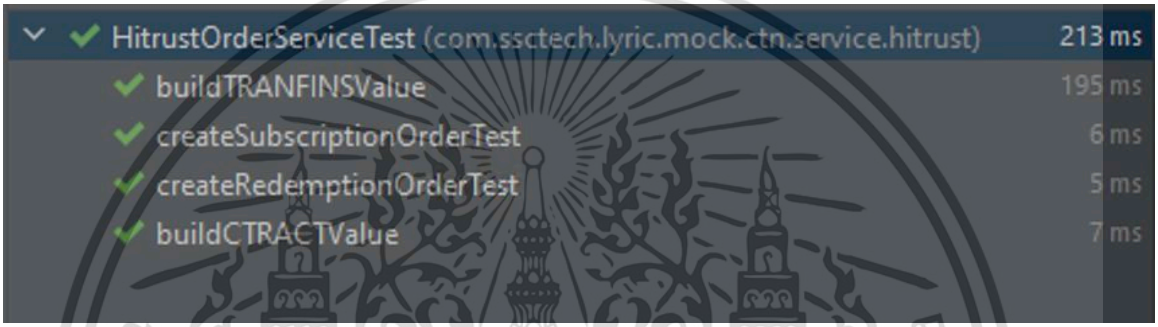
Mock HiTrust Integration ได้ทำการทดสอบแบบ Unit Testing โดยใช้ Junit ในการทดสอบ มีรายการ Testcase และผลดังนี้



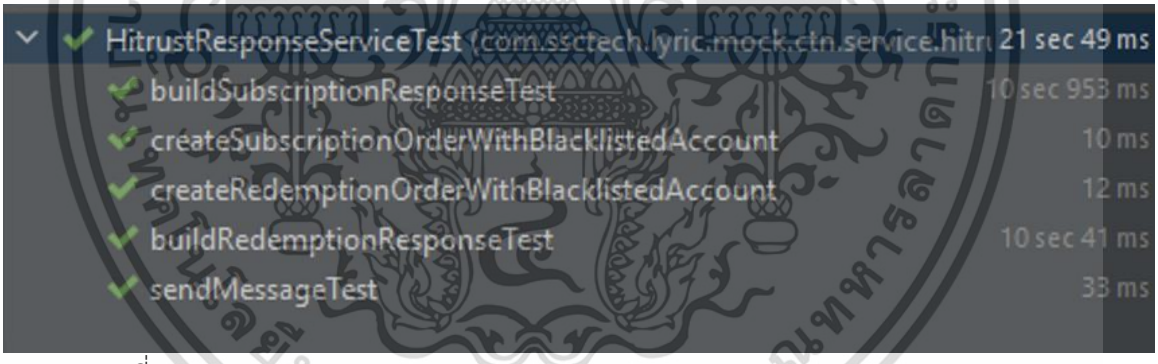
รูปที่ 4.42 ภาพแสดง Testcase ของ HiTrust Order Controller และผลการทดสอบ



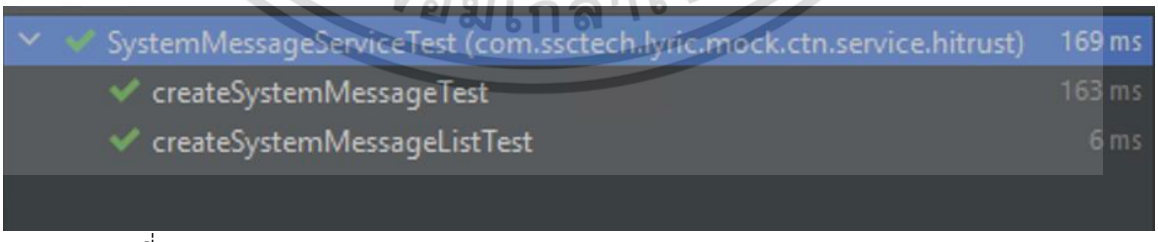
รูปที่ 4.43 ภาพแสดง Testcase ของ Blacklist Service และผลการทดสอบ



รูปที่ 4.44 ภาพแสดง Testcase ของ HiTrust Order Service และผลการทดสอบ

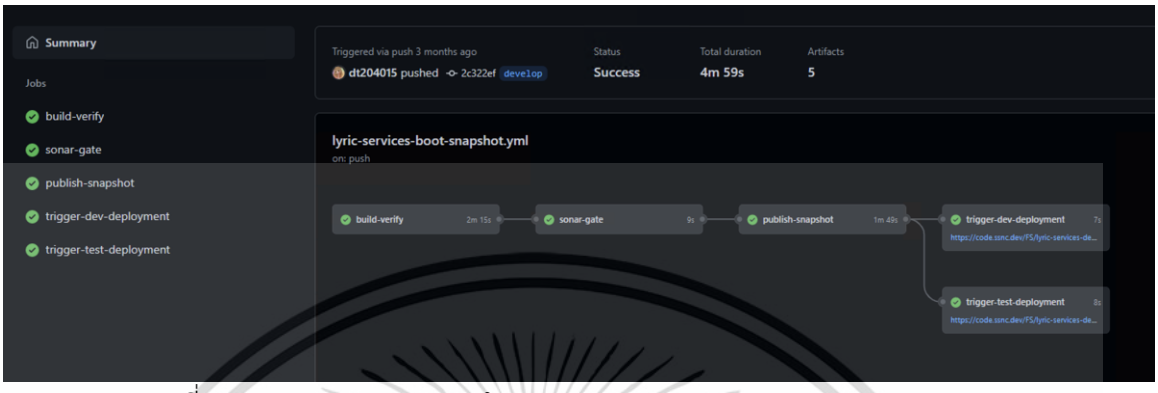


รูปที่ 4.45 ภาพแสดง Testcase ของ HiTrust Response Service และผลการทดสอบ



รูปที่ 4.46 ภาพแสดง Testcase ของ System Message Service และผลการทดสอบ

4.4 การทำ CI/CD



รูปที่ 4.47 ภาพแสดงของการทำงานของ Workflow Spring Boot Snapshot

รูปที่ 4.48 ภาพแสดงของการทำงานของ Workflow Java CodeQL Analysis

หลังจากผ่านการ Development, Automation Test และ Code Review เรียบร้อยแล้ว ก็จะทำการ Deploy Mock Calastone Service ขึ้นสู่ DEV Platform โดยการทำ CI / CD ผ่าน GitHub Actions โดยในขั้นตอนการทำ CI นั้น ได้มีการใช้งาน Workflow 2 ตัว ได้แก่ Workflow Spring Boot Snapshot และ Java CodeQL Analysis เมื่อกระบวนการ CI ผ่านแล้ว การทำ CD ก็จะเกิดขึ้นทันที หรือก็คือ Service ก็จะถูก Deploy ขึ้น Platform หลังจากที่การทำ CI เสร็จสมบูรณ์

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

ในระยะเวลา 6 เดือนที่ผู้จัดทำได้มีโอกาสเข้าร่วมโครงการสหกิจศึกษากับทางบริษัท เอสเอส แอนด์ซี เทคโนโลยี จำกัด เพื่อพัฒนาไมโครเซอร์วิสจำลองการทำงานของ Calastone API และ HiTrust โดยมีจุดประสงค์เพื่อลดค่าใช้จ่ายจำนวนมากในการนำ Calastone อันเป็นระบบภายนอกเข้ามาทดลองใช้งานจริงกับ Intelligence Service และ Event Messaging Service อีกทั้งยังแก้ปัญหาที่ไม่สามารถนำ HiTrust ซึ่งเป็นระบบที่ไม่สามารถใช้งานได้บน US Environment เข้ามาใช้ในการทดสอบอีกด้วย โดยกระบวนการทำงานจะเริ่มต้นขึ้นเมื่อ Intelligence Service ได้รับข้อมูล Order จาก Calastone จากนั้น Intelligence Service จะนำ Order ที่ได้ไปประมวลผลและส่งต่อข้อมูลไปยัง HiTrust เมื่อ HiTrust ได้รับข้อมูลจาก Intelligence Service ก็จะทำการประมวลผลและส่ง Kafka Message ไปยัง Kafka Topic ที่ Event Messaging Service เชื่อมต่ออยู่ และเมื่อครบกระบวนการทำงาน Intelligence Service ก็จะส่งข้อมูลเพื่ออัปเดตสถานะ Order ที่ Calastone ส่งมาตั้งแต่ขั้นแรก เพื่อเติมเต็มช่องว่างที่หายไปในการทดสอบเมื่อไม่มีระบบจริง Mock Calastone และ HiTrust Integration จึงเข้ามาแก้ปัญหาในส่วนนั้นและจำลองกระบวนการทำงานที่จำเป็น โดยในการพัฒนานั้นจะพัฒนาบน Spring Boot Framework ซึ่งเป็น Framework ที่ช่วยในการสร้าง Web Service มีการนำ H2 Database ซึ่งเป็น In-Memory Database เข้ามาสร้างฐานข้อมูลสำหรับใช้งานในระบบ รวมไปถึงการนำ Kafka เข้ามาใช้ในการรับและส่งข้อมูลระหว่าง Service ด้วย ในส่วนของการทำ Automation Testing นั้นได้ทำการทดสอบแบบ Unit Testing โดยใช้ Junit ในการทดสอบ นอกจากนี้ยังได้เรียนรู้กระบวนการ Deploy ชิ้นงานขึ้นสู่ Platform โดยอาศัยหลักการ CI/CD อีกด้วย

5.2 ข้อเสนอแนะ

ในการทำงานมักจะพบปัญหาเรื่องความรู้และประสบการณ์ที่ไม่เพียงพอ รวมถึงความไม่คุ้นชินกับเทคโนโลยีที่ไม่เคยใช้ ทำให้ต้องใช้เวลานานในการค้นคว้าและทำความเข้าใจ อีกทั้งยังขาดความรอบคอบในการทำงานหลาย ๆ ครั้ง จึงควรหมั่นทำความเข้าใจ ศึกษาหาความรู้กับเทคโนโลยีใหม่ ๆ ให้ละเอียดรวมไปถึงฝึกฝนและระมัดระวังการทำงาน ตรวจเช็คงานอย่างละเอียดรอบคอบ เพื่อเพิ่มประสิทธิภาพในการทำงานให้มากขึ้น

เอกสารอ้างอิง

[1] Spring Boot Team.**Spring Boot Reference Documentation**.[ออนไลน์].แหล่งที่มา : https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle.สืบค้นเมื่อ 8 ตุลาคม 2022

[2] Saritrat Jirakulphondchai..**[Spring Ep.01] Why Spring ?.**[ออนไลน์].แหล่งที่มา : https://saritrat.medium.com/spring-ep-01-why-spring-5e2bcc0d9dfa.สืบค้นเมื่อ 8 ตุลาคม 2022

[3] Teerawat Amornrattanakij.**Spring Boot มีไว้ทำอะไร?**.[ออนไลน์].แหล่งที่มา : https://medium.com/@Teerawat.amo/spring-boot-%E0%B8%A1%E0%B8%B5%E0%B9%84%E0%B8%A7%E0%B9%89%E0%B8%97%E0%B8%B3%E0%B8%AD%E0%B8%B0%E0%B9%84%E0%B8%A3-c1d84a7796d7. สืบค้นเมื่อ 8 ตุลาคม 2022

[4] TechTarget Contributor.**Spring Framework**.[ออนไลน์].แหล่งที่มา : https://www.techtarget.com/searchapparchitecture/definition/Spring-Framework. สืบค้นเมื่อ 8 ตุลาคม 2022

[5] **Kafka Documentation**.[ออนไลน์].แหล่งที่มา : https://kafka.apache.org/documentation/.สืบค้นเมื่อ 8 ตุลาคม 2022,

[6] Neng Liangpornrattana.**Apache Kafka ฉบับผู้เริ่มต้น #1: Hello Apache Kafka**. [ออนไลน์].แหล่งที่มา : https://medium.com/linedevth/apache-kafka-%E0%B8%89%E0%B8%9A%E0%B8%B1%E0%B8%9A%E0%B8%9C%E0%B8%B9%E0%B9%89%E0%B9%80%E0%B8%A3%E0%B8%B4%E0%B9%88%E0%B8%A1%E0%B8%95%E0%B9%89%E0%B8%99-1-hello-apache-kafka-242788d4f3c6.สืบค้นเมื่อ 8 ตุลาคม 2022

[7] Baeldung.**Spring Boot With H2 Database**.[ออนไลน์].แหล่งที่มา : https://www.baeldung.com/spring-boot-h2-database.สืบค้นเมื่อ 8 ตุลาคม 2022

[8] AmiyaRanjanRout.**Spring Boot – H2 Database**.[ออนไลน์].แหล่งที่มา : https://www.geeksforgeeks.org/spring-boot-h2-database.สืบค้นเมื่อ 8 ตุลาคม 2022

[9] **What is Postman?**.[ออนไลน์].แหล่งที่มา : https://www.postman.com/product/what-is-postman/.สืบค้นเมื่อ 8 ตุลาคม 2022

[10] Parikshit Hooda.(2022).**Introduction to Postman for API Development**.[ออนไลน์]. แหล่งที่มา : https://www.geeksforgeeks.org/introduction-postman-api-development/.สืบค้นเมื่อ 8 ตุลาคม 2022

[11] Shinji Kanai.**JUnit: A Complete Guide**.[ออนไลน์].แหล่งที่มา : https://www.headspin.io/blog/junit-a-complete-guide.สืบค้นเมื่อ 8 ตุลาคม 2022

[12] **What is IntelliJ IDEA?**.[ออนไลน์].แหล่งที่มา : https://www.jetbrains.com/idea/features/#:~:text=What%20is%20IntelliJ%20IDEA%3F,on%20all%20sorts%20of%20applications.สืบค้นเมื่อ 8 ตุลาคม 2022,

[13] **git - the stupid content tracker**.[ออนไลน์].แหล่งที่มา : https://git-scm.com/docs/git. สืบค้นเมื่อ 8 ตุลาคม 2022

[14] **Git พื้นฐานสุด ๆ จบในหน้าเดียว**.[ออนไลน์].แหล่งที่มา : https://www.borntodev.com/2020/03/30/git-%E0%B8%9E%E0%B8%B7%E0%B9%89%E0%B8%99%E0%B8%90%E0%B8%B2%E0%B8%99%E0%B8%AA%E0%B8%B8%E0%B8%94%E0%B9%86/.สืบค้นเมื่อ 8 ตุลาคม 2022

[15] Chris Richardson.**Microservices คืออะไร ใช้ยังไง ?**.[ออนไลน์].แหล่งที่มา : https://aws.amazon.com/microservices/#:~:text=Microservices%20are%20an%20architectural%20and,small%2C%20self%2Dcontained%20teams.สืบค้นเมื่อ 8 ตุลาคม 2022

[16] **Microservices**.[ออนไลน์].แหล่งที่มา : https://www.borntodev.com/2020/05/22/microservices-%E0%B8%84%E0%B8%B7%E0%B8%AD%E0%B8%AD%E0%B8%B0%E0%B9%84%E0%B8%A3/.สืบค้นเมื่อ 8 ตุลาคม 2022

[17] **Microservices**.[ออนไลน์].แหล่งที่มา : https://www.borntodev.com/2020/05/22/microservices-%E0%B8%84%E0%B8%B7%E0%B8%AD%E0%B8%AD%E0%B8%B0%E0%B9%84%E0%B8%A3/.สืบค้นเมื่อ 8 ตุลาคม 2022

[18] **What Is A RESTful API?**.[ออนไลน์].แหล่งที่มา : https://aws.amazon.com/what-is/restful-api/?nc1=h_ls.สืบค้นเมื่อ 8 ตุลาคม 2022

[19] Sakul Montha.**REST กับ RESTful API ต่างกันนะรู้ยัง**.[ออนไลน์].แหล่งที่มา : https://iamgique.medium.com/restful-api-%E0%B8%81%E0%B8%B1%E0%B8%9A-rest-api-%E0%B8%95%E0%B9%88%E0%B8%B2%E0%B8%87%E0%B8%81%E0%B8%B1%E0%B8%99%E0%B8%99%E0%B8%B0%E0%B8%A3%E0%B8%B9%E0%B9%89%E0%B8%A2%E0%B8%B1%E0%B8%87-2c70c42990e3.สืบค้นเมื่อ 8 ตุลาคม 2022

[20] **HTTP response status codes**.[ออนไลน์].แหล่งที่มา : https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.สืบค้นเมื่อ 8 ตุลาคม 2022

[21] Paul Kirvan.**multithreading**.สืบค้นเมื่อ 8 ตุลาคม 2022,จาก https://www.mindphp.com/forums/viewtopic.php?t=36677

[22] **Multithreading in Java – What is Java Multithreading?**.[ออนไลน์].แหล่งที่มา : https://www.mygreatlearning.com/blog/multithreading-in-java/#:~:text=In%20Java%2C%20Multithreading%20refers%20to,and%20share%20the%20process%20resources.สืบค้นเมื่อ 8 ตุลาคม 2022

[23] TechTarget Contributor.**Unit Testing**.[ออนไลน์].แหล่งที่มา : https://www.techtarget.com/searchsoftwarequality/definition/unit-testing.สืบค้นเมื่อ 8 ตุลาคม 2022

[24] pp_pankaj.**Unit Testing | Software Testing**.[ออนไลน์].แหล่งที่มา : geeksforgeeks.org/unit-testing-software-testing.สืบค้นเมื่อ 8 ตุลาคม 2022

[25] Suppawat K.**CI/CD คืออะไร? ช่วยให้ Developer ทำงานง่ายขึ้นได้มากขนาดไหน?**.[ออนไลน์].แหล่งที่มา : https://about.gitlab.com/topics/ci-cd.สืบค้นเมื่อ 8 ตุลาคม 2022

[26] **What is CI/CD?.[**ออนไลน์].แหล่งที่มา : https://blog.cloudhm.co.th/ci-cd/.สืบค้นเมื่อ 8 ตุลาคม 2022,จาก

ภาคผนวก

ภาคผนวก ก
การติดตั้ง IntelliJ IDEA

ขั้นตอนการดาวน์โหลดและติดตั้ง IntelliJ IDEA

1. เข้าไปที่ Website https://www.jetbrains.com/idea/download จากนั้นดาวน์โหลดตัวติดตั้งโปรแกรม IntelliJ IDEA โดยเลือกเป็น Community Edition

รูปที่ 5.1 หน้าดาวน์โหลด IntelliJ IDEA

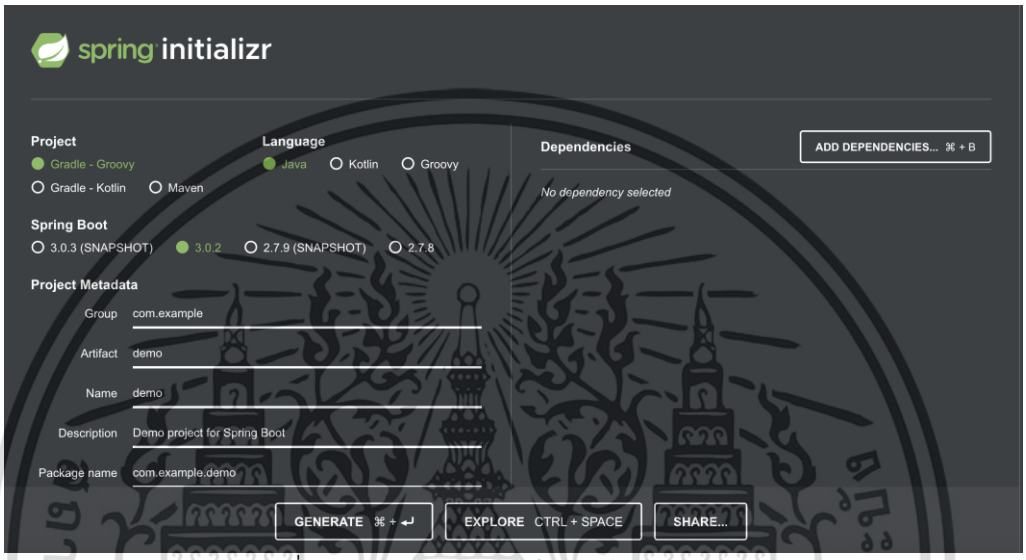
2. Double-click เข้าไปที่ไฟล์ IntelliJ IDEA เพื่อเริ่มการติดตั้ง และกดปุ่ม Next เพื่อดำเนินการต่อ
3. เลือกพื้นที่สำหรับติดตั้ง IntelliJ IDEA จากนั้นกด Next เพื่อดำเนินการต่อ
4. กดปุ่ม Install และรอจนกว่า IntelliJ IDEA ติดตั้งสำเร็จ
5. เมื่อ IntelliJ IDEA ติดตั้งสำเร็จ โปรแกรมจะพร้อมใช้งาน

ภาคผนวก ข

การสร้าง Spring Project ด้วย Spring Initializer

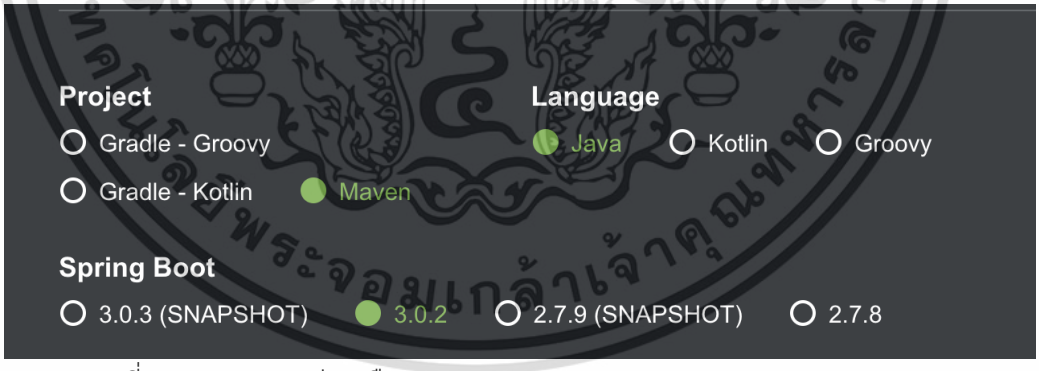
ขั้นตอนการสร้าง Spring Project ด้วย Spring Initializer

1. เข้าไปที่เว็บไซต์ https://start.spring.io/



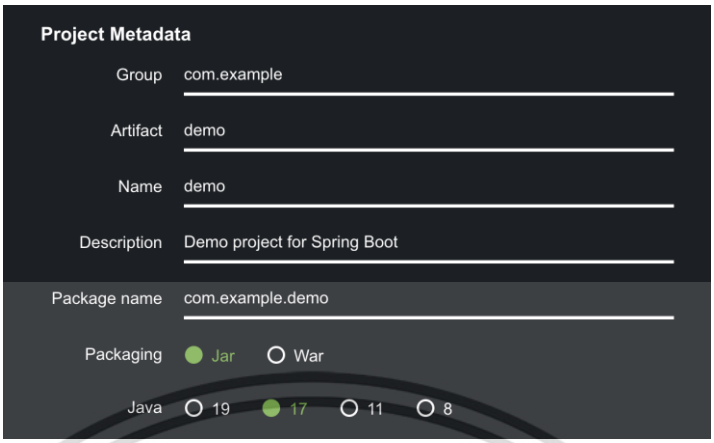
รูปที่ 5.2 ภาพแสดงหน้าเว็บ Spring Initialiizer

2. เลือกพื้นที่สำหรับติดตั้ง IntelliJ IDEA จากนั้นกด Next เพื่อดำเนินการต่อ



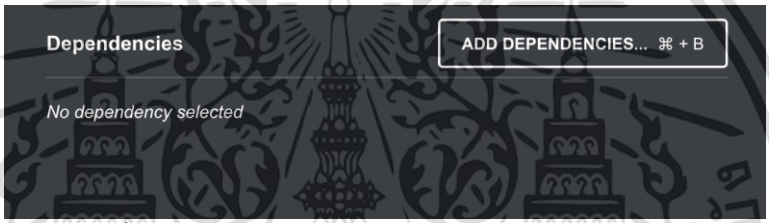
รูปที่ 5.3 ภาพแสดงส่วนเลือก ภาษาและ Spring Boot Version ของ Project

3. กำหนด Metadata ของ Project และเลือก Version ของ Java



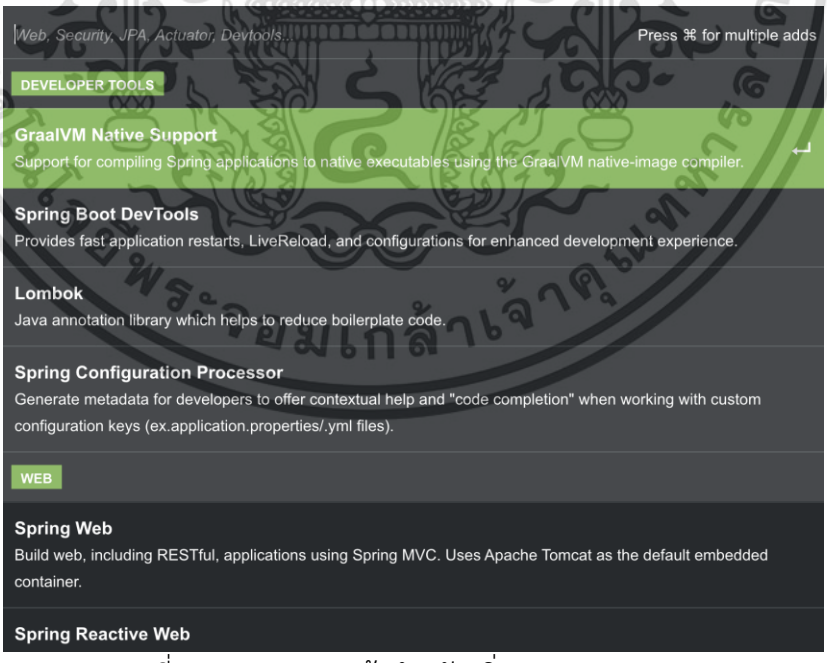
รูปที่ 5.4 ภาพแสดงส่วนกำหนด Metadata และเลือก Java Version ของ Project

4. กด Add Dependencies เพื่อเปิดหน้าต่างสำหรับเพิ่ม Dependency ที่ต้องการใช้ใน Project



รูปที่ 5.5 ภาพแสดงส่วน Dependencies ของ Project

5. เลือก Dependency ต้องการ



รูปที่ 5.6 ภาพแสดงหน้าสำหรับเพิ่ม Dependencies

6. เมื่อตั้งค่าทุกอย่างเสร็จแล้ว กดปุ่ม Generate เพื่อสร้าง Spring Project

งานทะเบียนคณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

คำรับรองเล่มโครงงานพิเศษ/ปัญหาพิเศษ/สหกิจศึกษา

วันที่ 26 เดือน มิถุนายน พ.ศ 2566

ข้าพเจ้า นางสาว ณัฐฑริกา ทองแถม รหัสประจำตัว 62050154

..... รหัสประจำตัว .....

นักศึกษาหลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชา วิทยาการคอมพิวเตอร์ ภาควิชา วิทยาการคอมพิวเตอร์

ขอรับรองว่า ปัญหาพิเศษ เรื่อง

ชื่อภาษาไทย ไมโครเซอร์วิสจำลองการทำงานของ Calastone API และ HiTrust Integration

ชื่อภาษาอังกฤษ MOCK CALASTONE SERVICE AND HITRUST INTEGRATION

ปีการศึกษา 2565

เป็นผลงานวิจัยที่มิได้คัดลอกหรือละเมิดลิขสิทธิ์ของผู้อื่นและได้ผ่านการตรวจสอบความซ้ำซ้อน เรียบร้อยแล้ว และได้แนบเอกสารการตรวจสอบการลอกเลียนงานวรรณกรรมที่ตรวจสอบจากเล่มโครงงานพิเศษ/ปัญหาพิเศษ/สหกิจศึกษาฉบับสมบูรณ์แล้ว

โปรแกรมอักขราวิสุทธิ์.....% หรือโปรแกรม Turnitin 12 %

ลงชื่อ ณัฐฑริกา ทองแถม ลงชื่อ.....

(นางสาว ณัฐฑริกา ทองแถม) (.....)

นักศึกษา นักศึกษา

ข้าพเจ้า ดร.วิชญะ ต่อวงศ์ไพชยนต์ อาจารย์ที่ปรึกษาปัญหาพิเศษ ได้ตรวจสอบปัญหาพิเศษของนักศึกษาข้างต้นแล้ว ขอรับรองว่าเป็นผลงานวิจัยของนักศึกษาจริงและมีเนื้อหาสมบูรณ์ จึงลงชื่อไว้เป็นหลักฐาน

ลงชื่อ.....

อาจารย์ที่ปรึกษา

คำรับรองเล่มสหกิจศึกษาโดยสถานประกอบการ

วันที่ 26 เดือน มิ.ย. พ.ศ. 2566

ข้าพเจ้า Wiphaphorn Phramanee ตำแหน่ง Dir Software Engineering

เป็นตัวแทนของสถานประกอบการ SS&C Technologies, Inc

ขอรับรองว่า ทางสถานประกอบการได้ตรวจสอบเล่มสหกิจศึกษา

เรื่อง ไมโครเซอร์วิสจำลองการทำงานของ Calastone API และ HiTrust Integration ไมโครเซอร์วิสจำลองการทำงานของ Calastone API และ HiTrust Integration ของนักศึกษาชื่อ ณัฐฑริกา ทองแถม ซึ่งเป็นนักศึกษาภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังเรียบร้อยแล้ว และไม่มีส่วนหนึ่งส่วนใดในเล่มสหกิจศึกษานี้ที่มีข้อมูลอ่อนไหว และ/หรือ ข้อมูลอันเป็นความลับอันจะก่อให้เกิดความเสียหายต่อสถานประกอบการ รวมทั้งอนุญาตให้สามารถเผยแพร่ต่อสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังได้ จึงลงชื่อไว้เป็นหลักฐาน

ลงชื่อ .....

(Wiphaphorn Phramanee)

ตัวแทนสถานประกอบการ

ข้าพเจ้า ดร.วิชญะ ต่อวงศ์ไพชยนต์ อาจารย์ที่ปรึกษาสหกิจศึกษา ได้ตรวจสอบเล่มสหกิจศึกษาแล้วและรับทราบว่าสถานประกอบการดำเนินการตรวจสอบเล่มสหกิจศึกษาแล้ว จึงลงชื่อไว้เป็นหลักฐาน

ลงชื่อ .....

(ดร.วิชญะ ต่อวงศ์ไพชยนต์)

อาจารย์ที่ปรึกษา