รายงานสหกิจศึกษาฉบับสมบูรณ์

ไมโครเซอร์วิสในระบบการเติมเงินออนไลน์สำหรับบัตรรถไฟฟ้าใต้ดิน

Pending Top-up Inquiry Request and Transaction Status Inquiry Request for Online Payment Solution System

นาย สุกฤษฎิ์ คุ้มครอง

ภาควิชาวิศวกรรมคอมพิวเตอร์ สาขาวิศวกรรมสารสนเทศ

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2562

ชื่อโครงงานสหกิจศึกษา ไมโครเซอร์วิสในระบบการเติมเงินออนไลน์สำหรับบัตรรถไฟฟ้าใต้ดิน
ชื่อ-สกุล นักศึกษา นายสุกฤษฎิ์ คุ้มครอง
คณะ วิศวกรรมศาสตร์ **ภาควิชา** วิศวกรรมคอมพิวเตอร์ **สาขาวิชา** วิศวกรรมสารสนเทศ
ชื่อ-สกุล อาจารย์นิเทศ ผศ.ดร.เกล็ดดาว สัตย์เจริญ
ชื่อ-สกุล ผู้นิเทศงาน นาย สุเมธัท อัศวประภา
สถานประกอบการ บริษัท สตรีม ไอ.ที.คอนซัลติ้ง จำกัด

บทคัดย่อ

ปัจจุบันการเดินทางในชีวิตประจำวันนั้นมีตัวเลือกที่ค่อนข้างมาก ซึ่งถ้ามองโดยปกติแล้วนั้น รถไฟฟ้าก็คงเป็นตัวเลือกหลักของคนกรุงเทพมหานครฯ แต่ด้วยสถิติแล้วมีคนใช้งานรถไฟฟ้าในการโดยสารภายในบริเวณกรุงเทพฯ และปริมณฑลเพียงแค่ร้อยละ 12.5 เท่านั้น เนื่องจากเส้นทางรถไฟฟ้าครอบคลุมพื้นที่เพียงเล็กน้อย แต่ด้วยการวางแผนในอนาคตที่จะมีการเพิ่มสายการให้บริการของรถไฟฟ้าที่ครอบคลุมมากขึ้นจึงมีการคาดการณ์ว่าผู้ใช้งานก็จะมีจำนวนมากขึ้นไปด้วย จึงได้ต้องการจัดทำระบบการเติมเงินในช่องทางออนไลน์ขึ้นมา เพื่อลดจำนวนงานของพนักงานทั้งในปัจจุบันและอนาคต ทางบริษัท สตรีม ไอ.ที.คอนซัลติ้ง จำกัด จึงได้รับการว่าจ้างในการทำระบบที่ใช้ในการจ่ายเงินออนไลน์ให้แก่ทางรถไฟฟ้าใต้ดิน ด้วยเหตุนี้จึงทำให้ผู้วิจัยที่ร่วมทำโครงงานสหกิจศึกษาร่วมกับทางบริษัท สตรีม ไอ.ที.คอนซัลติ้ง จำกัด นั้นได้รับหน้าที่ในการทำไมโครเซอร์วิสหลักจำนวนสองตัวจากทั้งแปดตัว โดยการจัดทำนั้นจะเป็นไปตามข้อตกลงที่เห็นชอบจากทุก ๆ ฝ่ายที่มีส่วนเกี่ยวข้องในการทำโครงการนี้เพื่อให้ระบบสามารถตอบสนองความต้องการในใช้งานให้กับทุกฝ่ายอย่างเท่าเทียมและมีประสิทธิภาพสูงสุด

คำสำคัญ: ระบบที่ใช้ในการจ่ายเงินออนไลน์, ไมโครเซอร์วิส

Cooperative Title: Pending Top-up Inquiry Request and Transaction Status Inquiry

Request for Online Payment Solution System

Student intern name: Mr. Sukrit Kumkrong

Faculty: Engineering **Department:** Computer Engineering (Information Engineering)

Advisor name: Asst.Prof.Dr.Kleddao Satcharoen

Mentor name: Mr.Sumethas Asavaprapha

Company: Stream I.T. Consulting Ltd.

ABSTRACT

Nowadays, travelling in normal life has quite many options. Which, if viewed normally, the skytrain would be the main choice for people who live in Bangkok or nearby. But with the statistics, there are people using skytrain in travelling within Bangkok and the parameter only 12.5% because the skytrain route covers only a small area. In the future there will be more lines of skytrain services that are more comprehensive, there are also predictions that the number of users will increase too. Thus, Stream I.T. Consulting Company is therefore hired to make Online Payment Solution System (OPSS) for online top-up system, for this reason the researcher was responsible for providing two of eight main microservices. The preparation will be in accordance with the agreement agreed by all parties involved in the project for equally and maximum efficiency

Keyword: Online Payment Solution System (OPSS), Microservices

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จขึ้นได้นั้นต้องขอขอบคุณ ดร.เกล็ดดาว สัตย์เจริญ ที่ให้คำปรึกษา ให้ความช่วยเหลือ รวมถึงพี่ ๆ ภายในทีม และนายสุเมธัท อัศวประภา ที่ทำหน้าที่เป็นผู้ดูแล และให้คำแนะนำที่ดี รวมไปถึงผู้บริหารในบริษัท สตรีม ไอ.ที.คอนซัลติ้ง จำกัด ที่ให้คำปรึกษาและแนวคิดต่างๆ ในการจัดทำโครงงานฉบับนี้ รวมทั้งดูแลตรวจสอบความถูกต้องจนกระทั่งสำเร็จเป็นปริญญาฉบับนี้ขึ้น

ขอขอบพระคุณอาจารย์ทุกท่าน ที่ได้ให้ความรู้และคำแนะนำที่มีคุณค่ามากมายมาตลอดทุกเทอมการศึกษา รวมถึงเพื่อนๆ พี่ๆ น้องๆ ที่คอยช่วยเหลือ ให้คำปรึกษา และเป็นกำลังใจให้

ท้ายที่สุด ผู้ทำวิจัยขอขอบพระคุณ บิดา มารดา บุคคลที่มีความสำคัญที่สุดที่คอยให้การสนับสนุนในทุกด้านและคอยให้กำลังใจตลอดมา ทางผู้ทำวิจัยหวังว่าโครงงานฉบับนี้จะก่อให้เกิดประโยชน์กับผู้ที่สนใจสูงสุด

สุกฤษฎิ์ คุ้มครอง

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ หน้า

สารบัญรูป

รูปที่ หน้า

สารบัญรูป (ต่อ)

รูปที่ หน้า

สารบัญรูป (ต่อ)

รูปที่ หน้า

สารบัญรูป (ต่อ)

รูปที่ หน้า

สารบัญรูป (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 แนวคิดและที่มาของโครงงาน

การเดินทางสัญจรในเมืองหลวงอย่างกรุงเทพมหานครฯ มีปัญหาเรื่องการจราจรติดขัดมานาน โดยเฉพาะในช่วงเวลาเร่งด่วนในแต่ละวัน ซึ่งแต่ละคนก็จะมีการจัดรูปแบบการใช้ชีวิตและรูปแบบการเดินทางที่แตกต่างออกไปตามกิจกรรมที่ต้องทำในวันนั้น ๆ ส่วนมากจะเป็นการออกไปเรียนหรือการไปทำงานซะส่วนใหญ่ ซึ่งทำให้รูปแบบการเดินทางมีผลเป็นอย่างมากต่อการดำเนินชีวิต เนื่องจากทุกคนก็ต้องรีบไปให้ทันเวลาเรียนหรือเวลาทำงานของตัวเอง ซึ่งเห็นได้ชัดว่าตัวเลือกรูปแบบการเดินทางในกรุงเทพมหานครฯ ณ ปัจจุบันมีหลากหลายช่องทางให้เลือกใช้ ยกตัวอย่างเช่น รถเมล์ รถไฟ รถไฟฟ้า เรือ รวมไปถึงรถส่วนตัวและอื่น ๆ อีกมากมายที่ไม่ได้กล่าวถึงในข้างต้น ถ้ากล่าวถึงวิธีการเดินทางในระยะไกลที่รวดเร็วและสะดวกสบายภายในบริเวณกรุงเทพฯและปริมณฑลก็คงจะเป็นการเดินทางด้วยรถไฟฟ้า

รถไฟฟ้าที่พบเห็นในกรุงเทพฯ และปริมณฑลจะมีทั้งรถไฟฟ้าบนดินและรถไฟฟ้าใต้ดินไม่ว่าจะเป็น Bangkok Transit System (BTS) Metropolitan Rapid Transit (MRT) หรือ Airport Rail Link (ARL) สำหรับคนกรุงเทพมหานครฯ ที่ต้องการเข้าไปทำงานในใจกลางเมืองให้ทันเวลาหรือกลับบ้านในช่วงเวลาเลิกงานก็จะเลือกใช้บริการรถไฟฟ้า หรือ จะเลือกการใช้รถส่วนตัวในการเดินทางรวมถึงรถโดยสารต่าง ๆ ที่ขับขี่บนท้องถนน แต่เนื่องจากผู้คนต้องการหนีปัญหาจราจรติดขัดบนท้องถนนในช่วงเวลาเร่งด่วน หรือจะเป็นสาเหตุมาจากบริษัทที่จอดรถไม่พอต่อพนักงานทุกคนเนื่องจากพื้นที่ใจกลางเมืองค่อนข้างราคาสูงบางบริษัทจึงจัดหาที่จอดรถให้ได้ไม่เพียงพอกับพนักงานทุกคน ด้วยเหตุผลข้างต้นอาจจะดูเหมือนว่าผู้คนจำนวนมากก็คงเลือกใช้บริการรถไฟฟ้าเป็นซะส่วนใหญ่และทำให้ดูเหมือนว่ารถไฟฟ้าเป็นทางเลือกที่คนกรุงเทพมหานครฯ เลือกใช้ในการเดินทางเป็นหลัก ประกอบกับการเห็นจำนวนคนพลุกพล่านในช่วงเวลาเร่งรีบทั้งในช่วงเวลาเช้าและเย็นมีสภาพที่แออัดจนไม่มีพื้นที่ยืน แต่ในความเป็นจริงแล้วคนกรุงเทพฯ และปริมณฑลมีการใช้รถไฟฟ้าสำหรับการเดินทางเพียงแค่ 12.5% เท่านั้น แต่มีการใช้รถโดยสารประเภทอื่น ๆ ถึง 81% นั้นเป็นเหตุมาจากการที่รถไฟฟ้าครอบคลุมพื้นที่ของกรุงเทพฯและปริมณฑลเพียงแค่ 10% ของพื้นที่ทั้งหมดเท่านั้น แต่อย่างที่ทราบหรืออาจจะมีการพบเห็นจะสังเกตุว่าได้มีโครงการสร้างสายการเดินทางรถไฟฟ้าเพิ่มขึ้นรอบกรุงเทพฯและปริมณฑล เนื่องด้วยมี

ความต้องการขยายพื้นที่บริการให้ครอบคลุมพื้นมากขึ้น การขยายสายการเดินรถไฟฟ้าครั้งนี้จะทำให้สายการเดินรถไฟฟ้าในกรุงเทพฯและปริมณฑลของผู้ให้บริการมทุกรายมีการครอบคลุมพื้นรวมที่รวมกันถึง 45-47% จึงทำให้มีการคาดการว่าในปี พ.ศ.2580 คนกรุงเทพฯและปริมณฑลจะหันมาเดินทางด้วยรถไฟฟ้ามีจำนวนสูงถึง 25% เลยทีเดียว คงจะแย่ไม่น้อยถ้าในอนาคตข้างหน้ามีผู้ใช้บริการมากขึ้นแต่แถวของลูกค้าที่รอใช้บริการหน้าเคาท์เตอร์ยังมีความยาวที่มากเหมือนกับทุกวันนี้

การเติมเงินของบัตรรถไฟฟ้า ในปัจจุบันจะสามารถเติมได้ที่หน้าเคาท์เตอร์ของแต่ละสถานีนั้น ๆ หรือ ถ้าเป็นบัตรสายสีม่วงจะสามารถเติมจากตู้อัตโนมัติที่สถานีของสายสีม่วงได้ เนื่องจากโครงสร้างการเติมเงินดังกล่าวทำให้มีปัญหาเกี่ยวกับการต่อแถวรอรับบริการบริเวณหน้าเคาท์เตอร์ที่มีจำนวนผู้คนต่อแถวค่อนข้างมากในช่วงเวลาเร่งด่วนเนื่องจากจะมีกลุ่มที่ต่อแถวซื้อบัตรโดยสารทั่วไปกับกลุ่มที่ต้องการเติมเงินจึงทำให้จำนวนผู้คนรอใช้บริการของผู้ใช้งานมีจำนวนที่มากโดยไม่จำเป็น เพราะฉะนั้นเป็นเหตุให้ทางบริษัทได้คิดโปรเจคการเติมเงินผ่านช่องทางออนไลน์หรือผ่านทางโทรศัพท์มือถือขึ้นมาเพื่อความสะดวกและรวดเร็วของลูกค้าและลดจำนวนลูกค้าที่รอใช้บริการการเติมเงินที่หน้าเคาท์เตอร์ และยังช่วยให้ทางพนักงานโดยที่ไม่ต้องทำงานหนักเกินความจำเป็น ทั้งนี้ก็เพื่อให้เกิดประโยชน์สูงสุดทั้งผู้ให้บริการและผู้ใช้บริการ โดยโปรเจคนี้จะเป็นการจัดทำ web service กลางระหว่าง payment provider หรือ operation support ให้มีข้อมูลที่สอดคล้องตรงกันกับทาง Online Payment System Solution (OPSS) server เพื่อนำข้อมูลดังกล่าวไปใช้ในขั้นตอนการเติมเงินแบบออนไลน์

1.2 วัตถุประสงค์ของโครงงาน

1.2.1 สามารถดูข้อมูลของเลขทำรายการและรายละเอียดต่าง ๆ ตามรูปแบบที่ตกลงกันไว้ได้

1.2.2 เพื่อให้หน้าเคาท์เตอร์สามารถตรวจเช็คสถานะของเลขทำรายการนั้น ๆ

1.2.3 บ่งบอกสถานะของบัตรที่ได้รับเข้ามาว่าสามารถเติมเงินได้หรือไม่

1.2.4 เพื่อให้เว็บเซอร์วิสที่เป็นเซอร์วิสเกี่ยวกับการทำรายการเติมเงิน เรียกใช้เพื่อเช็คสถานะของบัตรก่อนจะอนุมัติการทำรายการที่ได้รับมา

1.3 แนวคิดที่ใช้ในการออกแบบ

1.3.1 ตัวเซอร์วิสออกแบบให้เป็นไมโครเซอร์วิสที่สามารถทำงานได้ด้วยตัวเอง (stand alone) เพื่อที่จะสามารถทำงานต่อไปหากเซอร์วิสตัวอื่นเกิดมีปัญหาขัดข้อง

1.3.2 ใช้ Netflix Eureka ทำงานเป็น service discovery เพื่อใช้เป็นตัวสำหรับบ่งบอกถึงเซอร์วิสต่าง ๆ ในระบบ เมื่อมีการขอ request เข้ามา

1.3.3 เซอร์วิสมีความถูกต้องในการส่งข้อมูลกลับ

1.3.4 เซอร์วิสส่งรูปแบบโค้ดที่ได้ตกลงกันไว้ได้อย่างถูกต้อง

1.3.5 แสดงข้อมูลที่เกี่ยวข้อง (log) เมื่อมีการเรียกใช้งาน

1.3.6 มีการทำระบบติดตามการเดินทางของเซอร์วิส (tracing)

1.3.7 ทำการทดสอบตัวฟังก์ชันทุกตัวว่าได้ผลตามถูกต้องตามที่คาดการไว้หรือไม่ (unit test)

1.4 ขอบเขตของโครงงาน

โครงงานนี้จะเป็นการจัดทำเว็บเซอร์วิสสองเซอร์วิสในระบบของการจ่ายเงินออนไลน์โดยมีเซอร์วิสแรกคือ Transaction Status Inquiry โดยเซอร์วิสตัวนี้จะทำหน้าที่สำหรับให้ทางผู้ให้บริการด้านการเติมเงิน (Payment Provider Server) เรียกดูข้อมูลของเลขทำรายการที่ได้ทำเข้ามาว่ามีข้อมูลเป็นอย่างไรและสถานะของเลขการทำรายการนี้เป็นยังไงบ้างและอีกเซอร์วิสนึงก็คือ Pending Top-up Inquiry โดยเว็บเซอร์วิสนี้มีไว้สำหรับการเช็คสถานะของบัตร ว่าบัตรหมายเลขนี้ถูกเติมเงินมาจากผู้ให้บริการเติมเงินใดมาก่อนหรือไม่เพราะ จะอนุญาติให้ลูกค้าสามารถเติมเงินได้เพียงทีละครั้งเท่านั้นจากนั้นต้องนำบัตรไปแตะรับเงินที่เครื่องในสถานีก่อนถึงจะอนุญาติให้ลูกค้าเติมครั้งต่อไปได้หรือลูกค้าสามารถยกเลิกการทำรายการเก่าก็สามารถทำการเติมเงินอีกครั้งได้ ซึ่งสองเซอร์วิสนี้จะถูกนำไปใช้จริงในระบบของ Online Payment Solution System (OPSS) ที่จะนำไปใช้เป็นระบบกลางในการเชื่อมต่อระหว่างผู้ให้บริการและผู้บริการเกี่ยวกับการเติมเงิน(Payment Provider) โดยจะรับส่งข้อมูลแบบการร้องขอ (request) และตอบกลับ (response) ในรูปแบบของ JSON โดยตัวเซอร์วิสนั้นจะมีการปรับเปลี่ยนตลอดในระยะเวลาการทำ

1.5 ขั้นตอนการทำโครงงาน

1.5.1 ศึกษาค้นคว้าข้อมูลเกี่ยวกับการทำและพัฒนาเว็บเซอร์วิสให้มีประสิทธิภาพ

1.5.2 ออกแบบการทำงานของฟังก์ชันในแต่ละเซอร์วิส

1.5.3 สร้างและพัฒนาเว็บเซอร์วิสในส่วนที่ได้รับมอบหมายมา

1.5.4 ทดสอบการทำงานของเซอร์วิสต่าง ๆ ว่าได้ผลตามที่คาดหวังไว้หรือไม่

1.5.5 ปรับปรุงแก้ไขในส่วนที่ผิดพลาด

1.6 ประโยชน์ที่คาดว่าจะได้รับ

1.6.1 เรียนรู้การวางระบบสำหรับทำเว็บเซอร์วิสอย่างเป็นแบบแผนและควบคุมได้ง่าย

1.6.2 เรียนรู้การออกแบบฐานเก็บข้อมูลให้ง่ายและมีประสิทธิภาพต่อการพัฒนาเว็บเซอร์วิส

1.6.3 สามารถสร้างและออกแบบระบบของเว็บเซอร์วิสได้ด้วยตนเองในอนาคต

1.6.4 ได้รับความรู้เกี่ยวกับภาษาและเฟรมเวิร์ก (framework) ที่ใช้

1.6.5 ได้รับความรู้ในการใช้เทคโนโลยีต่าง ๆ เพื่อทำให้จัดการพัฒนาเว็บเซอร์วิสได้สะดวกมากยิ่งขึ้น

1.7 อุปกรณ์ที่ใช้ในโครงงาน

1.7.1 โปรแกรมซอฟต์แวร์

1.7.1.1 เครื่องมือที่ช่วยในการพัฒนาโปรแกรม

- IntelliJ

1.7.1.2 เครื่องมือควบคุมจัดการไลท์บรารี่

- maven

1.7.1.3 โปรแกรมระบบจัดการฐานข้อมูล

- MySQL

1.7.1.4 โปรแกรมจำลองสภาพแวดล้อมเพื่อเรียกใช้เซอร์วิสที่ต้องการ

- Docker

1.7.1.5 เครื่องมือสำหรับทดสอบเซอร์วิส

- Post Man

1.7.2 ภาษาและเฟรมเวิร์ก

1.7.2.1 ภาษา

-Java

1.7.2.2 เฟรมเวิร์ก

- Spring Boot
- Hibernate

1.8 เวลาการดำเนินงาน

ตารางที่ 1.1 แผนผังกำหนดการดำเนินงานของงานวิจัย

ID	Task	ส.ค. 2562	ก.ย. 2562	ต.ค. 2562	พ.ย. 2562	ธ.ค. 2562
1	ศึกษาเกี่ยวกับภาษา Java , spring boot, Git					
2	ทดลองทำ web service ใช้งานตามโจทย์ที่ได้รับมา					
3	ศึกษาการใช้ hibernate และ criteria builder					
4	ศึกษาการทำงานของภาพรวมระบบ					
5	ศึกษาการทำงานของ database และจุดประสงค์ของตาราง (Table)					
6	ประชุมวางแผนการทำงานของระบบ การทำงานของเซอร์วิสแต่ละตัวและระยะเวลาในการทำ					
7	จัดทำ แก้ไขเซอร์วิสทั้งสอง					
8	จัดทำต้นฉบับปริญญานิพนธ์					

บทที่ 2
แนวคิด ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีหลักที่เกี่ยวข้องกับการทำงานของแอปพลิเคชัน

2.1.1 Micro Service

ปกติแล้วการจะทำเว็บเซอร์วิสไว้เรียกใช้งานมักจะถูกทำในรูปแบบแอปพลิเคชันขนาดใหญ่เพียงอันเดียว ซึ่งทุกฟังก์ชันการทำงานทุกอย่างจะถูกรวมอยู่ภายในนั้นทั้งหมด จะเกิดอะไรขึ้นหากตัวเว็บเซอร์วิสนี้เกิดการล่ม (Server Down) หรือเกิดการเสียหายในส่วนใดส่วนนึง เป็นที่แน่นนอนหากเกิดปัญหานี้ขึ้นทุกฟังก์ชันทุกการทำงานก็จำเป็นที่จะต้องหยุดตัวลง เพราะฉะนั้นจึงได้เกิดแนวคิดการทำไมโครเซอร์วิสขึ้นมา นั่นก็คือถ้าเกิดฟังก์ชันใดฟังก์ชันหนึ่งเกิดความเสียหายแต่ผู้ใช้งานก็ยังคงสามารถใช้งานติดต่อฟังก์ชันอื่น ๆ ภายในระบบได้อยู่หรือจะรวมไปถึงการที่ตัวเซอร์วิสถูกเรียกใช้ในฟังก์ชันใดฟังก์ชันหนึ่งเป็นจำนวนมากจนทำให้เซิร์ฟเวอร์ (Server) ไม่สามารถรองรับจำนวนผู้ใช้งานได้เพียงพอ โดยรูปแบบของเว็บแอปพลิเคชั่นทั่วไปจะแสดงดังรูปที่ 2.1 วิธีแก้ปัญหาก็คงไม่ใช่การนำเว็บเซอร์วิสที่มีฟังก์ชันทุกอย่างไปขึ้นเซิร์ฟเวอร์เพื่อเพิ่มช่องทางการเข้าใช้งานสำหรับฟังก์ชันเพียงอันเดียว หลักการของไมโครเซอร์วิสสามารถนำมาแก้ปัญหาในส่วนนี้ได้โดยนำฟังก์ชันที่มีแนวโน้มที่จะถูกเรียกใช้งานสูงนำไปขึ้นบนเซิร์ฟเวอร์จำนวนหลายตัวเพื่อรองรับการเข้าใช้งานจากผู้ใช้งานที่มีจำนวนมาก แล้วทำการกระจายการเข้าถึงฟังก์ชันจากผู้ใช้งานให้เท่า ๆ กันตามจำนวนฟังก์ชันที่เปิดให้ใช้บริการ (load balance) ดังแสดงในรูปที่ 2.2

ไมโครเซอร์วิสถูกนิยามออกมาในหลายความหมายมากมายตามที่แล้วแต่บุคคลจะเข้าใจ เพราะฉะนั้นการจะอธิบายไมโครเซอร์วิสให้เข้าใจโดยง่ายก็คงต้องเริ่มจากการอธิบายแนวคิดของมันซะก่อน โดยแนวคิดของมันก็คือ การแยก (Decomposing) แอปพลิเคชันให้เป็นฟังก์ชั่นเดี่ยว ๆ (single-function) ด้วยการที่ต้องง่ายต่อการติดต่อใช้งาน (well-defined interfaces) ซึ่งแต่ละเซอร์วิสสามารถสร้าง (build) และนำขึ้นบนเซิร์ฟเวอร์ (deploy) ได้อย่างอิสระไม่ขึ้นอยู่กับใคร (Independent) โดยแต่ละความสามารถจะถูกอธิบายไว้ดังนี้

1. Decomposing

จากการที่ต้องทำเพียงแค่หนึ่งแอปพลิเคชันใหญ่ ๆ จะถูกนำมาแยกเป็นเป็นแอปพลิเคชันขนาดเล็กที่แตกต่างกันโดยจะถูกเรียกว่าเซอร์วิส โดยแต่ละเซอร์วิสจะถูกออกแบบให้ทำงานตามลักษณะธุรกิจแบบใดแบบหนึ่งและมีการทำ หน้าสำหรับรองรับการใช้งาน หน้าสำหรับตรรกะที่ใช้คิดคำนวณและการเชื่อมต่อกับฐานข้อมูล

2. Single-function

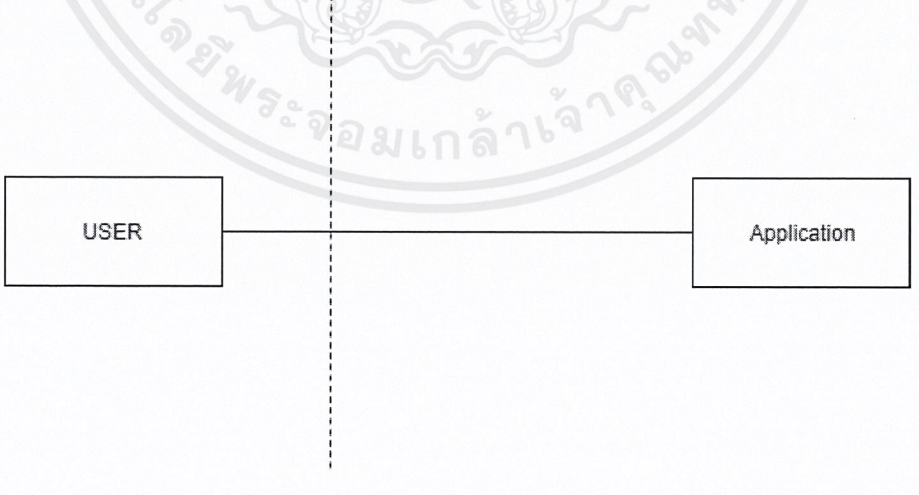
โดยแต่ละเซอร์วิสจะมีการทำงานของฟังก์ชันหรือรับผิดชอบในส่วนใดส่วนหนึ่งที่เฉพาะเจาะจงและตัวเซอร์วิสต้องสามารถรองรับการทำงานได้จำนวนมาก

3. Well-defined interfaces

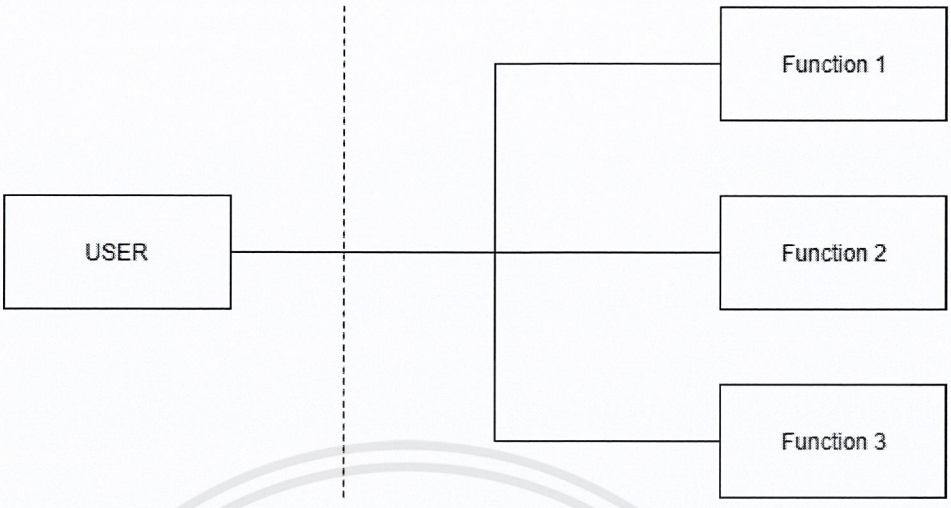
เซอร์วิสจำเป็นต้องกำหนดรูปแบบของหน้ารูปแบบการใช้งานให้สำหรับการติดต่อแต่ละเซอร์วิสนั้น โดยส่วนมากแล้วจะถูกนำเสนอด้วยรูปแบบข้อมูลขาเข้า(Input) และ ข้อมูลขาออก(Output)

4. Independent

เซอร์วิสไม่จำเป็นต้องรู้การทำงานของเซอร์วิสอื่น สามารถทดสอบ นำขึ้นบนเซิร์ฟเวอร์ และแก้ไขได้อย่างเป็นอิสระ ซึ่งอาจจะมีกรณีที่เซอร์วิสบางตัวไม่ได้ถูกเขียนด้วยภาษาโปรแกรมชนิดเดียวกันและอาจจะเชื่อมต่อฐานข้อมูลคนละตัวแต่ก็ไม่ได้หมายความว่าเซอร์วิสพวกนี้จะไม่ทำงานด้วย กัน ถ้ามีเซอร์วิสที่ต้องการใช้งานเซอร์วิสตัวดังกล่าวเพื่อต้องการทำให้เซอร์วิสตัวเองเสร็จสมบูรณ์ก็สามารถติดต่อกันผ่าน อินเทอร์เฟซของไมโครเซอร์วิสนั้นได้



รูปที่ 2.1 โครงสร้างแอปพลิเคชันทั่วไป



รูปที่ 2.2 โครงสร้างการทำระบบแบบไมโครเซอร์วิส

2.1.2 Web API

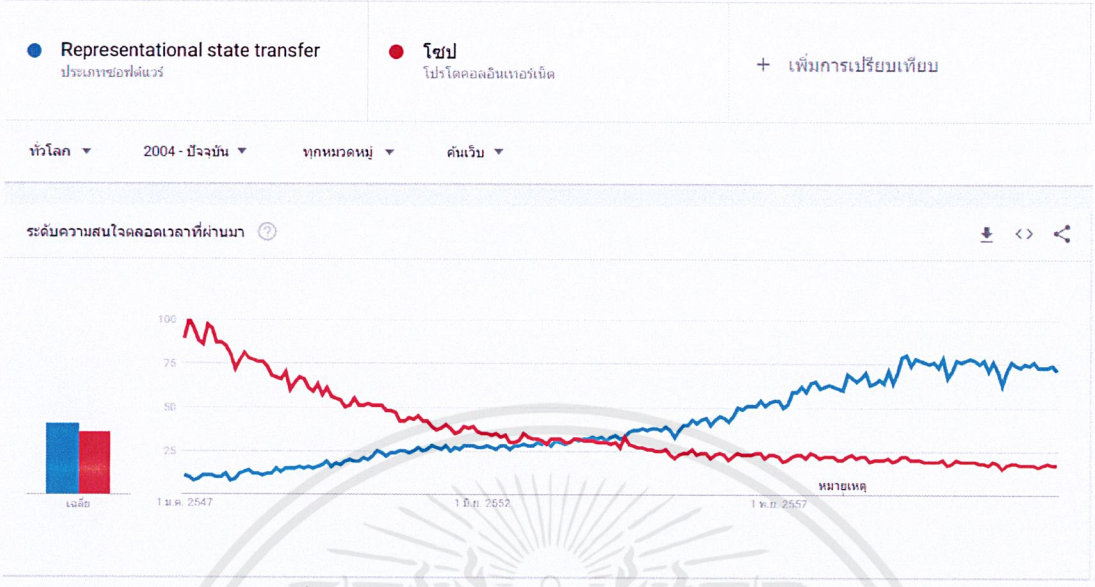
Web API หรือที่อาจจะรู้จักกันดีในชื่อเว็บเซอร์วิส ซึ่งเป็นส่วนหนึ่งในระบบที่จะเปิดให้ข้อมูลบริการแก่พวกแอปพลิเคชันต่าง ๆ เช่น Mobile Application, Web Application หรือ IoT Application โดยจะทำการรับ-ส่ง HTTP-request และ HTTP-response ไปบนเครือข่ายอินเทอร์เน็ต โดยจะทำงานคล้ายกับฟังก์ชันทั่ว ๆ ไปซึ่งก็คือการมี input เข้ามาและการมีการตอบกลับส่งกลับไป ซึ่งรูปแบบที่ HTTP สามารถอ่านเข้าใจได้และนำไปประมวลผลได้ก็จะมีเช่น XML, JSON ที่น่าจะรู้จักกันเป็นอย่างดี โดยตัว Web API นั้นจะมีอยู่ด้วยกัน 2 ชนิดนั้นก็คือ

- RESTful API

โดยข้อมูลที่ส่งผ่านนั้นจะอยู่ในรูปแบบของ JSON หรือ XML ก็ได้ และสามารถเรียกใช้งานตัวเซอร์วิสนั้น ๆ ได้ผ่านทาง URL โดยตรงอีกด้วย ซึ่งเป็นรูปแบบที่นิยมใช้ในปัจจุบัน

- SOAP-Based Web Services

ข้อมูลที่ส่งผ่านสามารถส่งผ่านกันได้แค่รูปแบบ XML เท่านั้น โดยถ้าจะเรียกใช้ก็จำเป็นต้องส่งคำขอไปในรูปแบบ SOAP (Simple Object Access Protocol) ซึ่งในปัจจุบันความนิยมในการใช้งานจะน้อยลงไปเรื่อย ๆ เมื่อเทียบกับตัว RESTful API ดังแสดงในรูปที่ 2.3



รูปที่ 2.3 เปรียบเทียบการใช้งานระหว่าง SOAP กับ REST

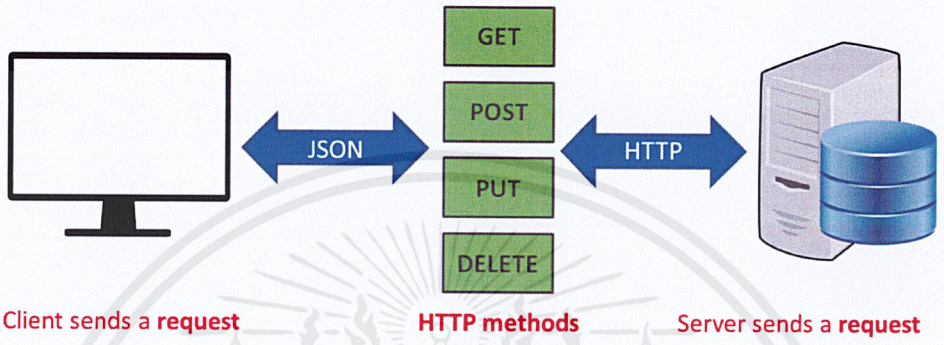
2.1.3 RESTful API

Representational state transfer หรือ REST หรือ RESTful ไม่ว่าจะถูกเรียกแบบไหนชื่อเหล่านี้จะหมายถึงสิ่งเดียวกันทั้งหมดซึ่ง REST จะเป็นแนวคิดในการทำแต่ถ้ามีเว็บเซอร์วิสไหนที่นำแนวคิดนี้มาใช้จะถูกเรียกว่า " RESTful Web Services " นั่นก็คือ การสร้าง webservice เพื่อสื่อสารกันบนอินเทอร์เน็ต โดยจะอาศัย URL (Uniform Resource Locator) ในการประมวลผลรับ-ส่งข้อมูลกันในรูปของ XML, JSON, HTML โดยอินเทอร์เฟซที่ใช้ติดต่อกันนั้นทั้งผู้รับ-ส่งต้องได้ตกลงกันอย่างเข้าใจว่าจะส่งกันในรูปแบบไหนได้ผลลัพธ์กลับไปเป็นอย่างไร ซึ่งส่วนที่จะรับ และตอบกลับไปนั้นจะถูกพัฒนาด้วยภาษาโปรแกรมมิ่งต่าง ๆ

RESTful API นั้นสามารถใช้ชื่อ URL ที่ซ้ำกันได้แต่ต้องกำหนด HTTP Request Method ที่แตกต่างกัน เพื่อจำแนกจุดประสงค์ของการให้บริการ ซึ่งตัว HTTP Request Method ที่ถูกใช้อย่างบ่อยครั้งก็จะมีดังนี้

1. GET ใช้สำหรับการดึงข้อมูล
2. POST ใช้สำหรับสร้างข้อมูล
3. PUT ใช้เมื่อต้องการแก้ไขข้อมูลใด ๆ ที่อยู่ในฐานข้อมูล
4. DELETE ใช้สำหรับลบข้อมูล

การทำงานของ RESTful API นั้นเริ่มต้นด้วยการที่ไคลเอนต์ (client) ส่งการร้องขอเข้ามายังตัวเซอร์วิสตามรูปแบบที่เซอร์วิสนั้นต้องการไปบน URL จากนั้นตัวเซอร์วิสก็จะรับและประมวลผลตาม business logic ของแต่ละเซอร์วิสแล้วส่งผลลัพธ์กลับไปให้กับไคลเอนต์ก็เป็นอันเสร็จการทำงาน โดยการทำงานจะวิ่งอยู่บน HTTP protocol



รูปที่ 2.4 การทำงานของ RESTful
(https://www.doprax.com/content/What-is-restful-API%3F/)

Path ที่ใช้เพื่อติดต่อกับ RESTful API นั้นจะสามารถตั้งเป็นชื่ออะไรก็ได้แต่ส่วนมากจะถูกตั้งชื่อให้สื่อความหมายกับสิ่งที่เซอร์วิสนั้นทำ หรือเกี่ยวข้องเช่น

- localhost:8080/student ใช้ในการขอข้อมูลนักศึกษาทั้งหมด
- localhost:8080/student/{studentId} ใช้เพื่อขอข้อมูลนักศึกษารายบุคคล
- localhost:8080/student?faculty=engineer&year=4 ใช้ในการระบุกลุ่มเงื่อนไขของนักศึกษาที่ต้องการ โดยชื่อของ URL นั้นจะจบหลังเจอกับเครื่องหมาย ? ต่อจากนั้นจะเป็นชื่อตัวแปรและค่าที่ต้องการส่งไปยังเซอร์วิส

เมื่อเกิดข้อผิดพลาดจาก API ก็จะมีการส่งการตอบกลับตอบกลับตามมาตรฐานของ HTTP โดย HTTP status ที่พบเห็นได้ทั่วไปจะมีดังนี้

1. 2xx: Successful
   - 200: OK - การรับส่งข้อมูลเป็นไปอย่างถูกต้อง
   - 201: Created - สร้างทรัพยากรใหม่แล้ว
   - 202: Accepted - ได้รับการร้องขอแล้วแต่ยังไม่ได้ประมวลผล

- 204: No Content - เซอร์วิสประมวลผลเรียบร้อยแล้วแต่ไม่ได้ส่งเนื้อหาอะไรกลับ

2. 4xx: Client Error

- 400: Bad Request - ทำงานไม่สำเร็จอาตเป็นเพราะ syntax ไม่ถูกต้อง

- 401: Unauthorized - ยังไม่ได้ระบุตัวตน

- 403: Forbidden - รับทราบการร้องขอแล้วแต่เซิร์ฟเวอร์ปฏิเสธการทำงานเพราะไม่มีสิทธิเข้าถึง

- 404: Not Found - ไม่พบหน้าที่ร้องขอ

- 405: Method Not Allowed - เนื่องจาก HTTP method ไม่ถูกต้องอาจจะเป็นข้อผิดพลาดในการใช้ GET, POST, PUT, DELETE สลับกัน

- 406: Not Acceptable - header ของการร้องขอไม่สัมพันธ์กัน

- 413: Request Entity Too Large - เซิร์ฟเวอร์ไม่สามารถประมวลผลการร้องขอที่ขนาดใหญ่เท่านี้ได้

- 414: Request-URI Too Long - URL ที่ส่งมายาวเกินไป

- 415: Unsupported Media Type - เซิร์ฟเวอร์ไม่รองรับชนิดของรูปหรือสื่อที่ส่งมา

3. 5xx: Server Error

- 500: Internal Server Error มีข้อผิดพลาดเกิดขึ้นที่เซิร์ฟเวอร์

- 501: Not Implemented เซิร์ฟเวอร์ไม่เข้าใจการร้องขอหรือไม่สามารถทำงานตามคำสั่งได้

- 502: Bad Gateway เซิร์ฟเวอร์เป็นเกตเวย์ (Gateway) หรือ พร็อกซี่ (Proxy) ได้รับการตอบกลับผิดพลาดมาจากเซิร์ฟเวอร์อื่น

- 503: Service Unavailable เซิร์ฟเวอร์ down หรือมีการปรับปรุงเซิร์ฟเวอร์อยู่

- 504: Gateway Timeout ไม่ได้รับการตอบสนองในเวลาที่กำหนด

2.1.4 JSON

JSON ถูกย่อมาจาก JavaScript Object Notation มันก็คือรูปแบบการจัดเก็บข้อมูลของจาวาสคริปต์ (JavaScript) นั่นเอง แต่จะถูกจัดเก็บแบบ JSON Object แต่ถึงอย่างนั้นก็ไม่ได้มีแค่ภาษา จาวาสคริปต์เพียงภาษาเดียวที่อ่านรูปแบบนี้ออกเท่านั้นภาษาอื่น ๆ ก็สามารถอ่านและเข้าใจได้ซึ่งจริง ๆ ตัว JSON ก็คือ Standard format อย่างหนึ่งที่เป็น text และสามารถอ่านออกได้ด้วยตาเปล่า ที่สร้างขึ้นมา

ใช้ในการสร้างออบเจ็ค (Object) เพื่อส่งข้อมูลรับ-ส่งระหว่างแอปพลิเคชัน โดย format ของ JSON นั้นจะอยู่ในรูปของแบบเป็นคู่ Key กับ Value หรือจะเป็นแบบ Array ก็ได้ โดยข้อมูลนั้นจะอยู่ภายใต้เครื่องหมายปีกกา ซึ่ง JSON นั้นถูกนำมาใช้งานในการสร้าง และแปลง format ไปมาระหว่างภาษาโปรแกรมมิ่งที่รองรับได้เช่น การแปลงออบเจ็คเป็น JSON หรือการแปลง JSON กลับไปเป็นออบเจ็คในภาษาจาวา

รูปแบบ JSON Object

ชุดข้อมูลที่เป็นคู่ Key-Value แบบ strings ใช้สัญลักษณ์ปีกกา {key1:value1, key2:value2} ใช้เครื่องหมายจุลภาคเป็นตัวแบ่งแต่ละคู่ และใช้เครื่องหมายทวิภาคเป็นตัวแบ่งระหว่าง key กับ value โดยจะไม่สนใจลำดับในการวางข้อมูล เพราะจะอ้างอิงจากตัว Key เพื่อแมพปิ้งกับออบเจ็คของภาษาโปรแกรมมิ่งนั้น ๆ ที่ต้องการจะแปลงไปมาระหว่าง key และ value

ชนิดข้อมูลของ JSON

รูปแบบข้อมูลของ JSON นั้นจะสามารถแบ่งออกได้ทั้งหมด 4 ชนิดดังนี้

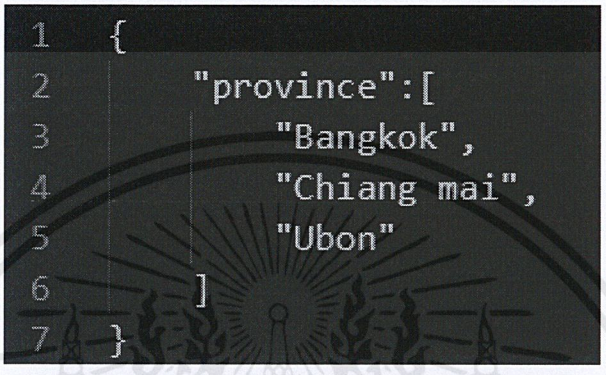
1. Number: เป็น type ชนิดตัวเลข ไม่จำเป็นต้องมีเครื่องหมายอัญประกาศ (“ ”) ครอบ
2. String: ขึ้นต้นปิดท้ายด้วยเครื่องหมายอัญประกาศ (“ ”)
3. Boolean: ค่าเป็น true หรือ false
4. Null: ค่าว่าง

```
{
    "studentId":"123456",
    "studentName":"Sukrit",
    "phone":"0123456789",
    "age":21
}
```

รูปที่ 2.5 ตัวอย่างรูปแบบข้อมูล JSON

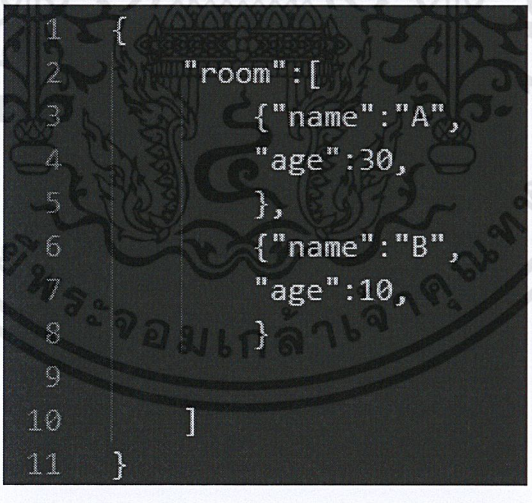
รูปแบบ JSON Array

เป็นรูปแบบที่สนใจลำดับชื่อ key ซึ่งจะเป็นชื่ออาร์เรย์ก่อนแล้วตามด้วยเครื่องหมายทวิภาค จากนั้นตามด้วยเครื่องหมายวงเล็บเหลี่ยม ([ ]) ซึ่งภายในเครื่องหมายวงเล็บเหลี่ยมนั้นจะเป็นสมาชิกของอาร์เรย์และแต่ละสมาชิกจะถูกคั่นด้วยเครื่องหมายจุลภาค ดังแสดงในรูปที่ 2.6



รูปที่ 2.6 ตัวอย่างรูปแบบข้อมูล JSON

ซึ่งในอาร์เรย์นั้นก็สามารถมีสมาชิกเป็นออบเจ็คได้เหมือนกันโดยสามารถเขียนได้ดังแสดงในรูปที่ 2.7

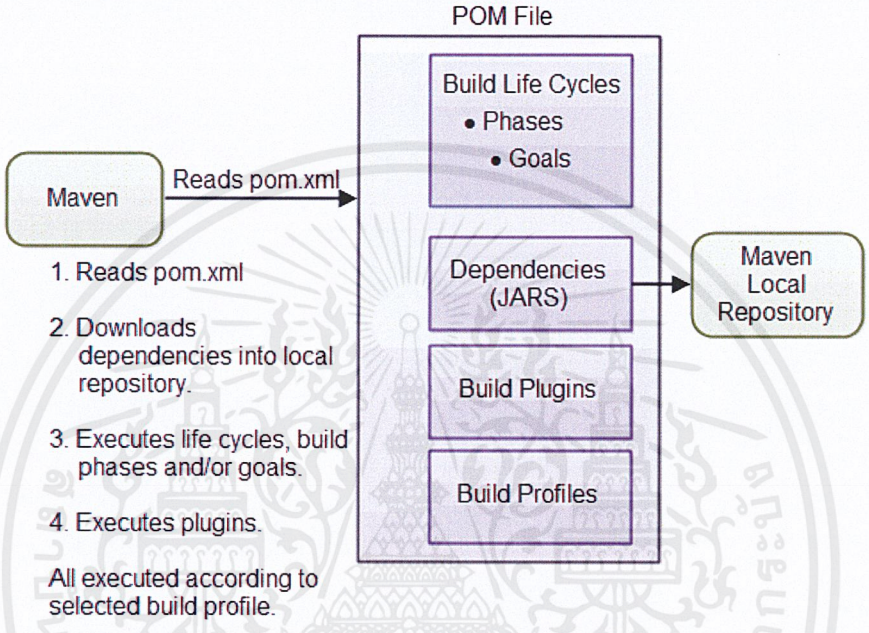


รูปที่ 2.7 ตัวอย่างรูปแบบข้อมูล JSON

2.1.5 Maven

Maven เป็นเครื่องมือที่จะนำมาใช้ในการบริหารและจัดการโปรเจคซึ่งจะช่วยให้เหล่า Java Developer พัฒนาระบบได้ง่ายยิ่งขึ้น โดยจะช่วยจัดการเรื่องของไลบรารี่ต่าง ๆ ที่ต้องการนำมาใช้ในโปร

เจคได้ง่ายมากยิ่งขึ้นเพียงแค่ทำการ config ซึ่งหัวใจหลักของ Maven นั้นคือ POM (Project Object Model) ซึ่งเป็นไฟล์ข้อมูลที่เกี่ยวกับการสร้างโปรเจคในรูปแบบ XML Configuration File เพื่อให้นักพัฒนาสามารถกำหนดการทำงาน (Library Dependencies) ต่าง ๆ ของโปรเจค โดย 1 โปรเจคก็จะ pom.xml เพียง 1 ไฟล์เท่านั้น โดยการทำงานพื้นฐานนั้นจะเป็นไปดังรูปที่ 2.8

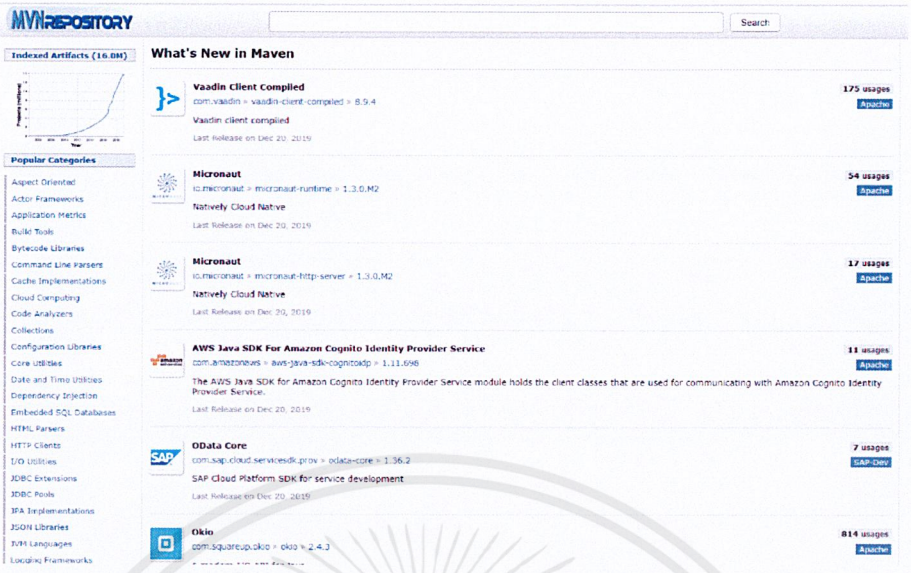


รูปที่ 2.8 การทำงานของ Maven (https://medium.com/thipwriteblog/สรุปสิ่งที่ได้จากการทบทวน-apache-maven-ไว้สักหน่อย-61d315a8f64d)

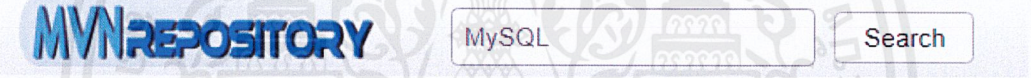
โดยนักพัฒนาสามารถนำไลบรารี่มาได้จาก 2 แหล่งด้วยกันนั่นก็คือ

1. เว็บไซต์ของผู้สร้างไลบรารี่นั้นโดยตรง
2. Central Maven Repository

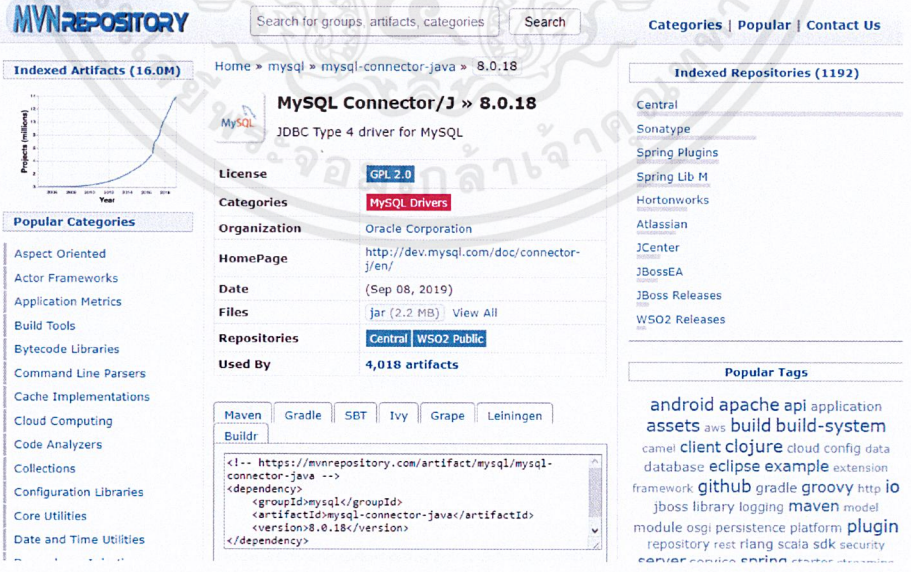
นักพัฒนาจะสามารถหาไลบรารี่เพื่อนำมากำหนด dependency ที่จะใช้ในโปรเจคได้จากทาง https://mvnrepository.com ดังรูปที่ 2.9 ซึ่ง dependency ต่าง ๆ ที่อยู่ภายในเว็บไซต์จะเป็นสิ่งที่บริษัทต่าง ๆ ยินยอมให้นำไลบรารี่เหล่านี้ไปใช้ได้โดยไม่ต้องเสียเงิน โดยรายละเอียดของต่าง ๆ ของการค้นหา dependency จะเป็นดังรูปที่ 2.10 และรูปที่ 2.11



รูปที่2.9 หน้าหลักของเว็บ Central Maven



รูปที่ 2.10 ช่องการค้นหา dependencies

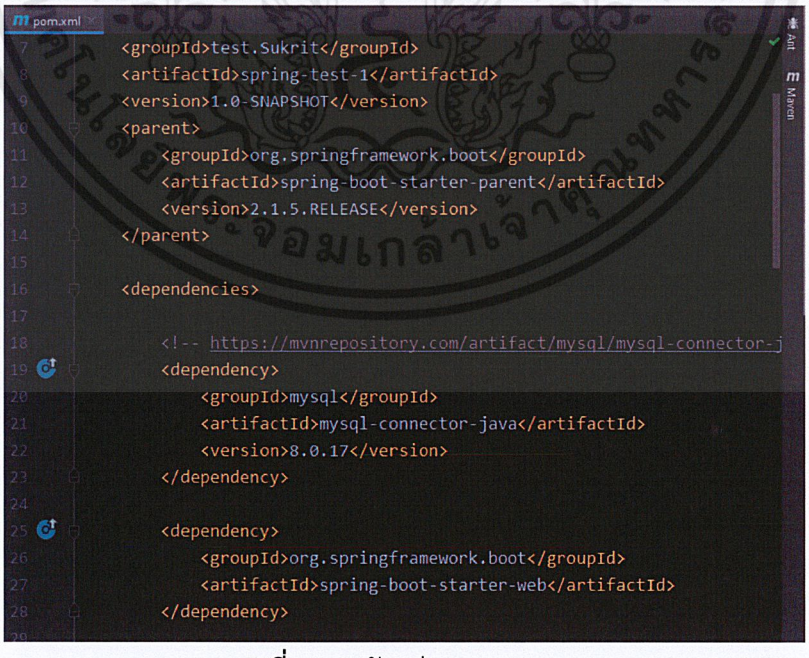


รูปที่ 2.11 รายละเอียดของ dependency

ซึ่งการเพิ่มไลบรารี่นั้นจะต้องเพิ่มใน pom.xml โดยข้อมูลไลบรารี่ที่นำมาใช้นั้นจะถูกวางอยู่ภายใต้แท็ก <dependencies> </dependencies> หลักจากที่กดบันทึก pom.xml แล้วนั้น Maven จะทำการสำรวจ Maven Local Repository ที่อยู่ภายในเครื่องเราก่อนเพื่อนำไฟล์ที่เกี่ยวข้องกับ dependencies ต่าง ๆ ที่เราได้ใส่ไว้เข้ามาในโปรเจคอัตโนมัติ แต่ถ้าหากไลบรารี่นั้นเราไม่เคยใช้เลย Maven จะติดต่อไปยัง Central Maven Repository เพื่อดาวน์โหลดไลบรารี่ที่ไม่ได้อยู่ภายในเครื่องเรามาเก็บไว้ในเครื่อง ซึ่งการดาวน์โหลดจากส่วนกลางนั้นจะถูกดาวน์โหลดเพียงครั้งเดียวคือครั้งแรกที่ถูกเรียกหาไลบรารี่นั้นโดยที่เครื่องเราไม่เคยดาวน์โหลดมาก่อน แต่หลังจากนั้นจะเป็นการนำไฟล์แคช (cache) มาเพิ่มโดยที่ไม่ต้องมีการดาวน์โหลดใหม่ แน่นอนว่าการดาวน์โหลดจาก Central Maven Repository จำเป็นต้องใช้งานการเชื่อมต่ออินเทอร์เน็ตด้วย

Maven นั้นจะมีคำศัพท์ที่จำเป็นต้องรู้ เพื่อให้การพัฒนานั้นมีประสิทธิภาพ ซึ่งตัวอย่างของไฟล์ Maven นั้นจะเป็นไปตามที่แสดงในรูปที่ 2.12 โดยจะมีคำศัพท์ทั้งหมด 4 ตัวที่จำเป็นต้องรู้จักดังนี้

1. groupId - เป็นชื่อ source code package hierarchy
2. artifactId - เป็นการตั้งชื่อปแบพลิเคชั่นโดยค่าที่ใช้จะต้องกำหนดเป็น unique
3. version - เป็นการตั้งค่า version ของแอปพลิเคชัน
4. dependency - เป็นการนำ Library Dependency ที่ต้องการใช้มาใส่ไว้ใน pom.xml



รูปที่ 2.12 ตัวอย่าง pom.xml

2.1.6 Unit Test

การทำซอฟแวร์นั้นนอกจากต้องคำนึงถึงการทำงานของโปรแกรมเป็นสำคัญแล้วนั้นนักพัฒนาก็ควรคำนึงถึงบั๊ก (bugs) ที่มีโอกาสจะเกิดขึ้นด้วย ดังนั้นจึงจำเป็นอย่างมากที่ตัวโปรแกรมจำเป็นต้องมีการทดสอบความถูกต้องในการทำงานของเมทธอดต่าง ๆ

โดยทั่วไปแล้วนักพัฒนาส่วนใหญ่จะไม่สนใจการทำการทดสอบเมทธอดของตัวเองมากนัก เพราะว่ามั่นใจกับโค้ดที่ตัวเองเขียนแต่ปัญหาจะอยู่ตอนที่ถ้าเกิดมีการเพิ่มหรือแก้ไขอะไรต่าง ๆ ภายในโค้ดส่วนที่เพิ่มไปนั้นอาจส่งผลกระทบต่อการทำงานเมทธอดอื่น ๆ ก็เป็นได้ ซึ่งเหตุการณ์นี้จะไม่เป็นปัญหาหากตัวโปรแกรมมีการเตรียมการทดสอบสำหรับเมทธอดต่าง ๆ เอาไว้ ถ้ารันการทดสอบแล้วไม่มีข้อผิดพลาดใด ๆ นั่นก็แสดงว่าสิ่งที่เราเพิ่มเข้ามาใหม่นั้นสามารถใช้งานได้ แต่ถ้ามีข้อผิดพลาดเกิดขึ้นเราก็จะสามารถตรวจสอบได้ว่าข้อผิดพลาดนั้นมาจากเมทธอดใด

unit testing คือการเขียนการทดสอบเล็ก ๆ ในส่วนของโค้ดภายในแอปพลิเคชัน โดยการเขียนการทดสอบนี้นั้นจะแยกการเขียนออกเป็นอิสระจากตัวโค้ดที่ทำงานจริง ๆ โดยทำการเขียนทดสอบเพื่อการตรวจสอบ behavior และ functionality ของโค้ดหรือเมทธอดนั้น ๆ ว่าทำงานได้อย่างถูกต้องมีผลลัพธ์ตามที่คาดหวังไว้รึป่าว การเขียน unit tests ที่ดี แต่ละ test cases ต้องทำงานอย่างเป็นอิสระและการทำงานของแต่ละ test cases ต้องไม่ขึ้นอยู่กับลำดับในการ run ของ test cases

อย่างที่ได้กล่าวไปนั้นว่าการทำ unit test ต้องทำงานอย่างเป็นอิสระไม่ยุ่งเกี่ยวกับ dependencies อื่น ๆ จากภายนอก (นอกเหนือจากเมทธอดที่ต้องการทำการทดสอบ) แต่ถ้ามี dependencies มาเกี่ยวข้องแล้วนั้นก็จำเป็นต้องทำการ Mock ให้แก่ dependencies ที่เกี่ยวข้อง

2.2 โปรแกรมที่ใช้พัฒนาโครงงาน

2.2.1 IntelliJ IDEA

IntelliJ IDEA คือ integrated development environment (IDE) ซึ่งใช้สำหรับการพัฒนาโปรแกรมซอฟแวร์บนคอมพิวเตอร์ในภาษา Java ถูกพัฒนาโดย JetBrains โดยเวอร์ชั่นแรกถูกปล่อยออกมาให้ใช้งานเมื่อตอน มกราคม พศ.2544 สำหรับนักพัฒนาแอปพลิเคชัน Android นั้นก็คงจะรู้จักโปรแกรม Android Studio เป็นอย่างดีซึ่งตัวโปรแกรมนี้ก็พัฒนาโดยใช้ตัว community edition of

IntelliJ IDEA เป็นตัวตั้งตนนี่เอง โดย IntelliJ นั้นจะมีเวอร์ชั่นอยู่ 2 แบบ คือ Community และ Ultimate editions ซึ่งจะเป็นแบบเปิดให้ใช้งฟรีกับต้องจ่ายเงินเพื่อใช้งานตามลำดับ

สเปคเครื่องคอมพิวเตอร์ที่สามารถใช้งานโปรแกรม IntelliJ IDEA

- RAM ต่ำสุดที่ 2 GB แนะนำ 8 GB สำหรับโปรเจคขนาดใหญ่
- Disk space ต้องการพื้นที่ว่าง 2.5 GB และ 1 GB สำหรับเก็บแคชรวมเป็น 16 GB
- Monitor resolution ต่ำสุดที่ 1024 * 768 และสูงสุดที่ 1920 * 1080
- Microsoft Windows 7 SP1 หรือมากกว่า
- MacOS 10.11 หรือมากกว่า
- Linux distribution ที่รองรับ Gnome, KDE, or Unity DE

คุณสมบัติของ IntelliJ IDEA

1. Coding assistance

IDE จะมีคุณสมบัติที่ช่วยให้เขียนโค้ดได้ง่ายยิ่งเช่น การคาดการณ์โค้ดของเราต้องการจะพิมพ์อะไรโดยวิเคราะห์จากบริบทที่เราพิมพ์อยู่, code navigation หรือก็คือผู้ใช้จะสามารถกดเข้าไปดูคลาสที่ประกาศไว้ได้โดยตรงผ่านทางโค้ด และยังช่วยแนะนำโค้ดที่เราได้เขียนไปว่าควรมีไรอะปรับเปลี่ยนบ้าง

2. Built in tools and integration

IDE จะจัดหาเครื่องมือจำพวก build/packaging ให้แก่เราเช่น grunt, bower, gradle, maven และ SBT และยังรองรับพวกระบบ version control ต่าง ๆ อีกด้วยเช่น Git, Mercurial, Perforce และ SVN รวมไปถึงสิ่งที่เกี่ยวกับฐานข้อมูลต่าง ๆ ไม่ว่าจะเป็น Microsoft SQL Server, Oracle, PostgreSQL, SQLite และ MySQL จะถูกเข้าถึงได้โดยตรงผ่านทาง IDE

3. Plugin ecosystem

รองรับการเพิ่ม plugin ต่าง ๆ ให้แก่ IDE ซึ่งจะช่วยเพิ่มฟังก์ชันในการใช้งานให้แก่ผู้ใช้ตามที่ผู้ใช้ได้เลือกดาวน์โหลด ซึ่ง IDE ทั้งแบบ Community และ Ultimate editions นั้นจะมีจำนวน plugin ให้เลือกจำนวนมากกว่า 3000 plugins

4. Supported languages

เนื่องจาก IntelliJ IDEA นั้นเปิดให้ใช้งานสองเวอร์ชั่นทั้งแบบ จึงทำให้สองเวอร์ชั่นนี้รองรับภาษาที่แตกต่างกันไปด้วย โดยภาษาที่ทั้งสองเวอร์ชั่นรองรับด้วยกันจะมีดังนี้

- Clojure (ติดตั้งผ่าน plugin)
- CloudSlang (ติดตั้งผ่าน plugin)
- Dart (ติดตั้งผ่าน plugin)
- Elm (ติดตั้งผ่าน plugin)
- Erlang (ติดตั้งผ่าน plugin)
- Go (ติดตั้งผ่าน plugin)
- Gosu (ติดตั้งผ่าน plugin)
- Groovy
- Haskell (ติดตั้งผ่าน plugin)
- Haxe (ติดตั้งผ่าน plugin)
- Java
- Julia (ติดตั้งผ่าน plugin)
- Kotlin
- Lua (ติดตั้งผ่าน plugin)
- Perl (ติดตั้งผ่าน plugin)
- Python (ติดตั้งผ่าน plugin)
- Rust (ติดตั้งผ่าน plugin)
- Scala (ติดตั้งผ่าน plugin)
- XML/XSL
- R (ติดตั้งผ่าน plugin)

และที่รองรับแค่เวอร์ชั่น Ultimate editions จะมีดังนี้

- ActionScript/MXML
- CoffeeScript
- HTML/XHTML/CSS
- JavaScript
- PHP (ติดตั้งผ่าน plugin)
- Ruby/JRuby

- SQL
- TypeScript

เทคโนโลยีและเฟรมเวิร์กที่ IDE รองรับทั้งสองเวอร์ชั่นร่วมกันจะมีดังนี้

- Android
- Ant
- Gradle
- JUnit
- JavaFX
- Maven
- Python
- SBT
- TestNG

และที่รองรับแค่เวอร์ชั่น Ultimate editions จะมีดังนี้

- Django
- EJB
- FreeMarker
- Geronimo
- GlassFish
- Google App Engine
- Google Web Toolkit
- Grails
- Hibernate/JPA
- JBoss Seam
- JBoss
- Jetty
- Java ME MIDP/CLDC

- JSF
- JSP
- Jelastic
- Node.js
- OSGi
- Play
- Ruby on Rails
- Spring
- Struts 2
- Tapestry
- Tomcat
- Velocity
- Web services
- Weblogic
- WebSphere

เป็นที่แน่นอนว่าการรองรับระบบ version control และ revision control ของทั้งสองเวอร์ชั่นก็จะแตกต่างกัน โดยระบบที่รองรับกับทั้งสองเวอร์ชั่นนั้นจะมีดังนี้

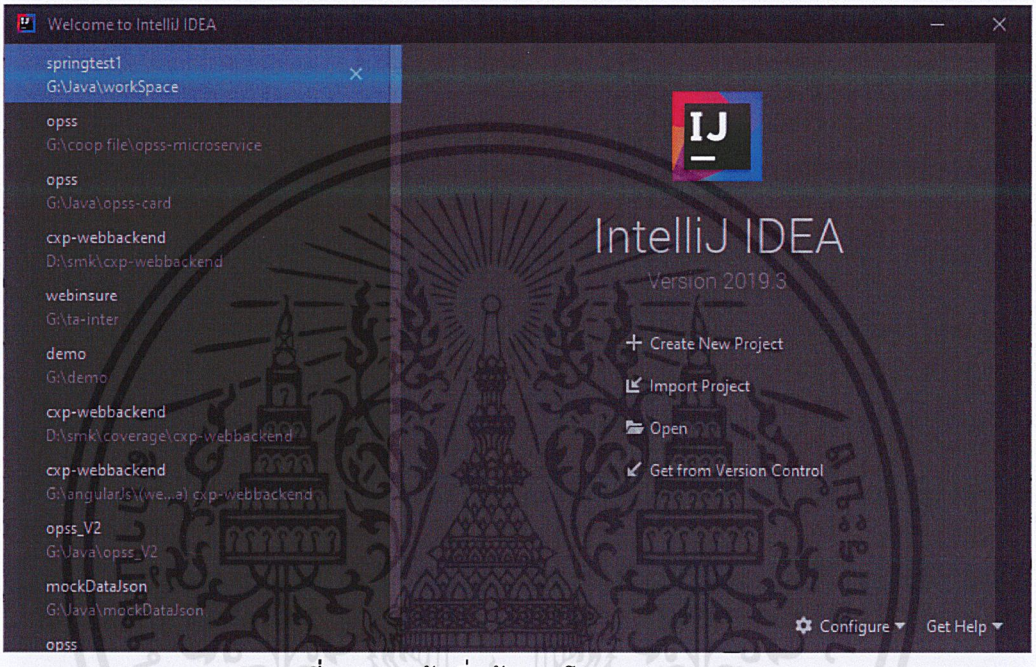
- CVS
- Git
- GitHub
- Mercurial
- Subversion
- Azure DevOps

และที่รองรับแค่เวอร์ชั่น Ultimate Edition จะมีดังนี้

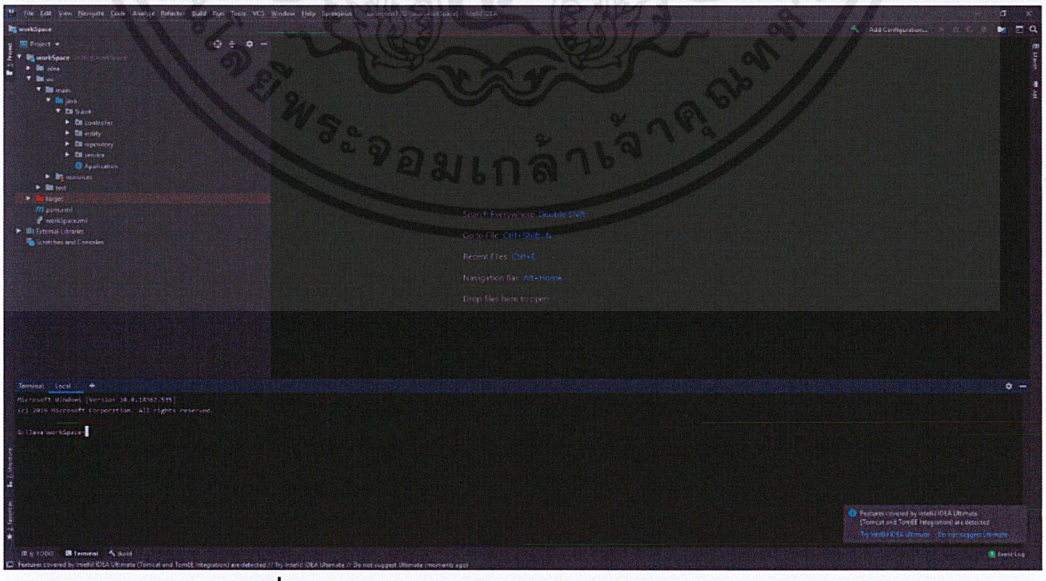
- Visual SourceSafe
- Perforce

- ClearCase

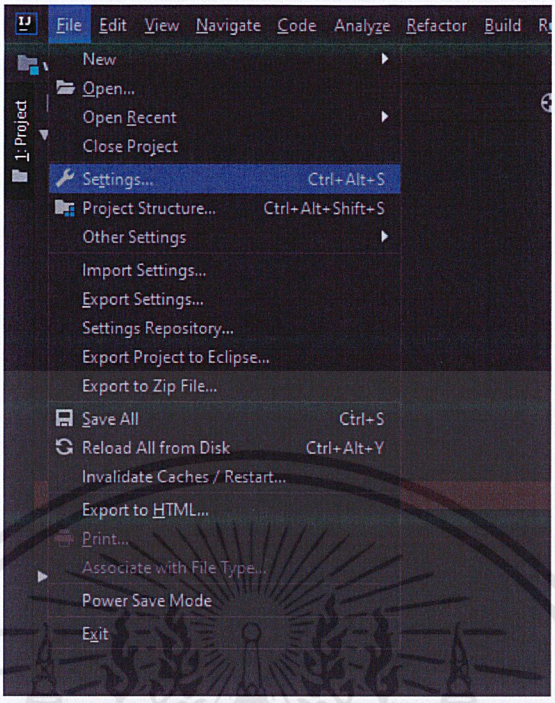
โดยหน้าเริ่มต้นของโปรแกรมและหน้าหลักนั้นจะเป็นไปดังแสดงในรูปที่ 2.14 และรูปที่ 2.15 ตามลำดับ และการจะดาวน์โหลด plugin ของตัวโปรแกรมนั้นจะดาวน์โหลดโดยการกดการตั้งค่า ดังแสดงในรูปที่ 2.16 และเข้าไปยังเมนู plugin ซึ่งจะสามารถดาวน์โหลด plugin ต่าง ๆ ได้ดังรูปที่ 2.17



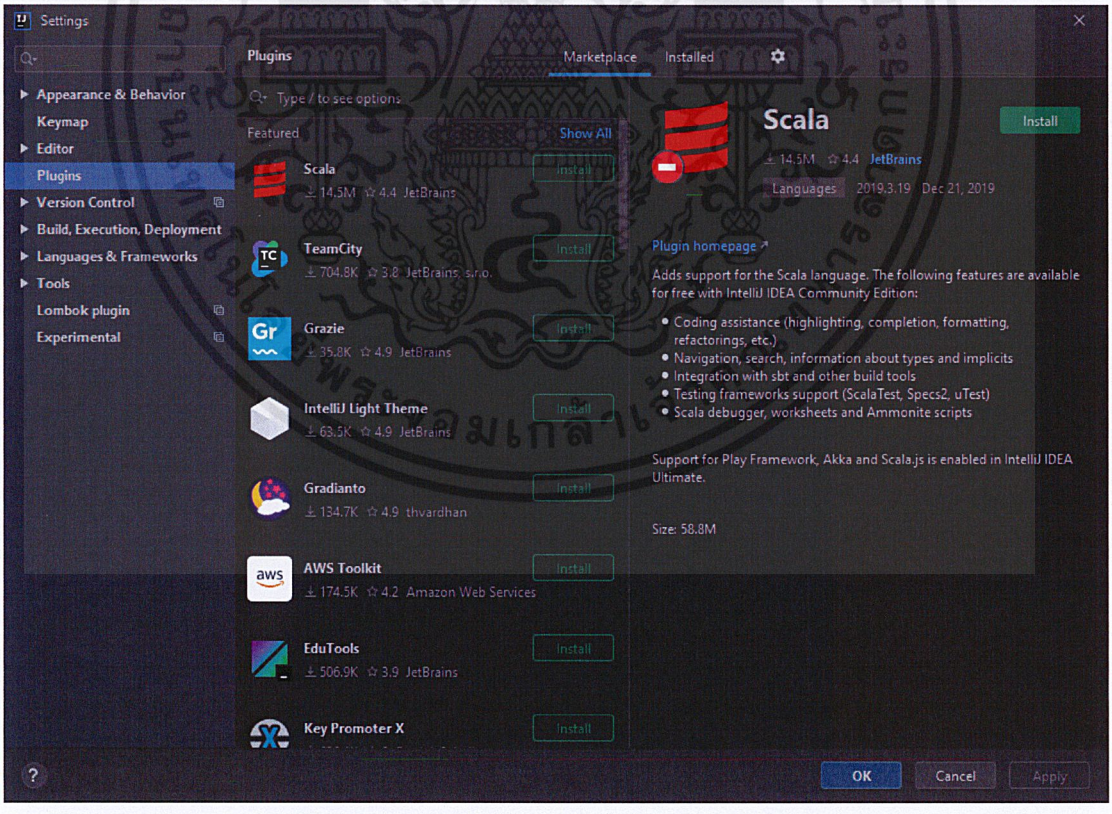
รูปที่ 2.13 หน้าเริ่มต้นของโปรแกรม IntelliJ



รูปที่ 2.14 หน้าหลักการใช้งานของโปรแกรม IntelliJ



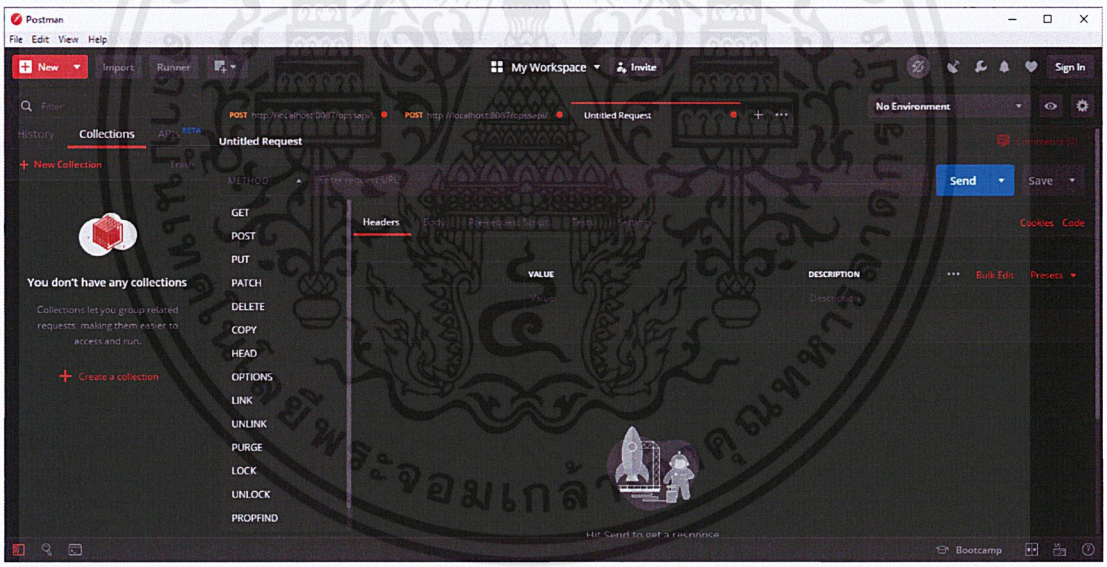
รูปที่ 2.15 ปุ่มการตั้งค่าของโปรแกรม IntelliJ



รูปที่ 2.16 หน้าดาวน์โหลด Plugin ของโปรแกรม IntelliJ

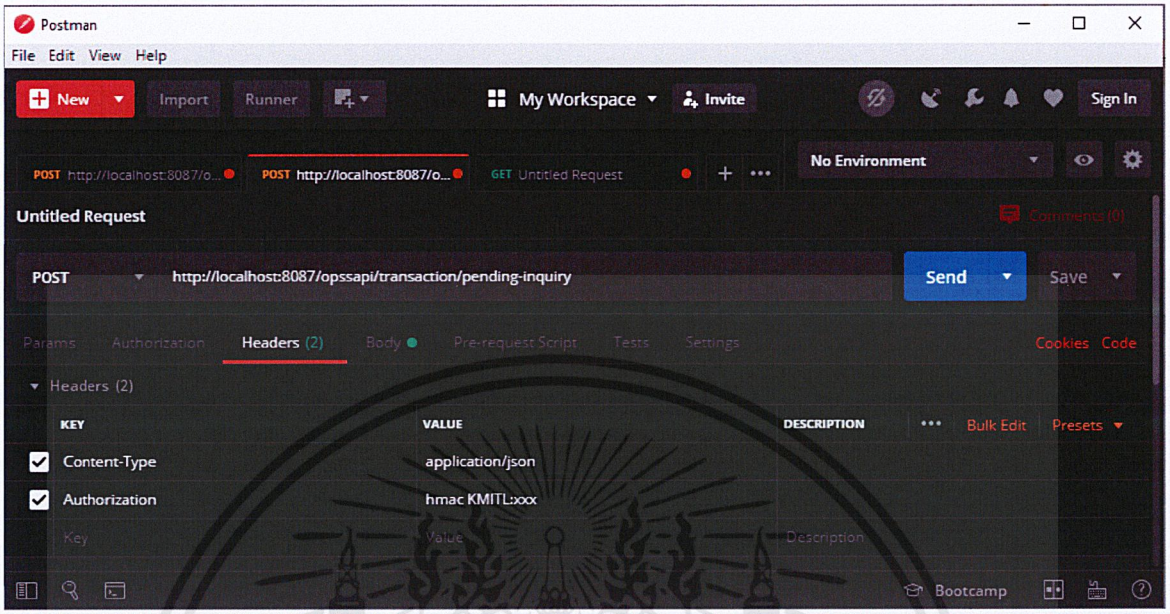
2.2.2 Postman

ความสำคัญของ API ในปัจจุบันมีผลเป็นอย่างมากต่อผู้พัฒนาโปรแกรมแอปพลิเคชันต่าง ๆ ซึ่ง API นั้นจะทำให้การแลกเปลี่ยนข้อมูล การโอนย้ายข้อมูลระหว่างสองแอปพลิเคชันที่ไม่ได้พัฒนามาร่วมกัน ให้สามารถติดต่อแลกเปลี่ยนสื่อสารกันได้อย่างง่ายดาย ซึ่งเป็นที่แน่นอนว่า API ของเราจำเป็นต้องมีการรับ-ส่งข้อมูลอย่างถูกต้องตาม business logic ที่ผู้พัฒนาได้ออกแบบวางแผนไว้ ซึ่งถ้าการให้บริการในการรับ-ส่งของเราไม่ถูกต้องหรือมีข้อผิดพลาดก็อาจจะเป็นเหตุให้ชื่อเสียงของบริษัทเสียหายได้ การจะตรวจสอบข้อผิดพลาดของตัว API ได้นั้นก็คงต้องใช้วิธีการทดสอบลองเรียกใช้ API และดูผลลัพธ์ต่าง ๆ ที่เกิดขึ้นว่ามีผลลัพธ์อันไหนผิดเพี้ยนไปรึป่าวโดยต้องเริ่มดูตั้งแต่การเริ่มรับข้อมูล การทำงานของฟังก์ชัน ความปลอดภัย ข้อผิดพลาดของ exception ต่าง ๆ จนกระทั่งไปถึงการส่งข้อมูลตอบกลับไป ซึ่ง Postman นั้นจะเป็น API ที่นิยมเป็นอย่างมากในการช่วยให้นักพัฒนาสามารถทดสอบ API ของตนเอง โดยหน้าหลักของโปรแกรมการจะแสดงดังรูปที่ 2.18

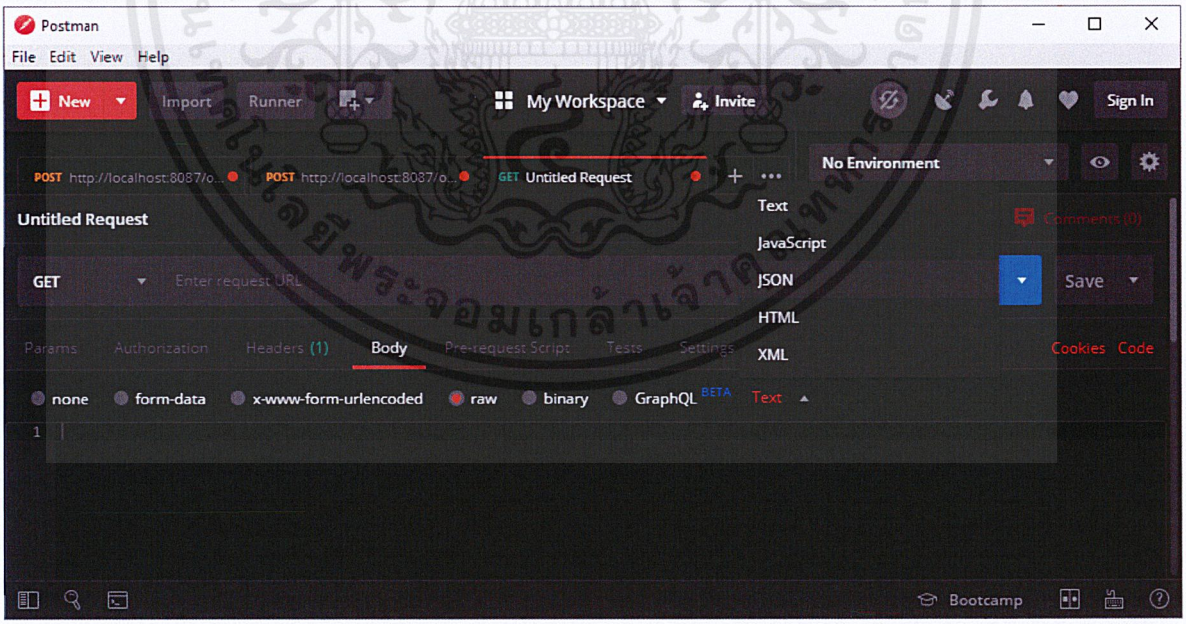


รูปที่ 2.17 หน้าหลักของโปรแกรม Postman

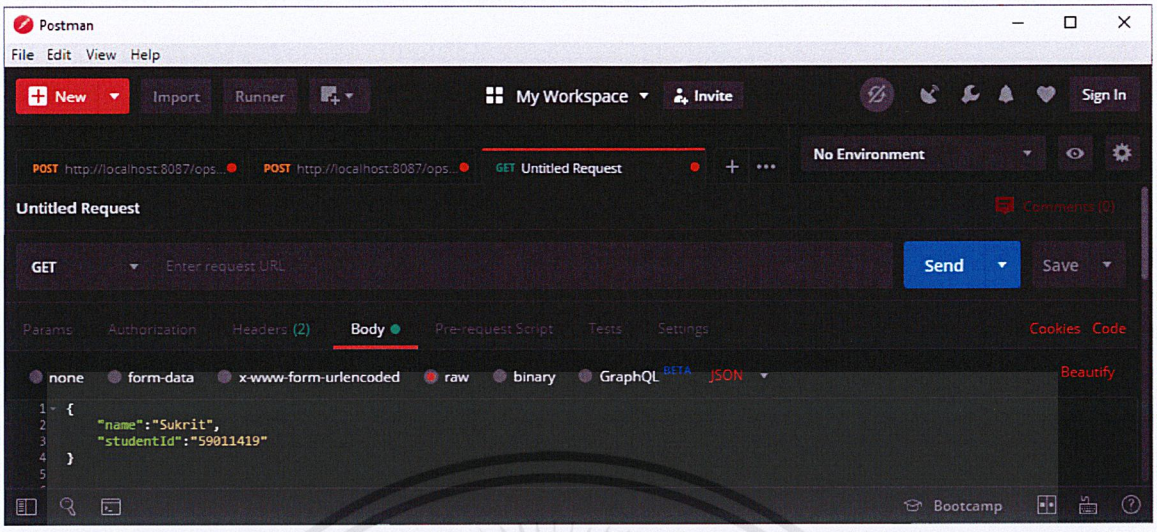
ใน Postman จะสามารถเลือกรูปแบบการส่งของ HTTP method ได้โดยจะมีให้เลือกมากมายตามรูปภาพข้างต้น Postman นั้นจะสามารถส่งทั้งส่วน header และ body ไปยัง API ของนักพัฒนาได้ ซึ่งก็เหมือนกับการติดต่อบน HTTP ทั่วไป โดยลักษณะของการกำหนด header และ body ใน Postman จะมีลักษณะดังรูปที่ 2.19 และรูปที่ 2.20 ตามลำดับ



รูปที่ 2.18 ส่วนของ header ในโปรแกรม Postman



รูปที่ 2.19 ส่วนของ body ในโปรแกรม Postman



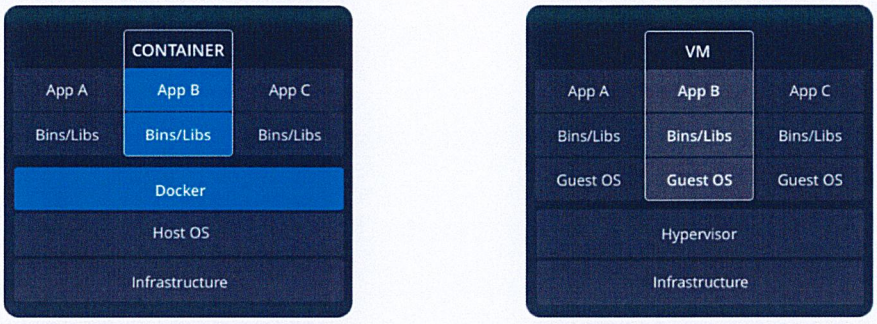
รูปที่ 2.20 การใส่ข้อมูลในส่วนของ body ในรูปแบบ JSON

ตัว Postman นั้นจะมีคุณสมบัติอีกมากมายสำหรับ Tester ที่ต้องการส่งข้อมูลเป็นชุดรวมกัน หรือการรอดูผลลัพธ์ว่ามีการส่งครั้งใดได้ผลลัพธ์ที่ผิดจากที่คาดไว้รึป่าว การทดสอบอันไหนมีสถานะการส่งสำเร็จหรือไม่สำเร็จ นั้นเป็นสิ่งที่ทำให้ Postman เป็นที่นิยมไม่ว่าจะสำหรับนักพัฒนาที่ต้องการทดสอบแค่เบื้องต้นหรือ Tester ที่ต้องการทดสอบการส่งแบบเป็นกลุ่มก็สามารถทำได้โดยง่าย

2.2.3 Docker

Docker นั้นคือ engine ชนิดหนึ่งที่มีการทำงานในรูปแบบที่จำลองสภาพแวดล้อมบนเครื่องเซิร์ฟเวอร์ เพื่อทำให้เซอร์วิสที่ต้องการสามารถทำงานได้บนเครื่องนั้น ๆ โดยจะทำงานคล้ายกับ Virtual Machine (VM) แต่จะแตกต่างกันตรงที่ถ้าหาก VM มีความต้องการในการใช้เซอร์วิสไหนก็ตามจำเป็นต้องไปทำการเพิ่มบน operating system (OS) นั้น ๆ แต่ docker จะใช้สิ่งที่เรียกว่า container ในการจำลองสภาพแวดล้อมที่เหมาะสำหรับการทำงานของเซอร์วิสนั้นขึ้นมาโดยไม่ต้องมีส่วนเกี่ยวข้องของ OS เข้ามายุ่งยากเหมือนกับ Virtual Machine โดยการเปรียบเทียบนั้นจะเป็นไปดังรูปที่ 2.23

Docker นั้นถูกนำมาใช้ในอย่างแพร่หลายในปัจจุบัน เนื่องด้วยความสามารถต่าง ๆ ของมันที่ถูกใจเหล่านักพัฒนา ซึ่งความสามารถเหล่านั้นจะมาช่วยลดระยะเวลาและความซับซ้อนในการนำเซอร์วิสขึ้นไปบนเซิร์ฟเวอร์ให้มีความสะดวกและง่ายมากยิ่งขึ้น ซึ่งเหมาะสำหรับการใช้ร่วมกับไมโครเซอร์วิสอีกด้วยเพราะว่าตัว Docker มีความสามารถในการรันการทำงานของเซอร์วิสที่เกิดการล่มไปแล้วขึ้นมาใหม่ได้โดยอัตโนมัติ รวมถึงการนำโปรแกรมต่าง ๆ ที่ได้เปิดให้ใช้บริการแบบไม่เสียเงินมาวางลงบนเครื่องของนักพัฒนาได้อย่างง่ายดาย



รูปที่ 2.21 เปรียบเทียบโครงสร้างระหว่าง Docker และ Virtual Machine (https://saladpuk.gitbook.io/learn/basic/docker)

2.2.3.1 Docker image

Docker image เป็นตัวต้นแบบของ container ซึ่งภายในจะประกอบด้วยแอปพลิเคชันต่าง ๆ ที่มีการติดตั้งไว้เพื่อให้เซอร์วิสเหล่านั้นทำงาน รวมทั้งมีการตั้งค่าในสิ่งที่จำเป็นต่อแอปพลิเคชันเอาไว้เรียบร้อยแล้ว จากนั้นก็นำมาสร้างเป็น docker image บน registry เพื่อนำไปใช้งาน ซึ่งผู้ใช้งานนั้นจะสามารถสร้าง docker image เป็นของตัวเองได้หรือจะนำ docker image ของคนอื่นมาใช้งานก็ได้ โดยสรุป Image คือชุดคำสั่งของ docker ที่เอาไว้สร้าง environment ในแบบที่เราต้องการ รวมถึงการตั้งค่าต่าง ๆ library ที่เราต้องการ และ source code พื้นฐานที่เราอยากให้มันมี (ทั้งหมดนี่อยู่ในสิ่งที่เรียกว่า Image)

2.2.3.2 Docker container

Docker container เปรียบเสมือนกล่องใบหนึ่งซึ่งจะ docker image ใส่เอาไว้หรือก็คือนำ docker image มาติดตั้งใน docker container นั้นเอง เพื่อให้สามารถใช้งานจากเซอร์วิสที่ต้องการจาก image ตัวนั้น ๆ ได้ โดยใน container แต่ละตัวจะมีการใช้งาน RAM, CPU, ไฟล์ config ต่าง ๆ เป็นของแต่ละ container เอง และผู้ใช้ยังสามารถสั่ง start, stop แก่ container ใด ๆ ที่ต้องการก็ย่อมได้ โดย 1 image นั้นสามารถถูกนำไปสร้าง container กี่ตัวก็ได้

2.2.3.3 Docker file

เป็นชุดคำสั่งที่จะเตรียมให้กับ Docker เพื่อนำไปสร้างเป็น Image โดยคำสั่งพวกนี้ผู้พัฒนาจะสามารถกำหนดได้หมดเลยว่าจะให้มันติดตั้งอะไรให้เราบ้าง เช่น MySQL version อะไร network เป็น

แบบไหนต้องการจะเปิดพอร์ต (port) อะไรบ้างจะให้ทำอะไรกับไฟล์ก่อนหรือไม่ และอีกหลากหลายความสามารถ

การสร้าง image และ container

การสร้าง image นั้นจะสามารถสร้างได้ผ่าน command prompt, powershell หรือ terminal ก็ได้ หลังจากเปิดขึ้นมาแล้วนั้นก็ทำการเข้าไปยังที่อยู่ของโฟลเดอร์ที่ต้องการจะสร้าง image โดยใช้คำสั่ง

- docker build -t <ชื่อของ image> .

ซึ่งถ้าหากจะตรวจสอบว่าว่าใน Docker มี image ใดอยู่บ้างก็สามารถดูได้ดังนี้ ดังแสดงในรูปที่ 2.26

- docker images

รูปที่ 2.22 ตัวอย่างการใช้คำสั่ง docker images

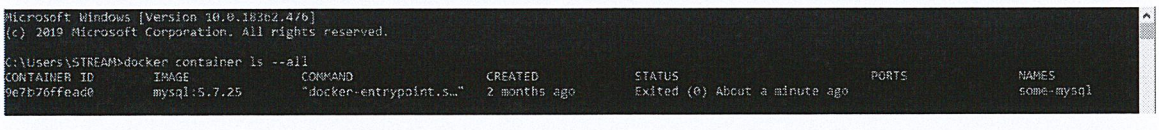
เมื่อมี image แล้วนั้นก็นำไปสร้าง container ด้วยคำสั่งต่อไปนี้ ดังแสดงในรูปที่ 2.26

- docker run -d -p server port : docker port <ชื่อ image>

รูปที่ 2.23 ตัวอย่างการสร้าง docker container

โดยการดู container ที่มีอยู่ก็สามารถดูได้ด้วยคำสั่งดังต่อไปนี้ ดังแสดงในรูปที่ 2.26

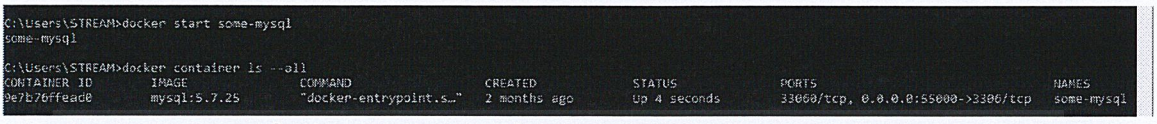
- docker container ls --all



รูปที่ 2.24 ตัวอย่างการดู docker container ที่มีอยู่

จะเห็นได้ว่า container ตัวนี้ไม่ได้ถูกใช้งานอยู่หรือก็คือมีสถานะเป็น stop นั่นเอง โดยเราจะสามารถสั่งให้ container ตัวนี้ให้ทำงานหรือหยุดทำงานด้วยคำสั่ง ดังแสดงในรูปที่ 2.27 และรูปที่ 2.28 ตามลำดับ

- docker start/stop <ชื่อ container>



รูปที่ 2.25 ตัวอย่างการเริ่มการทำงานของ docker container

รูปที่ 2.26 ตัวอย่างการหยุดการทำงานของ docker container

โดยที่กล่าวมาข้างต้นเป็นเพียงแนวทางส่วนหนึ่งในการใช้ Docker เท่านั้น ในความเป็นจริงแล้ว Docker มีคำสั่งและความสามารถอีกมากมายให้ได้ใช้ในการควบคุมการทำงานของแอปพลิเคชันบนเซิร์ฟเวอร์ได้ดีและสะดวกมากยิ่งขึ้น

2.2.4 Git

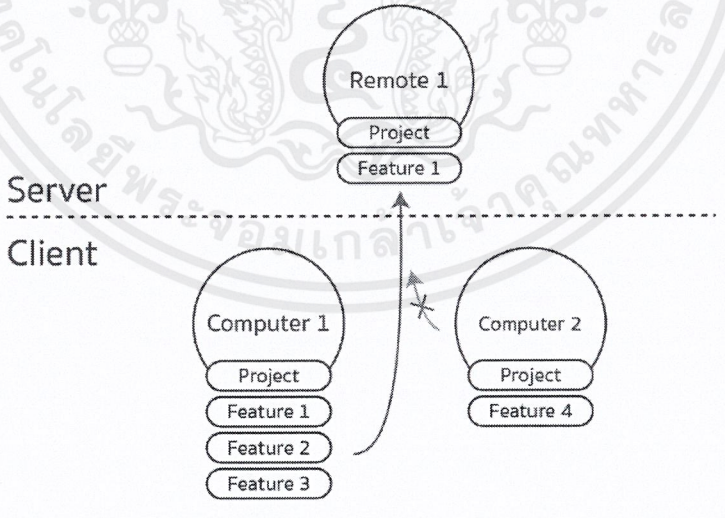
การพัฒาแอปพลิเคชันร่วมกันหลาย ๆ คนนั้นจะเกิดปัญหาที่พบได้บ่อยนั้นคือการที่จะนำโค้ดจากผู้พัฒนาหลาย ๆ คนมารวมกันซึ่งโดยปกติแล้วการจะรวมโค้ดกันนั้นก็คงไม่พ้นวิธีนำแฟลชไดร์ฟไปก็อปปี้ข้อมูลของแต่ละคนมารวมกันโดยวิธีการนี้ก็อาจทำให้ติดไวรัสต่าง ๆ มาด้วยหรือจะเป็นวิธีที่นำโค้ดขึ้นบน Google Drive, One Drive หรือ Dropbox ซึ่งวิธีการนี้ก็สามารถใช้ได้แต่ว่าก็ยังไม่ได้สะดวกอย่างที่ควรจะเป็นมากนัก

การแก้ปัญหาในปัจจุบันก็คือจะนำโปรแกรมที่ถูกเรียกว่า Version Control มาช่วยจัดการการแชร์โค้ดลักษณะนี้ให้สะดวกและง่ายมากยิ่งขึ้นโดย Version Control จะเข้ามาช่วยในการควบคุมการเปลี่ยนแปลงของโค้ดโดยประโยชน์ที่สามารถเห็นได้ชัดจะมีดังนี้

1. เก็บประวัติการทำงานการแก้ไขโค้ดสามารถดูได้ว่ามีการแก้ไขตรงไหนหรือใครเป็นผู้แก้ไข
2. การรวมโค้ดจะเป็นไปได้ง่ายขึ้นเพราะสามารถเทียบโค้ดเก่ากับโค้ดใหม่ได้

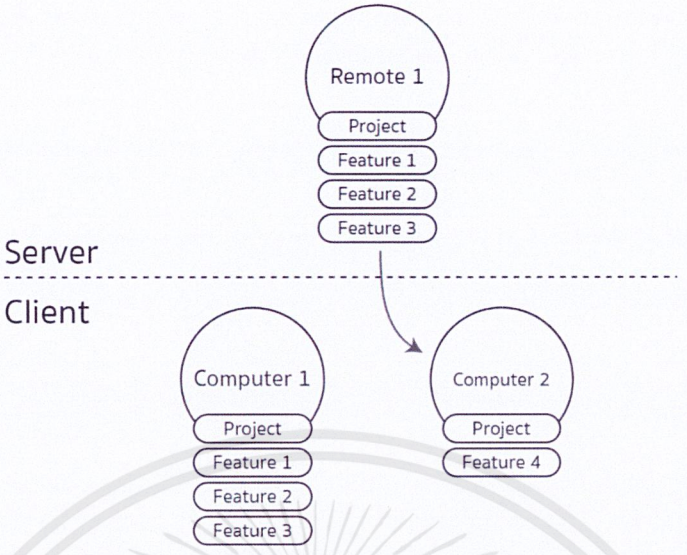
3. เมื่อเกิดข้อผิดพลาดในโค้ดหลังมีการแก้โค้ดก็สามารถหาได้ว่าส่วนไหนที่เพิ่มมเข้ามาทำให้เกิดปัญหา
4. เปรียบเสทือนการทำ backup เอาไว้ซึ่งเวลาโค้ดมีปัญหาก็สามารถ rollback กลับไปใช้โค้ดชุดเก่าได้

Version Control ที่ได้รับการยอมรับและนักพัฒนารู้จักกันเป็นอย่างดีก็คงไม่พ้น Git นั่นเองซึ่ง Git นั้นก็เป็นหนึ่งใน Version Control แบบ Distributed Version Control Systems โดยจะเป็นที่นิยมใช่งานมากในปัจจุบันนี้ Git นั้นจะไม่ผูกขาดกับเซิร์ฟเวอร์เพราะเมื่อเวลาที่ผู้ใช้ไม่ได้เชื่อมต่ออินเทอร์เน็ตก็ยังคงสามารถทำงานต่อได้โดยข้อมูลจะถูกเก็บอยู่เป็นแบบโลคอลภายในเครื่องแต่ถ้ามีความต้องการจะนำโค้ดขึ้นไปบนเซิร์ฟเวอร์แล้วก็มีความจำเป็นต้องเชื่อมต่ออินเทอร์เน็ตเพื่อทำการซิงค์ (sync) ข้อมูล การควบคุมการทำงานเพื่อให้ซิงค์ข้อมูลกันนั้น Git ก็ไม่ได้ทำให้โดยอัตโนมัตินักพัฒนาจำเป็นต้องทำการควบคุมหรือตัดสินใจต่าง ๆ เองเพื่อป้องกันปัญหาเล็กน้อยที่อาจเกิดขึ้นได้ แล้วถ้าหากมีนักพัฒนาจะซิงค์เพื่ออัพเดทโค้ดขึ้นบนเซิร์ฟเวอร์พร้อมกัน Git จะเอาโค้ดของคนที่เอาขึ้นเซิร์ฟเวอร์ไวสุดมาอ้างอิงเป็นเวอร์ชั่นล่าสุด ซึ่ถ้างคนที่สองที่จะเอาโค้ดขึ้นเซิร์ฟเวอร์ก็จะเกิดความยุ่งยากขึ้นนั้นก็คือจะนำโค้ดล่าสุดมาใช้ด้วยรีป่าว หรือจะนำโค้ดล่าสุดมาใช้แล้วปรับให้เข้ากับโค้ดตัวเองก็ได้ทั้งนั้น ซึ่งต้องอยู่ที่ทีมพัฒนาจะตกลงกัน



รูปที่ 2.27 ตัวอย่างการทำงานของ Git

(https://blog.nextzy.me/มาเรียนรู้-git-แบบง่ายๆกันเถอะ-427398e62f82)



รูปที่ 2.28 ตัวอย่างการทำงานของ Git

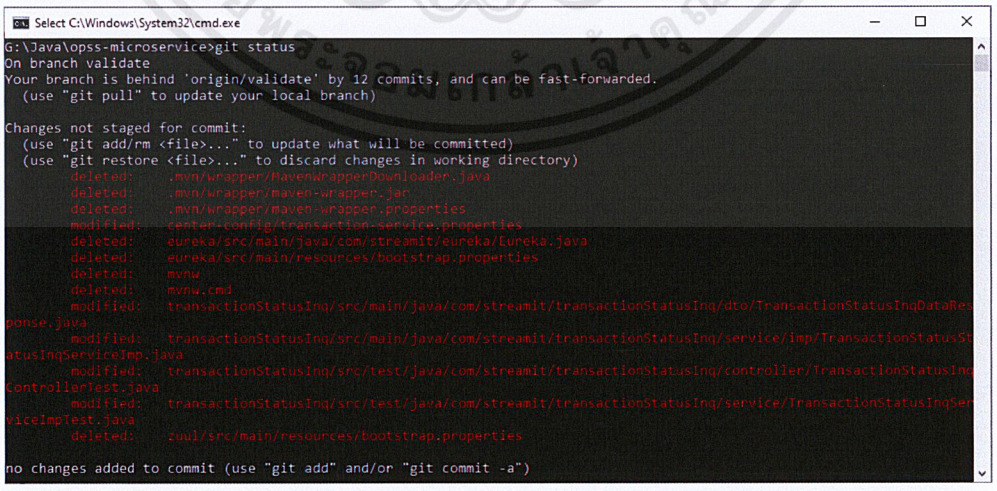
(https://blog.nextzy.me/มาเรียนรู้-git-แบบง่ายๆกันเถอะ-427398e62f82)

การเรียกใช้งาน Git

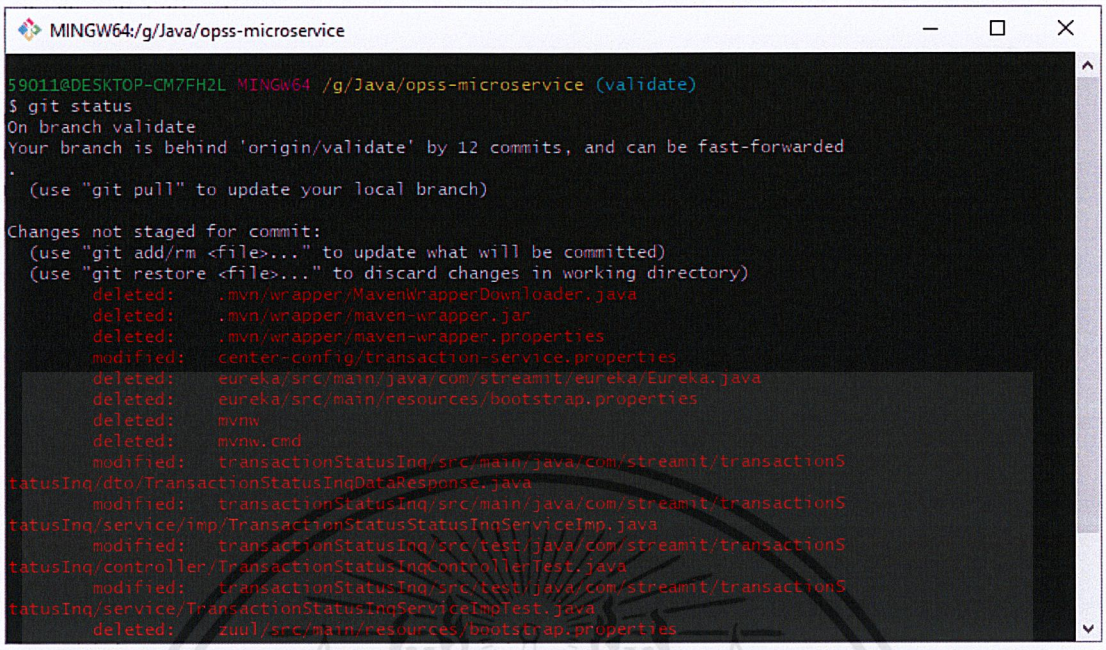
การใช้งาน Git นั้นจะใช้งานได้ 2 แบบหลัก ๆ นั้นก็คือ

1. Command Line

ควบคุมการเปลี่ยนแปลงโดยทำการพิมพ์คำสั่งผ่าน Git Terminal หรือ ผ่าน Command Prompt โดยตรง ดังแสดงในรูปที่ 2.32 และ 2.33 ตามลำดับ



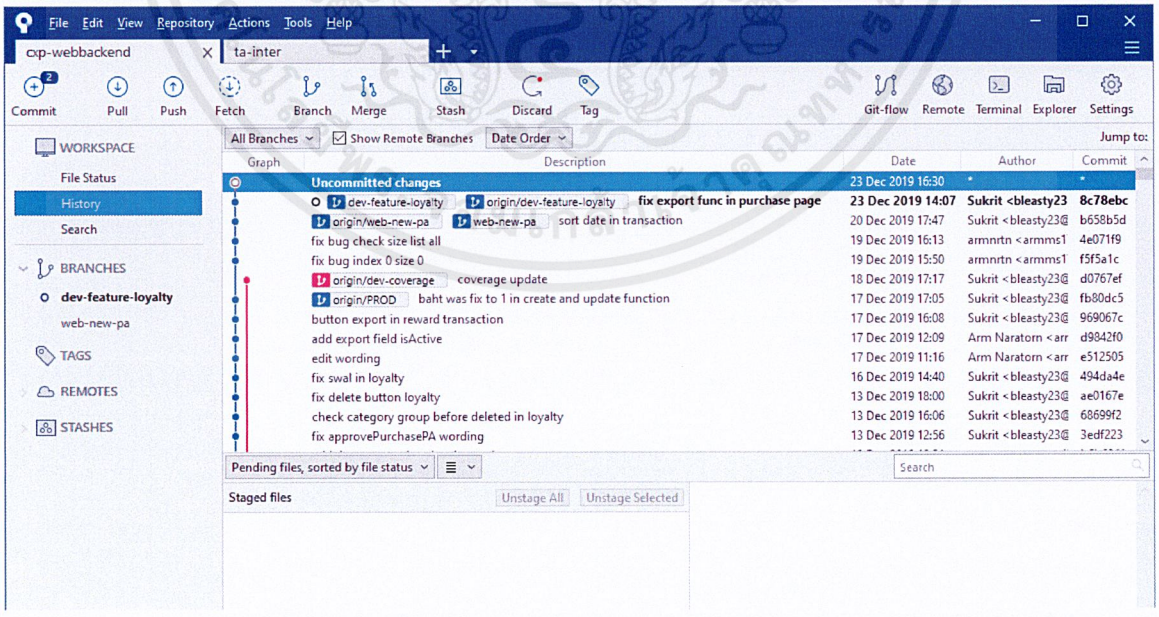
รูปที่ 2.29 ใช้งาน Git ผ่าน Command Prompt



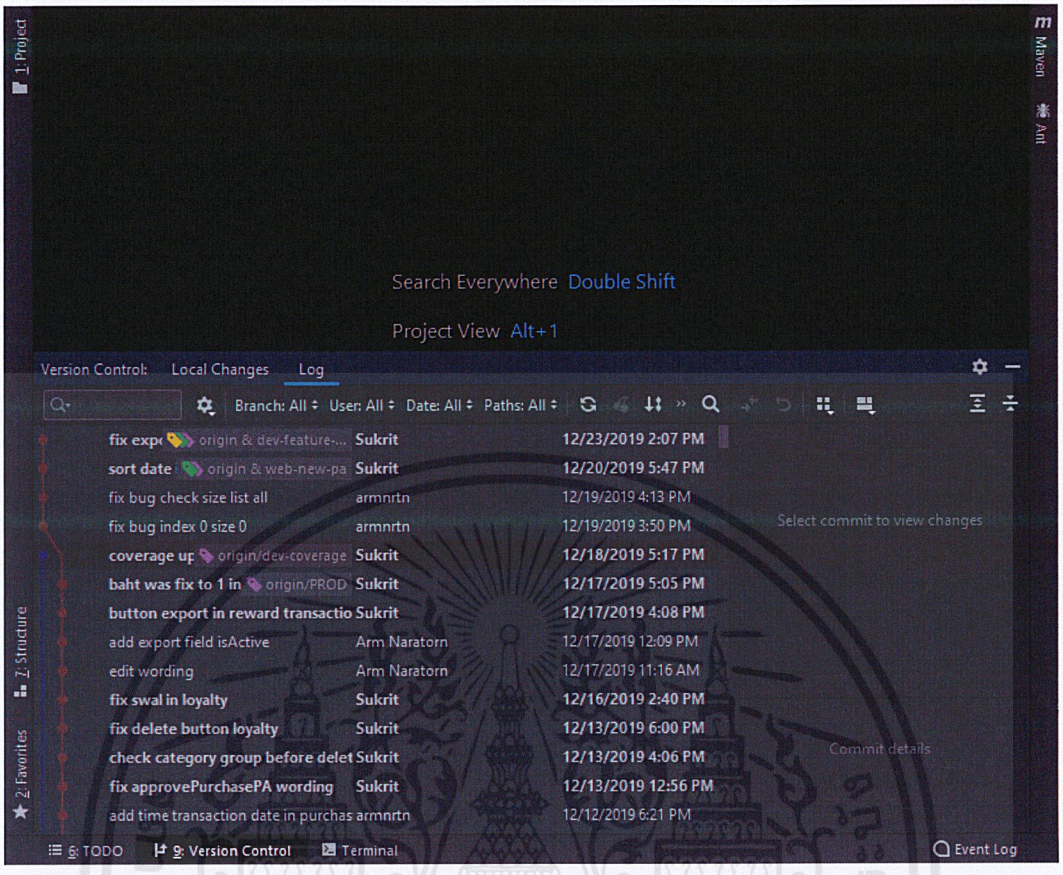
รูปที่ 2.30 ใช้งาน Git ผ่าน Git Terminal

2. GUI

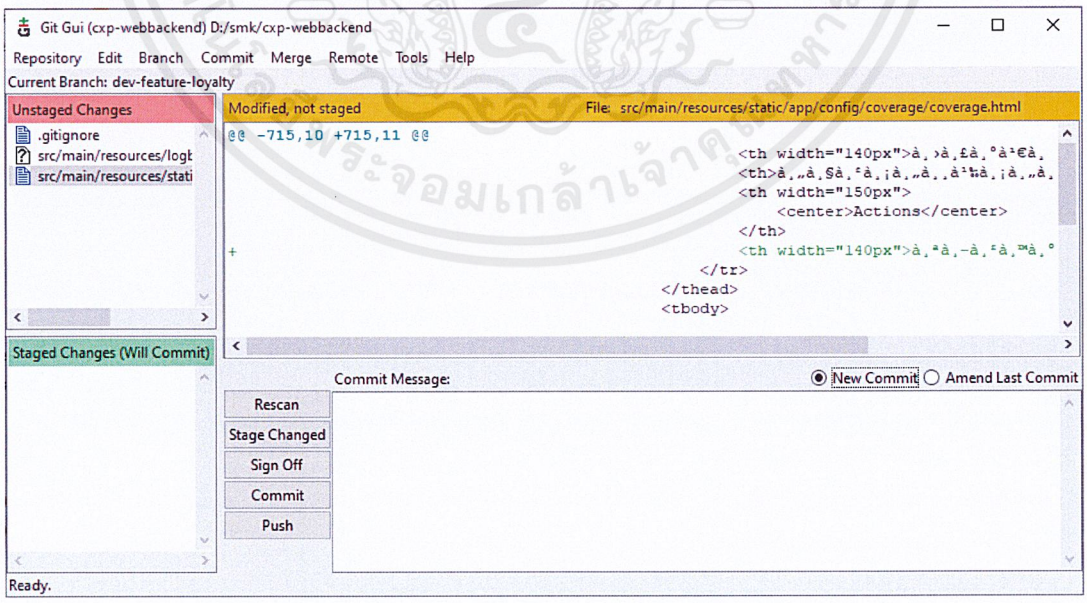
ใช้โปรแกรมที่มีหน้าตาแสดง User Interface (UI) แบบที่เข้าใจได้ง่ายโดยโปรแกรม Git GUI ที่นิยมใช้จะมีเช่น SourceTree ดังแสดงในรูปที่ 2.34 และ IDE ในที่นี้จะใช้ IntelliJ ดังแสดงในรูปที่ 2.35 รวมไปถึง GitHub Desktop ดังแสดงในรูปที่ 2.36 เป็นต้น



รูปที่ 2.31 ใช้งาน Git บนโปรแกรม SourceTree



รูปที่ 2.32 ใช้งาน Git บนโปรแกรม IntelliJ



รูปที่ 2.33 ใช้งาน Git บนโปรแกรม Git Desktop

คำสั่ง Git ที่จำเป็น

1. Clone เป็นการดึง source code จากเซิร์ฟเวอร์มาเพื่อใช้พัฒนาโค้ดร่วมกัน

   - คำสั่งที่ใช้: git clone <GIT_REPOSITORY_URL>

2. Add เมื่อไฟล์ใดถูกเปลี่ยนแปลงก็จำเป็นต้องใช้คำสั่งนี้เลือกว่าจะเอาไฟล์ใดบ้างที่ถูกเปลี่ยนนำไปทำการ commit ต่อไป

   - คำสั่งที่ใช้: git add . ใช้คำสั่งนี้เพื่อเลือกทุกไฟล์ หรือgit add <FILE_NAME> โดยใช้คำสั่งนี้เพื่อเลือกทีละไฟล์

3. Commit เป็นการบอกว่าโค้ดที่อยู่ภายในเครื่องเราตอนนี้เราพอใจกับมันแล้วซึ่งการ commit แต่ละครั้งก็จะสามารถดูได้

   - คำสั่งที่ใช้: git commit -m "comment about this commit"

4. Push เป็นการนำโค้ดที่ถูก commit หรือโค้ดในเครื่องของเราขึ้นไปบนเซิร์ฟเวอร์เพื่อทำการปรับเปลี่ยนโค้ดใหม่แต่ถ้าไม่ได้มีการ commit อะไรเพิ่มเติมจากการ push ครั้งที่แล้วก็จะไม่มีอะไรเกิดขึ้น

   - คำสั่งที่ใช้: git push

5. Pull เป็นการอัพเดทโค้ดของเราให้ตรงกับกับโค้ดที่อยู่บนเซิร์ฟเวอร์

   - คำสั่งที่ใช้: git pull

2.3 ภาษาและเฟรมเวิร์กที่ใช้พัฒนาในโครงงาน

2.3.1 ภาษา

2.3.1.1 Java

ภาษาจาวา (Java programming language) เป็นภาษาโปรแกรมเชิงวัตถุ (Object Oriented Programming) ซึ่งถูกพัฒนามาเพื่อใช้งานแทนภาษาซีพลัสพลัส (C++) โดยที่มีความสามารถหลากหลาย

มากยิ่งขึ้น แม้ว่าชื่อของภาษาจาวา กับภาษาจาวาสคริปต์นั้นจะคล้ายคลึงกัน แต่จริง ๆ แล้วทั้งสองภาษาไม่ได้มีความเกี่ยวข้องใด ๆ ต่อกันเลย ปัจจุบันมาตรฐานของภาษาจาวาดูแลโดย Java Community Process ซึ่งเป็นกระบวนการอย่างเป็นทางการ ที่อนุญาตให้ผู้ที่สนใจเข้าร่วมกำหนดความสามารถในจาวาแพลตฟอร์มได้

โดยโปรแกรมต่าง ๆ ที่ถูกสร้างขึ้นนั้นจะถูกสร้างภายในคลาส (Class) และโปรแกรมเหล่านั้นก็จะถูกเรียกว่าเมทธอด (Method) หรือพฤติกรรม (Behavior) โดยจะมีสถานะ (State) และรูปพรรณ (Identity) ประจำพฤติกรรมนั้น ๆ ซึ่งคลาสนั้นจะถูกเรียกว่าวัตถุ โดยแต่ละวัตถุก็จะมีพฤติกรรมที่แตกต่างต่างกันออกไป เมื่อมองภาพรวมของโปรแกรมนั้นก็เหมือนกับการนำวัตถุหลาย ๆ ตัว หรือคลาสมารวมไว้เป็นกลุ่มเดียวกัน ซึ่งการเขียนโปรแกรมในลักษณะนี้นั้นแต่ละคลาสอาจจะมีคุณสมบัติของ การปกป้อง (Encapsulation) การสืบทอด (Inheritance) การพ้องรูป (Polymorphism) หรือ การเรียกเกินกำลัง (Overloading) นั่นก็คือแนวคิดการเขียนโปรแกรมเชิงวัตถุนั่นเอง จึงสรุปได้ว่าคลาสนั้นคือต้นแบบของการสร้างวัตถุ

จุดมุ่งหมาย

จุดมุ่งหมายหลัก 4 ประการ ในการพัฒนาจาวามีทั้งหมด 4 อย่างดังนี้

1. ใช้ภาษาโปรแกรมเชิงวัตถุ (OOP Concepts)
2. ไม่ขึ้นกับแพลตฟอร์มใด ๆ (สถาปัตยกรรม และ ระบบปฏิบัติการ)
3. เหมาะกับการใช้ในระบบเครือข่ายเพราะมีไลบรารีสนับสนุน
4. เรียกใช้งานจากระยะไกลได้อย่างปลอดภัย

แนวคิดของการโปรแกรมเชิงวัตถุ (OOP Concepts)

1. การปกป้อง (Encapsulation) คือ การรวมกลุ่มของข้อมูลให้รวมกันเป็นเหมือนแคปซูลหนึ่งอัน เพื่อการปกป้อง และเลือกตอบสนอง
2. การสืบทอด (Inheritance) คือ การยอมให้นำไปใช้ หรือเขียนขึ้นมาทดแทนของเดิม
3. การพ้องรูป (Polymorphism) ซึ่งมี 2 หลักการที่สำคัญคือ โอเวอร์โหลดดิ้ง (Overloading) คือการที่มีชื่อเมทธอดเดียวกัน แต่รายการตัวแปร (Parameter List) ต่างกัน และโอเวอร์ไรดิ้ง (Overriding) คือการมีชื่อและตัวแปรเหมือนกันแต่ต้องการเขียนพฤติกรรม (Behavior) ขึ้นมาใหม่แทนอันที่มีอยู่

ข้อดี

1. ภาษาจาวาสนับสนุนการเขียนโปรแกรมเชิงวัตถุจึงเหมาะสำหรับพัฒนาระบบที่มีความซับซ้อน การพัฒนาโปรแกรมแบบวัตถุจะช่วยให้เราสามารถใช้คำหรือชื่อต่าง ๆ ที่มีอยู่ในระบบงานนั้นมาใช้ในการออกแบบโปรแกรมได้ง่ายยิ่งขึ้น
2. โปรแกรมที่เขียนขึ้นโดยใช้ภาษาจาวาจะสามารถนำไปใช้ในระบบปฏิบัติการที่ต่างกันได้โดยไม่จำเป็นต้องแก้โค้ดเช่น หากเขียนโปรแกรมบนเครื่อง Sun โปรแกรมนั้นก็สามารถถูก compile และ run บนเครื่องพีซีธรรมดาได้
3. ภาษาจาวาจะตรวจสอบโปรแกรมในขณะ compile time และ runtime ทำให้ช่วยลดข้อผิดพลาดที่อาจจะเกิดขึ้นได้ในโปรแกรม และการทำ debug แก้โปรแกรมก็ทำได้ง่าย
4. ภาษาจาวามีความซับซ้อนน้อยกว่าภาษา C++ จึงทำให้การพัฒนาโปรแกรมเป็นไปได้ง่ายกว่าและข้อผิดพลาดน้อยลง
5. มีความปลอดภัยสูง
6. มี IDE, application server, และ library ต่าง ๆ มากมายสำหรับจาวาที่เราสามารถใช้งานได้โดยไม่ต้องเสียค่าใช้จ่าย

ข้อเสีย

1. ภาษาจาวาจะทำงานได้ช้ากว่า native code (โปรแกรมที่ compile ให้อยู่ในรูปของภาษาเครื่อง) หรือโปรแกรมที่เขียนขึ้นด้วยภาษาอื่น อย่างเช่น C หรือ C++ นั้นเป็นเพราะว่าภาษาจาวาจำเป็นต้องถูกแปลงเป็นภาษากลางก่อนจากนั้นเมื่อโปรแกรมทำงานคำสั่งของภาษากลางนี้ก็จะถูกเปลี่ยนเป็นภาษาเครื่องอีกทีละคำสั่งใน runtime ทำให้ทำงานช้ากว่า native code ซึ่งอยู่ในรูปของภาษาเครื่องแล้วตั้งแต่ compile เพราะฉะนั้นโปรแกรมไหนที่ต้องการความรวดเร็วจึงไม่เหมาะกับภาษาจาวา ซึ่งข้อเสียก็พอเป็นที่รับได้สำหรับนักพัฒนาที่ชอบในข้อดีของภาษาจาวา

2.3.1.2 MySQL

MySQL เป็นโปรแกรมจัดการฐานข้อมูล Relational Database Management System (RDBMS) เป็นฐานข้อมูลที่สามารถ เก็บข้อมูล ดึงข้อมูล แก้ไขข้อมูล และลบข้อมูลได้ MySQL มีความสามารถให้ผู้ใช้งานเข้าใช้งานด้านข้อมูลได้เป็นจำนวนมากในเวลาเดียวกัน และมีการเข้าถึงข้อมูลที่

รวดเร็ว รวมถึงมีการกำหนดการเข้าใช้งานของผู้ใช้งานในลักษณะที่แต่ละคนจะแตกต่างกันออกไปตามความเหมาะสมและความปลอดภัย

ปัจจุบัน MySQL ได้ใช้งานแพร่หลายโดยเป็นโปรแกรมที่เปิดให้ใช้งานได้แบบไม่เสียค่าบริการ แต่ก็มีแบบคิดค่าบริการให้ใช้งานด้วยเช่นกัน โดยคุณสมบัติจะแตกต่างกันออกไป โดย MySQL นั้นมีความสามารถในการสนับสนุนการทำงานได้เกือบทุกระบบปฏิบัติการณ์ อาทิเช่น Windows และ Linux เป็นต้น จึงทำให้ MySQL เป็นที่ถูกใจและไว้วางใจต่อนักพัฒนาจำนวนมาก นอกจากนั้น MySQL ยังเป็นที่นิยมในการนำไปใช้งานกับ Web Application เป็นอย่างมากและยังทำงานร่วมกับเครื่องบริการเว็บ (Web Server) เพื่อให้บริการแก่ภาษาสคริปต์ที่ทำงานฝั่งเครื่องบริการ (Server-Side Script) เช่น ภาษา php ภาษา aps.net หรือภาษาเจเอสพี อีกทั้งยังสามารถทำงานร่วมกับโปรแกรมประยุกต์ (Application Program) เช่น ภาษาวิชวลเบสิกดอทเน็ต ภาษาจาวา หรือภาษาซีชาร์ป ได้อีกด้วย

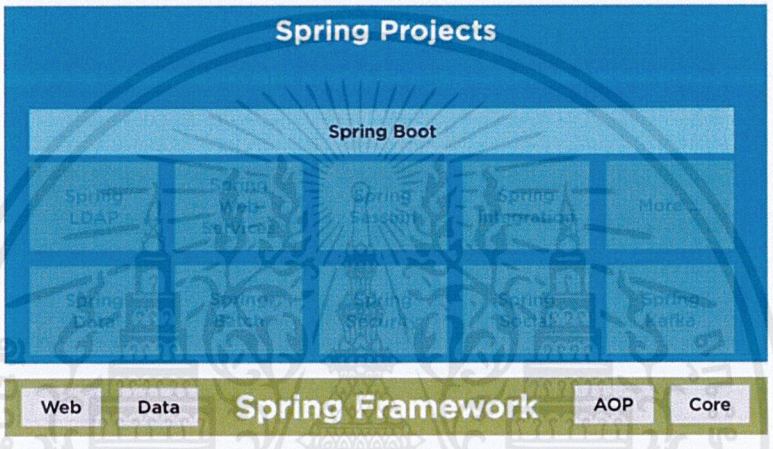
ฐานข้อมูลนั้นมีลักษณะที่เป็นโครงสร้างของการเก็บรวบรวมข้อมูลโดยการที่จะเพิ่มข้อมูล อ่านข้อมูล หรือประมวลผลข้อมูลที่เก็บในฐานข้อมูล นั้นจะต้องมีระบบการจัดการฐานข้อมูลเป็นเป็นตัวจัดการโดยจะทำหน้าที่เป็นตัวกลางในการจัดการกับข้อมูลที่อยู่ภายในฐานข้อมูล ทั้งสำหรับการใช้งานเฉพาะและรวมไปถึงการรองรับการทำงานของแอปพลิเคชันอื่น ๆ ที่ต้องการติดต่อเข้าถึงข้อมูลที่อยู่ภายในฐานข้อมูล เพื่อความสะดวกในการจัดการกับข้อมูลจำนวนมาก ดังนั้นตัว MySQL จึงเข้ามาทำหน้าที่เป็นทั้งตัวฐานข้อมูล และระบบจัดการฐานข้อมูล โดยฐานข้อมูล MySQL นั้นจะมีรูปแบบเป็นแบบ relational ซึ่งจะทำการเก็บข้อมูลทั้งหมดในรูปแบบของตารางแทนการเก็บลงไปในไฟล์เดียวส่งผลให้การทำงานมีความรวดเร็วและมีความยืดหยุ่น โดยแต่ละตารางที่เก็บข้อมูลจะมีความสัมพันธ์ที่สามารถเชื่อมโยงเข้าหากันทำให้สามารถรวม หรือจัดกลุ่มข้อมูลได้ตามต้องการ โดยจะใช้ภาษา SQL ที่เป็นส่วนหนึ่งของโปรแกรม MySQL ซึ่งเป็นภาษามาตรฐานในการติดต่อเข้าถึงฐานข้อมูล

2.3.2 เฟรมเวิร์ก

2.3.2.1 Spring Boot

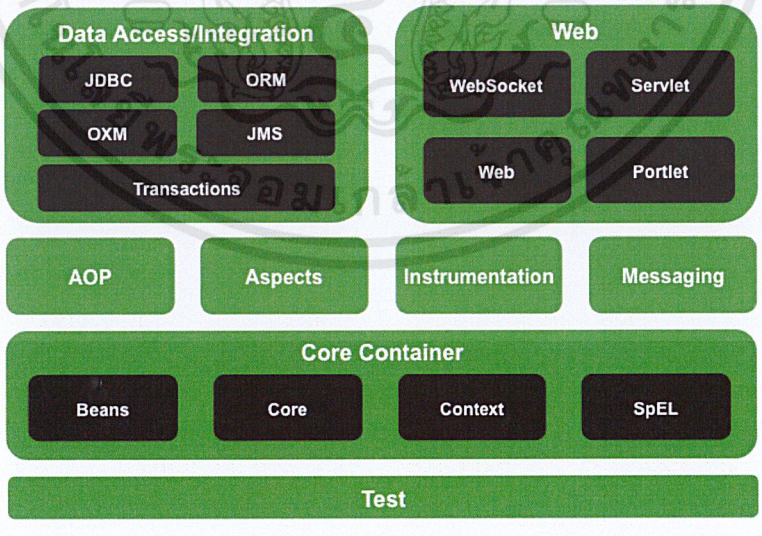
Spring Boot คือ เฟรมเวิร์กสำหรับภาษาจาวาที่เปิดให้ใช้งานโดยไม่เสียค่าบริการใด ๆ ส่วนใหญ่แล้วนักพัฒนาจะนำมาใช้ในการสร้าง REST API โดยตัว spring boot ถูกพัฒนามาจาก spring framework ซึ่ง spring boot ไม่ได้มาแทนที่ตัว spring framework แต่อย่างใดเพราะว่าการทำงานของ spring boot นั้นจะเป็นเหมือนแค่เพียงนำตัว spring boot ไปวางไว้ชั้นบนสุดของโครงสร้าง spring framework แบบเก่า ซึ่งจะทำให้ตัว spring boot นั้นสามารถทำการติดตั้ง (set up) และการตั้งค่า

(Configure) ได้โดยง่ายดายกว่าตัว spring framework แบบเก่ามาก เพราะตัว spring framework แบบเก่านั้นจำเป็นต้องทำการตั้งค่าด้วยตัวผู้ใช้งานเองทั้งหมด จึงทำให้มีจำนวนไฟล์ในการตั้งค่าที่ค่อนข้างเยอะเป็นจำนวนมากเช่น ไฟล์ .xml ดังนั้นผู้พัฒนาเลยคิดค้น spring boot เพื่อมาแก้ปัญหาในจุดนี้ อีกทั้งยังมี Embed Tomcat, Jetty or Undertow อยู่ภายใน จึงทำให้ผู้พัฒนาไม่จำเป็นต้องทำการนำไฟล์ WAR ขึ้นเซิร์ฟเวอร์ แต่สามารถแปลงโค้ดที่เขียนให้อยู่ในรูปของ JAR และนำขึ้นบนเซิร์ฟเวอร์ได้เลย โดยปกติแล้วนักพัฒนาจะนำ spring boot มาใช้เพื่อพัฒนาในการทำ REST API โดยโครงสร้างของ spring boot และ spring จะมีโครงสร้างดังรูปที่ 2.40 และรูปที่ 2.41 ตามลำดับ



รูปที่ 2.34 โครงสร้าง Spring Boot Framework

(https://dzone.com/articles/what-is-spring-boot)



รูปที่ 2.35 โครงสร้าง Spring Framework

(http://avaldes.com/introduction-to-spring-framework-ioc-and-injection/)

จากรูปดังกล่าวจะเห็นได้ว่า spring boot นั้นจะมีความสามารถที่หลากหลายให้นักพัฒนาเลือกใช้ ซึ่งไม่ได้จำเป็นต้องนำมาทุกส่วนใช้งานทั้งหมดโดยจะขึ้นอยู่กับโปรเจคว่าควรจะนำส่วนไหนมาใช้งานบ้างโดยแต่ละส่วนจะมีความสามารถดังนี้

Core Container

ประกอบไปด้วย 4 โมดูลดังนี้

1. Core module เป็นเหมือนองค์ประกอบพื้นฐานที่สำคัญของเฟรมเวิร์กและยังรวมไปถึงการทำ Inversion of Control (IoC) และ Dependency Injection (DI)
2. Bean module ให้บริการ BeanFactory ในรูปแบบ factory pattern
3. Context module ถูกสร้างโดย Core และ Beans modules โดยโมดูลตัวนี้จะทำหน้าที่เป็นการในการเข้าถึงออบเจ็คต่าง ๆ และยังสามารถกำหนดค่าต่าง ๆ ให้แก่ออบเจ็คเหล่านั้น
4. SpEL module (Spring Expression Language) เป็นภาษาของพิเศษของ spring framework ที่ใช้จัดการกับ query และควบคุมออบเจ็คในขณะ runtime

Data Access/Integration

ประกอบไปด้วย 5 โมดูลดังนี้

1. JDBC module ทำให้การเชื่อมต่อ JDBC โดยที่ไม่จำเป็นต้องเขียนโค้ดให้ยุ่งยากเพียงแค่ใส่ dependency สำหรับติดต่อฐานข้อมูลใด ๆ ที่ต้องการใช้แล้วทำการตั้งค่าเกี่ยวกับข้อมูลฐานข้อมูลก็สามารถติดต่อกับฐานข้อมูลได้ทันที
2. ORM module เป็นโมดูลที่จะทำการ mapping object ของภาษา java ให้สามารถติดต่อสื่อสารกับฐานข้อมูลได้เข้าใจ ซึ่งจะประกอบด้วย JPA, JDO, Hibernate, และ iBatis
3. OXM module ใช้ในการ mapping object กับภาษา XML ให้เข้าใจกัน ซึ่งต้องใช้ใน JAXB, Castor, XMLBeans, JiBX และ XStream
4. Java Messaging ให้บริการ JMS (Java Messaging Service) module โดยจะมีฟังก์ชันสำหรับการรับส่งข้อความ

5. Transaction module สนับสนุน transaction management สำหรับ class ซึ่งไม่ได้ใช้ได้แค่เฉพาะกับคลาสที่ได้ implement special interface นี้เท่านั้นแต่จะสามารถใช้ POJOs (plain old Java objects) ทุกตัว

Web layer

ประกอบด้วย 4 โมดูลดังนี้

1. Web module จะจัดหาให้บริการการทำงานพื้นฐานของเว็บทั่วไป เช่น multipart file-upload รวมไปถึงการสร้าง Ioc container
2. Web-MVC module คือ model-view-controller (MVC) ของ spring framework ซึ่งใช้ในการพัฒนาเว็บแอปพลิเคชัน
3. Web-Socket module รองรับการทำงานของโปรเจคที่ต้องการใช้ websocket โดยจะให้บริการการสื่อสารสองทางระหว่างไคลเอนต์และเซิร์ฟเวอร์ใน web application
4. Web-Portlet รองรับการทำงาน MVC บน portlet โดยฟังก์ชันต่าง ๆ จะคล้ายกับตัว web-socket

Miscellaneous module

ประกอบไปด้วย 5 โมดูลดังนี้

1. AOP module เป็นแนวคิด aspect-oriented programming (AOP) ซึ่งมีไว้ใช้สำหรับต้องการที่จะแทรกการทำงานในช่วงใดช่วงนึงในโปรแกรมก็ได้โดยไม่ต้องแก้โค้ดเก่าให้ยุ่งยากหรือก็คือการ cross-cutting นั่นเอง
2. Aspects module เป็นเฟรมเวิร์กสำหรับใช้พัฒการเขียน AOP ในภาษา java
3. Instrumentation module เป็นโมดูลที่ใช้จัดการกับคลาส
4. Messaging module เป็นโมดูลที่เป็นพื้นฐานสำหรับของแอปพลิเคชันแบบ messaging-based รวมไปถึงแอนโนเทชัน (Annotation) ต่าง ๆ เพื่อใช้สำหรับการ mapping ระหว่าง message กับ method ทำงานคล้ายกับ Spring MVC
5. Test module เป็นโมดูลที่รองรับการทดสอบแบบ integration และ unit test โดยใช้ JUnit หรือ TestNG เฟรมเวิร์ก

การสร้างโปรเจค

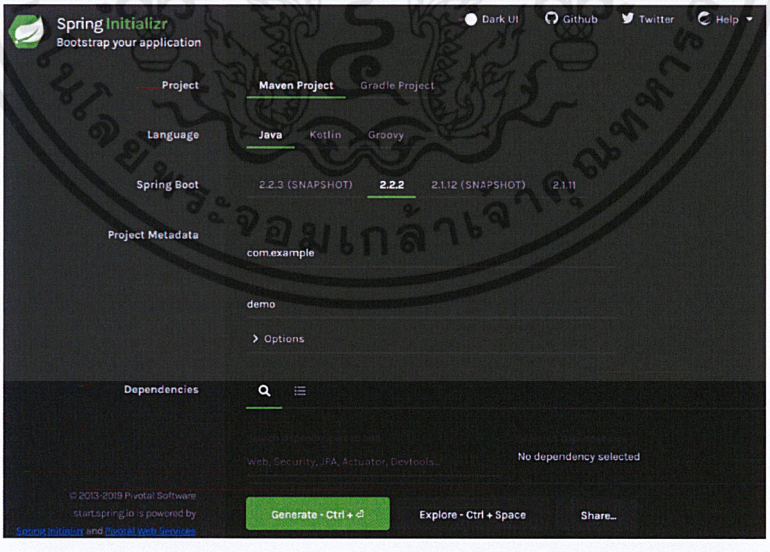
จากที่ได้กล่าวมาข้างต้นว่า spring boot ได้ลดความซับซ้อนการ configure ของโปรเจคให้มีความง่ายมากยิ่งขึ้น ทำให้การจะนำไลบรารี่ข้างนอกมาใช้ปกติแล้วก็จำเป็นต้องดาวน์โหลดหรือกำหนดค่าในการตั้งค่าต่าง ๆ ของไลบรารี่ที่เราจะนำมาใช้ แต่ตัว spring boot ไม่จำเป็นต้องทำแบบนั้นเพียงแค่สร้างโปรเจคเป็นแบบ Maven หรือ Gradle โดยถ้าต้องการจะเพิ่ม JAR Files หรือ dependency อื่น ๆ ให้แก่โปรเจคก็จะสามารถนำ dependency ไปวางไว้ในไฟล์ pom.xml หรือ build.gradle ตามลำดับ โดยไม่จำเป็นต้องไปทำการ configure อะไรเพิ่มเติมอีก การสร้างโปรเจค spring boot จะทำได้โดยสองวิธีดังนี้

1. ใช้ Integrated Development Environment (IDE) ในการสร้าง

IDE ที่ใช้พัฒนาภาษาจาวาที่เป็นที่รู้จักดีก็คงจะเป็น Eclipse, IntelliJ และ Netbeans ซึ่งวิธีการสร้างของแต่ละ IDE ก็จะไม่แตกต่างกันมากนักสามารถสร้างและใช้งานได้อย่างง่ายดาย

2. สร้างโปรเจคผ่านเว็บไซต์

อีกทางเลือกนึงที่สะดวกต่อการสร้าง project ของ spring boot นั้นสามารถสร้างได้ผ่านทางเว็บไซต์ https://start.spring.io โดยหน้าเว็บไซต์จะมีรูปแบบหน้าตาดังแสดงในรูปที่ 2.42



รูปที่ 2.36 หน้าการสร้างโปรเจค Spring Boot ผ่านทางเว็บไซต์

ซึ่งจะสามารถเลือกภาษาที่ใช้ รูปแบบของโปรเจคครวมถึง dependency ต่าง ๆ ที่ต้องการนำเข้ามาใช้ได้โดยง่าย เมื่อทำการเลือกทุกอย่างเสร็จเรียบร้อยก็ทำการบันทึกออกมาจะได้เป็นไฟล์ .zip จากนั้นนำไปแตกไฟล์ก็จะได้โปรเจคมาใช้งาน

การทำงานของ Spring Framework

การทำงานของ spring framework นั้นจะใช้แนวคิด Inversion of Control (IoC) และ Dependency Injection (DI) ซึ่งจะทำให้ตัวโค้ดที่เราเขียนมีความยืดหยุ่นไม่ผูกมัดกันมากนัก (lossely coupled) ซึ่งแนวคิดของ IoC จะถูกนำมาใช้ใน spring framework จะถูกรียกว่า Spring IoC container ซึ่งตอนเริ่มทำงานจะทำการโหลดไฟล์ configure และ bean ต่าง ๆ เก็บไว้ใน container ซึ่งจะผูกความสัมพันธ์ของออบเจ็คให้เองอัติโนมัติโดยที่ไม่ต้องสร้างออบเจ็คขึ้นมาแล้วเซ็ตค่าต่าง ๆ ให้เหมือนปกติแต่อย่างใด และจะทำงานร่วมกับแนวคิดของ DI ซึ่งก็คือการนำคลาสของออบเจ็คที่ได้ถูกสร้างเป็น bean ไว้ไปเซ็ตค่าให้แก่อีกออบเจ็คที่มีการเรียกใช้ bean นี้โดยการเซ็ตค่าผ่านทาง constructor, setter method หรือ field injection ดังแสดงในรูปที่ 2.43 ซึ่งการทำงานที่ให้ไคลเอนต์สามารถติดต่อกับตัวเซอร์วิสนั้นจะทำโดยใช้หลักการของ Spring MVC ในการติดต่อแลกเปลี่ยนข้อมูลไปมาระหว่างทั้งสองฝั่งได้อยากสะดวกและรวดเร็ว โดยการเริ่มต้นการทำงานของเฟรมเวิร์กนั้นจะทำงานโดยเริ่มจากแอนโนเทชันของทาง Spring Framework นั่นก็คือ @SpringBootApplication ซึ่งแอนโนเทชันตัวนี้จะรวมการทำงานของสามแอนโนเทชันไว้ด้วยกันนั่นก็คือ

1. @Configuration

โดยจะประกาศ annotation ตัวนี้เมื่อต้องการที่จะกำหนดการตั้งค่าต่าง ๆ ด้วยตนเอง

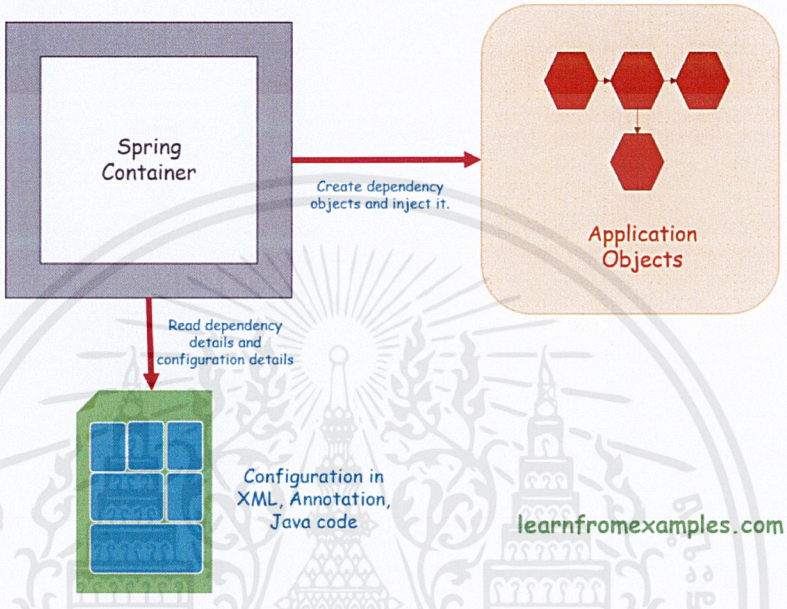
2. @EnableAutoConfigutation

โดยจะประกาศ annotation ตัวนี้เพื่อให้ตัว spring boot มีการ configure application ให้อัตโนมัติ โดยจะพิจารณาจาก dependencies (.JAR) ในโปรเจค

3. @ComponentScan

โดยจะประกาศ annotation ตัวนี้เพื่อค้นหาคลาสที่เกี่ยวข้องกับตัวโปรเจคทั้งหมด หรือก็คือการค้นหา bean ที่ถูกประกาศไว้ในแพคเกจนั้นและแพคเกจย่อย ๆ

เมื่อโปรแกรมได้ค้นหาชิ้นส่วนต่าง ๆ ที่ได้ถูกออกแบบไว้ภายในระบบเสร็จสิ้นแล้วนั้นก็จะนำชิ้นส่วนต่าง ๆ ที่จำเป็นในการทำงานของเซอร์วิสมาประกอบเข้าด้วยกัน หลังจากที่ทำการรันโปรแกรมแล้วนั้นก็จะพร้อมให้ทางไคลเอนต์สามารถติดต่อเข้าใช้งานกับทางเซอร์วิสได้ทันที



รูปที่ 2.37 การทำงานของ Spring IoC
(https://www.zoltanraffai.com/blog/how-does-spring-work-internally/)

Bean

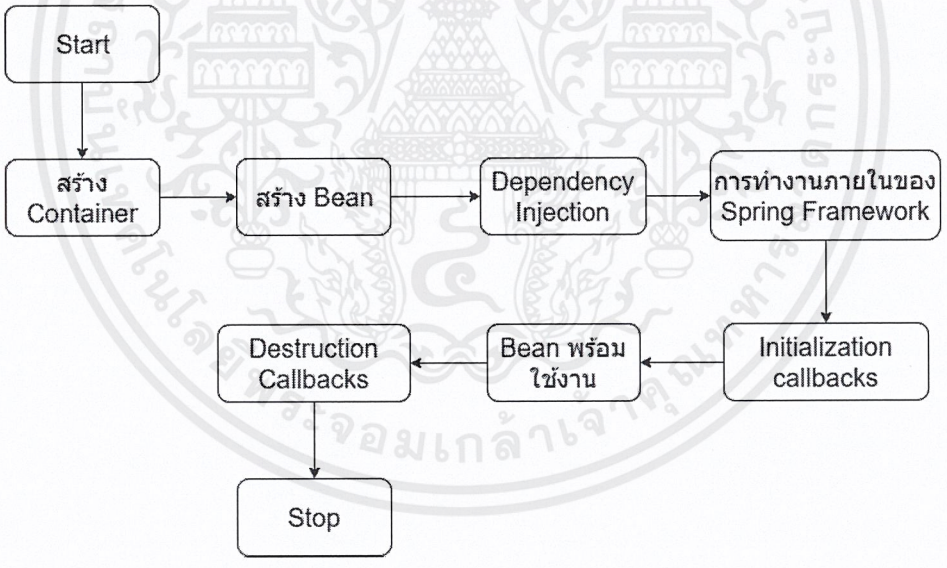
อย่างที่ได้กล่าวไปข้างต้นว่า bean นั้นจะถูกสร้างแล้วเก็บไว้ใน IoC container ของ spring frameworkโดยวิธีการสร้าง bean นั้นก็มีด้วยกัน 3 วิธีได้แก่

- XML Based Configuration เป็นการ config ผ่าน XML files
- Annotation Based Configuration เป็นการ config โดยใช้ Annotation
- Java Based Configuration เป็นการ config ด้วยโค้ดของภาษา Java

ซึ่ง bean นั้นก็จะสามารถถูกกำหนดขอบเขต (Scope) ในการให้ใช้งานได้โดยแบ่งออกเป็น 5 ขอบเขตได้แก่

1. Singleton หมายถึง bean หนึ่งตัวจะมีแค่ 1 instance ใน container เมื่อมีการเรียกใช้งาน bean ตัวดังกล่าวก็จะใช้งาน instance เดิมที่ได้ถูกสร้างเอาไว้แล้ว โดยจะไม่มีการสร้าง instance ใหม่ขึ้นมา
2. Prototype หมายถึง bean จะถูกสร้าง instance ขึ้นมาใหม่ทุกตรั้งที่ bean ชื่อนี้มีการถูกเรียกใช้งาน
3. Request หมายถึง bean จะถูกสร้างขึ้นสำหรับการใช้งาน HTTP request
4. Session หมายถึง bean จะถูกสร้างขึ้นสำหรับการใช้งาน HTTP session
5. Global-session หมายถึง bean จะถูกสร้างขึ้นสำหรับการใช้งาน HTTP session ที่มีการเข้าถึงได้แบบ global

ซึ่งถ้า scope ของ bean ไม่ได้ถูกกำหนดโดยผู้ใช้งาน ก็จะมีค่าเริ่มต้นเป็น singleton โดย bean เองนั้นก็จะมีวงจรชีวิตการทำงานของมันโดยจะถูกเรียกว่า bean life cycle ซึ่งการทำงาน โดยการทำงานของ bean life cycle จะเป็นดังรูปที่ 2.44



รูปที่ 2.38 การทำงานของ Bean

จากรูปจะเห็นได้ว่านักพัฒนาสามารถเพิ่มการทำงานของ bean ได้เพราะเมื่อ bean ถูกสร้างขึ้นอาจจะจำเป็นที่ต้องการให้ bean ตัวนั้นทำงานบางอย่างเพื่อให้ bean นั้นมีความพร้อมที่จะถูกนำไปใช้งาน หรือการทำลายค่าอะไรบางอย่างก่อน bean ตัวนั้นจะถูกทำลายซึ่งจะถูกเรียกว่า Initialization callbacks และ Destruction callbacks ตามลำดับ

Spring MVC

การที่เซอร์วิสจะติดต่อกับผู้ใช้งานนั้นจำเป็นต้องมีอินเทอร์เฟซในการติดต่อส่งข้อมูลกัน ซึ่ง spring framework ได้นำแนวคิดแบบ MVC (Model , View ,Controller) นั่นก็คือการแบ่งการทำงานออกเป็นสามส่วนได้แก่

- Controller เป็นเหมือนตัวกลางในการรับ-ส่งข้อมูลระหว่าง View และ Model
- Model มีไว้ประมวลผลสำหรับส่วนของ business logic ไม่ว่าจะเป็นการทำ CRUD (create, read, update และ delete) หรือจะเป็นการประมวลผลติดต่อกับฐานข้อมูลและผลลัพธ์เหล่านี้จะส่งข้อมูลไปยังส่วน View เพื่อแสดงผล
- View เป็นชุดคำสั่งสำหรับแสดงหน้าอินเทอร์เฟซให้แก่ผู้ใช้ โดยข้อมูลเหล่านี้จะมาจาก Model

โดยเฟรมเวิร์กนี้จะใช้สำหรับการสร้างเว็บแอปพลิเคชันโดยจะแยกส่วนและหว่าง model กับ view เพื่อที่จะให้นักพัฒนา front-end และ back-end พัฒนาไปควบคู่กันได้

Spring Actuator

Spring Boot Actuator มีคุณสมบัติเกี่ยวกับการ monitoring และ metrics ของแอปพลิเคชัน โดยที่ผู้พัฒนาไม่จำเป็นต้องเขียนโค้ดเองแต่อย่างใดก็สามารถดูสถานะของเซอร์วิสเราหรือสถานะการเชื่อมต่อกับฐานข้อมูลได้อย่างสะดวกสบาย การที่จะทำให้แอปพลิเคชันของเราสามารถใช้งานคุณสมบัตินี้ได้นั้นก็ เพียงแค่ใส่ dependency ของตัว spring actuator เข้าไปใน pom.xml ของเราดังรูปที่ 2.45

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

รูปที่ 2.39 Dependencies ของ Spring Actuator

เพียงเท่านี้แอปพลิเคชันของเราก็สามารถใช้งาน spring boot actuator ได้ ซึ่งจะสามารถทำให้ผู้พัฒนาดูคุณสมบัติต่าง ๆ ของแอปพลิเคชันได้จาก REST endpoint โดยมี Actuator endpoints เบื้องต้นดังนี้

- GET: /configprops ดู configuration properties รวมไปถึงค่า default config

- GET: /beans ดูรายละเอียดของ beans ไม่ว่าจะเป็น context หรือ relationship ในแอปพลิเคชัน

- GET: /env ดู environment properties ทั้งหมดในแอปพลิเคชั่น

- GET: /env/{name} ดู environment properties แบบเจาะจงโดยส่งชื่อผ่านบนพารามิเตอร์

- GET: /health ดูสถานะ up-down ของแอปพลิเคชัน

- GET: /info ดูการตั้งค่า info จากไฟล์ appication.properties ในแอปพลิเคชันของเราโดยจะถูกตั้งชื่อนำหน้าด้วย info.xxx

- GET: /mappings ดู URL paths ที่ถูก map อยู่กับ controller ได้ว่ามีอะไรบ้าง

- GET: /metrics ดูข้อมูลเช่น memory usage และ HTTP request counters

- GET: /metrics/{name} ดู metric แบบเจาะจงจากชื่อโดยส่งค่าไปบนพารามิเตอร์

โดยการจะเข้าดูรายละเอียดเหล่านี้นั้นจะเข้าผ่านโดยมี prefix ข้างหน้าเป็น /actuator เช่น localhost:8080/actuator/health, localhost:8080/actuator/info ดังแสดงในรูปที่ 2.46 และรูปที่ 2.47


```
{
  "activeProfiles": [],
  "propertySources": [
    {
      "name": "server.ports",
      "properties": {
        "local.server.port": {
          "value": 8087
        }
      }
    },
    { ▶ 2 properties, 1 KB "name": "configService:file:center-config/transaction… },
    { ▶ 2 properties, 51 bytes "name": "servletContextInitParams", "properties": {}},
    { ▶ 2 properties, 5 KB "name": "systemProperties", "properties": {}},
    { ▶ 2 properties, 7 KB "name": "systemEnvironment", "properties": {}},
    { ▶ 2 properties, 168 bytes "name": "springCloudClientHostInfo", "properties": {}},
    { ▶ 2 properties, 314 bytes "name": "applicationConfig: [classpath:/bootstrap.pro… },
    { ▶ 2 properties, 162 bytes "name": "defaultProperties", "properties": {}}
  ]
}
```


รูปที่ 2.40 ตัวอย่าง Spring Actuator Info

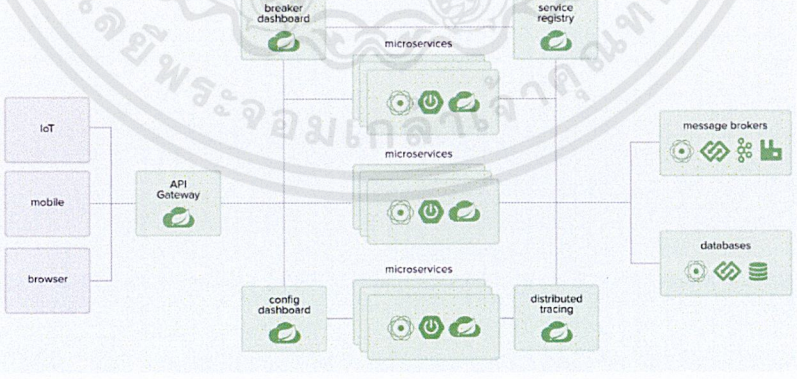
```
{
  "status": "UP",
  "details": {
    "diskSpace": {
      "status": "UP",
      "details": {
        "total": 31457275904,
        "free": 28891348992,
        "threshold": 10485760
      }
    },
    "db": {
      "status": "UP",
      "details": {
        "database": "MySQL",
        "hello": 1
      }
    },
    "refreshScope": {
      "status": "UP"
    },
    "discoveryComposite": {
      "status": "UP",
      "details": {
        "discoveryClient": {
          "status": "UP",
          "details": {
            "services": [
              "transaction-service"
            ]
          }
        },
        "eureka": {
          "description": "Remote status from Eureka server",
          "status": "UP",
          "details": {
            "applications": {
              "TRANSACTION-SERVICE": 1
            }
          }
        }
      }
    },
    "clientConfigServer": {
      "status": "UP",
      "details": {
        "propertySources": [
          "file:center-config/transaction-service.properties"
        ]
```


รูปที่ 2.41 ตัวอย่าง Spring Actuator Health

Spring Cloud

เป็นอีกหนึ่งโปรเจคที่ spring framework ได้ทำออกมาเพื่อให้นักพัฒนาได้พัฒนา spring application ที่ทำงานบนระบบคลาวน์ได้ดีมากยิ่งขึ้น ซึ่งยังรองรับการทำงานร่วมกับ Netflix OSS ในชื่อ Spring Cloud Netflix อีกด้วย (Eureka, Hystrix, Zuul, Archaius, etc.) ซึ่งโครงสร้างจะแสดงแดงดังรูปที่ 2.48 โดย spring cloud นั้นยังรอบรับการทำ distributed system ด้วยระบบต่าง ๆ ดังนี้

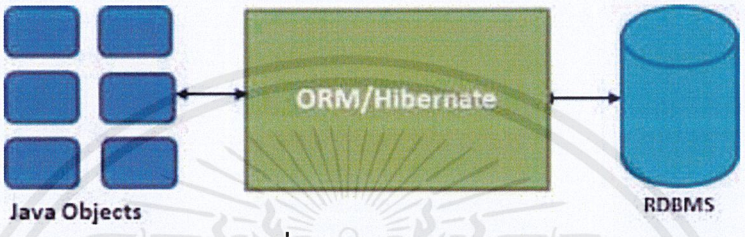
- Distributed Configuration - แยกไฟล์ที่ใช้สำหรับ config ออกมาจากตัวไมโครเซอร์วิส
- Service Discovery – กรณีที่เกตเวย์จะเข้าถึงตัว service ได้นั้นจะต้องมาถาม URL จากเซอร์วิสที่เป็นตัวกลางก่อนถึงจะได้ URL ของเซอร์วิสที่ต้องการเรียกใช้งานจริง ๆ กลับไป
- Routing - ติดต่อไปยังจุดหมายปลายทางได้ถูกต้อง
- Distributed Tracing - เมื่อเซอร์วิสถูกเรียกใช้งานระหว่างกันไปมาเลยจำเป็นต้องตรวจดูเส้นทางว่าผ่านเซอร์วิสไหนใช้เวลาเท่าไหร่
- Circuit Breaker - การเรียกใช้ระหว่างเซอร์วิสถ้ามีความเสียหายจากเซอร์วิสที่ถูกเรียกใช้แต่เซอร์วิสนั้นไม่ได้มีการรับมือกับความผิดพลาดที่ดีพอก็อาจทำให้เซอร์วิสนั้นพัง spring cloud จึงมี Circuit Breaker ไว้รองรับโดยมีคุณสมบัติ fault tolerance
- Load Balancing – เมื่อมีการเพิ่มจำนวนของไมโครเซอร์วิสโดยที่มีพอร์ตแตกต่างกันก็จะช่วยกระจายการร้องขอที่ได้รับเข้ามาไปยังแต่ละพอร์ตแบบเท่า ๆ กัน
- Service-to-Service Call - แต่ละเซอร์วิสสามารถติดต่อกันได้ด้วย RESTful API



รูปที่ 2.42 โครงสร้าง Spring Cloud
(https://medium.com/@phayao/สร้าง-microservices-ด้วย-spring-boot-spring-cloud-b3ea6f4a8b92)

2.3.2.2 Hibernate

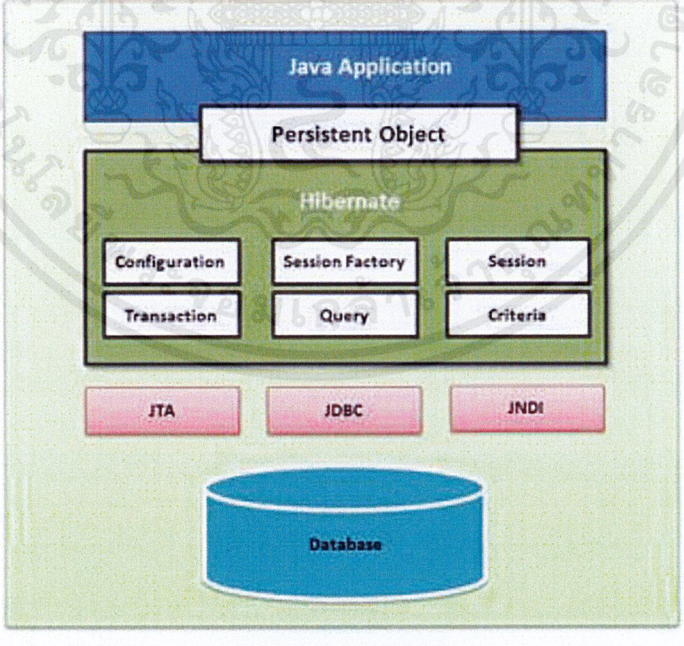
การติดต่อฐานข้อมูลสามารถจัดการโดยการใช้ Hibernate เข้ามาช่วยในการทำงาน โดยการวิธี ORM - Object Relational Mapping เป็นการแปลงจากออบเจ็คไปเป็นฐานข้อมูลหรือจากฐานข้อมูลมาเป็นออบเจ็คโดยที่ไม่จำเป็นต้องใช้งานคำสั่ง SQL เช่น Select, Insert, Update หรือ Delete ดังแสดงในรูปที่ 2.49 ซึ่งตัว Hibernate นั้นเป็นไลบรารี่ที่เปิดให้ใช้งานแบบไม่คิดค่าบริการจึงเป็นที่นิยมใช้



รูปที่ 2.43 วิธีการ ORM

(http://programminghunter.blogspot.com/2015/07/what-is-hibernate-framework.html)

ซึ่งรายละเอียดในส่วนของย่อยของการทำหน้าที่เป็นตัวกลางการสื่อสารระหว่าง Application Layer กับ Database ของ Hibernate Framework นั้นสามารถแยกย่อยได้ดังรูปที่ 2.50



รูปที่ 2.44 รายละเอียดของ Hibernate Framework

(http://programminghunter.blogspot.com/2015/07/what-is-hibernate-framework.html)

ประโยชน์ของ Hibernate

1. มีเวาลาในการพัฒนา business logic ของโปรแกรมอย่างเต็มที่
2. ทำให้การบำรุงรักษาง่ายขึ้นเนื่องจากโคดที่เขียนน้อยลง สะอาด เข้าใจง่าย
3. ไม่ยึดติดกับฐานข้อมูล เราสามารถใช้ฐานข้อมูลได้หลากหลาย
4. มี APIs ที่เรียบง่ายที่ใช้ในการบันทึกและอ่านข้อมูลจากฐานข้มูลได้โดยตรงผ่าน จาวา ออบเจ็คหากมีการเปลี่ยนแปลงโครงสร้างฐานข้อมูลหรือตารางต่าง ๆ ก็สามารถแก้ไขได้อย่างสะดวกสบาย เพียงแค่แก้ที่ ไฟล์ XML เท่านั้น
5. Hibernate ไม่ต้องเพิ่ม application server เพื่อใช้งานแต่อย่างใด
6. สามารถจัดการความสัมพันธ์ที่ซับซ้อนของตารางต่าง ๆ ได้อย่างง่ายดาย
7. จัดการการเข้าถึงฐานข้อมูลได้อย่างมีประสิทธิภาพ ทำให้ลดภาระของ Database Server ซึ่งส่งผลให้การเชื่อมต่อเร็วขึ้น

การใช้งาน Hibernate Framework นั้นจำเป็นต้องสั่งงานผ่านแอนโนเทชัน โดยประกาศเพื่อบ่งบอกลักษณะของโค้ดเราให้ตรงกับตารางในฐานข้อมูล ซึ่ง data type ในโค้ดของเรานั้นไม่ได้จำเป็นต้องเป็น data type เดียวกันกับในฐานข้อมูลแต่ส่วนใหญ่แล้วจะกำหนดข้อมูลให้เหมือน ๆ กันเพื่อความถูกต้องโดยการแมพปิ้งเบื้องต้นนั้นแสดงได้ดังรูปที่ 2.51

Mapping type	Java type	ANSI SQL Type
integer	int or java.lang.Integer	INTEGER
long	long or java.lang.Long	BIGINT
short	short or java.lang.Short	SMALLINT
float	float or java.lang.Float	FLOAT
double	double or java.lang.Double	DOUBLE
big_decimal	java.math.BigDecimal	NUMERIC
character	java.lang.String	CHAR(1)
string	java.lang.String	VARCHAR
byte	byte or java.lang.Byte	TINYINT
boolean	boolean or java.lang.Boolean	BIT
yes/no	boolean or java.lang.Boolean	CHAR(1) ('Y' or 'N')
true/false	boolean or java.lang.Boolean	CHAR(1) ('T' or 'F')

รูปที่ 2.45 ความสัมพันธ์ของ data type ระหว่าง Java และ Hibernate (https://www.tutorialspoint.com/hibernate/hibernate_mapping_types.htm)

Query Language

การดึงข้อมูล (query) โดยทั่วไปเราจะต้องเขียนคำสั่ง SQL เพื่อดึงข้อมูลมาแสดงผลในรูปแบบของตาราง แต่เมื่อเราใช้ Hibernate ก็จะมีสิ่งที่คล้ายกับ SQL ให้เราสามารถ query ขึ้นมาในรูปแบบออบเจ็คได้เลย และสามารถทำความสัมพันธ์ได้เสมือนการ JOIN ของตารางได้ด้วย ซึ่งการเขียน query เพื่อติดต่อกับฐานข้อมูลนั้นสามารถทำได้ 3 วิธีนั่นก็คือ

1. Native Query

เป็นวิธีการเขียนที่เหมือนเขียนใน SQL โดยตรงเลยเช่น

- SELECT * FROM STUDENTS WHERE NAME = 'ABC'

2. Hibernate Query Language (HQL)

การเขียนโดยวิธีนี้นั้นจะอ้างอิงโดยการใช้ class หรือ attribute แทนการเรียกใช้จากชื่อจริงของตารางและคอลัมน์ (column) ซึ่งผลลัพธ์ของการใช้ HQL จะออกมาในลักษณะออบเจ็คพร้อมใช้งานได้เลย โดยการเขียนนั้นจะมีลักษณะดังนี้

- SELECT * FROM StudentAccount s LEFT JOIN FETCH s.studentId
- SELECT * FROM StudentAccount s WHERE s.studentName = 'ABC'

3. Criteria Builder

วิธีนี้นั้นเป็นการเขียนที่เกี่ยวข้องใดกับภาษา SQL พื้นฐานเลย โดยจะอ้างอิงจากตัววัตุต่าง ๆ ของคลาสในการควบคุมการทำงานทั้งหมด ซึ่งวิธีนี้จะให้การ query ของเรามีความหลากหลายได้โดยไม่ต้องเขียน query ซ้ำ ๆ เพื่อใช้งานในลักษณะที่ต่างกัน โดยรูปแบบการเขียนนั้นจะแสดงดังรูปที่ 2.52

```
Session session = HibernateUtil.getHibernateSession();
CriteriaBuilder cb = session.getCriteriaBuilder();
CriteriaQuery<Item> cr = cb.createQuery(Item.class);
Root<Item> root = cr.from(Item.class);
cr.select(root);

Query<Item> query = session.createQuery(cr);
List<Item> results = query.getResultList();
```

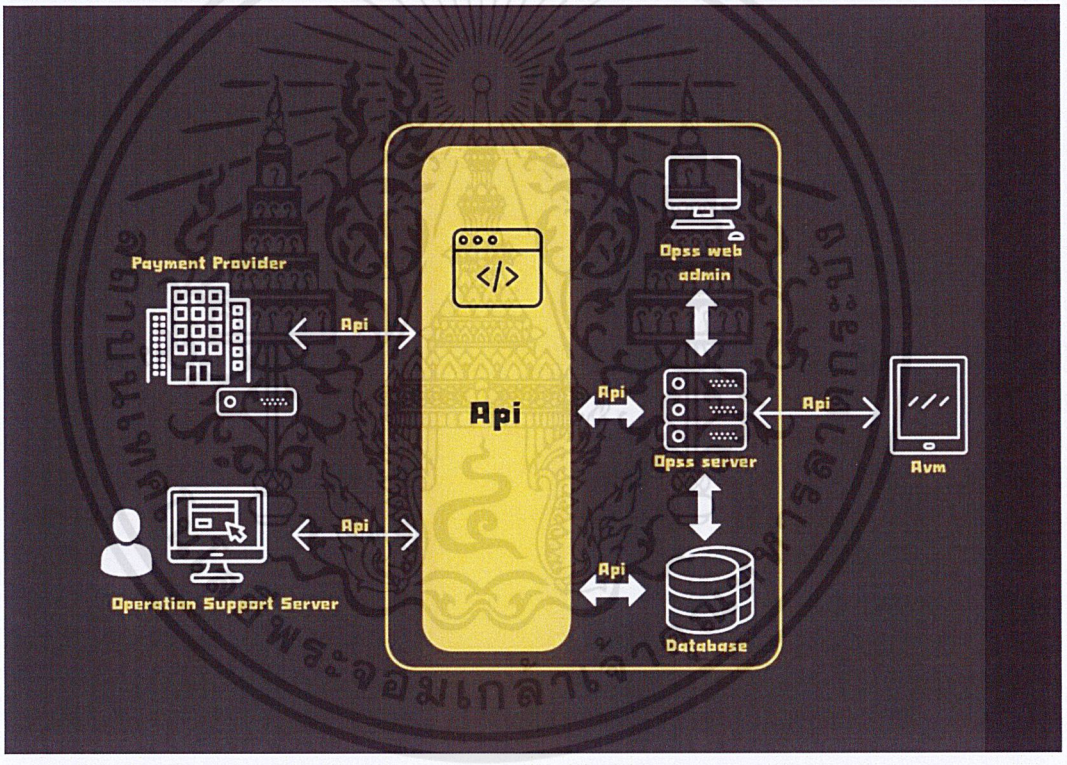
รูปที่ 2.46 ตัวอย่าง Criteria Builder

(https://www.baeldung.com/hibernate-criteria-queries)

บทที่ 3
วิธีดำเนินการวิจัย

3.1 การออกแบบโครงสร้างของระบบการเติมเงินออนไลน์

ระบบที่สร้างขึ้นมานี้นั้นจะทำหน้าที่เป็นตัวกลางระหว่างผู้ payment provider และ ผู้ให้บริการรถไฟฟ้าใต้ดินซึ่งต้องรับ-ส่งข้อมูลให้แก่ทั้งสองฝ่ายเพื่อให้ระบบสามารถดำเนินการได้อย่างสมบูรณ์โดย การทำงานของระบบนั้นจะมีภาพรวมแสดงดังรูปที่ 3.1

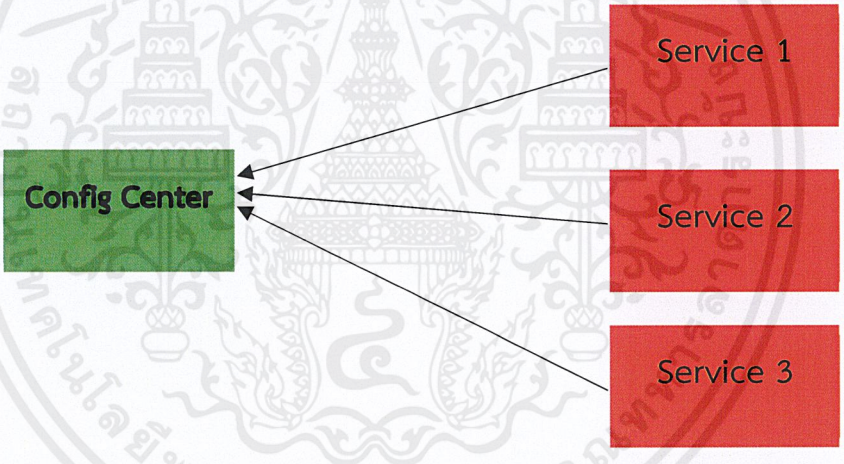


รูปที่ 3.1 ภาพรวมระบบการเติมเงินออนไลน์

เป็นที่แน่ชัดอยู่แล้วว่าระบบนั้นจะต้องถูกเรียกใช้บ่อยครั้งอยู่แล้วเพราะทุกการเติมเงินของผู้ใช้งานจะต้องทำงานผ่านการเรียก API เหล่านี้ การออกแบบโครงสร้างของระบบที่มีผู้ใช้งานจำนวนมากนั้นจำเป็นต้องคำนึงถึงประสิทธิภาพความสามารถในการรองรับผู้ใช้งานเหล่านั้นให้ได้รับประสบการณ์ที่ดีต่อการใช้บริการ การจะออกแบบให้คล้ายคลึงกับเว็บแอปพลิเคชั่นทั่วไปนั้นคงเป็นไปได้ยากเพราะว่าการทำงานของระบบนั้นมีหลากหลายส่วนมากมาย จะเป็นอย่างไรถ้าเกิดการให้บริการใดบริการหนึ่งเกิด

เสียหายแล้วทำทั้งระบบเสียหายใช้งานไม่ได้ นั่นคงเป็นเรื่องแย่สำหรับผู้ใช้บริการและผู้ให้บริการ ดังนั้นจึงได้นำแนวคิดของ Spring Cloud มาใช้ รวมไปถึงการทำระบบทั้งหมดให้อยู่ในรูปของไมโครเซอร์วิส เพื่อให้ฟังก์ชันที่ไม่ได้เกี่ยวข้องกันสามารถทำงานต่อได้เมื่ออีกฟังก์ชันนึงเสียหาย

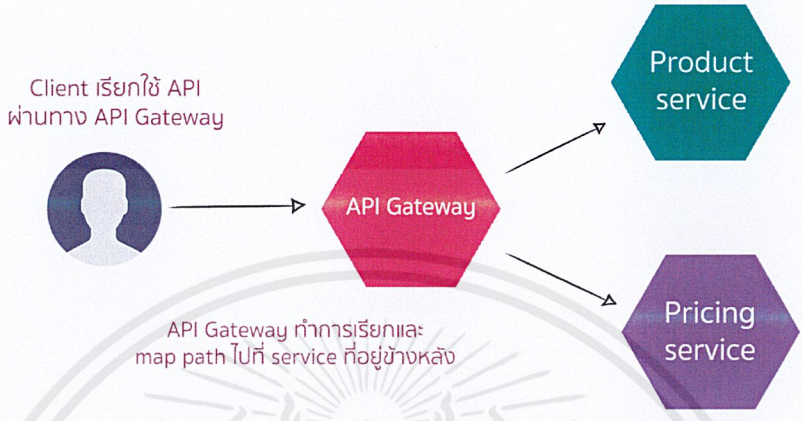
การออกแบบนั้นจะเริ่มที่การที่จะทำการแยกไฟล์สำหรับการตั้งค่า (Config) ออกมาจากโมดูลของแต่ละเซอร์วิสก่อนดังแสดงในรูปที่ 3.2 เหตุผลที่ทำแบบนี้เพราะว่าถ้าเกิดมีความต้องการจะเปลี่ยนการตั้งค่าในแต่ละโมดูลจะทำได้ง่ายกว่าการวางไฟล์ตั้งค่าไว้ในตัวโมดูล เป็นที่แน่นอนอยู่แล้วว่าระบบนี้จะทำเป็นไมโครเซอร์วิสดังนั้นไมโครเซอร์วิสแต่ละตัวก็ต้องถูกนำขึ้นเซิร์ฟเวอร์เป็นจำนวนหลายตัวอยู่แล้วเพื่อรองรับการรับ-ส่งข้อมูลโดยที่ระบบจะไม่เกิดการล่มเมื่อมีผู้ใช้จำนวนมาก ดังนั้นถ้าจะเปลี่ยนการตั้งค่าก็คงต้องไปเปลี่ยนให้กับทุกตัวที่นำขึ้นไปหรือจะเป็นการนำไฟล์ใหม่ที่ได้แก้ไขแล้วไปส่งขึ้นเซิร์ฟเวอร์ใหม่ก็จะยุ่งยากเสียเวลา การทำไฟล์ config แยกจึงมีประโยชน์อย่างมากในส่วนนี้เพียงแค่ผู้พัฒนาเปลี่ยนการตั้งค่าที่เดียวเซอร์วิสที่ชี้มายังที่การตั้งค่านี้ก็จะเปลี่ยนตามไปกันหมด



รูปที่ 3.2 การทำงานของ Config Center

โดยระบบจะมี Zuul Netflix เป็นตัวเซอร์วิสที่ทำหน้าที่เป็น gate way และใช้ Eureka Netflix เป็นตัวทำ service discovery โดยการทำงานผ่านเข้าออกของทุกการแลกเปลี่ยนข้อมูลนั้นจะต้องผ่าน zuul service ทุกครั้งเพื่อให้สามารถจัดการเรื่อง security (authentication และ authorization), logging และ tracking users และถ้ามีการเรียกใช้เซอร์วิสใด ๆ ในระบบก็จำเป็นต้องมาถามหาตำแหน่งจากทาง Eureka Service ซะก่อน การทำ service discovery นั้นจะช่วยเป็นอย่างมากเมื่อมีการเพิ่ม พอร์ตหรือมีการเปลี่ยนพอร์ตต่าง ๆ ของตัวเซอร์วิสโดยผู้ที่จะเข้ามาเรียกใช้งานไม่จำเป็นต้องเปลี่ยน URL ที่ใช้ติดต่อโดยหน้าที่จะตกไปอยู่ที่ Eureka Service ในการหาเซอร์วิสเหล่านั้นให้แล้วนำไปบอกยังผู้ที่ต้องการ

เรียกใช้งานเซอร์วิสในระบบ โดยปกติแล้ว Zuul จะทำการหาข้อมูลของเซอร์วิสโดยใช้ Eureka ดังแสดงในรูปที่ 3.3



รูปที่ 3.3 การทำงานของ Zuul Gateway (https://developers.ascendcorp.com/สร้าง-api-gateway-ด้วย-netflix-zuul-และ-spring-cloud-e50a10fec4e5)

ในการหาที่อยู่ของเซอร์วิสจาก Service IDs และใช้ Netflix Ribbon ในการทำ client-side load balancing ของ requests ใน Zuul ทำให้ไม่เกิดการส่งการร้องขอไปแค่เพียงที่เดียว

การให้บริการของ API ส่วนกลางนั้นจะมี API ให้บริการดังนี้

1. Key Exchange
2. Card Status Inquiry
3. Top-up Request
4. Transaction Status Inquiry
5. Pending Top-up Inquiry
6. Void
7. Reload Update Notification
8. Reload Cancellation Notification

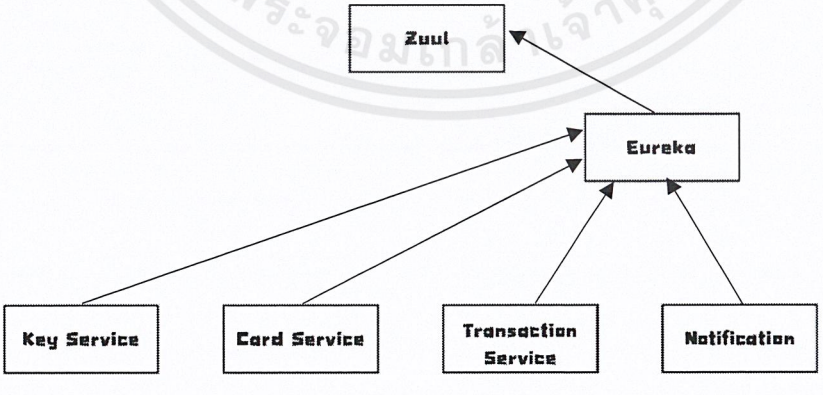
ซึ่งแต่ละเซอร์วิสจะถูกแบ่งกลุ่มให้อยู่ในลักษณะงานที่ทำคล้ายคลึงกันโดยจะแบ่งได้ดังตารางที่ 3.1

ตารางที่ 3.1 กลุ่มเซอร์วิสในระบบ

Key Service	Card Service	Transaction Service	Notification
Key Exchange	Card Status Inquiry	Top-up Request	Update Notification
		Transaction Status Inquiry	
		Pending Top-up Inquiry	Cancellation Notification
		Void	

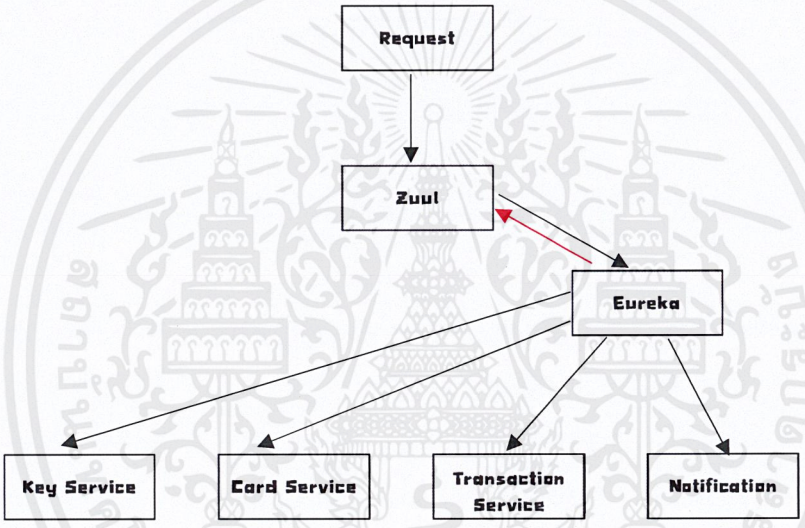
API ดังกล่าวนี้จะมีพื้นฐานโครงสร้างเหมือนกันเกือบทั้งหมดแตกต่างแค่ business logic ภายในที่ทำหน้าที่ต่างกันรวมไปถึงรูปแบบข้อมูลการรับส่งก็จะแตกต่างตามกันไปแล้วแต่จุดประสงค์ของการทำงาน โดยในที่นี้จะกล่าวถึงการทำงานของเซอร์วิส Transaction Status Inquiry และ Pending Top-up Inquiry ซึ่งการรับ-ส่งข้อมูลนั้นจะส่งในรูปของ JSON โดยวิ่งอยู่บน HTTP protocol โดยใช้รูปแบบการส่ง post method เพื่อเพิ่มความปลอดภัยของข้อมูลลูกค้าให้มากที่สุด โดยรูปแบบของฟิลด์ (fields) รับ-ส่งนั้นจะเป็นไปในรูปแบบตามที่ได้ตกลงกันไว้ โดยการทำงานของทั้งระบบจะสรุปได้ดังนี้

- นำเซอร์วิสแต่ละตัวเชื่อมต่อไปยัง service discovery ก่อน นั่นก็คือตัว Eureka โดยตัว Eureka เองนั้นก็จะเชื่อมต่อกับ Zuul ซึ่งเป็นเกตเวย์เซอร์วิส เท่านี้ทั้งระบบก็พร้อมให้ไคลเอนต์ส่งข้อมูลการร้องขอมาใช้งาน แสดงดังรูปที่ 3.4



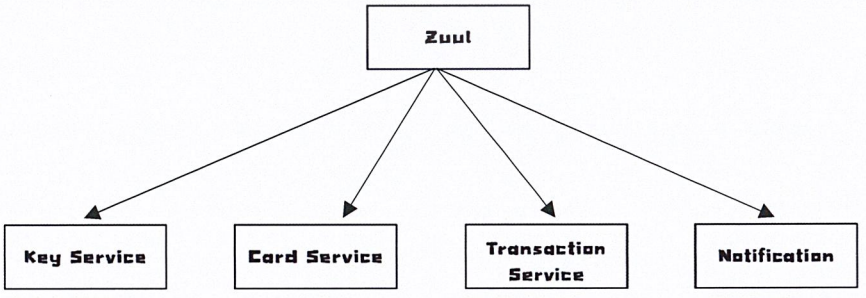
รูปที่ 3.4 การทำงานของระบบ

- หลังจากที่ไคลเอนต์ส่งการร้องขอมายัง Zuul gateway service ของระบบ ทาง Zuul gateway service ก็จะทำการตรวจสอบการร้องขอนั้นว่าเป็นผู้ใช้งานที่อนุญาติให้เข้ามาใช้งานเซอร์วิสต่าง ๆ เหล่านี้ได้รึป่าว ซึ่งถ้าไม่ผ่านการตรวจสอบจาก Zuul gateway service แล้วนั้นก็จะจบการทำงานแต่ถ้าผ่านการยืนยันตัวตนว่าเป็นการการร้องขอมาจากบุคคลที่ใช้งานได้ทาง Zuul gateway service ก็จะตรวจสอบดูว่าการร้องขอนั้นต้องการใช้ service ตัวใดจากนั้นก็จะส่งข้อมูลนั้นไปยัง service discovery เพื่อหาเส้นทางของเซอร์วิสที่การร้องขอนั้นจะส่งไปจากนั้น service discovery ก็จะนำเส้นทางของเซอร์วิสนั้น ๆ ไปบอกแก่ยัง Zuul gateway service ว่าเซอร์วิสที่ได้ขอมานั้นอยู่ที่ URL ใด แสดงดังรูปที่ 3.5



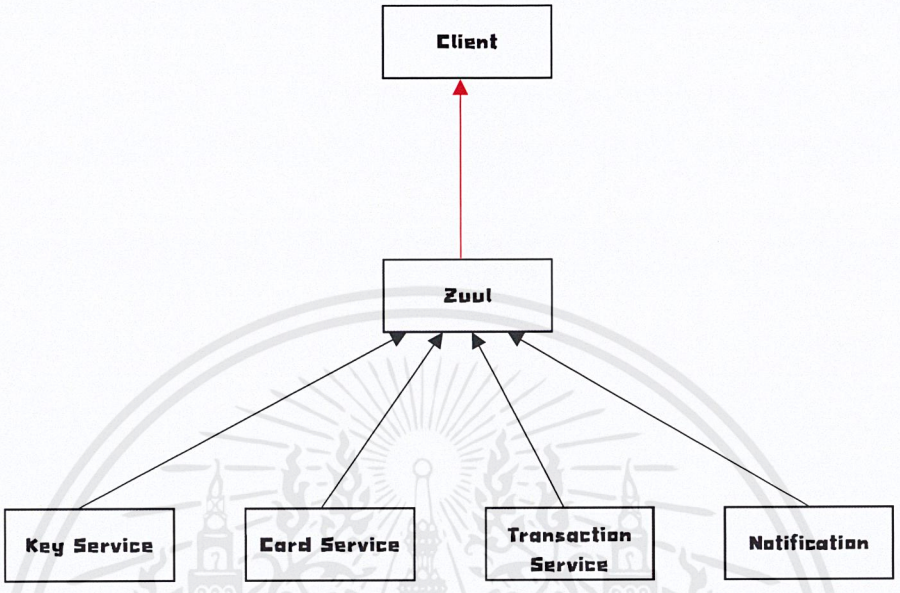
รูปที่ 3.5 การทำงานของระบบ

- หลังจากที่ Zuul gateway service ทราบถึง URL เซอร์วิสนั้นแล้วก็จะทำการส่งการร้องขอนั้นไปยังเซอร์วิสต่าง ๆ แสดงดังรูปที่ 3.6



รูปที่ 3.6 การทำงานของระบบ

- เมื่อเซอร์วิสได้ทำหน้าที่ของแต่ละเซอร์วิสนั้น ๆ เสร็จสิ้นแล้วก็จะนำผลลัพธ์ที่ได้ส่งกลับยัง Zuul gateway service ซึ่งตัวเกตเวย์ก็จะทำการส่งข้อมูลการตอบกลับไปยังไคลเอนต์ที่ขอมา ดังรูป 3.7



รูปที่ 3.7 การทำงานของระบบ

Zuul Gateway Service นั้นไม่ได้มีหน้าที่แค่ทำการติดต่อพูดคุยแค่ระหว่างไคลเอนต์กับเซอร์วิสเท่านั้นแต่ยังมีหน้าที่ในการให้เซอร์วิสติดต่อกับเซอร์วิสอีกด้วย โดยการพูดคุยของทั้งสองแบบนั้นจะต่างกันที่การคุยระหว่างไคลเอนต์กับเซอร์วิสนั้นจำเป็นต้องมีการตรวจสอบระบุตัวตนของผู้ใช้งาน รวมไปถึงการทำงานพื้นฐานของเกตเวย์ที่ควรจะเป็นอีกด้วยดังนั้นจึงจำเป็นที่ต้องตรวจสอบโดยละเอียดเพื่อความปลอดภัยของระบบ ต่างกับการพูดคุยระหว่างเซอร์วิสด้วยกันเอง ที่สามารถพูดคุยกันได้ สามารถเรียกถึงอีกเซอร์วิสได้โดยไม่จำเป็นต้องผ่านการตรวจสอบอะไรเนื่องจากเป็นเซอร์วิสที่อยู่ภายในระบบเดียวกันอยู่แล้ว

ระบบนี้นั้นจะใช้ Maven เป็นตัวจัดการไลบรารี่ของโปรเจค ซึ่งเซอร์วิสแต่ละตัวก็อาจจำเป็นต้องใช้ไลบรารี่เดียวกัน จึงจำเป็นต้องต้องมีการใส่ dependencies ไว้ใน pom.xml เพื่อที่แต่ละเซอร์วิสจะได้ไม่จำเป็นต้องนำ dependencies เหล่านี้มาใส่เพิ่มไว้ใน pom.xml ของแต่ตนเองอีก และยังมีประโยชน์ตรงที่สามารถควบคุมเวอร์ชั่นการใช้งานให้ตรงกันอีกด้วย เพื่อป้องกันปัญหาที่ว่าเมื่อนำขึ้นโปรดักชั่นแล้วนั้นเวอร์ชั่นที่ใช้กับไลบรารี่แตกต่างกันออกไปจึงทำให้เกิดบั๊กขึ้นมา ซึ่งก็คงต้องมานั่งเปลี่ยนทุกเซอร์วิสก็คงจะเสียเวลา ดังนั้นจึงมี dependencies ที่ใช้ร่วมกันดังรูปที่ 3.8

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>io.micrometer</groupId>
        <artifactId>micrometer-registry-prometheus</artifactId>
        <version>${prometheus.version}</version>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>${lombok.version}</version>
        <optional>true</optional>
    </dependency>

    <!-- https://mvnrepository.com/artifact/net.logstash.logback/log
    <dependency>
        <groupId>net.logstash.logback</groupId>
        <artifactId>logstash-logback-encoder</artifactId>
        <version>${logstash-logback-encode.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
```

รูปที่ 3.8 Dependencies รวม

ถ้าทำแบบนี้แล้วนั้นเมื่อมีความต้องการจะเปลี่ยนเวอร์ชั่นที่ใช้ก็เปลี่ยนเพียงแค่ pom.xml ตัวนี้ที่เดียวก็เพียงพอไม่จำเป็นต้องไปแก้เวอร์ชั่นใน pom.xml ของทุก ๆ เซอร์วิส ดังแสดงในรูปที่ 3.9

```
<properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Greenwich.SR2</spring-cloud.version>
    <lombok.version>1.18.8</lombok.version>
    <prometheus.version>1.2.0</prometheus.version>
    <mysql.version>8.0.17</mysql.version>
    <logstash-logback-encode.version>6.1</logstash-logback-encode.version>
    <jjwt.version>0.9.0</jjwt.version>
</properties>
```

รูปที่ 3.9 การกำหนดเวอร์ชั่นใน pom.xml

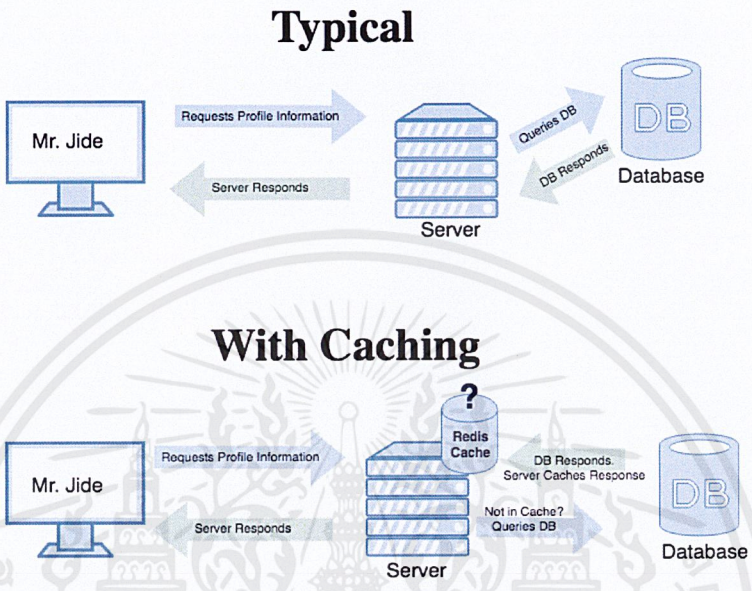
3.2 การออกแบบโครงสร้างเซอร์วิส

3.2.1 การออกแบบโครงสร้างพื้นฐาน

เซอร์วิสทั้งสองนั้นจะใช้ spring boot framework เป็นหลักในการทำงานเพราะฉะนั้นการทำงานก็จะคล้ายคลึงกันในภาพรวม โดยจะเริ่มจากการที่ต้องนำไลบรารี่กลางของโปรเจคมา install ก่อนเพื่อที่จะนำข้อมูลบางส่วนที่เหมือนกันมาใช้ ซึ่งก็คือการทำ mapping response หรือคือการทำรูปแบบการตอบกลับนั้นเองรวมถึงการทำ error exception ด้วย ที่ต้องใช้งานร่วมกันเป็นเพราะว่ารูปแบบการตอบกลับได้ถูกตกลงกันไว้แล้วจึงจำเป็นต้องมีรูปแบบที่เหมือนกัน โดยโค้ดของการตอบสนองนั้นก็จะเก็บอยู่ที่ตารางนึงในฐานข้อมูลจะขอเรียกข้อมูลนี้ว่า response code ซึ่งข้อมูลที่ตารางนี้เก็บไว้นั้นจะเป็นเหมือนโค้ดที่ทุกฝั่งเข้าใจว่าผลลัพธ์ออกมามีข้อผิดพลาด หรือการทำงานเป็นอย่างไร เพราะฉะนั้นการที่จะทำให้เซอร์วิสต้องไปเรียกเพื่อดึงข้อมูลจากฐานข้อมูลในส่วนนี้บ่อยครั้งก็อาจจะทำให้การพูดคุยระหว่างเซอร์วิสกับฐานข้อมูลมีมากเกินความจำเป็นจนทำให้คุณภาพของระบบต่ำลงได้ ดังนั้นจึงได้ออกแบบให้ตัวเซอร์วิสนั้นเรียกข้อมูลนี้จากฐานข้อมูลเพียงครั้งแรกเท่านั้นโดยจะสร้างเป็น bean เพื่อให้ทำงานตั้งแต่การเริ่มรันของเซอร์วิสจากนั้นจะเก็บไว้เป็นแคชแทน

เมื่อโปรแกรมเริ่มทำงาน bean ตัวดังกล่าวที่ทำหน้าที่ในการติดต่อฐานข้อมูลเพื่อเรียกข้อมูลของ response code ต่าง ๆ ที่จะตอบกลับไปนั้นก็จะทำงานและนั่นเป็นการเชื่อมต่อฐานข้อมูลของตารางนี้เพียงแค่ครั้งเดียวของการรันเซอร์วิสตัวนั้นเพราะหลังจากนี้หากมีการเรียกใช้ bean ตัวดังกล่าว bean ตัวนี้จะไม่ทำการเชื่อมต่อฐานข้อมูลโดยตรงแต่จะเป็นการไปนำค่าที่เคยทำงานไว้แล้วมาแสดงแทน โดยการ

จะใช้การออกแบบนี้ได้นั้นต้องมั่นใจว่าความเปลี่ยนแปลงของตารางดังกล่าวมีน้อยมาก หรือแทบไม่มีเลย ไม่อย่างนั้นจะทำให้ข้อมูลในเซอร์วิสกับข้อมูลในฐานข้อมูลเกิดความผิดพลาดและคลาดเคลื่อนได้ การเก็บแคชนั้นจะไม่ได้ทำงานซับซ้อนอะไรมากนักดังแสดงในรูปที่ 3.10



รูปที่ 3.10 การทำ Cache

(https://medium.com/@phayao/ทำ-caching-ด้วย-spring-redis-cache-f00b9009796)

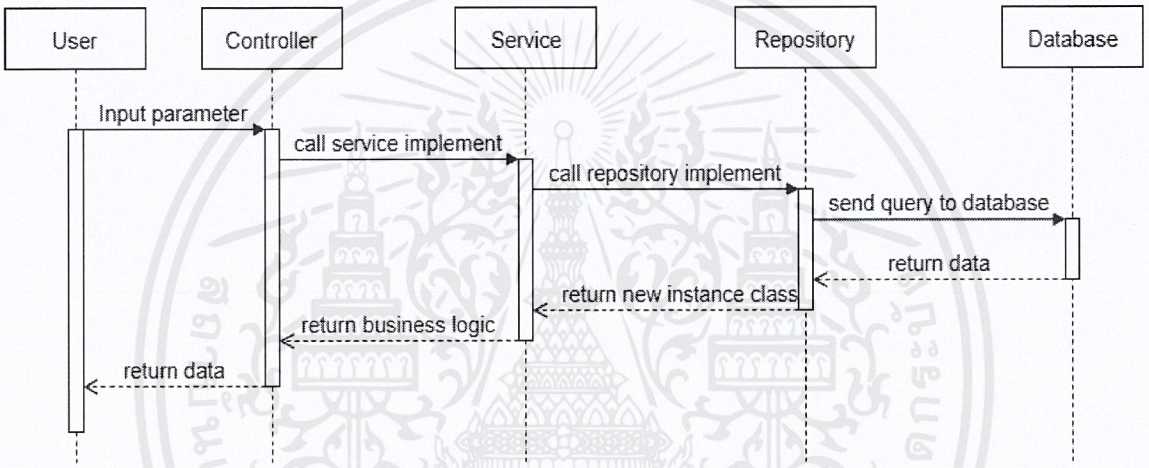
การประกาศ bean นั้นจะถูกออกแบบให้ประกาศในคลาสที่มีแอนโนเทชั่น @Configuration ซึ่ง bean ใด ๆ ก็ตามที่ต้องการสร้างไว้ใช้งานก็จะมาประกาศการทำงานไว้ในนี้ เมื่อมีการรันโปรแกรมเกิดขึ้น spring boot ก็จะมาอ่านค่าจากคลาสดังกล่าวแล้วนำไปสร้าง bean ให้ใช้งาน

ในส่วนของการทำ error exception นั้นก็จะคล้ายคลึงกันจึงได้ออกแบบการทำคลาสใหญ่เอาไว้ที่เดียวเมื่อมีความต้องการในการเพิ่มการดักจับ error ใหม่ก็สามารถ extend จากคลาสดังกล่าวได้เลยจึงช่วยให้ลดเวลาการสร้าง error exception อย่างมาก ซึ่งพารามิเตอร์ที่จะส่งเข้าไปสร้างนั้นก็มาจาก response code ที่เราได้สร้างเอาไว้นั่นเอง ดังนั้นเราจะได้กำหนดข้อมูลของ error exception ที่เกิดขึ้นให้ตรงกับ response code ในฐานข้อมูล

การออกแบบการจัดการกับฐานข้อมูลนั้นจะใช้ Docker มาช่วยในส่วนการนำเซิร์ฟเวอร์ของ MySQL มาไว้บนเครื่องที่ใช้พัฒนา เนื่องจากเวอร์ชั่นที่ใช้นั้นมีความไม่แน่นอน หรืออาจจะมีการเปลี่ยนแปลง ซึ่งจะช่วยให้การพัฒนาไม่ติดขัดหากเกิดการเปลี่ยนเวอร์ชั่นขึ้นเพียงแค่นำ Docker container อีกเวอร์ชั่นหนึ่งมาใช้ง้านแทนที่เวอร์ชั่นเดิมก็สามารถพัฒนาต่อได้

3.2.2 การออกแบบการทำงานของเซอร์วิส

เนื่องจากทั้งสองเซอร์วิสทำงานโดยใช้ Spring Boot Framework เป็นหลัก ดังนั้นการทำงานจะถูกออกแบบให้มีการเดินทางของข้อมูลในรูปแบบลักษณะที่เหมือนกัน โดยการเดินทางข้อมูลแบบนี้นั้น Spring Boot เป็นคนแนะนำซึ่งเราจะทำตามแบบนี้หรือไม่ทำก็ได้แต่เป็นที่แน่นอนอยู่แล้วว่าโครงสร้างที่ผู้สร้างเฟรมเวิร์กแนะนำมานั้นก็มีเหตุผลที่ดีจึงได้ตัดสินใจนำมาเป็นโครงสร้างของทั้งสองเซอร์วิสโดยมีการทำการงานในลักษณะดังรูปที่ 3.11 ซึ่งในขั้นตอนของการติดต่อฐานข้อมูลนั้นจะใช้วิธี criteria builder ในการติดต่อสื่อสารกัน



รูปที่ 3.11 Sequence Diagram ของเซอร์วิส

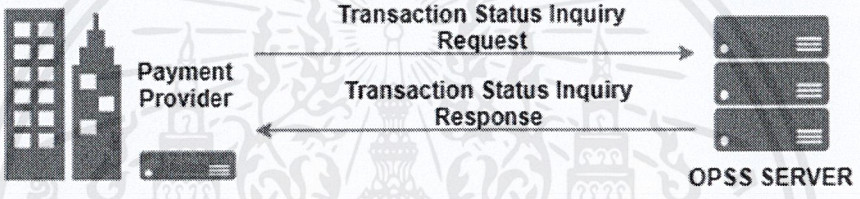
การทำงานนั้นจะอธิบายเป็นชั้น ๆ ไปโดยชั้นที่สำคัญจะมีด้วยกันทั้งหมด 4 ชั้นและมีรายละเอียดดังนี้

1. Controller - เมื่อได้รับการร้องขอเข้ามาจะถูกส่งมายัง controller ซึ่งเป็นเหมือนชั้นติดต่อสื่อสารโดยใช้ Spring MVC โดยจะถูกอ้างอิงจากชนิดของ HTTP request method และ Path URL
2. Service - ชั้น service ซึ่งก็คือชั้นที่จะทำงาน business logic ต่าง ๆ ที่ถูกกำหนดไว้ใน controller ว่าจะวิ่งไปหาเซอร์วิส ไหนแล้วตัวเซอร์วิสนั้นทำอะไรบ้าง
3. Repository - จะเป็นชั้นที่ทำ query เกี่ยวกับการกระทำต่อฐานข้อมูลต่าง ๆ โดยการเรียกข้อมูลจะเป็นไปตามจุดประสงค์ของแต่ละเซอร์วิส
4. Entity – เป็นการสร้างคลาสให้สอดคล้องกับตารางในฐานข้อมูลเพื่อแมพปิ้งกัน โดยชั้น repository จะทำงานผ่านคลาส entity เหล่านี้

3.3 การออกแบบเซอร์วิส Transaction Status Inquiry

3.3.1 การออกแบบภาพรวมการทำงานของเซอร์วิส

การออกแบบนั้นจะอ้างอิงจากจุดประสงค์หลักของเซอร์วิสเป็นสำคัญซึ่งจุดประสงค์ของเซอร์วิสนี้นั้นมีไว้สำหรับให้ ให้ Payment Provider Server เรียกดูข้อมูลของรายการ (transaction) เพื่อขอเรียกดูสถานะของเลขรายการที่ได้ทำผ่านเซอร์วิส top-up request มา โดยข้อมูลที่ถูกนำมาใช้ค้นหานั้นจะเฉพาะเจาะจงของตามข้อมูลของเลขทำรายการที่ต้องการค้นหา ทั้งหมดนี้ก็เพื่อให้ทาง payment provider นั้นได้รับทราบว่าการ top-up request ที่ผ่านไปนั้นมีผลลัพธ์สำเร็จหรือไม่สำเร็จ โดยการทำงานรับ-ส่งจะแสดงดังรูปที่ 3.12



รูปที่ 3.12 การทำงานของเซอร์วิส Transaction Status Inquiry

ซึ่งการทำงานจากรูปจะเป็นการเชื่อมต่อแลกเปลี่ยนข้อมูลรับส่งระหว่าง Payment Provider Server และ OPSS Server โดยการทำงานจะออกแบบไว้ดังนี้ เริ่มต้นที่ทาง Payment Provider Server ส่งเลขการทำรายการที่เฉพาะเจาะจงมายังเซอร์วิส transaction status inquiry ที่อยู่บน OPSS Sever จากนั้น OPSS Sever จะนำเลขทำรายการนั้นไปตรวจสอบในฐานข้อมูลในตารางที่ได้เก็บข้อมูลรายการต่าง ๆ เอาไว้โดยเซอร์วิส top-up request ถ้าเลขทำรายการที่ได้ส่งมาตรวจสอบนั้นตรงกับเลขรายการที่มีอยู่จริงในฐานจ้อมูล OPSS Sever ก็จะส่งรายละเอียดของเลขทำรายการนั้นกลับไป ซึ่งฟิลด์ที่จะใช้ในการหาเลขการทำรายการนั้นจะมีดังนี้

1. payment provider id
2. card number
3. original transaction id
4. original amount
5. original timestamp
6. reference id

3.3.2 การออกแบบข้อมูลการรับ-ส่ง

ซึ่งเมื่อทาง Payment Provider Server ทำรายการเสร็จแล้วจะได้รับเลขทำรายการ (original transaction id) หรือเลขอ้างอิง (reference id) กลับไปและถ้าหากมีความต้องการในการขอดูเลขทำรายการที่ได้ทำไปแล้วนั้นก็จะนำเลข transaction id หรือ reference id มาหาที่เซอร์วิส Transaction Status Inquiry ซึ่งการหานั้นจะสามารถหาจากเลขใดก็ได้หรือหาจากทั้งสองเลขก็ได้ แต่ก็จำเป็นต้องใช้ฟิลด์อื่นในการหาอีกด้วยเพื่อให้เซอร์วิสสามารถหาเลขทำรายการได้อย่างไม่คลาดเคลื่อนดังนั้นจะได้รูปแบบในการรับข้อมูลที่เข้าใจตรงกันดังตารางที่ 3.2

ตารางที่ 3.2 Field สำหรับการส่ง request เข้ามายังเซอร์วิส

Field name	Data type	Section	Request
Authorization	String	Header	M
Pid	String	Body	M
Timestamp (เวลาของการทำรายการนี้)	String	Body	M
Transaction Id	String	Body	M
Ref Id	Number	Body	C
Card Number	Number	Body	M
Real Transaction Id	String	Body	C
Real Amount	Number	Body	M
Real Timestamp	String	Body	M

โดยที่

M = Mandatory หรือก็คือฟิลด์นี้จำเป็นต้องส่งไม่ส่งไม่ได้

C = Conditional หรือก็คือฟิลด์นี้มีเงื่อนไขในการในใช้งาน

O = Optional หรือก็คือฟิลด์นี้จะไม่ส่งกลับหากไม่มีข้อมูล

อย่างทีได้กล่าวเอาไว้ว่า transaction id และ reference id นั้นสามารถส่งอย่างใดอย่างหนึ่งมาก็ได้แต่ถ้าหากส่งมาพร้อมกันทั้ง 2 ฟิลด์จะทำให้เงื่อนไขในการหาเป็นการใช้เงื่อนไข AND condition ซึ่งถ้าตัวใดตัวหนึ่งค่าไม่ตรงกับฐานข้อมูล ผู้ที่ส่งการร้องขอมาก็จะไม่ได้รับการตอบกลับของเลขรายการนั้นกลับไป โดยส่วนของ header นั้นจะมีไว้แค่การทำ validate เท่านั้น แต่ส่วนที่นำมาใช้หาในฐานข้อมูลจะใช้แค่ส่วน body และถ้าข้อมูลของ body ที่ได้ส่งมามีค่าตรงกับการทำรายการใดรายการหนึ่งเซอร์วิสก็จะตอบกลับด้วยฟิลด์ดังตารางที่ 3.3

ตารางที่ 3.3 Field สำหรับการส่ง response ออกไปจากเซอร์วิส

Field name	Data type	Section	Response
Timestamp (เวลาของการทำรายการนี้)	Date	Body	M
Transaction Id	String	Body	M
Response Code	Number	Body	M
Response Description	String	Body	M
Data	JSON	Body	C

จะเห็นได้ว่า Data Field นั้นเป็น Conditional ดังนั้นการตอบกลับไปทุกครั้งจะมี 4 ฟิลด์ที่ถูกตอบกลับไปเสมอ ตัว data นั้นจะไม่ถูกส่งออกมาด้วยเหตุผลดังนี้

- ฟิลด์ต่าง ๆ ทีได้ส่งเข้ามาไม่ตรงกับข้อมูลใด ๆ ที่ถูกเก็บในฐานข้อมูล

- ฟิลด์ใด ๆ ก็ตามเกิดข้อผิดพลาดด้วยรูปแบบของของ data type หรืออาจจะเป็นเหตุมาจากความสั้นหรือความยาวที่ส่งเข้ามาก็ล้วนแล้วแต่อยู่ในความผิดพลาดนี้
- เกิด error ใด ๆ ก็ตามที่จะสามารถเกิดขึ้นได้ในระบบ

ฟิลด์ Data นั้นคือชุดข้อมูลรายละเอียดต่าง ๆ ของเลขการทำรายกาย อย่างที่ได้กล่าวไปว่าไม่ใช่ทุกการร้องขอจะได้รับส่วนของ data นี้กลับไป ซึ่งถ้าการร้องขอไหนส่งข้อมูลมาถูกต้องก็จะได้รับการตอบกลับของ data ซึ่งมีรายละเอียดต่าง ๆ ดังตารางที่ 3.4

ตารางที่ 3.4 Field สำหรับการ response ในส่วนของ data

Field name	Data Type	Description	response
Card Number	Number	หมายเลขเฉพาะของ Card	M
Uid	String	Physical ID ของ Card	M
Card Issuer	String	บริษัทผู้ออกหลักทรัพย์ของ Card	O
Card Type	String	ชนิดของ Card	O
Real Timestamp	String	เวลาการทำรายการ	M
Pid	String	Payment Provider ที่ทำรายการเข้ามา	M
Real Transaction Id	String	เลขของการทำรายการที่มีความเป็น unique เอาไว้สำหรับหาเลขทำรายการนั้นๆ	M
Real Amount	String	จำนวนเงินที่ต้องการจะเติมเข้าบัตร	M
Ref Id	Number	เลขอ้างอิงการทำรายการ	M
Transaction Status	String	สถานะของรายการนี้ว่าถึงสถานะหรือขั้นตอนไหนอยู่	M
Reason Desc	String	เป็นฟิลด์ที่จะถูกส่งออกไปก็ต่อเมื่อการทำรายการนั้นเป็นรายการที่ทำไม่สำเร็จ	C

จะเห็นได้บางฟิลด์อาจจะมีเงื่อนไขเป็น optional นั้นเป็นเหตุผลมาจากการที่เลขทำรายการนั้นอ้างอิงไปยังข้อมูลที่ไม่มีอยู่ในฐานข้อมูล หรืออาจจะมีในตอนแรกแต่ภายหลังได้ถูกนำออกไปเลยอาจจะทำให้เกิดปัญหาดังกล่าว นั่นคือการที่จะหาข้อมูลอ้างอิงทีได้รับมาไม่เจอจึงจำเป็นต้องให้ฟิลด์เหล่านี้เป็น optional ซึ่งจะไม่มีการส่งกลับถ้าหากไม่มีข้อมูล และจะมีฟิลด์ที่เป็น conditional นั่นก็คือ Reason Desc ซึ่งฟิลด์นี้นั้นจะมีการส่งกลับไปก็ต่อเมื่อเลขการทำรายการนั้นเป็นรายการที่มีสถานะไม่สำเร็จแล้วแน่นอน ซึ่งข้อมูลของการทำไม่สำเร็จนั้นจะถูกเก็บแยกออกไปอีกตารางนึง

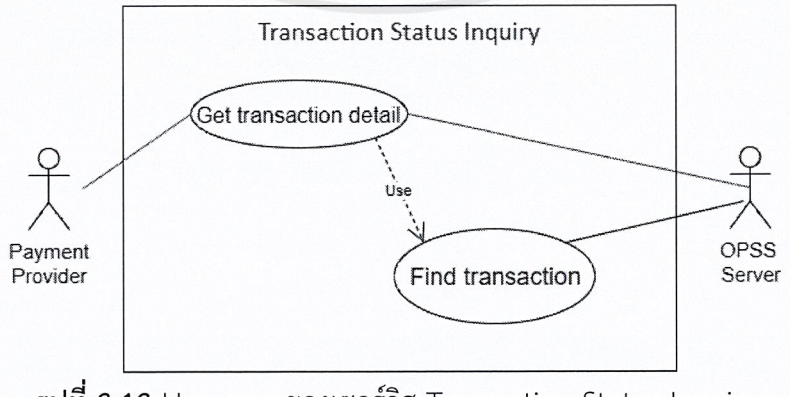
โดยการตอบกลับนั้นก็จำเป็นต้องทำให้ทั้งสองฝ่ายเข้าใจร่วมกันด้วยว่าการทำงานมีผลลัพธ์เป็นอย่างไรโดยการตอบกลับการทำงานจะเป็นไปดังตารางที่ 3.5

ตารางที่ 3.5 Response code

Response Code	Http Status Code	Description
00	200	Approved.
17	400	Invalid field in Body.
41	200	Transactions ID not found.
99	400	System Error.

3.3.3 การออกแบบทางเทคนิคของเซอร์วิส

ขั้นตอนการทำงานนั้นจะออกแบบเพื่อรองรับคำขอการร้องขอสำหรับการตรวจสอบข้อมูลของเลขทำรายแบบเฉพาะเจาะจง โดยหลักการทำงานนั้นคือ มีผู้ใช้งานมาร้องขอข้อมูลเลขทำรายการจากทางเซิฟเวอร์จากนั้นทางเซิฟเวอร์ก็จะส่งข้อมูลดังกล่าวกลับไป จึงออกแบบการทำงาน use case ไว้แสดงดังรูปที่ 3.13



รูปที่ 3.13 Use case ของเซอร์วิส Transaction Status Inquiry

ตารางที่ 3.6 แสดงรายละเอียด use case เซอร์วิส Transaction Status Inquiry

Use Case ID:	01
Use Case Name:	ภาพรวมการทำงานของเซอร์วิส Transaction Status Inquiry
Actors:	Payment Provider
Description:	ผู้ใช้งานสามารถตรวจสอบเลขทำรายการที่มีอยู่ในฐานข้อมูลได้
Trigger:	ผู้ใช้งานส่งรายละเอียดของเลขทำรายการ
Pre-conditions:	1. ต้องมีการใส่รายละเอียดให้ฟิลด์ต่าง ๆ ตามที่กำหนดไว้
Post-conditions:	-
Normal Flow:	1. ผู้ใช้งานอินพุตค่าของรายละเอียดเลขทำรายการที่ต้องการจะหา 2. ระบบตรวจสอบเลขรายละเอียดของเลขทำรายการทีได้รับมา 3. ระบบส่งผลลัพธ์เกี่ยวกับเลขทำรายการดังกล่าว
Exception:	1. กรณีที่ Payment Provider ส่งฟิลด์โดยที่รูปแบบผิดจากเงื่อนไขที่ตกลงกันไว้ หรือไม่ได้ส่งฟิลด์ที่จำเป็นมาตามที่กำหนด

การออกแบบเทคนิคของเซอร์วิสนั้นจะกล่าวถึงการออกแบบทำงานในแต่ละส่วนรวมไปถึงการนำไลบรารี่ต่าง ๆ มาใช้งานเพื่อให้เซอร์วิสมีประสิทธิภาพ ซึ่งการออกแบบข้อมูลการร้องขอและการตอบกลับได้ถูกกล่าวถึงไปแล้วขั้นตอนต่อไปจะเป็นการพูดถึงการทำงานของตัวเซอร์วิส โดยจะ อธิบายเป็นชั้นการทำงานของเซอร์วิสโดยแบ่งออกเป็นแต่ละชั้นไปดังนี้

1. Controller

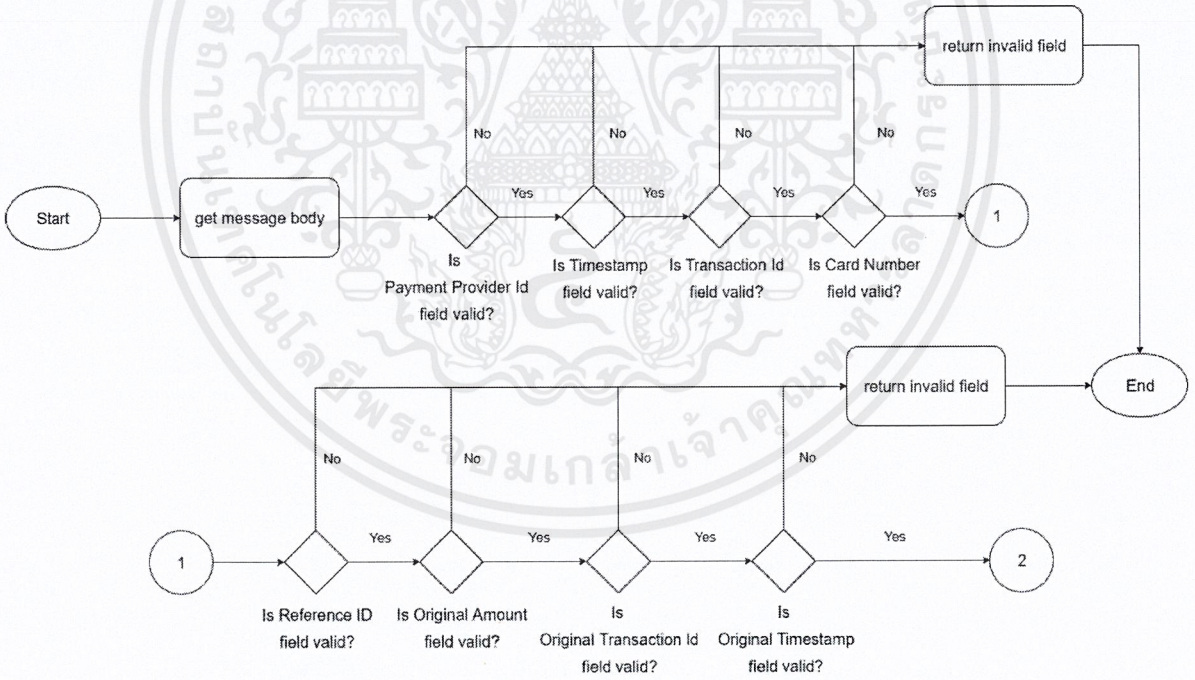
พารามิเตอร์ที่ไว้รับค่ามีดังนี้

- HTTP Headers : เป็นพารามิเตอร์ที่รับส่วน header ของการร้องขอเข้ามา

- JSON Body : เป็นพารามิเตอร์ที่รับ body ที่มาพร้อมกับการร้องขอโดยรูปแบบของฟิลด์จะเป็นไปตามที่ออกแบบเอาไว้

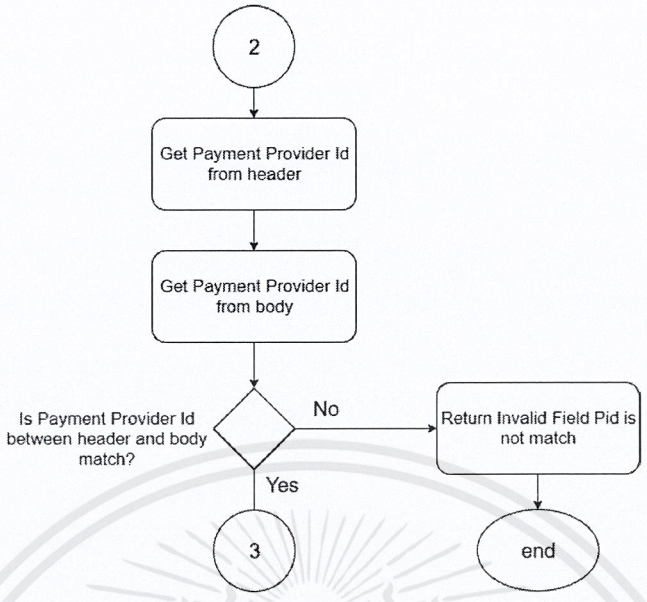
หลังจากที่รับค่าต่าง ๆ ตามพารามิเตอร์มาแล้วก็จะนำข้อมูลเหล่านั้นมาเข้า flow การทำงานโดยการทำงานนั้นจะเป็นดังต่อไปนี้

- ตรวจสอบฟิลด์ต่าง ๆ จาก JSON body ก่อนว่าถูกต้องตาม data type รึป่าวขนาด max min เกินหรือขาดตามที่กำหนดรึป่าว ถ้ามีข้อผิดพลาดใดเกิดขึ้นการทำงานก็จะหยุดลงตั้งแต่ตรงนี้ เพราะฉะนั้นการทำงานในขั้นตอนแรกจะเป็น แสดงดังรูปที่ 3.14



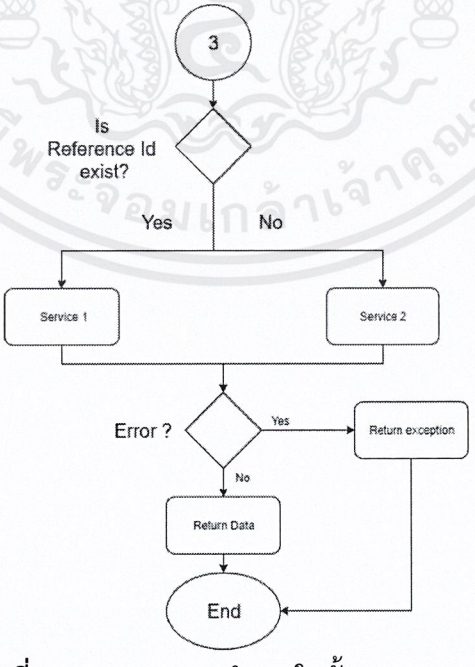
รูปที่ 3.14 Flow ตรวจสอบความถูกต้องของ field ที่ได้รับมา

- จากนั้นจะทำการตรวจสอบ payment provider id ที่อยู่ใน header กับ body ก่อนว่าเหมือนกันหรือไม่ถ้าไม่เหมือนก็จะสิ้นสุดการทำงานทันที แสดงดังรูปที่ 3.15



รูปที่ 3.15 Flow การตรวจสอบ payment provider id

- หลังจากตรวจฟิลด์และ payment provider id เรียบร้อยแล้วนั้นก็จะเริ่มทำงานจากการตรวจเช็คฟิลด์ที่ชื่อว่า Reference Id ซึ่งฟิลด์นี้นั้นจะเป็นตัวอ้างอิงของเลขรายการที่มีความเฉพาะเจาะจงเป็นพิเศษเนื่องจากในตารางที่เก็บเลขทำรายการทั้งสองตารางนั้นจะมีการแบ่งเลข Id เป็นช่วงเอาไว้เพื่อให้ทราบได้ว่าเลขรายการไหนอยู่ในตารางใดอย่างรวดเร็ว ดังนั้นจะตรวจสอบก่อนว่าถ้า body ส่ง Reference Id มาก็จะแยกเข้าชั้น service ที่ต่างกัน แสดงดังรูปที่ 3.16



รูปที่ 3.16 Flow การทำงานในชั้น controller

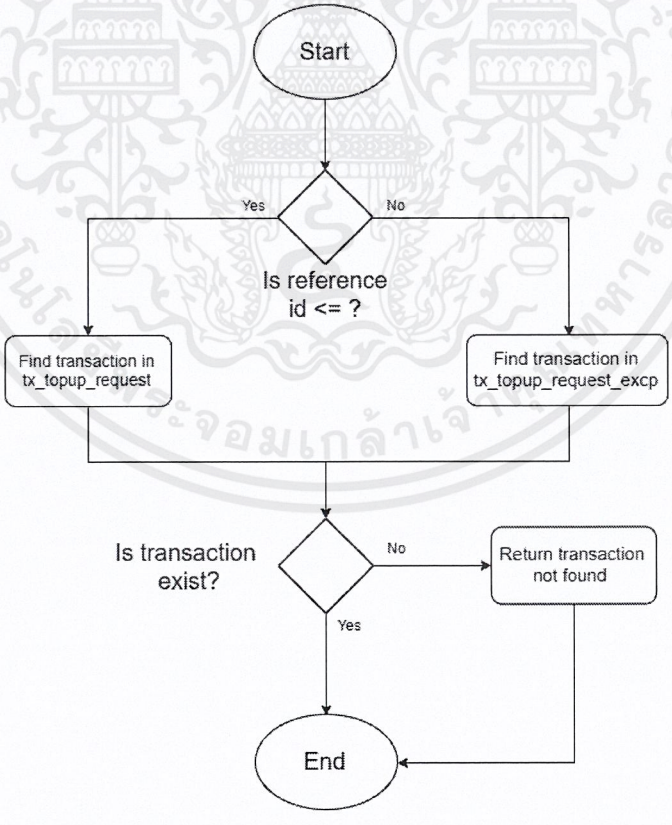
ในส่วนของ Controller จะสิ้นสุดเพียงเท่านี้ต้องรอจนกระทั่งชั้นอื่นทำหน้าเสร็จถึงจะส่งผลลัพธ์ออกไปได้ไม่ว่าจะสำเร็จหรือ error ก็ตาม

2. Service

ในการทำงานของชั้น service นั้นจะถูกเรียกใช้งานจากทาง controller ว่าฟังก์ชันไหนจะทำงาน โดยเงื่อนไขก็จะเป็นอย่างที่บอกไปในชั้นของ controller นั่นก็คือการแยกจาก Reference Id โดยจะอธิบายต่อจากชั้นของ controller ที่เข้าไปในการทำงานของเซอร์วิสที่ต่างกันดังนี้

- Service 1

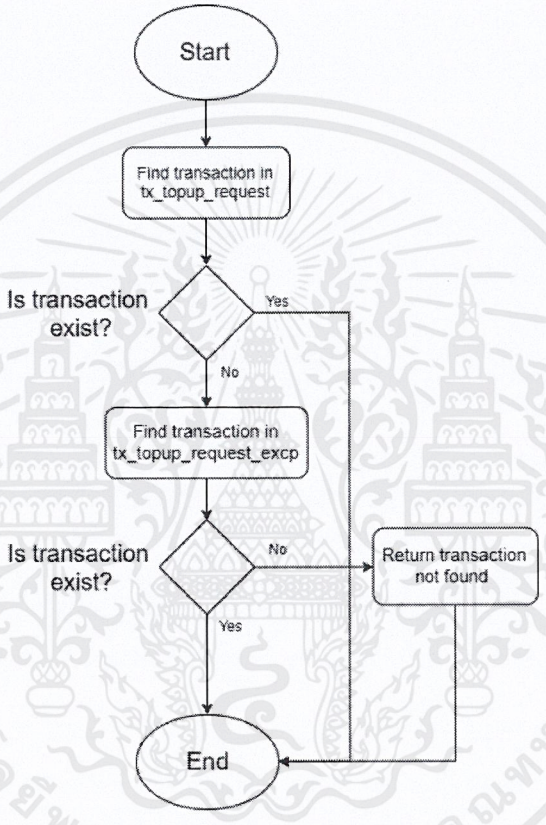
จากเงื่อนไขในชั้น controller ที่ต้องการ reference id มาพร้อมกับ request body แล้วนั้นก็จะนำมาสู่การทำงานของ service 1 โดยการทำงานนั้นจะอ้างอิงจาก reference id ที่ได้รับเข้ามาว่ามีค่าเป็นเท่าใด โดยเซอร์วิสจะแยกช่วงของเลข reference id นั้นว่าอยู่ในตารางฐานข้อมูลที่ปกติหรือไม่ปกติ และจะทำการตรวจสอบดูว่ามีข้อมูลการทำรายการอยู่จริงในฐานข้อมูลรึป่าว ดังนั้นการทำงานจะเป็นลักษณะดังรูปที่ 3.17



รูปที่ 3.17 การทำงาน Service 1

- Service 2

การทำงานนี้จะทำก็ต่อเมื่อ request body ไม่ได้ทำการส่ง reference id มา ดังนั้นจึงจำเป็นต้องใช้ข้อมูลอื่น ๆ ในการค้นหาโดยข้อมูลเหล่านั้นไม่สามารถระบุเลขทำรายการแบบเฉพาะเจาะจงได้ว่าอยู่ในตารางไหนของฐานข้อมูล จึงทำมีความจำเป็นต้องหาในทุกตารางที่เก็บเลขการทำรายการเอาไว้ โดยการทำงานจะเป็นดังรูปที่ 3.18



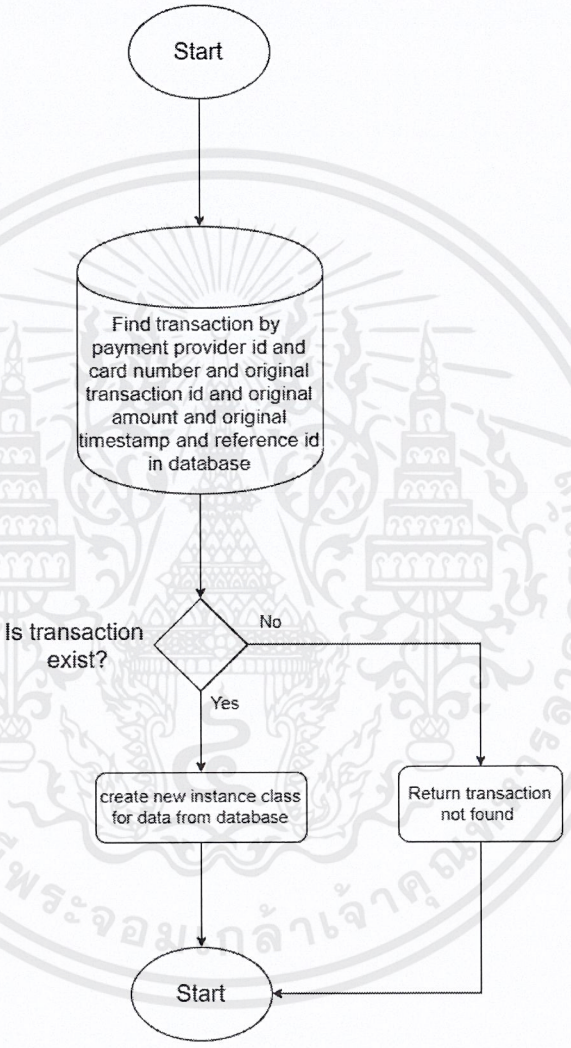
รูปที่ 3.18 การทำงาน Service 2

ซึ่งการจะหาข้อมูลของรายการที่ถูกได้ทำไว้แล้วนั้นจำเป็นต้องหาในฐานข้อมูล ดังนั้นจะต้องมีการทำ query ต่อฐานข้อมูลเพราะฉะนั้นชั้นที่จะรับหน้าที่ในการพูดคุยกับฐานข้อมูลนั้นก็จะต้องเป็นชั้นของ Repository

3. Repository

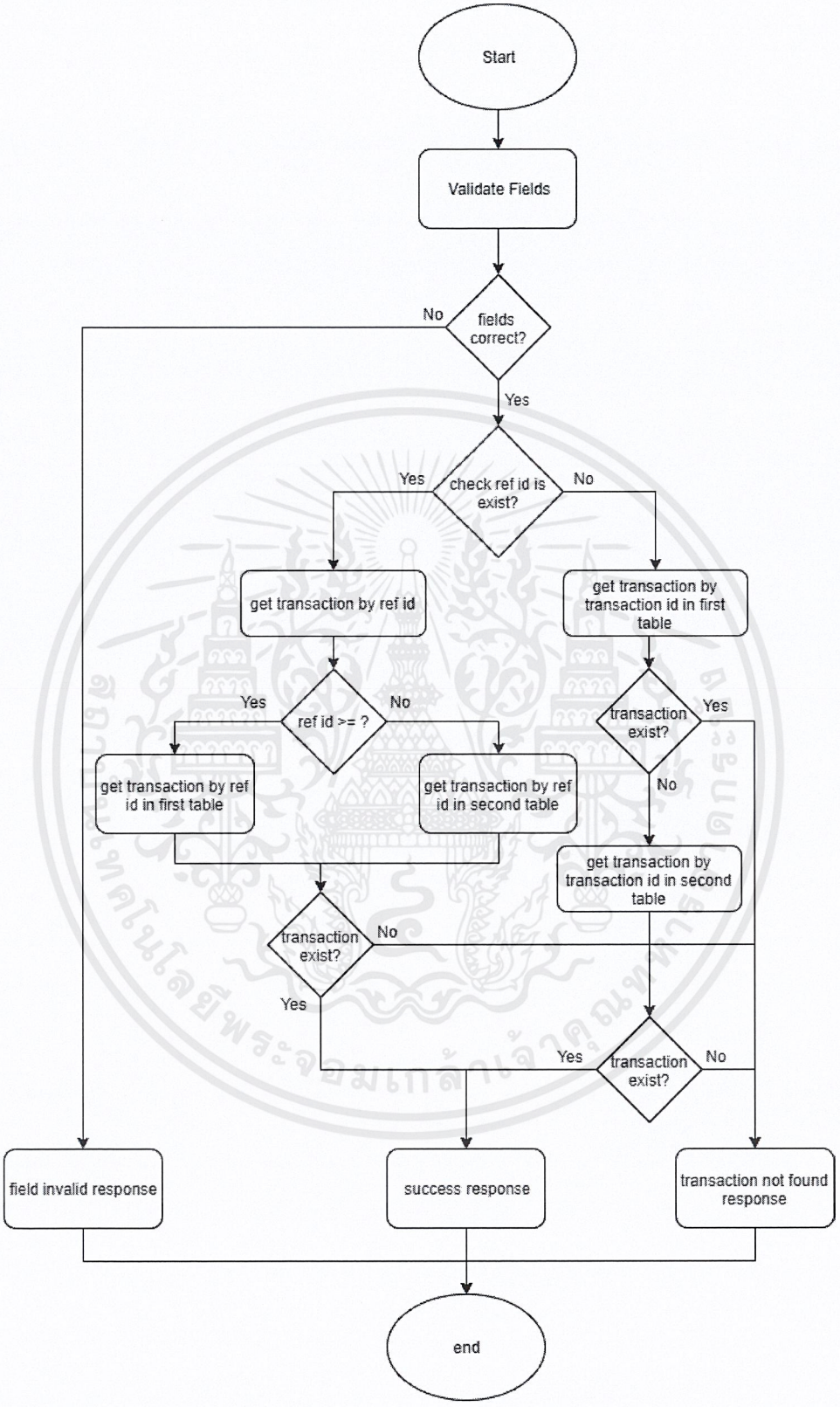
ดังที่ได้กล่าวไปการกระทำใด ๆ ที่เกี่ยวข้องกับฐานข้อมูลนั้นจำเป็นต้องอยู่ในชั้นที่ถูกเรียกว่า Repository โดยชั้นนี้จะทำหน้าที่ในการสร้าง query แล้วส่งไปกระทำกับฐานข้อมูลแล้วรับผลลัพธ์กลับมา ซึ่งในที่นี้นั้นจะมีการ query ไปยัง 2 ตาราง แต่ลักษณะการทำงานจะคล้ายคลึงกัน นั่นก็คือการ

ดึงข้อมูลของเลขรายการนั้นออกมาโดยการทำงานนั้นจะแตกต่างกันแค่ความสัมพันธ์ของตารางในฐานข้อมูลที่ต้องทำการเชื่อมต่อโดยให้มีความสอดคล้องกับการสร้าง entity ซึ่งทั้งสอง repository ก็จะอ้างอิงถึงตารางที่แตกต่างกันหรือคล้ายกันดังนั้นจึงจำเป็นต้องแยกการทำงาน แต่โดยพื้นฐานการทำงานนั้นจะทำงานดังรูปที่ 3.19

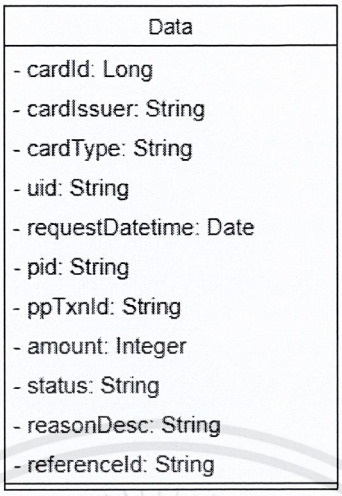


รูปที่ 3.19 การทำงานในชั้น Repository

หลังจากที่ได้ข้อมูลมาจากฐานข้อมูลแล้วนั้นก็จำเป็นที่ต้องสร้างจาวาคลาสดังรูปที่ 3.21 เพื่อรองรับข้อมูลที่ส่งกลับมาด้วยเพื่อที่จะนำข้อมูลจากฐานข้อมูลส่งกลับไปให้ยังไคลเอนต์ โดยข้อมูลที่ส่งมาจากฐานข้อมูลนั้นก็จะถูกเลือกเอาเฉพาะฟิลด์ ที่มีความจำเป็นในการนำไปใช้งาน โดยการทำงานภาพรวมนั้นจะเป็นไปดังรูปที่ 3.20



รูปที่ 3.20 การทำงานทั้งระบบของเซอร์วิส Transaction Status Inquiry



รูปที่ 3.21 Object Diagram ที่ใช้รองรับข้อมูลจากฐานข้อมูล

4. Entity

การจะทำให้ repository ทำงานได้นั้นเซอร์วิสก็จำเป็นต้องสร้าง entity จำลองให้ฟิลด์และความสัมพันธ์เหมือนกับตารางในฐานข้อมูลโดยตารางที่เกี่ยวข้องมีดังนี้

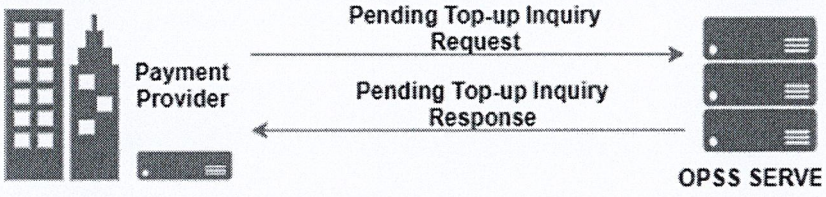
1. config_card_issuer
2. const_cancel_reason
3. const_excp_reason
4. const_txn_status
5. response_code
6. tp_card_sale
7. tx_topup_request
8. tx_topup_request_excp

3.4 การออกแบบเซอร์วิส Pending Top-up Inquiry

3.4.1 การออกแบบภาพรวมการทำงานของเซอร์วิส

การออกแบบนั้นจะอ้างอิงจากจุดประสงค์หลักของเซอร์วิสเป็นสำคัญซึ่งจุดประสงค์ในการออกแบบเซอร์วิสนี้ทำมาสำหรับให้ Payment Provider Server ส่งหมายเลขการ์ดมาเพื่อตรวจสอบดูว่า

การ์ดใบนี้นั้นไม่ได้มีการเติมเงินจากที่ใดมาก่อน ซึ่งการเติมเงินแต่ละครั้งนั้นจำเป็นต้องตรวจสอบก่อนอนุญาติให้ทำรายการเติมเงิน แสดงดังรูปที่ 3.22



รูปที่ 3.22 การทำงานของเซอร์วิส Pending Top-up Inquiry

ซึ่งการทำงานจากรูปจะเป็นการเชื่อมต่อแลกเปลี่ยนข้อมูลรับส่งระหว่าง Payment Provider Server และ OPSS Server โดยการทำงานจะออกแบบไว้ดังนี้ โดยเริ่มจากการที่ Payment Provider Server ส่งการร้องขอมายัง OPSS Sever ซึ่งข้อมูลในการส่งนั้นจะมีฟิลด์ที่สำคัญเพียงอันเดียวนั่นก็คือ card number ซึ่งจะส่งมาเพื่อตรวจสอบว่าการ์ดหมายเลขท่ส่งมานั้นอนุญาติให้เติมรึป่าว จากนั้น OPSS Sever ก็จะส่งผลลัพธ์กลับไปว่าอนุญาติรึป่าว

3.4.2 การออกแบบข้อมูลการรับ-ส่ง

แน่นอนอยู่แล้วว่าการรับและส่งก็ต้องมีรูปแบบที่ชัดเจนเป็นที่เข้าใจทั้งสองฝั่งอยู่แล้ว โดยจะเริ่มอธิบายการออกแบบรูปแบบของข้อมูลต้องส่งเข้ามาโดยมีรายละเอียดดังตารางที่ 3.7

ตารางที่ 3.7 Field สำหรับการส่ง request เข้ามายังเซอร์วิส

Field name	Data type	Section	Request
Authorization	String	Header	M
Timestamp (เวลาของการทำรายการนี้)	Date	Body	M
Payment Provider Id	String	Body	M
Transaction Id	String	Body	M
Card Number	Number	Body	M

โดยที่

M = Mandatory หรือก็คือฟิลด์นี้จำเป็นต้องส่งไม่ส่งไม่ได้

C = Conditional หรือก็คือฟิลด์นี้มีเงื่อนไขในการในใช้งาน

O = Optional หรือก็คือฟิลด์นี้จะไม่ส่งกลับหากไม่มีข้อมูล

หลังจากที่เซอร์วิสนำข้อมูลใน body มาทำ business logic แล้วนั้นก็จะต้องตอบกลับไปในรูปแบบที่ได้ตกลงกันไว้อีกด้วยเช่นกันโดยจะตอบกลับดังตารางที่ 3.8

ตารางที่ 3.8 Field สำหรับการส่ง response ออกไปจากเซอร์วิส

Field name	Data type	Section	Response
Timestamp	Date	Body	M
Transaction Id	String	Body	C
Response Code	Number	Body	M
Response Description	String	Body	M

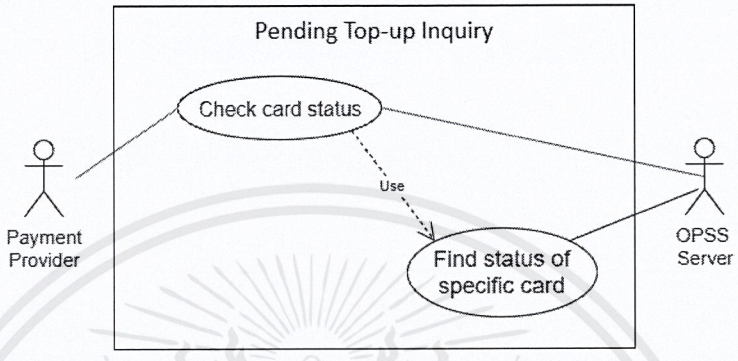
ซึ่งเซอร์วิสตัวนี้ไม่จำเป็นต้องส่งตัว data กลับเนื่องมาจากเซอร์วิสมีหน้าที่เพียงตรวจสอบสถานะเท่านั้นโดยสถานะที่จะอนุญาติหรือไม่อนุญาติให้เติมเงินก็จะอยู่ในฟิลด์ของ response description

ตารางที่ 3.9 Response Code

Response Code	Http Status Code	Description
00	200	Approved.
17	400	Invalid field in Body.
33	200	Card is pending update.
99	400	System Error.

3.4.3 การออกแบบทางเทคนิคของเซอร์วิส

ขั้นตอนการทำงานนั้นจะออกแบบเพื่อรองรับคำร้องขอ สำหรับการตรวจสอบสถานะการอนุญาติในการเติมเงินของบัตรโดยสาร ซึ่งหลักการทำงานก็คือ มีผู้ใช้งานมาส่งเลขบัตรที่ต้องการดูสถานะเข้ามาตรวจสอบว่าบัตรใบนี้อนุญาติให้เติมเงินได้รึป่าว โดยได้ออกแบบ use case การทำงานไว้ดังรูปที่3.23



รูปที่ 3.23 Use case ของเซอร์วิส Pending Top-up Status Inquiry

ตารางที่ 3.10 แสดงรายละเอียด use case เซอร์วิส Pending Top-up Status Inquiry

Use Case ID:	02
Use Case Name:	ภาพรวมการทำงานของเซอร์วิส Pending Top-up Inquiry
Actors:	Payment Provider
Description:	ผู้ใช้งานสามารถตรวจสอบสถานะของบัตรที่ต้องการทำรายการเติมเงินออนไลน์ได้
Trigger:	ผู้ใช้งานส่งหมายเลขบัตรและรายละเอียดตามที่กำหนด
Pre-conditions:	1. ต้องมีการใส่รายละเอียดให้ฟิลด์ต่าง ๆ ตามที่กำหนดไว้
Post-conditions:	-
Normal Flow:	1. ผู้ใช้งานอินพุตค่าของเลขบัตรและรายละเอียดต่าง ๆ เข้ามา 2. ระบบตรวจสอบสถานะของบัตรหมายเลขนั้นและดูสถานะว่าอนุญาติให้เติมเงินรึป่าว 3. ระบบส่งผลลัพธ์ว่าบัตรใบนี้สามารถทำการเติมเงินได้รึป่าว
Exception:	1. กรณีที่ Payment Provider ส่งฟิลด์โดยที่รูปแบบผิดจากเงื่อนไขที่ตกลงกันไว้ หรือไม่ได้ส่งฟิลด์ที่จำเป็นมาตามที่กำหนด

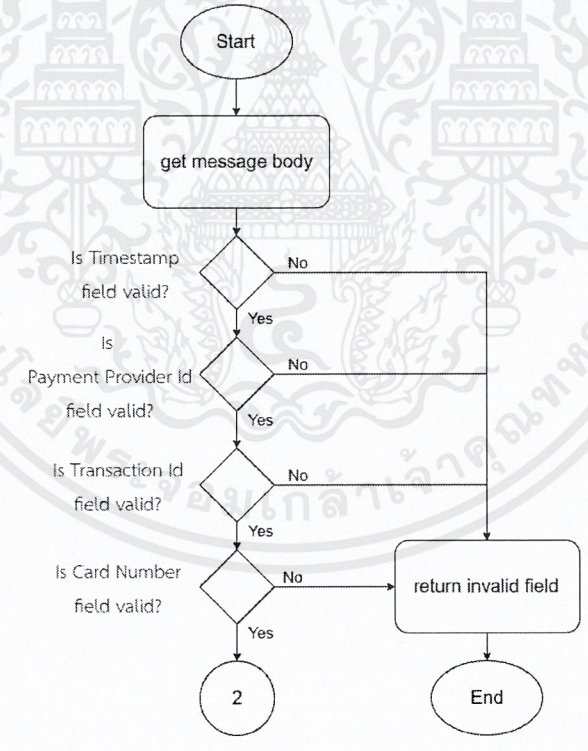
การทำงานนั้นก็จะใช้หลักการเดียวกันกับเซอร์วิส Transaction Status Inquiry นั่นก็คือจะแบ่งออกเป็นหลาย ๆ ชั้น ซึ่งการทำงานของเซอร์วิสนี้จะอธิบายการออกแบบได้ดังนี้

1. Controller

เนื่องจากเป็นชั้นแรกที่การร้องขอได้เข้ามาเจอก่อนดังนั้นจึงต้องทำพารามิเตอร์ไว้รับค่าดังนี้

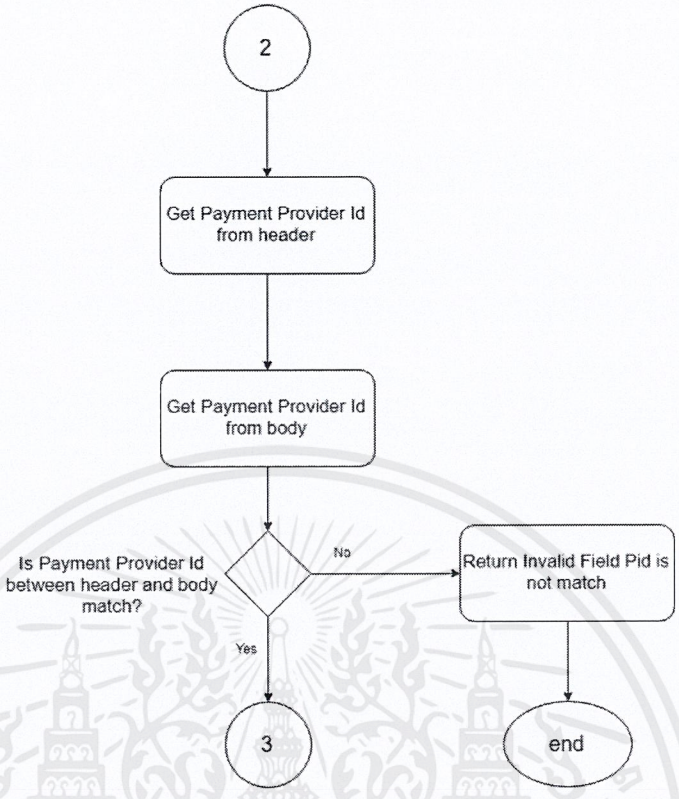
- HTTP Headers : เป็นพารามิเตอร์ที่รับส่วน header ของการร้องขอเข้ามา
- JSON Body : เป็นพารามิเตอร์ที่รับ body ที่มาพร้อมกับการร้องขอโดยรูปแบบของฟิลด์จะเป็นไปตามที่ออกแบบเอาไว้

ซึ่งการทำงานหลังจากได้รับการร้องขอมาแล้วนั้นก็จะมีขั้นตอนที่เหมือนกับเซอร์วิส Transaction Status Inquiry นั่นก็คือจะตรวจสอบความถูกต้องของฟิลด์ต่าง ๆ และตรวจสอบว่า payment provider id ใน header กับ body นั้นเหมือนกันหรือไม่โดยการตรวจสอบฟิลด์ของเซอร์วิสจะเป็นไปดังรูปที่ 3.24



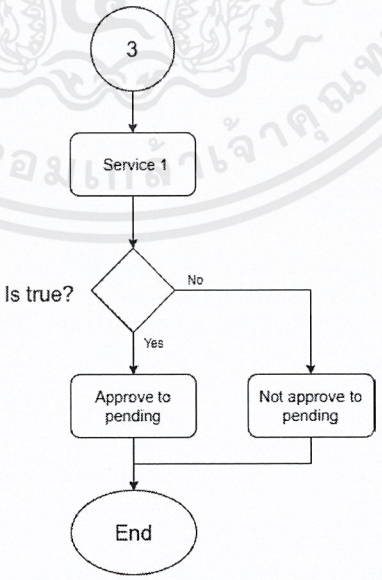
รูปที่ 3.24 Flow ตรวจสอบความถูกต้องของ field ที่ได้รับมา

ซึ่งถ้าฟิลด์ใดผิดรูปแบบของ data type หรือข้อกำหนดที่ได้ตกลงกันไว้นั้นก็จะไม่อนุญาติให้ทำงานต่อทันที และการตรวจสอบ payment provider id ก็เป็นไปดังรูปที่ 3.25



รูปที่ 3.25 Flow การตรวจสอบ payment provider id

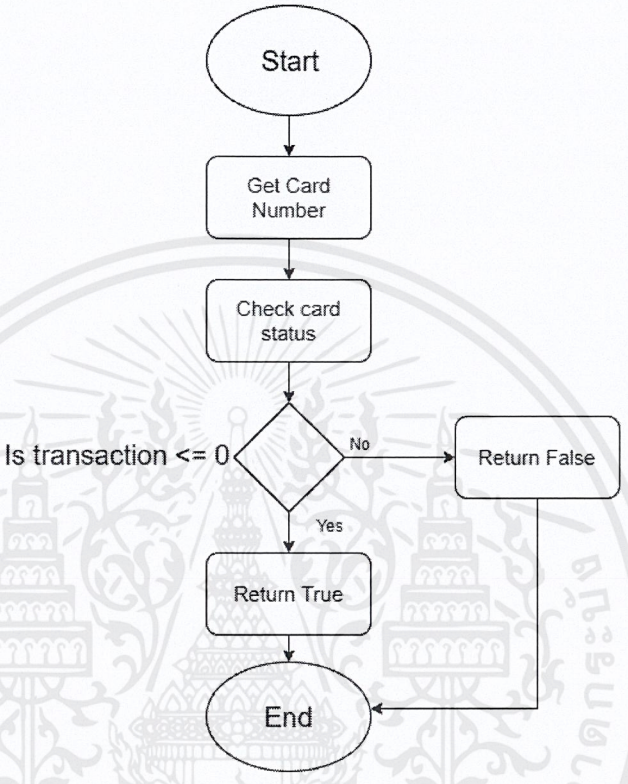
หลังจากที่ทำการตรวจสอบต่าง ๆ แล้วนั้นก็จะเข้าสู่การทำงานของตัวเซอร์วิสโดยตรงเลยโดยเซอร์วิสนี้นั้นจะทำการตรวจสอบค่าและส่งค่ากลับมาเป็น Boolean ว่าอนุญาติให้เติมรึป่าว แสดงดังรูปที่ 3.26



รูปที่ 3.26 Flow การทำงานในชั้น Controller

2. Service

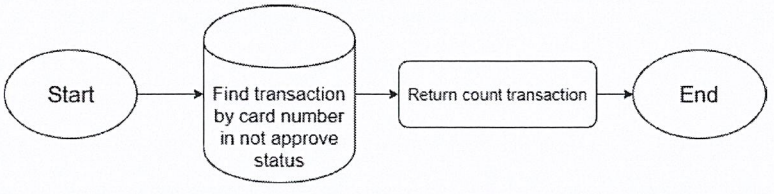
ชั้นนี้นั้นจะทำการตรวจสอบว่าหมายเลขการ์ดที่ได้รับเข้ามานั้นสถานะเป็นอะไรอนุญาติให้เติมรึป่าว โดยจะส่งเลขการ์ดนี้ไปตรวจสอบในชั้นของ repository แสดงดังรูปที่ 3.27



รูปที่ 3.27 Flow การทำงานในชั้น Service

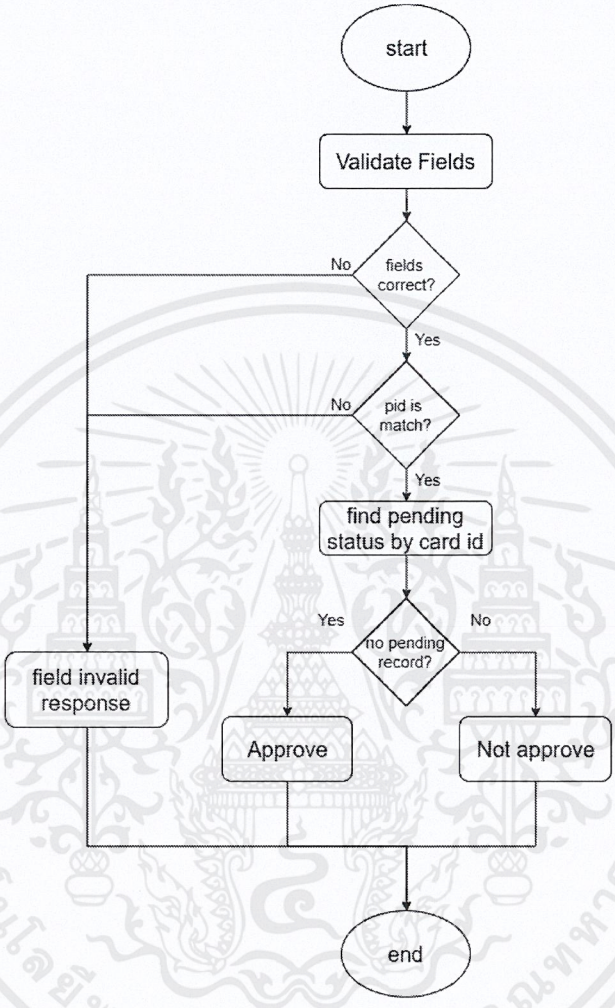
3. Repository

ชั้นนี้จะทำการตรวจสอบข้อมูลในฐานข้อมูลโดยใช้หมายเลขการ์ดและสถานะที่ไม่อนุญาติให้เติมเงินในการทำ query ซึ่งหมายเลขการ์ดนั้นจะถูกส่งมาพร้อมกับ JSON Body อยู่แล้วแต่สถานะต่างนั้นจะไปกำหนดไว้ที่ไฟล์ application.properties เพื่อให้สามารถเปลี่ยนค่าได้ง่ายเมื่อมีความต้องการเพิ่ม ลด หรือเปลี่ยนชื่อสถานะ แสดงดังรูปที่ 3.28



รูปที่ 3.28 Flow การทำงานในชั้น Repository

โดยการทำงานทั้งหมดจะเป็นไปดังรูปที่ 3.29



รูปที่ 3.29 Flow การทำงานทั้งหมดของเซอร์วิส Pending Top-up Inquiry

4. Entity

โดย entity ที่เซอร์วิสตัวนี้ต้องสร้างนั้นจะมีดังนี้

1. const_txn_status
2. response_code
3. tx_topup_request

3.5 การออกแบบฐานข้อมูล

ระบบของฐานข้อมูลที่นำมาใช้นั่นก็คือ MySQL โดยการออกแบบฐานข้อมูลนั้นก็จะถูกออกแบบให้มีตารางสำหรับเก็บเลขทำรายการต่าง ๆ ด้วยกันถึง 2 ตาราง นั่นก็คือ ตารางที่เก็บเลขทำรายการปกติ และอีกตารางนึงสำหรับรายการที่ไม่สำเร็จ นอกจากนี้ทั้งสองตารางยังเชื่อมต่อกับตารางอื่น ๆ ที่เก็บข้อมูลของที่เกี่ยวข้องกับรายละเอียดต่าง ๆ ใน data ที่ต้องการส่งกลับไป โดยจะใช้ foreign key เป็นตัวอ้างอิงความสัมพันธ์ของรายการต่าง ๆ

โดยตารางหลักนั้นจะเป็นตารางเก็บรายการที่มีสถานะปกติอยู่ซึ่งจะมีข้อมูลฟิลด์ที่มากกว่าอีกตารางนึงเพื่อนำข้อมูลไปใช้ประโยชน์กับเซอร์วิสอื่น ๆ ส่วนอีกตารางนั้นจะเก็บรายการที่ผิดพลาดต่าง ๆ ซึ่งฟิลด์บางอันไม่ได้ถูกบังคับสถานะให้เป็น not null ดังนั้นก็มีโอกาสที่จะไม่มีข้อมูลส่งกลับไปได้อย่างที่ได้อธิบายไปข้างต้น ส่วนข้อมูลรายละเอียดที่ต้องส่งกลับไปนั้นจะถูกอ้างอิงไปยังตารางที่เกี่ยวข้องโดยอ้างอิงจาก primary key นั่นเอง

3.6 การออกแบบวิธีทดสอบในเซอร์วิส

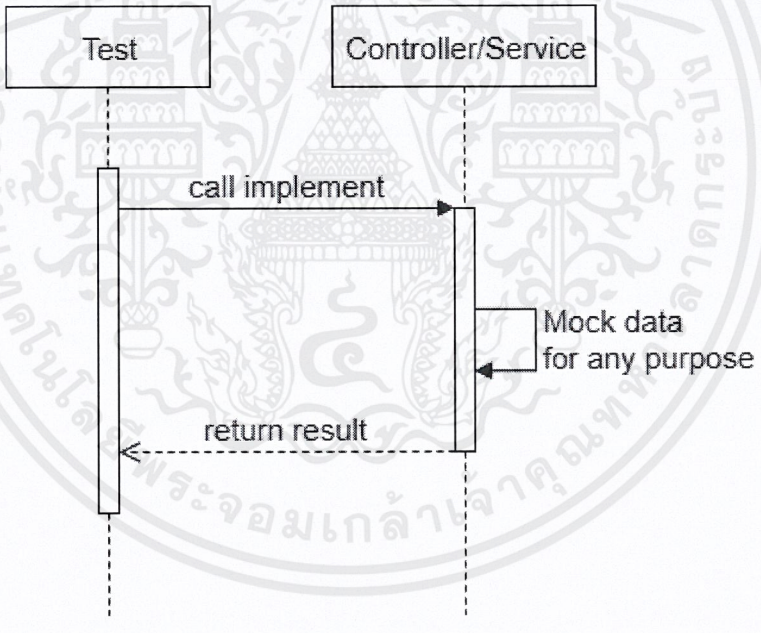
การทดสอบความถูกต้องของเซอร์วิสภายในตัวเองนั้น จะใช้รูปแบบการทดสอบแบบ unit test โดยใช้ไลบรารี่ Junit test เป็นหลัก การออกแบบนั้นจะทดสอบเป็นชั้น ๆ ไปคือ Controller, Service และ Repository ซึ่งต้องออกแบบการทดสอบให้ตรงตามจุดประสงค์ของชั้นนั้นอีกด้วย โดยเทคนิคที่ใช้ในการทดสอบจะแยกออกเป็น 2 วิธีดังนี้

1. การทดสอบที่ต้องเชื่อมต่อกับฐานข้อมูล

การทดสอบนี้มีจุดประสงค์ที่จะตรวจสอบว่า query ที่เราส่งเข้าไปยังฐานข้อมูลนั้นได้ผลลัพธ์แบบที่เราต้องการมาจริง ๆ ดังนั้นการทำงานก็จะเหมือนการรันปกติเลย แต่จำเป็นต้องสร้างฐานข้อมูลขึ้นมาใหม่เพื่อสำหรับใช้ทดสอบโดยเฉพาะในที่นี้จะใช้ H2 database ในการทดสอบซึ่งจะได้ไม่เปลืองทรัพยากรมาก การทดสอบโดยวิ่งเข้าฐานข้อมูลนั้นจะได้ข้อมูลที่เราสร้างไว้ก็จริงแต่การทดสอบของแต่ละชั้นนั้นจะใช้เวลานานมากเนื่องจากจะสร้างและทำลายฐานข้อมูลในทุกครั้งที่มีการทำการทดสอบในชั้นใหม่รวมถึงข้อมูลที่จะถูกเก็บอยู่ภายในฐานข้อมูลด้วย ซึ่งจะทำให้กินเวลานานมากกว่าจะรันการทดสอบแต่ละครั้งจะเสร็จ การทดสอบแบบนี้จะได้ความถูกต้องของข้อมูลที่แน่นอนแต่จะกินเวลานานซึ่งจะใช้ทดสอบกับชั้น Repository

2. การทดสอบโดยไม่เชื่อมต่อกับฐานข้อมูล

การทดสอบนี้ใช้ทดสอบแค่ business logic ของเราว่ามันถูกต้องเป็นไปอย่างที่ควรรึป่าว โดยการทดสอบแบบนี้จะไม่จำเป็นต้องเชื่อมต่อฐานข้อมูลโดยตรง โดยจะใช้ Mockito มาจำลองข้อมูลให้แทน ซึ่งจะใช้ตรวจสอบว่าเมทธอดนั้นได้ถูกเรียกใช้รึป่าวหากมีการเรียกใช้เมทธอดนั้นก็จะทำการส่งค่า mock up ที่เราได้กำหนดไว้กลับมาซึ่งวิธีนี้จะช่วยประหยัดเวลาในการรันของโปรแกรมได้ดีมากเนื่องจากไม่จำเป็นต้องสร้างฐานข้อมูลเพื่อใช้ทดสอบและถ้า business logic ของเราไม่ถูกต้องแล้วเราไม่ได้คาดหวังให้การรันนั้นไปเจอกับเมทธอดใด ๆ ก็ตามนั้นเมทธอดนั้นจะไม่ส่งค่าอะไรกลับมาแต่ โดยการทดสอบแบบนี้จะนำมาใช้ในชั้นของ Service และ Controller แต่ชั้น Controller นั้นจะแตกต่างออกไปนิดหน่อยตรงที่จำเป็นต้องส่งข้อมูลเข้า URL ตัวเองแล้วตรวจเช็คสถานะของ HTTP แต่โดยภาพรวมแล้วจะทำงานคล้ายคลึงกัน ซึ่งจะทดสอบโดยเรียกชั้นการทำงานของตนเองแล้วดูผลลัพธ์ โดยการทำงานจะแสดงดังรูปที่ 3.32



รูปที่ 3.30 Sequence Diagram ของการทดสอบแบบ unit test

บทที่4
ผลการวิจัย

4.1 ผลการทดสอบการทำงานของระบบ

ในส่วนของการทดสอบการทำงานของทั้งระบบนั้นจะเป็นการทดสอบว่าการตั้งค่า และแนวคิดต่าง ๆ ที่ได้ถูกออกแบบไว้ทำงานไปได้ด้วยดี ซึ่งตัวเซอร์วิสนั้นจะดึงการตั้งค่าจากส่วนกลาง และทำงานร่วมกับ service discovery ได้อย่างถูกต้อง โดยทั้งหมดของภาพรวมนั้นจะถูกพูดถึงการทำแคชแก่ response code ที่ตอบกลับอีกด้วย โดยทดสอบจะเป็นไปดังนี้

4.1.1 ผลการทดสอบการตั้งค่าของระบบกลาง

การจะทดสอบว่าการตั้งค่าระบบกลางนั้นทำงานได้ปกติจะตรวจสอบได้จากการเข้าถึงไฟล์การตั้งค่าต่าง ๆ ภายในระบบกลางว่าสามารถดูรายละเอียดในนั้นได้รึป่าว ซึ่งไฟล์เหล่านั้นจะบ่งบอกการตั้งค่าที่ได้ถูกกำหนดเอาไว้ โดยการทดสอบนั้นก็จะเข้าผ่าน URL ตามพอร์ตทีได้ตั้งค่าให้แก่ระบบกลางในที่นี้จะใช้เป็นพอร์ต 8888 ดังนั้น URL ที่ใช้ตรวจสอบคือ http://localhost:8888/pending-service/default และ http://localhost:8888/transaction-service/default โดยผลลัพธ์จะแสดงดังรูปที่ 4.1 และรูปที่ 4.2 ตามลำดับ


```
{
  "name": "pending-service",
  "profiles": [
    "default"
  ],
  "label": null,
  "version": null,
  "state": null,
  "propertySources": [
    {
      "name": "file:center-config/pending-service.properties",
      "source": {
        "server.port": "8086",
        "server.servlet.context-path": "/opssapi/transaction",
        "spring.datasource.url": "jdbc:mysql://127.0.0.1:3306/bem_opss?useTimezone=true&serverTimezone=Asia/Bangkok",
        "spring.datasource.username": "root",
        "spring.datasource.password": "root",
        "spring.jpa.hibernate.ddl-auto": "validate",
        "spring.jpa.properties.hibernate.show_sql": "true",
        "spring.jpa.properties.hibernate.dialect": "org.hibernate.dialect.MySQL5InnoDBDialect",
        "spring.jpa.hibernate.naming.physical-strategy": "org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl",
        "spring.jpa.properties.hibernate.format_sql": "true",
        "eureka.client.service-url.default-zone": "http://localhost:8761/eureka-server/"
      }
    }
  ]
}
```


รูปที่ 4.1 การตั้งค่าของเซอร์วิส Pending Top-up Inquiry

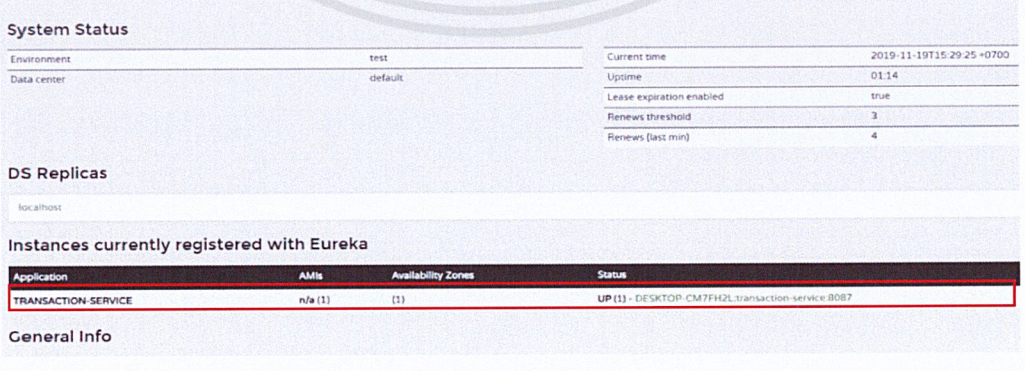
```
{
  "name": "transaction-service",
  "profiles": [
    "default"
  ],
  "label": null,
  "version": null,
  "state": null,
  "propertySources": [
    {
      "name": "file:center-config/transaction-service.properties",
      "source": {
        "server.port": "8087",
        "server.servlet.context-path": "/opssapi/transaction",
        "spring.datasource.url": "jdbc:mysql://127.0.0.1:3306/bem_opss?useTimezone=true&serverTimezone=Asia/Bangkok",
        "spring.datasource.username": "root",
        "spring.jpa.hibernate.ddl-auto": "validate",
        "spring.jpa.properties.hibernate.show_sql": "true",
        "spring.jpa.properties.hibernate.dialect": "org.hibernate.dialect.MySQL5InnoDBDialect",
        "spring.jpa.hibernate.naming.physical-strategy": "org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl",
        "spring.jpa.properties.hibernate.format_sql": "true",
        "transaction.card.padding": "%08d",
        "transaction.refId.padding": "%020d",
        "service.ordinary.transaction.start": "101",
        "service.ordinary.transaction.end": "200",
        "service.exception.transaction.start": "1",
        "service.exception.transaction.end": "100",
        "module.pending.status": "3,2,1",
        "eureka.client.service-url.default-zone": "http://localhost:8761/eureka-server/",
        "management.endpoints.web.exposure.include": "*",
        "management.endpoint.health.show-details": "always"
      }
    }
  ]
}
```


รูปที่ 4.2 การตั้งค่าของเซอร์วิส Transaction Status Inquiry

เมื่อเราสามารถเข้าถึงการตั้งค่าระบบกลางได้แล้วนั้นก็หมายความว่าเซอร์วิสอื่น ๆ ในระบบก็สามารถมาค้นหาหาไฟล์ที่ใช้ในการตั้งค่าสำหรับเซอร์วิสของตนเองภายในนี้ได้ด้วยเช่นกัน

4.1.2 ผลการทดสอบการเชื่อมต่อกับ service discovery

เนื่องจากเซอร์วิสทุกตัวในระบบจำเป็นต้องเชื่อมต่อกับ service discovery ซึ่งเป็นตัว Eureka ดังนั้นเมื่อเซอร์วิสของเราสามารถเชื่อมต่อกับ Eureka ได้นั้นก็จะสามารถตรวจสอบได้จากพอร์ตที่รันการทำงานของเซอร์วิส Eureka อยู่นั่นเองโดยผลลัพธ์เมื่อมีการนำเซอร์วิสมาเชื่อมต่อได้ผลลัพธ์ดังแสดงในรูปที่ 4.3 ซึ่งทำงานได้ตามปกติ เนื่องจากมีชื่อเซอร์วิสที่เรารันการทำงานมาขึ้นอยู่ในหน้าของ Eureka แล้ว



รูปที่ 4.3 หน้าต่างแสดงเซอร์วิสที่เชื่อมต่อกับ Eureka Service

4.1.3 ผลการทดสอบการทำ Cache

การตอบกลับของทุกเซอร์วิสจำเป็นต้องใช้การตอบกลับโดยอ้างอิงข้อมูลจากตารางในฐานข้อมูล นั้นจะถูกทำงานเพียงแค่ครั้งแรกในการรันเซอร์วิส ซึ่งจะสามารถดูได้จาก command line ที่รันเซอร์วิสนั้น โดยลักษณะของการทำงานตั้งแต่เริ่มต้นจะสังเกตุได้จากรูปที่ 4.4

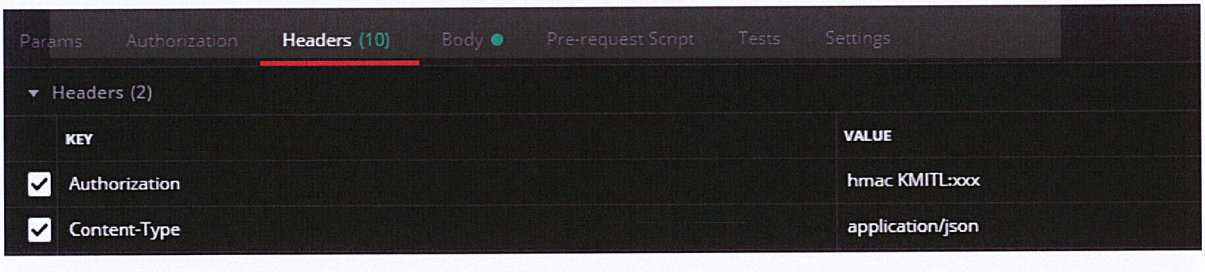
```
Hibernate:
    select
        responseco0_.RESP_ID as RESP_ID1_3_,
        responseco0_.RESP_CODE as RESP_COD2_3_,
        responseco0_.RESP_DESC as RESP_DES3_3_
    from
        CONST_RESPONSE_CODE responseco0_
```

รูปที่ 4.4 Query ในการทำ cache

จะเห็นได้ว่าเมื่อทำการรันเซอร์วิสนั้นขึ้นมา Hibernate framework จะทำ query ในการเรียก response code จากตาราง CONST_RESPONSE_CODE ให้ทันทีและสามารถนำกลับมาใช้ในระบบได้ตลอดการทำงาน

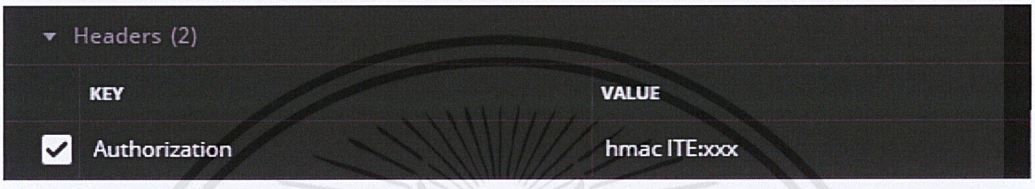
4.2 ผลการทดสอบการทำงานของเซอร์วิส

ในที่นี้จะใช้โปรแกรม Postman ในการทดสอบเซอร์วิสต่าง ๆ โดยจะส่งทั้ง header และ body เข้าไปยัง URL ของเซอร์วิสทั้งสอง และดูผลลัพธ์ที่เกิดขึ้นว่าทำงานได้ถูกต้องรึป่าวโดยจะอ้างอิงจาก response code และ http status ด้วยว่าส่งมาถูกต้องตามข้อตกลงรึป่าว ซึ่งทั้งสองเซอร์วิสนั้นจะใช้ข้อมูลที่อยู่ภายใน header ที่ทดสอบนั้นอยู่ในรูปแบบเดียวกันลักษณะเดียวกันดังแสดงในรูปที่ 4.5



รูปที่ 4.5 Header ที่ส่ง

ซึ่งทั้งสองเซอร์วิสนั้นจำเป็นต้องมีการตรวจเช็คในเรื่อง Payment Provider Id ซะก่อน ถ้า Authorization field นั้นไม่ตรงกับ Payment Provider Id field การทำงานก็จะหยุดลง โดยจะทดสอบจากการที่ส่งสองฟิลด์ในค่าที่ต่างกัน โดยส่งส่งค่าจาก header เป็น ITE แต่ใน body เป็น KMITL ดังแสดงในรูปที่ 4.6 ซึ่งทั้งสองเซอร์วิสนั้นจะใช้การ validate ในรูปแบบเดียวกันดังนั้นจึงจะทำการทดสอบเพียงแค่เซอร์วิสใดเซอร์วิสหนึ่งเท่านั้นก็จะได้รับผลลัพธ์ที่เหมือนกัน โดยจะทดสอบโดยการส่ง JSON body ดังที่แสดงในรูปที่ 4.7 และการทดสอบจะได้ผลลัพธ์ดังแสดงในรูปที่ 4.8



รูปที่ 4.6 Header ที่แก้ไขเป็น ITE

รูปที่ 4.7 JSON Body ที่ส่ง

รูปที่ 4.8 ผลลัพธ์การทดสอบความถูกต้องของ Payment Provider Id

ดังนั้นการทดสอบในเรื่องที่เกี่ยวกับการระบุตัวตนระหว่าง header และ body นั้นจะให้ผลลัพธ์ตามที่ถูกออกแบบไว้ ซึ่งรวมถึงในแง่ของการยืนยันที่ header และ body ตรงกันนั้นก็ได้ผลลัพธ์ตามที่ได้ออกแบบไว้

4.2.1 ผลการทดสอบเซอร์วิส Transaction Status Inquiry

ในส่วน body นั้นใช้ฟิลด์ต่าง ๆ ตามที่ได้ออกแบบเอาไว้ ซึ่งจะนำชื่อฟิลด์ที่ได้กำหนดเอาไว้นั้นไปใส่ไว้ส่วน body ของโปรแกรม Postman เพื่อส่งเข้าตัวเซอร์วิสโดยเซอร์วิสนี้นั้นจะมีฟิลด์ที่ใช้อ้างถึง ดังแสดงในรูปที่ 4.9

```
3         "paymentProviderId":"",
4         "timestamp":"",
5         "transactionId":"",
6         "cardNumber":"",
7         "originalTransactionId":"",
8         "originalAmount":"",
9         "originalTimestamp":"",
10        "referenceId":""
```

รูปที่ 4.9 JSON body request

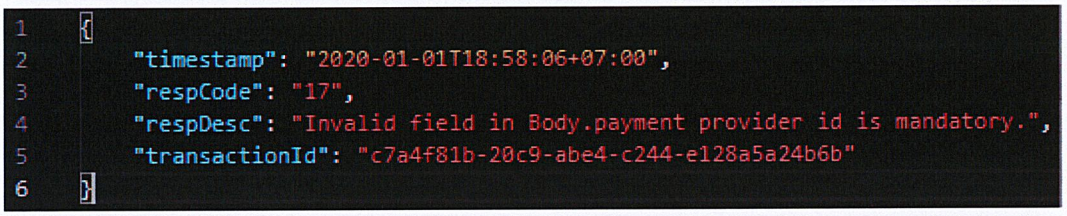
จะเริ่มทดสอบจากการทดสอบเกี่ยวกับการตรวจสอบความถูกต้องของฟิลด์ซะก่อนว่าฟิลด์ต่าง ๆ นั้นเป็นไปตามที่วางแผนเอาไว้หรือไม่ ซึ่งจะทดสอบไปทีละฟิลด์ โดยตามข้อตกลงแล้วนั้นจะมีฟิลด์ที่จำเป็นจะต้องส่งข้อมูลมา ซึ่งถ้าฟิลด์เหล่านี้หายไปหรือส่งค่ามาเป็นค่าว่าง การทำงานก็จะหยุดทันที และส่งผลลัพธ์ที่ผิดพลาดกลับมา โดยจะแยกเป็นการส่งของฟิลด์การร้องขอ และการตอบกลับตามลำดับ ซึ่งการทดสอบจะเป็นดังต่อไปนี้

1. Payment Provider Id

จะส่งคำร้องไปโดยไม่ใส่ฟิลด์ Payment Provider Id ดังแสดงในรูปที่ 4.10 และผลลัพธ์ที่ได้จะเป็นไปตามรูปที่ 4.11

```
1 {
2         "timestamp":"2019-09-27T12:00:00+07:00",
3         "transactionId":"c7a4f81b-20c9-abe4-c244-e128a5a24b6b",
4         "cardNumber":"10345627",
5         "originalTransactionId":"oriTxnId",
6         "originalAmount":"2500",
7         "originalTimestamp":"2019-10-05T16:25:00+07:00",
8         "referenceId":1
9 }
```

รูปที่ 4.10 การส่ง Request โดยไม่มีค่า Payment Provider Id

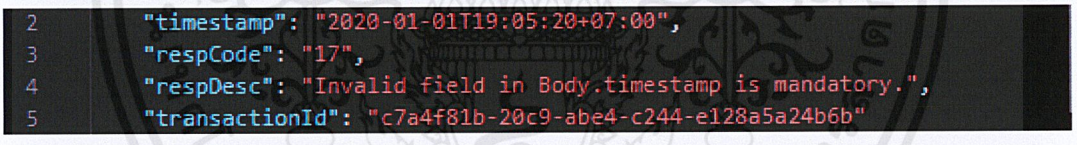


รูปที่ 4.11 Response ของ การส่ง Request โดยไม่มีค่า Payment Provider Id

2. Timestamp

จะส่งคำร้องไปโดยไม่ใส่ฟิลด์ Timestamp ดังแสดงในรูปที่ 4.12 และผลลัพธ์ที่ได้จะเป็นไปตามรูปที่ 4.13

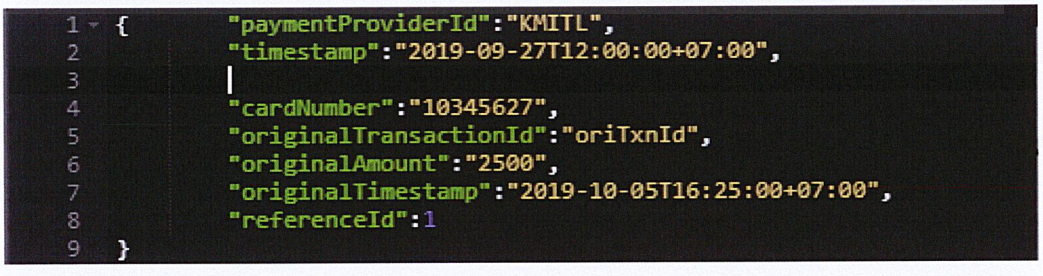
รูปที่ 4.12 การส่ง Request โดยไม่มีค่า Timestamp



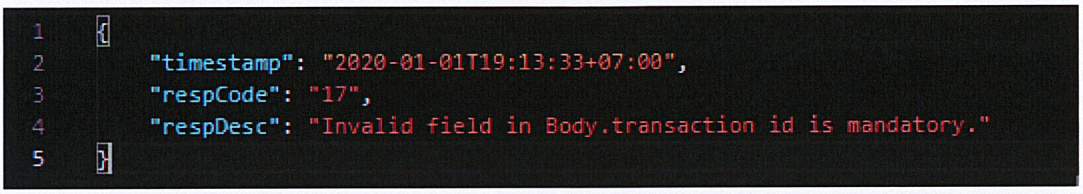
รูปที่ 4.13 Response ของ การส่ง Request โดยไม่มีค่า Timestamp

3. Transaction Id

จะส่งคำร้องไปโดยไม่ใส่ฟิลด์ Transaction Id ดังแสดงในรูปที่ 4.14 และผลลัพธ์ที่ได้จะเป็นไปตามรูปที่ 4.15



รูปที่ 4.14 การส่ง Request โดยไม่มีค่า Transaction Id

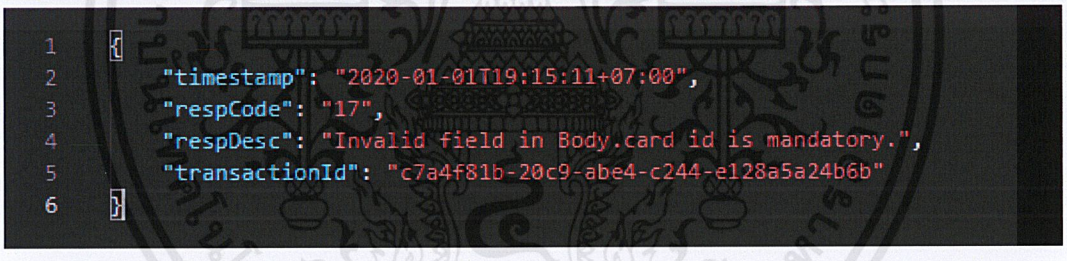


รูปที่ 4.15 Response ของ การส่ง Request โดยไม่มีค่า Transaction Id

4. Card Number

จะส่งคำร้องไปโดยไม่ใส่ฟิลด์ Card Number ดังแสดงในรูปที่ 4.16 และผลลัพธ์ที่ได้จะเป็นไปตามรูปที่ 4.17

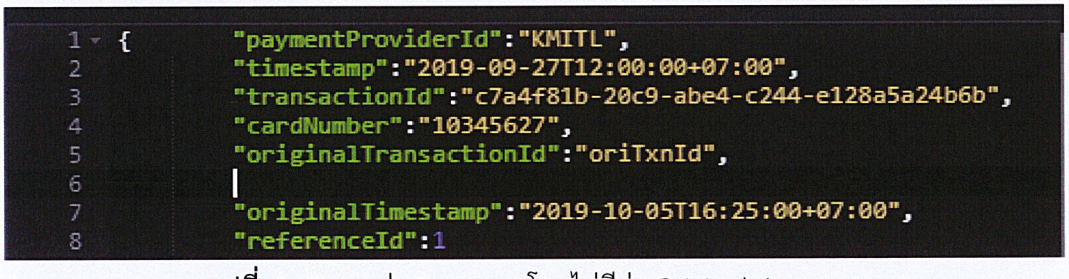
รูปที่ 4.16 การส่ง Request โดยไม่มีค่า Card Number



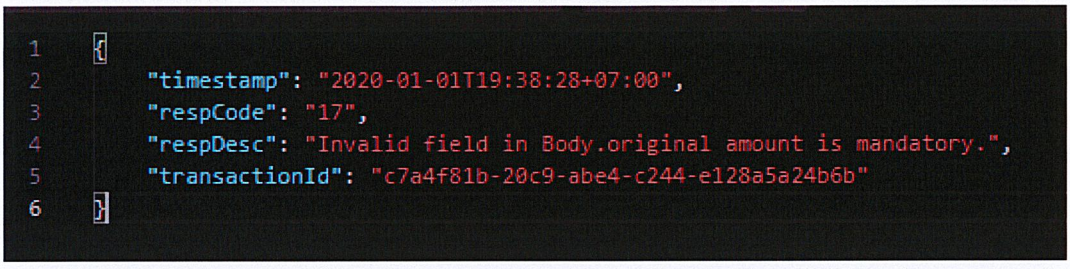
รูปที่ 4.17 Response ของ การส่ง Request โดยไม่มีค่า Card Number

5. Original Amount

จะส่งคำร้องไปโดยไม่ใส่ฟิลด์ Original Amount ดังแสดงในรูปที่ 4.18 และผลลัพธ์ที่ได้จะเป็นไปตามรูปที่ 4.19



รูปที่ 4.18 การส่ง Request โดยไม่มีค่า Original Amount

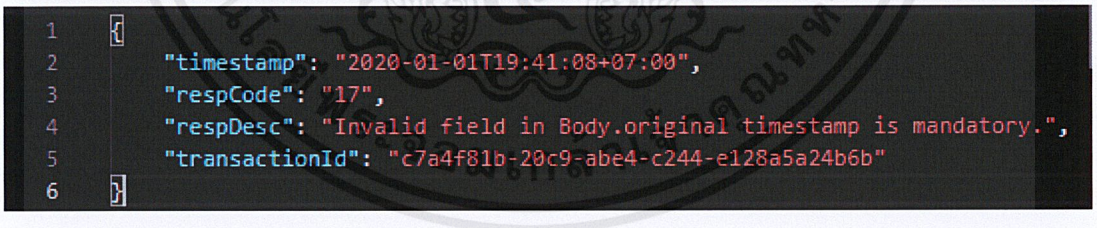


รูปที่ 4.19 Response ของ การส่ง Request โดยไม่มีค่า Card Number

6. Original Timestamp

จะส่งคำร้องไปโดยไม่ใส่ฟิลด์ Original Timestamp ดังแสดงในรูปที่ 4.20 และผลลัพธ์ที่ได้จะเป็นไปตามรูปที่ 4.21

รูปที่ 4.20 การส่ง Request โดยไม่มีค่า Original Timestamp



รูปที่ 4.21 Response ของ การส่ง Request โดยไม่มีค่า Original Timestamp

โดยฟิลด์เหล่านี้นั้นถ้าไม่ได้มีการส่งค่ามาก็จะไม่เกิดการทำงานอย่างแน่นอน แต่ก็ยังมีบางฟิลด์ที่ไม่จำเป็นต้องส่งมาก็สามารถทำงานได้ซึ่งนั่นก็คือ

- Original Transaction Id
- Reference Id

โดยการทดสอบสองฟิลด์นี้นั้นจะทำการทดสอบรวมกับผลลัพธ์ที่เซอร์วิสทำงานแบบสำเร็จไม่มีข้อผิดพลาดใดและมีเลขทำรายการที่ต้องการหาอยู่จริงในฐานข้อมูลเพื่อให้การทดสอบของสองฟิลด์นี้ชัดเจนมากยิ่งขึ้น จะเริ่มการทดสอบจากการที่ไม่ส่งค่าใดค่านึงออกไปแล้วดูผลลัพธ์

- การร้องขอและการตอบกลับโดยที่ไม่มีการส่งค่าของ Original Transaction Id โดยผลลัพธ์จะเรียงจากบนลงล่างตามลำดับ ดังแสดงในรูปที่ 4.22

รูปที่ 4.22 ผลลัพธ์การทดสอบเมื่อไม่มีค่า Original Transaction Id

- การร้องขอและการตอบกลับโดยที่ไม่มีการส่งค่าของ Reference Id โดยผลลัพธ์จะเรียงจากบนลงล่างตามลำดับ ดังแสดงในรูปที่ 4.23

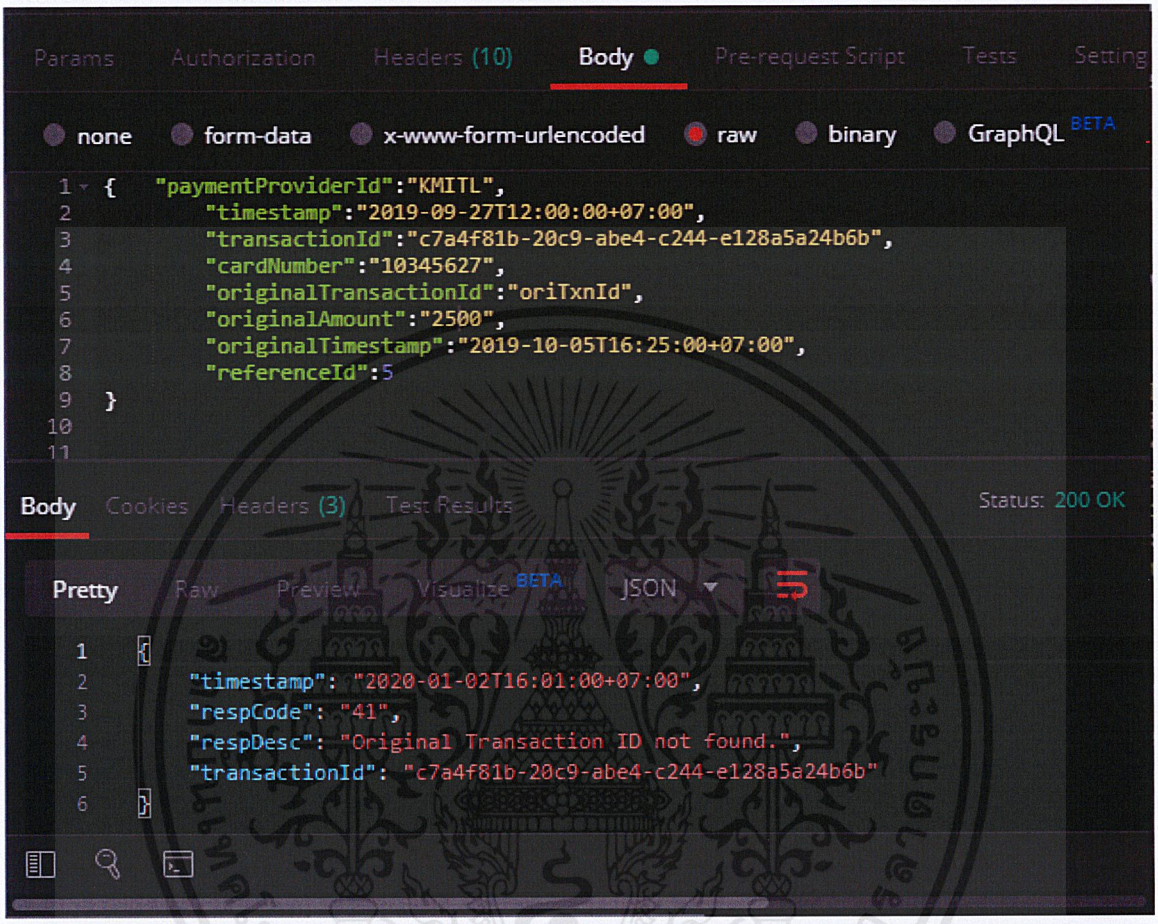
รูปที่ 4.23 ผลลัพธ์การทดสอบเมื่อไม่มีค่า Reference Id

จะเห็นได้ว่าผลลัพธ์ของทั้งสองการทำงานนั้นได้ผลลัพธ์เดียวกันออกมาแต่แตกต่างกันที่ timestamp เนื่องจากการร้องขอกันคนละเวลา จึงทำให้มั่นใจได้ว่าการเลือกส่งค่าใดค่าหนึ่งนั้นได้ผลถูกต้องตามที่ได้ออกแบบเอาไว้ โดยต่อไปจะแสดงให้เห็นถึงการที่ส่งค่ามาทั้งสองฟิลด์แล้วดูว่าผลลัพธ์นั้นจะเป็นไปตามที่ออกแบบรึป่าวนั้นก็คือจะใช้สองฟิลด์นี้หาร่วมกัน โดยจะแบ่งเป็นการส่งค่าถูกต้องกับทั้งสองฟิลด์และการส่งฟิลด์ใดฟิลด์หนึ่งผิด

- ผลลัพธ์ของการส่งค่าถูกทั้งสองฟิลด์ ดังแสดงในรูปที่ 4.24

รูปที่ 4.24 ผลลัพธ์การทดสอบเมื่อส่งมาครบทุกค่า

- ผลลัพธ์ของการส่งฟิลด์ใดฟิลด์หนึ่งผิด ดังแสดงในรูปที่ 4.25

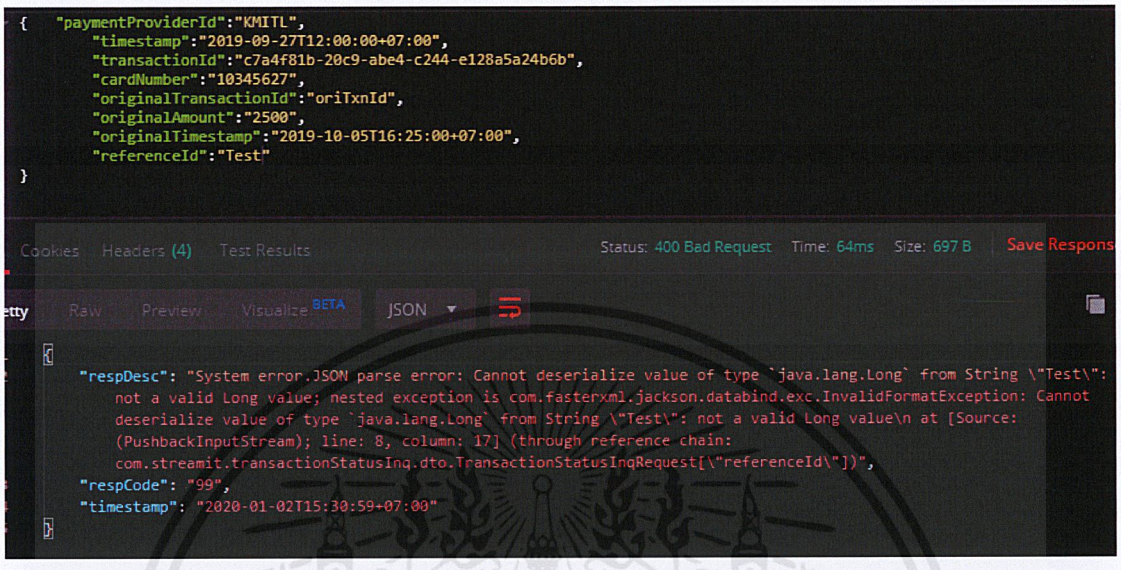


รูปที่ 4.25 ผลลัพธ์การทดสอบเมื่อส่งรายละเอียดไม่ถูกต้อง

เห็นได้ว่าการที่เปลี่ยนเป็นส่งค่า Reference Id เป็น 5 แทนนั้นจะทำให้เซอร์วิสไม่สามารถหาเลขทำรายการเจอเนื่องจากมีค่าใดค่าหนึ่งไม่ถูกต้องซึ่งนั่นก็คือ Reference Id จึงทำให้ผลลัพธ์ที่ได้ออกมานั้นคือการหาเลขทำรายการไม่เจอ ซึ่งจะเป็นไปในทางเดียวกันหางการส่งค่า Reference Id ถูกต้องแต่ค่า Original Transaction Id ผิดก็จะได้ผลลัพธ์แบบเดียวกัน วิ่งในการส่งค่าที่ผิดนั้นจะเป็นไปในทางเดียวกันกับทุกฟิลด์ที่ผลต่อการค้นหาตามที่ได้ออกแบบไว้

ต่อไปจะเป็นการทดสอบหากมีการส่งรูปแบบที่ผิดตั้งแต่ JSON format ก็จำเป็นต้องทำการตอบกลับข้อผิดพลาดที่ได้เกิดขึ้นกับตัวโปรแกรม โดยข้อความของข้อผิดพลาดเหล่านั้นจะเป็นข้อความจากทาง JSON โดยตรงแต่ต้องส่งกลับในรูปแบบที่ได้ตกลงกันเอาไว้ด้วยโดยจะทดสอบด้วยการส่งค่าที่เป็น String

ไปในฟิลด์ที่ต้องการค่า Number และการทดสอบด้วยการส่งค่า String และ Number รวมกันไปใน value ของ JSON โดยผลลัพธ์จะเป็นไปดังแสดงในรูปที่ 4.26 และรูปที่ 4.27



รูปที่ 4.26 ผลลัพธ์การส่งค่า String แทนการส่ง Number

รูปที่ 4.27 ผลลัพธ์การส่งค่า String และ Number ไปด้วยกัน

จะเห็นได้จากการทดสอบต่าง ๆ ว่าการส่งการร้องขอและการตอบกลับในทุกการทดสอบนั้นเป็นไปตามที่ออกแบบไว้ทั้งหมด โดยสังเกตุได้จากข้อมูลการส่งและการตอบกลับ รวมถึง response code ที่ส่งกลับมาตามผลลัพธ์การทำงานของตัวเซอร์วิส และ HTTP status ก็มีผลลัพธ์ตรงตามที่ออกแบบไว้

4.2.2 ผลการทดสอบเซอร์วิส Pending Top-up Inquiry

การทดสอบเซอร์วิสนี้ก็จะเริ่มจากการทดสอบความถูกต้องของฟิลด์ที่ส่งมาเหมือนกัน แต่ตัวเซอร์วิสนี้จะไม่มีฟิลด์ที่เป็นเงื่อนไขเพราะฉะนั้นฟิลด์ทุกตัวจำเป็นต้องส่งค่ามา หลังจากทดสอบโดยการไม่ส่งค่าไปในเซอร์วิส Transaction Status Inquiry แล้วนั้นก็ได้ผลลัพธ์ตามที่ออกแบบไว้ ดังนั้นในการทดสอบเซอร์วิสตัวนี้จะใช้การทดสอบโดยการส่งค่าที่เป็นค่าว่างให้แก่ทุกฟิลด์แทน ดังแสดงในรูปที่ 4.28

```
{
    "paymentProviderId":"",
    "timestamp":"",
    "transactionId":"",
    "cardNumber":""
}
```

รูปที่ 4.28 JSON Body Request

โดยต่อไปจะทำการตรวจสอบทีละฟิลด์ไปดังนี้

1. Payment Provider Id

   จะส่งคำร้องไปโดยไม่ใส่ฟิลด์ Payment Provider Id ดังแสดงในรูปที่ 4.29 และผลลัพธ์ที่ได้จะเป็นไปตามรูปที่ 4.30

```
{
    "paymentProviderId":"",
    "timestamp":"2019-08-13T01:58:03+07:00",
    "transactionId":"44e128a5",
    "cardNumber":"12345678"
}
```

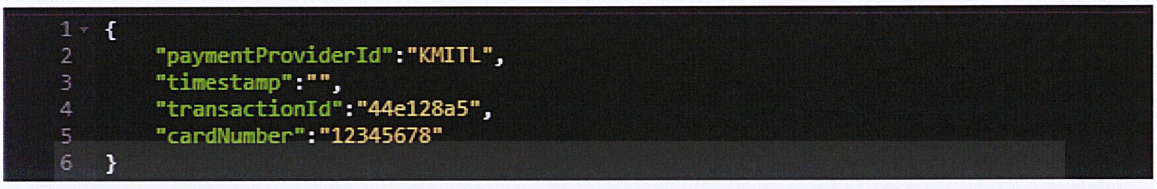
รูปที่ 4.29 การส่งค่า Request โดยส่งค่า Payment Provider Id เป็นค่าว่าง

```
{
    "timestamp": "2020-01-02T18:04:01+07:00",
    "respCode": "17",
    "respDesc": "Invalid field in Body.payment provider id is mandatory.",
    "transactionId": "44e128a5"
}
```

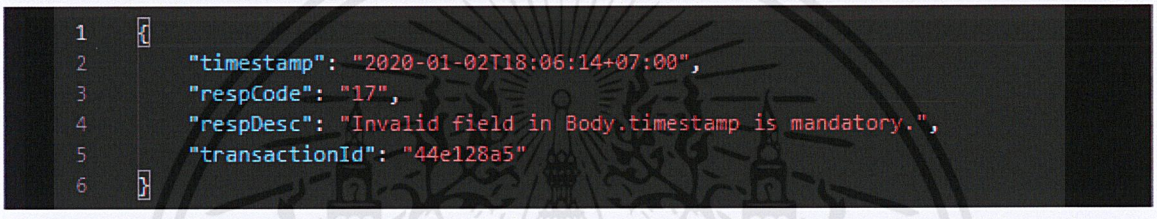
รูปที่ 4.30 Response ของ การส่ง Request โดยไม่มีค่า Payment Provider Id

2. Timestamp

จะส่งคำร้องไปโดยไม่ใส่ฟิลด์ Timestamp ดังแสดงในรูปที่ 4.31 และผลลัพธ์ที่ได้จะเป็นไปตามรูปที่ 4.32



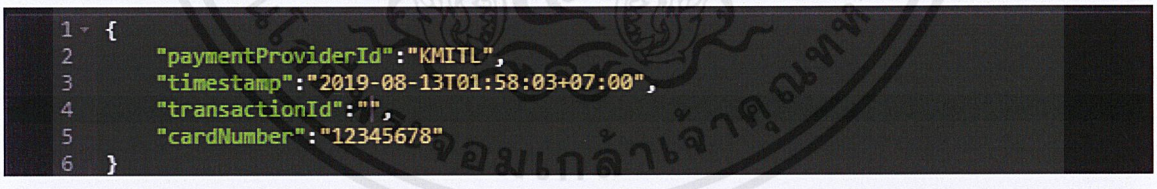
รูปที่ 4.31 การส่งค่า Request โดยส่งค่า Timestamp เป็นค่าว่าง



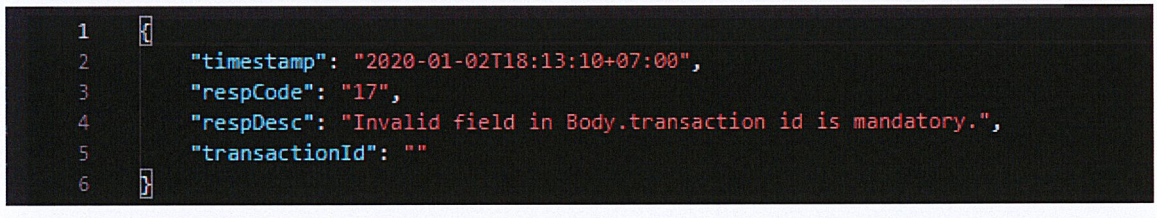
รูปที่ 4.32 Response ของ การส่ง Request โดยไม่มีค่า Timestamp

3. Transaction Id

จะส่งคำร้องไปโดยไม่ใส่ฟิลด์ Transaction Id ดังแสดงในรูปที่ 4.33 และผลลัพธ์ที่ได้จะเป็นไปตามรูปที่ 4.34



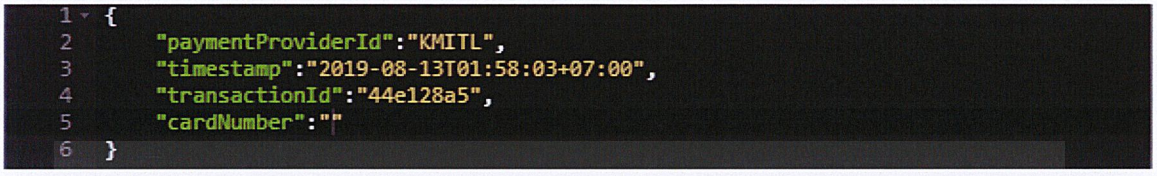
รูปที่ 4.33 การส่งค่า Request โดยส่งค่า Transaction Id เป็นค่าว่าง



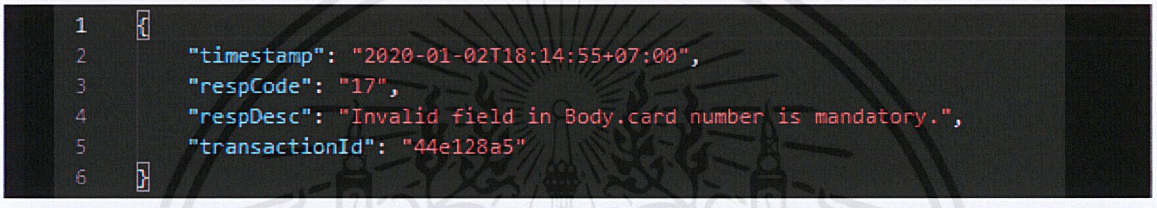
รูปที่ 4.34 Response ของ การส่ง Request โดยไม่มีค่า Transaction Id

4. Card Number

จะส่งคำร้องไปโดยไม่ใส่ฟิลด์ Card Number ดังแสดงในรูปที่ 4.35 และผลลัพธ์ที่ได้จะเป็นไปตามรูปที่ 4.36



รูปที่ 4.35 การส่งค่า Request โดยส่งค่า Card Number เป็นค่าว่าง



รูปที่ 4.36 Response ของ การส่ง Request โดยไม่มีค่า Card Number

และเมื่อไม่มีการส่งฟิลด์ใด ๆ มาเลยจะได้ผลลัพธ์ดังแสดงในรูปที่ 4.37

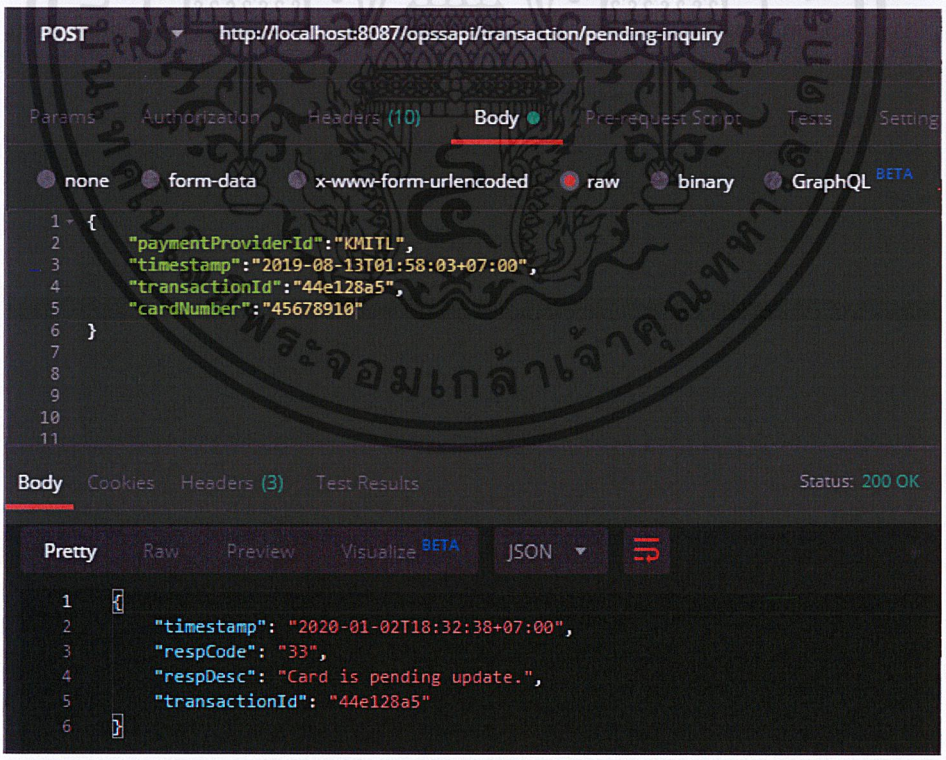
รูปที่ 4.37 Response ของ การส่ง Request โดยไม่มีค่าใดเลย

ซึ่งจะเห็นได้ว่าผลลัพธ์ของข้อมูลที่ส่งกลับมาและ HTTP status นั้นได้ผลลัพธ์ตรงตามที่ได้ออกแบบเอาไว้ ซึ่งในส่วนต่อไปนั้นจะทำการทดสอบเกี่ยวกับผลลัพธ์ที่จะได้รับกลับมาว่าบัตรใบดังกล่าวที่ได้ส่งเข้ามานั้นอนุญาติให้เติมเงินรึป่าว จากการทดสอบเซอร์วิส Transaction Status Inquiry ไปนั้นจะเห็นได้ว่าข้อมูลการทำรายการล่าสุดของบัตรหมายเลข 45678910 นั้นมีสถานะที่เป็น PENDING อยู่ดังแสดงในรูปที่ 4.38 ดังนั้นการ์ดใบนี้จะไม่อนุญาติให้เติมเงินอย่างแน่นอน ดังนั้นถ้าเกิดมีการเรียกเซอร์วิส

Pending Top-up Inquiry เพื่อดูสถานะของบัตรใบนี้แล้วนั้นสถานะที่ได้กลับมาก็จะเป็น PENDING เหมือนกัน ซึ่งเป็นตัวบ่งบอกสถานะว่าบัตรใบนี้ไม่อนุญาติให้เติม โดยเป็นสถานะที่ได้ออกแบบไว้ตามข้อตกลง ดังนั้นผลลัพธ์ในการทดสอบก็จะเป็นดังแสดงในรูปที่ 4.39

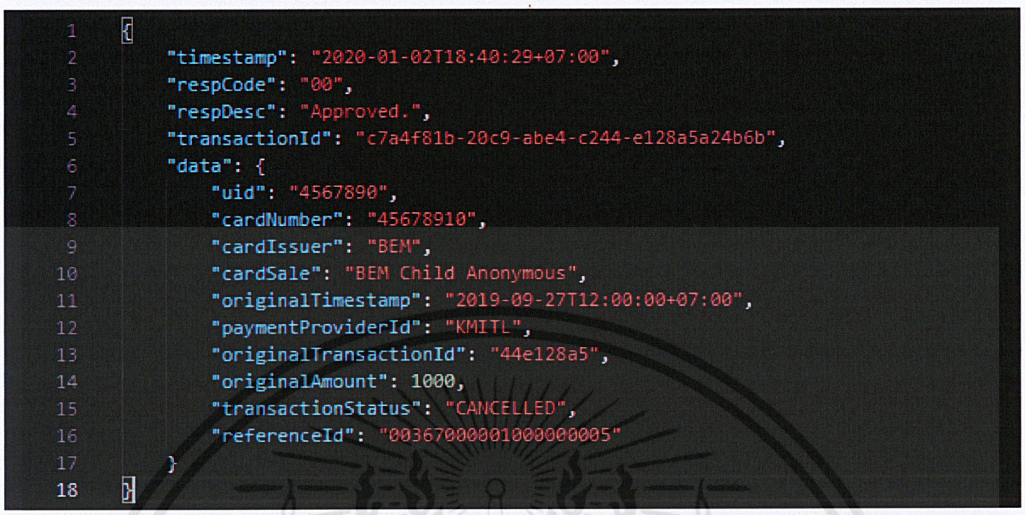

```
{
    "timestamp": "2020-01-01T20:16:47+07:00",
    "respCode": "00",
    "respDesc": "Approved.",
    "transactionId": "c7a4f81b-20c9-abe4-c244-e128a5a24b6b",
    "data": {
        "uid": "4567890",
        "cardNumber": "45678910",
        "cardIssuer": "BEM",
        "cardSale": "BEM Child Anonymous",
        "originalTimestamp": "2019-09-27T12:00:00+07:00",
        "paymentProviderId": "KMITL",
        "originalTransactionId": "44e128a5",
        "originalAmount": 1000,
        "transactionStatus": "PENDING",
        "referenceId": "00367000001000000005"
    }
}
```


รูปที่ 4.38 รายละเอียดของการทำรายการล่าสุดของบัตรหมายเลข 45678910

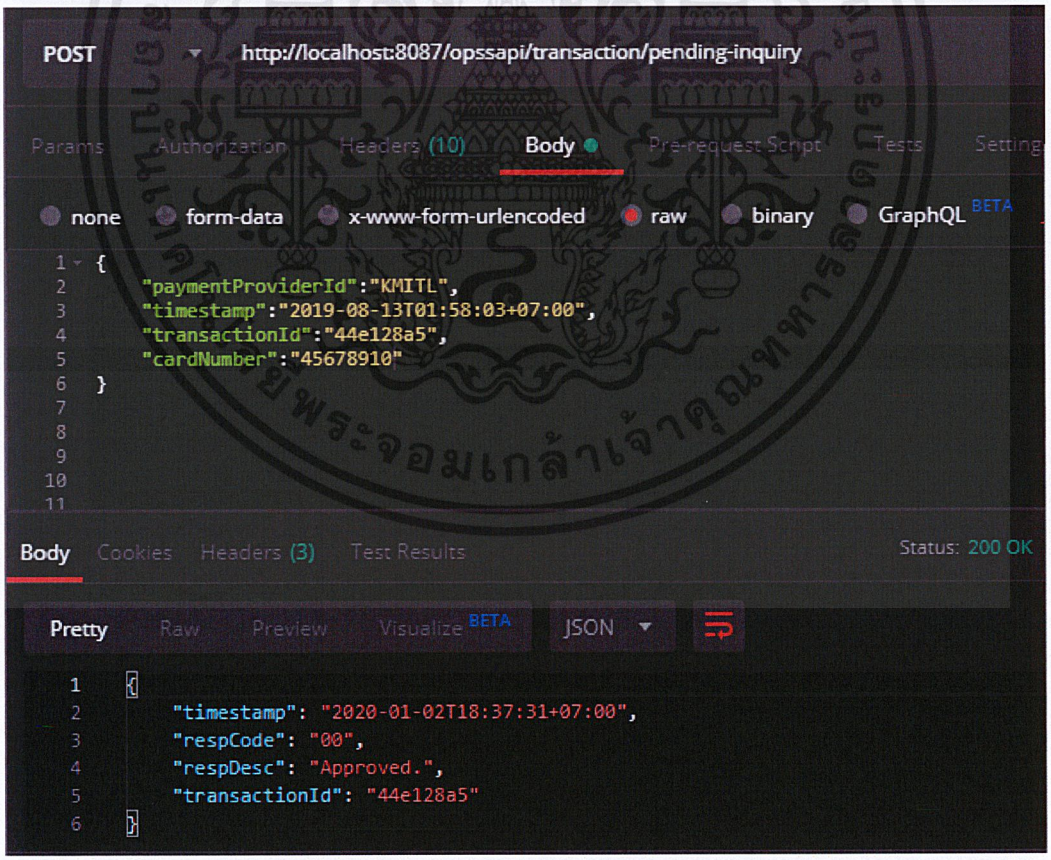


รูปที่ 4.39 ผลลัพธ์การตรวจสอบสถานะการเติมเงินของบัตร

ถ้าลองเปลี่ยนสถานะของการทำรายการล่าสุดเป็นสถานะที่อนุญาติให้บัตรนี้ทำรายการต่อได้ดังแสดงใน รูปที่ 4.40 จากนั้นลองทดสอบใหม่จะได้ผลดังรูปที่ 4.41

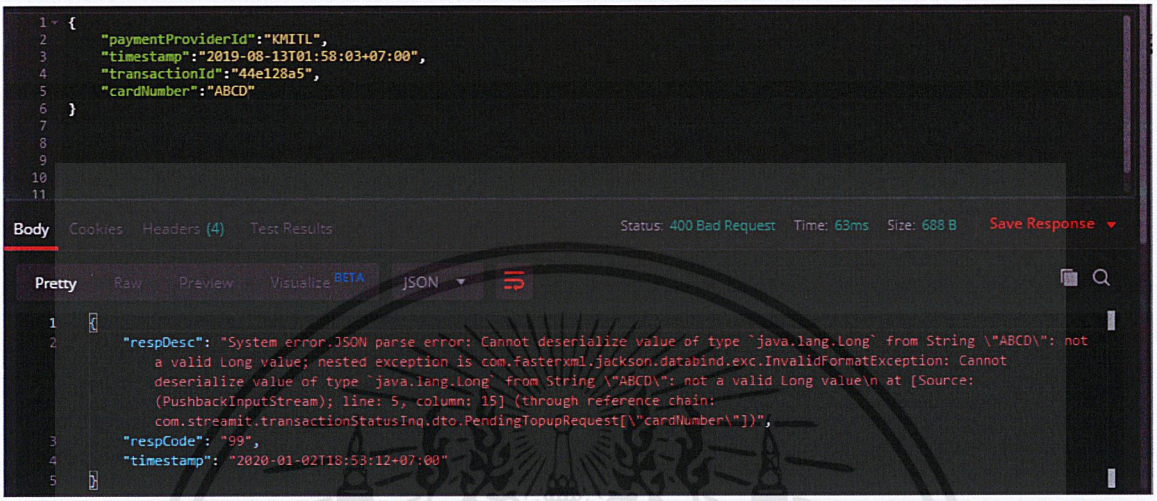


รูปที่ 4.40 รายละเอียดของการทำรายการล่าสุดของบัตรหมายเลข 45678910



รูปที่ 4.41 ผลลัพธ์การตรวจสอบสถานะการเติมเงินของบัตร

จะเห็นได้ว่าสถานะที่ตอบกลับมาจะเปลี่ยนเป็น Approve ซึ่งก็คือบัตรใบนี้สามารถทำรายการต่อไปได้ ดังนั้นการออกแบบในส่วนของเซอร์วิสได้ผลลัพธ์ที่ถูกต้องตามที่ได้ออกแบบเอาไว้ รวมถึงการส่งรูปแบบที่ผิด JSON format เข้ามายังเซอร์วิสจะได้ผลลัพธ์ตอบกลับดังแสดงในรูปที่ 4.42



รูปที่ 4.42 ผลลัพธ์ในการส่งค่า JSON โดยผิดรูปแบบที่ตกลงไว้

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการทดลอง

วิทยานิพนธ์นี้เป็นงานวิจัยที่ได้ผู้วิจัยได้มีโอกาสเข้าร่วมโครงการสหกิจศึกษากับทางบริษัท สตรีม ไอ.ที.คอนซัลติ้ง จำกัด โดยเป้าหมายของโปรเจคคือการทำระบบเกี่ยวกับการเติมเงินบัตรโดยสารออนไลน์ ซึ่งโปรเจคที่ทางบริษัทได้รับมานั้นจะเป็นการทำส่วนของทางด้าน back-end ทั้งหมด โดยการออกแบบระบบของทีมที่ผู้วิจัยอยู่นั้นได้วางแผนการออกแบบระบบในรูปแบบของไมโครเซอร์วิส เพื่อสร้างความยืดหยุ่นในการเข้าใช้งานในกรณีที่เกิดเหตุการณ์มีจำนวนของเข้าใช้การผู้เติมเงินที่มากเกินไป โดยผู้วิจัยนั้นจำเป็นที่จะต้องศึกษาการทำงานของระบบที่ได้วางแผนเอาไว้ และรูปแบบฐานข้อมูลต่าง ๆ เพื่อนำข้อมูลเหล่านั้นมาใช้พัฒนาไมโครเซอร์วิสที่ได้รับมอบหมายสำหรับในการทำวิจัย ซึ่งโปรเจคนี้นั้นเป็นการทำงานร่วมกับบริษัทอื่นอีกหลายบริษัทจึงทำให้ตัวไมโครเซอร์นั้นต้องเปลี่ยนแปลงเพื่อให้ทำงานร่วมกับทุกฝ่ายได้อย่างพึงพอใจ ดังนั้นจึงจำเป็นต้องมีการประชุมทั้งภายนอกและภายในบริษัท จึงทำให้เกิดการเปลี่ยนแปลงต่อตัวเซอร์วิสแต่ละตัวอย่างบ่อยครั้ง ทั้งนี้ทางทีมพัฒนายังได้มีการประชุมถึงระยะเวลาในการทำเซอร์วิสแต่ละตัวว่ามีระยะเวลาในการทำมากน้อยเท่าใด ซึ่งการที่รูปแบบในฐานข้อมูลเปลี่ยนไปก็ต้องปรับเปลี่ยนเซอร์วิสให้ใช้งานได้อย่างรวดเร็วอย่างสม่ำเสมอเพื่อให้ทันกำหนดการที่ทุกทีมทุกบริษัทได้วางกำหนดการไว้ ซึ่งผู้ทำงานวิจัยนั้นได้รับมอบหมายในการทำไมโครเซอร์วิสจำนวนสองตัวจากทั้งแปดตัว ซึ่งเซอร์วิสที่ได้รับมาเป็นเซอร์วิสที่คาดว่ามีผลกระทบน้อยที่สุดหากมีการเปลี่ยนแปลงใด ๆ ในอนาคต

โดยผู้ทำวิจัยนั้นจำเป็นต้องศึกษาทดสอบความรู้ในเรื่องต่าง ๆ ให้ดีซะก่อน และจากนั้นก็มาทดลองทำเซอร์วิสเพื่อทดสอบความรู้ในอีกหลากหลายรูปแบบเพื่อให้เข้าใจการทำงานของ Spring Boot Framework มากยิ่งขึ้น ซึ่งการจะทำเซอร์วิสนั้นอาจจะไม่ใช่เรื่องยากสักเท่าไหร่หากไม่ได้ใส่ใจในโครงสร้างของระบบ แต่การทำแบบนั้นก็ทำให้ระบบขาดความน่าเชื่อถือ และส่งผลเสียในภายหลังได้ ดังนั้นผู้ทำวิจัยจึงต้องศึกษารูปแบบการสร้างเซอร์วิสที่ใช้งานร่วมกับ Spring Boot และ Hibernate Framework ให้สะดวกและได้ผลดีที่สุด จึงต้องเริ่มศึกษาว่าเฟรมเวิร์กแต่ละตัวมีความสามารถในการทำอะไรบ้างอย่างละเอียด รวมถึงเลือกรูปแบบการทำงานต่อฐานข้อมูลโดยคำนึงถึงประสิทธิภาพของระบบเป็นหลัก เพื่อรองรับการใช้งานจากผู้ใช้งานจำนวนมาก จากนั้นก็ศึกษาความสัมพันธ์ระหว่างแต่ละตาราง

ในฐานข้อมูลที่เกี่ยวข้องกับทั้งสองเซอร์วิสรวมถึง data type ระหว่างฐานข้อมูลกับภาษาจาวาให้ทำงานรองรับกันให้ได้ตรงตามที่ตกลงกันไว้มากที่สุด ซึ่งเซอร์วิสที่ทำก็จำเป็นต้องเขียน unit test เพื่อรองรับการทำงานของเมทธอดทุกตัว เพื่อตรวจสอบความผิดพลาดภายในก่อนนำขึ้นโปรดักชั่น ดังนั้นการศึกษาการทำการทดสอบก็มีความสำคัญเหมือนกันเพราะรูปแบบการทดสอบนั้นมีมากมาย จึงจำเป็นต้องเลือกใช้ตัวที่เหมาะสม ซึ่งการทำทุกขั้นตอนนั้นต้องเน้นความถูกต้อง และมีประสิทธิภาพให้มากที่สุดเนื่องจากเป็นเซอร์วิสที่จะถูกนำไปใช้งานได้จริง และต้องทำรูปแบบในการตอบกลับอย่างที่ตกลงกันไว้ไม่ว่าจะเกิดข้อผิดพลาดหรือไม่ผิดพลาดก็ตาม

5.2 ปัญหาและอุปสรรคที่พบ

1. ข้อมูลในบางฟิลด์จำเป็นต้องอ้างอิงจากตารางอื่นในฐานข้อมูล แต่รูปแบบที่เก็บอยู่ในฐานข้อมูลจริง ๆ นั้นไม่ได้มีการเชื่อมความสัมพันธ์ทั้งสองตารางไว้ด้วยกัน จึงจำเป็นที่จะต้องใส่ความสัมพันธ์ที่ไม่มีจริงลงไปในโค้ด
2. การรันตัวไมโครเซอร์วิสที่มีจำนวนมากทำให้เซิร์ฟเวอร์มี RAM รองรับที่ไม่เพียงพอจึงต้องนำบางเซอร์วิสมารวมกันเพื่อให้สามารถรันบนเซิร์ฟเวอร์ของบริษัทได้

5.3 ข้อเสนอแนะและแนวทางในอนาคต

1. ควรคุยกับทุกฝ่ายให้เข้าใจอย่างรวดเร็วเพื่อลดการเปลี่ยนแปลงที่เกิดขึ้นบ่อยครั้ง
2. เนื่องจากยังมีการพัฒนาต่อหลังจากที่ผู้วิจัยได้สิ้นสุดการเข้าร่วมโครงการสหกิจศึกษากับทางบริษัท ซึ่งเป็นเซอร์วิสที่มีรากฐานคล้ายคลึงกับเซอร์วิสที่ผู้วิจัยได้เป็นคนทำเอาไว้จึงได้นำโครงสร้างเซอร์วิสดังกล่าวไปใช้พัฒนาเซอร์วิสใหม่

เอกสารอ้างอิง

[1] Thip. 2017. ทบทวน Spring Core ใน Spring Framework. [Online].

Available : https://medium.com/thipwriteblog/ทบทวน-spring-core-f1af9182b5ee.

[2] Marketeer. 2019. รถไฟฟ้ากรุงเทพ ยังวิ่งแพ้คนอื่นในเอเชีย?. [Online].

Available : https://marketeeronline.co/archives/100038.

[3] Phayao Boonon. 2019. สร้าง Microservices ด้วย Spring Boot + Spring Cloud. [Online].

Available : https://medium.com/@phayao/สร้าง-microservices-ด้วย-spring-boot-spring-cloud-b3ea6f4a8b92.

[4] Spring. 2019. Introduction to the Spring Framework. [Online].

Available :. https://docs.spring.io/spring/docs/4.3.24.RELEASE/spring-framework-reference/html/overview.html

[5] ASSANAI MANURAT. 2015. ทำความรู้จัก Spring IoC Container และ Dependency Injection. [Online].

Available : http://assanai.com/spring-ioc-container-and-dependency-injection/.

[6] Zoltan Raffai. 2018. How does spring work internally? [Online].

Available : https://www.zoltanraffai.com/blog/how-does-spring-work-internally/.

[7] Zoltan Raffai. 2018. Spring Boot: The Most Notable Features You Should Know. [Online].

Available : https://dzone.com/articles/what-is-spring-boot.

[8] Wattanachai Prakobdee. 2017. สร้าง API Gateway ด้วย Netflix Zuul และ Spring Cloud. [Online].

Available : https://developers.ascendcorp.com/สร้าง-api-gateway-ด้วย-netflix-zuul-และ-spring-cloud-e50a10fec4e5.

[9] Saladpuk. 2018. Docker ขั้นพื้นฐาน. [Online].

Available : https://saladpuk.gitbook.io/learn/basic/docker.

[10] Ake Exorcist. 2017. มาเรียนรู้ Git แบบง่ายๆกันเถอะ. [Online].

Available : https://blog.nextzy.me/มาเรียนรู้-git-แบบง่ายๆกันเถอะ-427398e62f82.

[11] Rachata Tongpagdee. 2018. Docker คืออะไร ใช้งานอย่างไร. [Online].

Available : https://medium.com/@rachatatongpagdee/docker-คืออะไร-ใช้งานอย่างไร-7e77145967b6.

ประวัติผู้เขียน

หัวข้อโครงงาน ไมโครเซอร์วิสในระบบการเติมเงินออนไลน์สำหรับบัตรรถไฟฟ้าใต้ดิน

ชื่อ - สกุล นายสุกฤษฎิ์ คุ้มครอง

รหัสนักศึกษา 59011419

คณะ วิศวกรรมศาสตร์

ภาควิชา วิศวกรรมคอมพิวเตอร์

สาขาวิชา วิศวกรรมสารสนเทศ

ประวัติส่วนตัว

วันเดือนปีเกิด 7 พฤษภาคม 2541

ที่อยู่ บ้านเลขที่ 314 ซอยอ่อนนุช 39 ถนนสุขุมวิท 77 แขวงสวนหลวง เขตสวนหลวง จังหวัดกรุงเทพมหานคร 10250

ประวัติการศึกษา

2559 - ปัจจุบัน สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (ระดับปริญญาตรี)

2553 - 2558 โรงเรียนเตรียมอุดมศึกษาพัฒนาการ (ระดับมัธยมศึกษา)

2546 - 2552 โรงเรียนกฤตศิลป์วิทยา (ระดับประถมศึกษา)