**บทที่ 3**

**ขั้นตอนการพัฒนาและออกแบบระบบ**

ในบทนี้จะกล่าวถึงขั้นตอนในการพัฒนาซอฟต์แวร์ทดสอบอัตโนมัติ โดยในการพัฒนานั้นได้ดำเนินการตามขั้นตอนที่ได้วางแผนไว้ดังต่อไปนี้

1. การศึกษาระบบ MicroData

2. การทำความเข้าใจจุดประสงค์และขอบเขตในการพัฒนา

3. การออกแบบซอฟต์แวร์

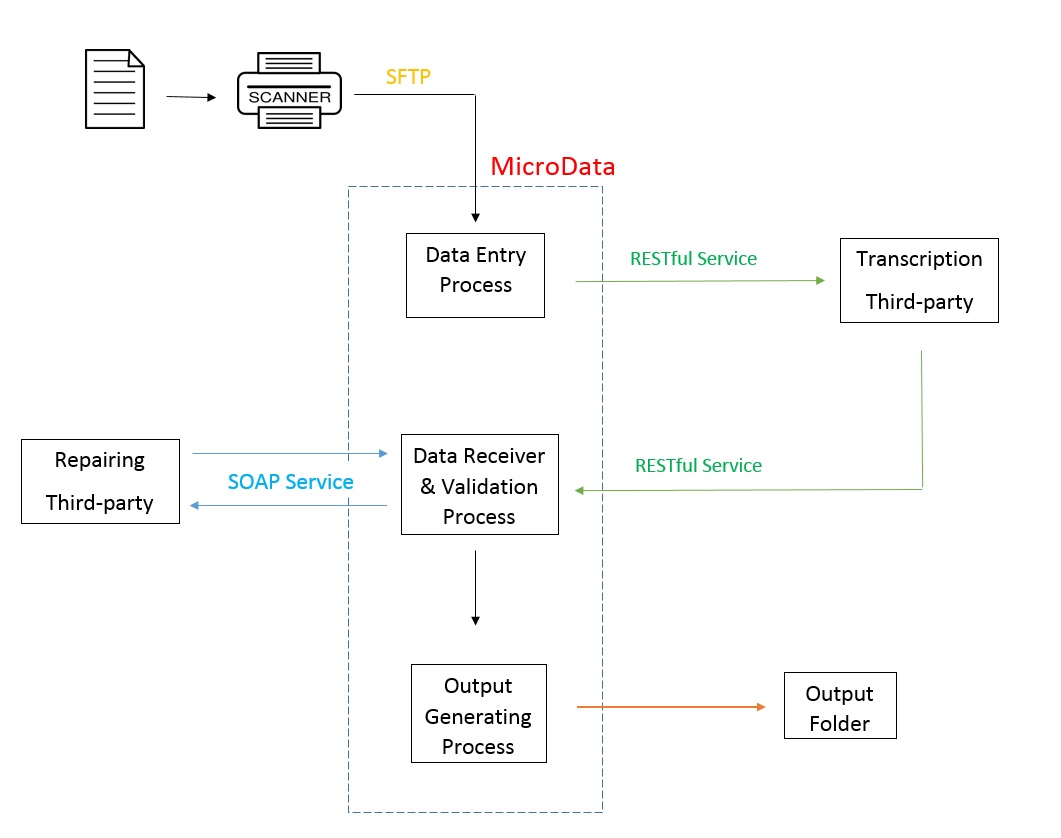
4. การพัฒนาซอฟต์แวร์

**3.1 การศึกษาระบบ MicroData**

เนื่องจากซอฟต์แวร์ทดสอบอัตโนมัติที่ทำการสร้างขึ้นนั้นมีจุดประสงค์เพื่อที่จะทดสอบความถูกต้องในการทำงานของระบบ MicroData โดยเฉพาะ ดังนั้นจึงมีความจำเป็นที่ผู้เขียนจะต้องทำความเข้าใจถึงการทำงานของระบบ MicroData ทั้งในมุมมองด้านธุรกิจ และมุมมองทางด้านเทคนิคในเชิงลึก เพื่อที่จะสามารถสร้างซอฟต์แวร์ทดสอบระบบที่ครอบคลุมกรณีปัญหาต่างๆที่เกิดขึ้นได้จริงในการทำงานอย่างครบถ้วน

โดยระบบ MicroData นั้นเป็นระบบที่ถูกสร้างขึ้นเพื่อช่วยในการดำเนินการแปลงเอกสาร แบบฟอร์มต่างๆที่อยู่ในรูปแบบของกระดาษ (Paper-based) ให้เปลี่ยนแปลงไปอยู่ในรูปของข้อมูลอิเล็กทรอนิก เพื่อที่สามารถนำไป จัดเก็บ ประมวลผล หรือดำเนินการอื่นๆ ด้วยระบบคอมพิวเตอร์ต่อไปได้ ซึ่งขั้นตอนการทำงานในการแปลงนี้ ถ้าให้มนุษย์เป็นผู้ดำเนินการ อาจทำให้เกิดข้อผิดพลาดในการทำได้ ถ้าแบบฟอร์มนั้นมีความซับซ้อนสูง หรือมีจำนวนมาก อีกทั้งยังเป็นการสิ้นเปลืองทรัพยากรมนุษย์ในการทำขั้นตอนต่างๆเหล่านี้ เนื่องจากแบบฟอร์มที่มีความซับซ้อน เช่นแบบฟอร์มทางด้านการเงิน จำเป็นต้องใช้คนทำงานใช้ความรู้ (Knowledge worker) ดังนั้นการใช้ระบบ MicroData มาทำงานในส่วนนี้แทน จึงมีประโยชน์อย่างมากต่อองค์กรที่ต้องทำงานกับเอกสารจำนวนมาก ช่วยลดต้นทุน ลดจำนวนทรัพยากรที่ต้องเสียไปในการทำการแปลงเอกสารเหล่านี้

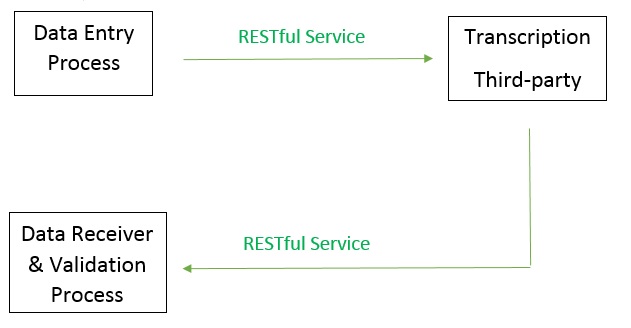
โดยวิธีการทำงานของระบบ MicroData จะแสดงด้วยแผนผังในรูปที่ 3.1



**รูปที่ 3.1** แผนผังการทำงานของระบบ MicroData

โดยจากแผนผังในรูปที่ 3.1 นั้น มีวิธีการทำงานโดย เริ่มจากการที่ลูกค้าที่ใช้ระบบทำการ แสกนเอกสารหรือแบบฟอร์มของตนเอง ที่ต้องการนำมาแปลงเป็นข้อมูล เมื่อแสกนเอกสารแล้วจะได้ไฟล์มาเป็นไฟล์รูปภาพ ซึ่งนามสกุลไฟล์นั้นขึ้นอยู่กับว่าได้ทำการตั้งค่าไว้ว่าจะรองรับไฟล์นามสกุลใดเอาไว้บ้าง แล้วจากนั้นจึงทำการอัพโหลดไฟล์ภาพที่ได้รับขึ้นมาบนระบบของ MicroData ผ่านทางโพรโตคอล SFTP เนื่องจาก ระบบของ MicroData นั้นเป็นระบบที่มีการรักษาความปลอดภัยเพื่อรักษาข้อมูลของลูกค้า หลังจากนั้นไฟล์ที่ถูกอัพโหลดขึ้นมานั้น จะถูกส่งต่อเข้าสู่โปรเซส Data entry

ซึ่งมีหน้าที่ในการตรวจสอบไฟล์ภาพ เช่น นามสกุลไฟล์ ขนาดไฟล์ต่างๆ และทำหน้าที่ในการบันทึกไฟล์นี้ลงไปในระบบของ MicroData หลังจากข้อมูลถูกบันทึกแล้วก็จะมี third party server ตัวนึงมาทำการดึงไฟล์ภาพไปทำการแปลงเอกสารเป็นข้อมูลอิเล็กทรอนิก หรือเรียกว่าการทำ data transcription หลังจากเอกสารของเราถูกแปลงเป็นข้อมูลอิเล็กทรอนิกเรียบร้อย third party server ก็จะทำการเรียกใช้ web Service ที่ระบบ MicroData เตรียมเอาไว้ โดยเป็น Web Service ประเภท RESTFul (Representational state transfer) เพื่อทำการส่งข้อมูลหลังการทำ data transcription กลับมายังระบบ MicroData โดยการส่งข้อมูลระแหว่างโปรเซสจะเห็นได้ดังรูปที่ 3.2



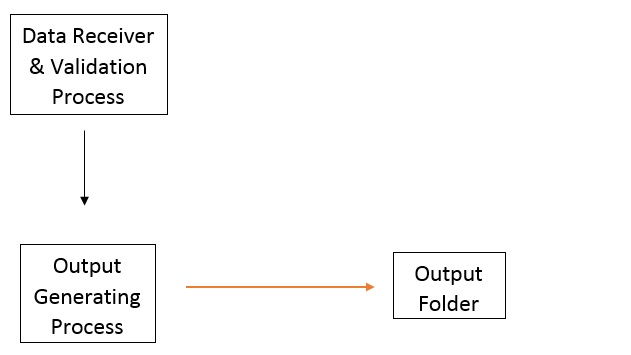
**รูปที่ 3.2** แผนผังจำลองการทำงานของ Transcription Third party

เมื่อไฟล์ข้อมูลถูกส่งกลับมาที่ server ของระบบ MicroData แล้ว ข้อมูลก็จะถูกนำเข้าสู่โปรเซสที่มีชื่อว่า data validation เพื่อทำการตรวจสอบความถูกต้องของข้อมูลที่กลับมาจากการทำ transcription โดยระบบจะมีการสร้างข้อกำหนดสำหรับข้อมูลของเอกสารเอาไว้ก่อนล่วงหน้า เช่นจำนวนฟิลด์ของข้อมูลที่ต้องได้รับจากการทำโปรเซส transcription หรือประเภทของข้อมูลที่จะต้องถูกส่งกลับมา ถ้าข้อมูลที่ส่งกลับมามีความคลาดเคลื่อน หรือไม่ตรงกับข้อกำหนด ก็จะถูกส่งข้อมูลที่ผิดพลาดไปยัง third party อีกตัวหนึ่ง ซึ่งมีหน้าที่ในการซ่อมข้อมูลที่มีข้อผิดพลาดที่ส่งมาให้ถูกต้อง โดยที่ระบบ third party ตัวนี้มีส่วนติดต่อกับโปรแกรมเป็น web service ประเภท SOAP (Simple Object Access protocol) ซึ่งเมื่อ third party ตัวนี้ทำการซ่อมข้อมูลเสร็จเรียบร้อยแล้วก็จะทำการส่งข้อมูลที่ถูกซ่อมกลับมาให้ระบบ MicroData ผ่านทาง SOAP web service ที่ระบบ MicroData เตรียมเอาไว้เพื่อทำการติดต่อกับ Repairing third party ตัวนี้เพื่อนำข้อมูลกลับไป โดยการส่งข้อมูลระหว่างโปรเซสนี้จะจำลองดังรูปที่ 3.3



**รูปที่ 3.3** แผนผังจำลองการทำงานของ Repairing Third party

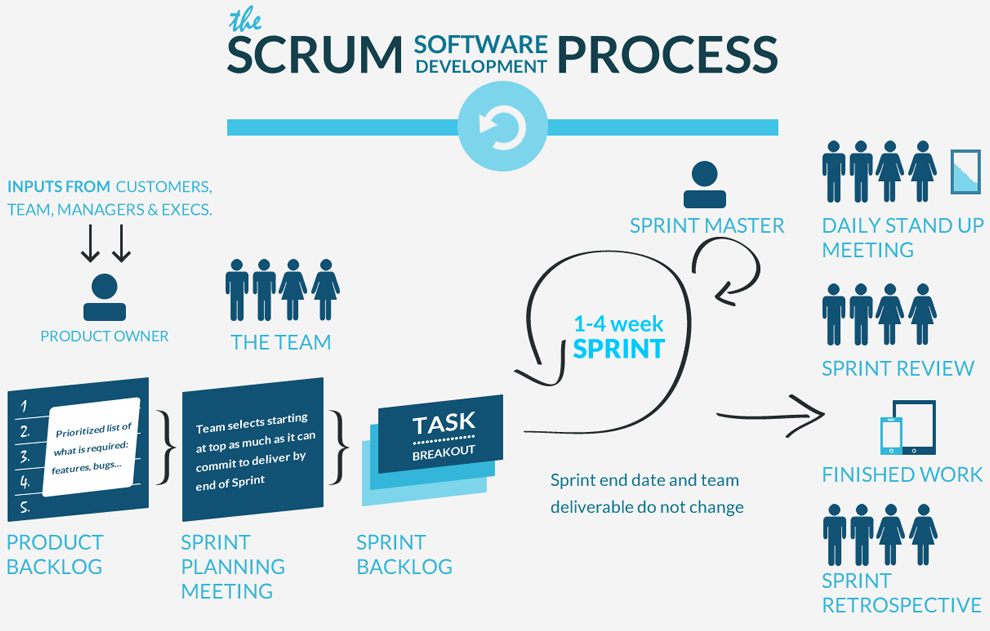
หลังจากข้อมูลถูกส่งกลับมาจากการซ่อมแล้ว ข้อมูลนั้นก็จะถูกนำเข้าสู่กระบวนการ data validation อีกรอบเพื่อตรวจสอบความถูกต้องของข้อมูลซ้ำอีกครั้ง. เมื่อข้อมูลผ่านการ validation อย่างถูกต้องแล้ว. ข้อมูลก็จะนำเข้าไปสู่โปรเซส data generator เพื่อทำการแปลงข้อมูล สร้างออกมาเป็น ไฟล์อิเล็กทรอนิกตามรูปแบบที่ลูกค้าต้องการ เพื่อให้ลูกค้านำไปใช้ประโยชน์ต่อไป โดยการทำงานในส่วนนี้จะแสดงดังรูปที่ 3.4



**รูปที่ 3.4** แผนผังจำลองการทำงานของการสร้างไฟล์ผลลัพธ์

**3.2 การทำความเข้าใจจุดประสงค์และขอบเขตในการพัฒนา**

จากหัวข้อที่ 3.1 การศึกษาโปรเซสการทำงานของระบบ MicroData ตั้งแต่ลูกค้า ทำการ อัพโหลดไฟล์ขึ้นมาจนถึงขั้นตอนที่ระบบ MicroData ทำการสร้างไฟล์ผลลัพธ์ออกมานั้น ระหว่างทางนั้นมีโปรเซสต่างๆ ทั้งการตรวจสอบความถูกต้อง การบันทึกข้อมูล การสร้างไฟล์ ย้ายไฟล์ การติดต่อกับ third party ด้วย web service และโปรเซสอื่นๆอีกมากมาย ซึ่งจะเห็นได้ว่าระบบ MicroData นั้นมีกระบวนการทำงานที่ค่อนข้างจะซับซ้อน ซึ่งในปัจจุบันการทดสอบความถูกต้องของระบบนั้นใช้การทดสอบด้วยมือหรือ manual testing ซึ่งใช้เวลามาก ในการตรวจสอบแต่ละครั้ง เนื่องจากการทดสอบจะต้องทำการ manual ในจุดต่างๆ เช่น การรับส่งข้อมูลจาก third party การสร้างไฟล์ การอัพโหลดไฟล์ และในการตรวจสอบนั้นก็จะต้องไปไล่ดูข้อมูลในจุดต่างๆเอง ซึ่งอาจเกิดการตรวจสอบได้ไม่ทั่วถึง และอีกข้อสำคัญก็คือ ระบบ MicroData นั้นถูกพัฒนาขึ้นภายใต้วงจรการพัฒนาซอฟต์แวร์แบบ Scrum framework ซึ่งมีรูปแบบการทำงานดังรูปที่ 3.5



**รูปที่ 3.5** แผนผังการทำงานของวรจรการพัฒนาซอฟต์แวร์แบบ Scrum

โดยจากรูปที่ 3.5 จะเห็นได้ว่าวงจรการพัฒนาแบบ Scrum นั้นจะมีกรอบเวลาในการพัฒนาในแต่ละรอบค่อนข้างสั้นประมาณ 1-4 อาทิตย์ และในทุกๆรอบนั้นจะต้องทำการนำซอร์สโค้ดที่พัฒนาขึ้นในแต่ละรอบ เข้าไปรวมกับซอร์สโค้ดหลักของระบบ ซึ่งก่อนที่จะนำไปรวมในทุกๆครั้งจะต้องจะต้องมีการตรวจสอบ ความถูกต้องของระบบว่า ซอร์สโค้ดใหม่ที่ทำในแต่ละรอบนั้น ทำให้ระบบมีข้อผิดพลาดเกิดขึ้นตรงไหนหรือไม่ โดยใช้การทำ manual testing ซึ่งถ้ามีการเปลี่ยนแปลงซอร์สโค้ดบ่อยๆ หรือมีกรอบเวลาในการพัฒนาที่สั้นแบบนี้ การทำ manual testing นั้นอาจจะทำได้ไม่ทัน และสิ้นเปลืองทรัพยากร และเวลาอย่างมากในการทดสอบ.

ดังนั้น โครงการพัฒนาซอฟต์แวร์ทดสอบอัตโนมัติสำหรับระบบ MicroData ด้วยการพัฒนาแบบขับเคลื่อนจากพฤติกรรมผู้ใช้ จึงถูกสร้างขึ้นมาเพื่อช่วยทดสอบความถูกต้องในการทำงานของระบบ MicroData โดยการใช้การทดสอบแบบอัตโนมัติ (automated test) ซึ่งจะช่วยให้การทดสอบ

สามารถทำได้ง่าย รวดเร็ว เพื่อตอบสนองต่อความต้องการในการทำการทดสอบซอฟต์แวร์บ่อยครั้งภายใต้กรอบเวลาในการพัฒนาที่สั้น โดยซอฟต์แวร์ตัวนี้ จะช่วยให้สามารถตรวจสอบการทำงานของระบบได้ครบถ้วน ครอบคลุม โดยเน้นไปที่กรณีต่างๆที่เกิดขึ้นเป็นปรกติ ในทุกๆครั้งที่ทำการทดสอบ โดยใช้การเขียน กรณีทดสอบ (test case) ที่จำเป็นต้องทดสอบทุกครั้งเอาไว้ล่วงหน้า ดังนั้นการทดสอบแบบอัตโนมัตินั้นจะทำการตรวจสอบทุกกรณีที่เขียนไว้ได้อย่างครบถ้วน ในทุกๆครั้งที่ทำการรันกรณีทดสอบ โดยการทดสอบอัตโนมัติจะทำการตรวจสอบความถูกต้องในการทำงานของระบบตั้งแต่เริ่มการทำงานในส่วนของการรับไฟล์ที่ลูกค้าทำการอัพโหลด ตรวจสอบในทุกส่วนของระบบ ไปจนกระทั่งไฟล์ผลลัพธ์ถูกสร้างออกมา ซึ่งขั้นตอนต่างๆ เช่น การอัพโหลดไฟล์ขึ้นไปบนระบบ การส่งข้อมูลที่ถูกแปลงแล้ว หรือการซ่อมเอกสารต่างๆ จะเป็นไปแบบอัตโนมัติทั้งหมด ทำให้ไม่จำเป็นที่จะต้องทำการอัพโหลดไฟล์ หรือทำการเรียกใช้ web service แทน third party ด้วยมือทั้งหมด. ซึ่งในส่วนของวิธีการทดสอบนั้น จะใช้การตรวจสอบไปที่ 3 ส่วนหลักๆของระบบ คือ 1.การตรวจสอบจากบันทึกการทำงานของระบบ (Processing log) 2.ข้อมูลภายในฐานข้อมูลที่ถูกบันทึกระหว่างการทำงาน และ 3.การตรวจสอบไฟล์ผลลัพธ์ที่ถูกสร้างออกมาให้กับลูกค้า โดยการพัฒนาซอฟต์แวร์ทดสอบอัตโนมัติตัวนี้ จะใช้รูปแบบการพัฒนาแบบขับเคลื่อนจากพฤติกรรมผู้ใช้ (Behavior Driven Development) ซึ่งเป็นวิธีที่นำมาใช้ช่วยให้การพัฒนาการทดสอบแบบอัตโนมัติทำได้สะดวกและง่ายดายขึ้น.

**3.3 การออกแบบซอฟต์แวร์**

ในด้านการออกแบบการพัฒนาของซอฟต์แวร์ทดสอบอัตโนมัตินั้น เราจะพูดถึงการออกแบบในส่วน ของกรณีทดสอบ (test script) ซึ่งเป็นสิ่งที่จะถูกใช้ในการรันการทดสอบแบบอัตโนมัติ โดย test script ของเรานั้นจะถูกเขียนขึ้นด้วย framework ที่เรียกว่า Cucumber โดยเราจะเรียก script ที่ถูกเขียน ขึ้นด้วย Cucumber framework ว่า “Feature file” ซึ่ง feature file นั้นจะรับผิดชอบในการทดสอบอัตโนมัติสำหรับแต่ละฟังก์ชั่นในระบบที่ต้องการทดสอบ โดยตัวอย่างของ feature file นั้นจะแสดงดังรูปที่ 3.6



**รูปที่ 3.6** ตัวอย่างของ feature file

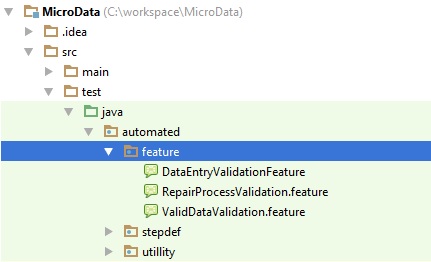
ดังนั้นในส่วนของการออกแบบของซอฟต์แวร์นั้น ทางผู้เขียนจึงจะมุ่งเน้นไปที่ การอธิบายถึงการออกแบบการเขียน feature file ว่าแต่ละ feature file ที่ถูกเขียนขึ้นนั้น ถูกสร้างขึ้นเพื่ออะไรมีจุดประสงค์ในการทดสอบอะไร ทดสอบฟังก์ชั่นไหนของระบบ และทดสอบฟังก์ชั่นนั้นอย่างไร เป็นหลัก โดยสำหรับทุกๆ feature นั้น หลักๆแล้วในการทดสอบฟังก์ชั่นต่างๆของระบบ MicroData จะมีวิธีการพื้นฐานเดียวกันก็คือ เราจะทำการจำลองการทำงานพื้นฐานของระบบเช่น การสร้างไฟล์ การอัพโหลดไฟล์สู่ระบบ และการจำลองการ รับ-ส่ง ข้อมูลกับ third party เพื่อให้เกิดการทำงานของโปรเซสต่างๆภายในระบบ ทำให้มีการสร้าง processing log และข้อมูลจริงในระบบเกิดขึ้น หลังจากนั้นจึงทำการตรวจสอบค่าที่เกิดขึ้น ว่าระบบทำงานได้ถูกต้องหรือไม่ ซึ่งทุกโปรเซสที่กล่าวมานั้นจะเป็นการทำงานแบบอัตโนมัติทั้งหมด โดยทุก feature file หรือ test script ที่เขียนขึ้นนั้นก็จะทำการดำเนินงานตามขั้นตอนข้างตั้นทั้งหมด ดังนั้นสิ่งที่จะเป็นข้อแตกต่างในแต่ละ feature file ที่เราจะทำการเขียนขึ้นนั้นก็คือ แต่ละ feature file จะทำหน้าที่เป็นตัวทดสอบสำหรับแต่ละกรณีที่เกิดขึ้นได้ในการทำงานจริง เช่น จะมี feature ที่ใช้ทดสอบว่าถ้าเราส่งอินพุตแบบต่างๆเข้ามาในระบบจะเกิดอะไรขึ้น หรือ feature ที่ใช้ในการตรวจ สอบการทำงานของการซ่อมข้อมูลที่เสียหาย เป็นต้น

ดังนั้นผู้เขียนจึงได้ออกแบบโดยการแบ่ง feature file ออกเป็น 3 ไฟล์หลักๆ คือ

1. DataEntryValidation Feature

2. ValidDataValidation Feature

3. RepairProcessValidation Feature

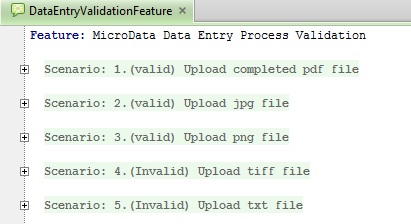


**รูปที่ 3.7** MicroData feature file

โดยรายละเอียดขั้นตอนการทำงานของแต่ละ Feature จะอธิบายดังต่อไปนี้

1. Data entry validation feature เป็น feature file ที่ใช้ในการตรวจสอบความถูกต้องใน

การทำงานของโปรเซส Data entry ในระบบ MicroData ซึ่งทำงานเกี่ยวกับการรับไฟล์ภาพที่ถูกอัพโหลดขึ้นมาบนเซิฟเวอร์ และบันทึกข้อมูลเบื้องต้น ดังนั้นจุดประสงค์ของ Data entry validation feature คือจะเป็น feature ที่รวบรวมสถานการณ์ทดสอบ (scenario) ที่เกี่ยวข้องกับการสร้างและอัพโหลดไฟล์ประเภทต่างๆขึ้นมาบนระบบเช่น อัพโหลด file นามสกุล pdf, jpg ,png เป็นต้น เพื่อตรวจสอบถึงผลลัพธ์ว่า ถ้าอัพโหลดไฟล์ที่ระบบรองรับแล้ว ตัวไฟล์ processing log ของระบบ จะต้องบันทึกข้อความที่ควรจะเกิดขึ้นถ้าระบบทำงานถูกต้อง ลงในระบบ และบันทึกข้อมูลที่เกี่ยวข้องลงในฐานข้อมูลในเบื้องต้นอย่างถูกต้อง และถ้าในส่วนขอวไฟล์ประเภทที่ระบบไม่รองรับ ไฟล์ processing log จะต้องมีการบันทึกข้อความที่แสดงถึงข้อความที่ปฎิเสธไฟล์เหล่านั้น รวมถึงข้อมูลในฐานข้อมูล จะต้องไม่มีไฟล์เหล่านั้นบันทึกลงไปด้วยเช่นกัน โดย scenario ต่างๆใน Data entry validation feature จะมีดังรูปที่ 3.8



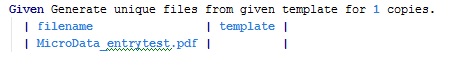
**รูปที่ 3.8** Data entry validation feature

จากรูป 3.8 จะเห็นได้ว่าแต่ละ scenario ใน feature นั้นจะทำการอัพโหลดไฟล์ที่มีนามสกุลต่างๆกัน ขึ้นไปบนระบบเพื่อทำการตรวจสอบถึงผลที่ออกมา สำหรับรายละเอียดการทำงานของแต่ละ scenario จะอธิบายจากรูปที่ 3.9



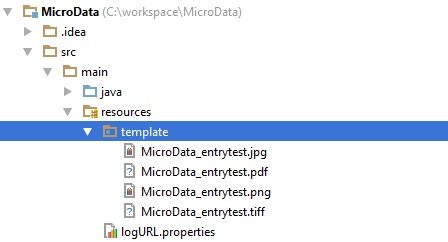
**รูปที่ 3.9** รายละเอียดของ Data entry scenario

โดย scenario ที่นำมายกตัวอย่างนี้ คือกรณีที่ทำการสร้างไฟล์นามสกุล pdf แล้วทำการ อัพโหลด ไฟล์นี้ขึ้นสู่ระบบ จากที่เห็นในรูป แต่ละบรรทัดภายใน scenario ที่มี keyword นำหน้าเป็นสีส้ม ที่เขียนว่า Given, When, Then นั้นจะถูกเรียกว่า step ซึ่งแต่ละ step นั้นจะมีการเชื่อมโยงกับการทำงานด้านหลัง ซึ่งเขียนด้วยภาษา Java เพื่อทำงานตามประโยคที่เขียน แต่ภายในหัวข้อนี้ จะอธิบาย เพียงแค่หน้าที่ของแต่ละ step ส่วนการ implementation หรือส่วนของ Java จะทำการอธิบาย ในหัวข้อต่อไป เริ่มต้นที่ step แรก ดังรูปที่ 3.10



**รูปที่ 3.10** step การสร้างไฟล์จาก template

โดย step Given หรืออินพุตของระบบ นี้มีหน้าที่ในการสร้างไฟล์ที่มีชื่อและนามสกุลของไฟล์ตามที่ระบุเอาไว้ในช่อง file name ซึ่งในที่นี้คือ MicroData\_entrytest ซึ่งมีนามสกุลเป็น pdf ส่วนในช่องที่ 2 ด้านข้างนั้นคือช่อง template ซึ่งเอาไว้ใส่ path ของไฟล์ที่เราต้องการนำมาใช้เป็น template โดยโปรแกรมจะทำการคัดลอก template ไฟล์มาใส่ให้ไฟล์ที่ถูกสร้างขึ้น ไฟล์ที่ถูกสร้างขึ้นจะมีชื่อไฟล์ตามที่ระบุในช่อง filename และมีเนื้อหาของไฟล์เป็นเนื้อหาของไฟล์ template ส่วนในกรณีที่ไม่ได้ระบุ template เอาไว้หรือปล่อยช่อง template ว่างไว้ ซอฟต์แวร์จะถือว่าเป็นการใช้ default template และจะไปหาไฟล์ที่มีชื่อเดียวกันใน folder template และคัดลอกไฟล์มาให้กับไฟล์ของเรา โดย default template จะอยู่ในโปรเจค ดังรูป 3.11



**รูปที่ 3.11** template โฟลเดอร์

โดยตัวเลขในประโยค Given ซึ่งในที่นี้คือ 1 copies หมายถึง ต้องการสร้างไฟล์จำนวนกี่ไฟล์ซึ่งเราสามารถใช้ประโยชน์จากการสร้างไฟล์หลายๆไฟล์ได้ ในกรณีที่เราต้องการจำลองการส่งไฟล์ขึ้นสู่ระบบหลายๆไฟล์ในเวลาเดียวกัน เพื่อทำ Load test หลังจากเราได้ทำการสร้างไฟล์ขึ้นมา จาก template file ด้วย step Given ในขั้นตอนก่อนหน้าสำเร็จ ขั้นตอนต่อไป คือขั้นตอนในการอัพโหลดไฟล์ทั้งหมดที่สร้างขึ้นไปบนเซิฟเวอร์ ด้วย step When ดังรูปที่ 3.12

deWhen.jpg

**รูปที่ 3.12** step การอัพโหลดไฟล์สู่เซิฟเวอร์

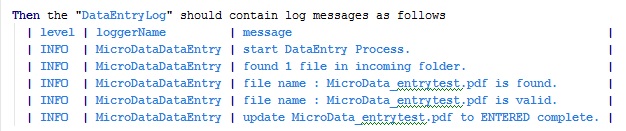
การทำงานของ Step นี้ จะเริ่มจากการทำการเชื่อมต่อกับ Secure shell server โดยเขียนโปรแกรมด้วย library Jsch ของภาษา Java เชื่อมต่อกับ server ผ่าน SFTP protocol แล้วทำการนำเอาไฟล์ทั้งหมดที่ถูกสร้างไปวางไว้ที่ folder ของระบบ MicroData โดย ระบบ MicroData จะมี กระบวนการตรวจสอบที่จะนำไฟล์เหล่านี้เข้าสู่โปรเซสต่อๆไปโดยอัตโนมัติ ดังนั้นหน้าที่ของ Step นี้ก็คือการนำไฟล์ที่ถูกสร้างไปวางไว้ใน incoming folder ของระบบเพื่อทำให้โปรเซสต่างๆ เริ่ม ทำงาน ในส่วนของการ Implementation จะอธิบายในหัวข้อต่อไป

เมื่อทำการอัพโหลดไฟล์เข้าสู่ระบบเสร็จเรียบร้อยแล้ว การทำงานของโปรเซสต่างๆในระบบ MicroData จะเริ่มต้นการทำงานขึ้นโดยอัตโนมัติ. โดยขั้นตอนต่อไปของเราก็คือการตรวจสอบ. โดยการตรวจสอบขั้นแรกของเราคือ การตรวจสอบ ไฟล์บันทึกการทำงานระบบ (Processing log) แต่เนื่องจากการทำงานในแต่ละโปรเซส อาจจะมีการใช้เวลาเล็กน้อยก่อนที่จะทำการบันทึกข้อความการดำเนินงานลงในไฟล์ ดังนั้นหากเราทำการดึงไฟล์ขึ้นมาตรวจสอบข้อความเลยทันที อาจพบว่า การทดสอบนั้นแสดงผลผิดพลาด ไม่พบกับข้อความที่เราต้องการ เนื่องจาก log ยังไม่ถูก save ลงในไฟล์ ดังนั้นเราจึงมี step ในการตั้งค่า เพื่อให้การทดสอบนั้นยืดหยุ่น โดยอนุญาติให้มีการดึงไฟล์ขึ้นมาใหม่เพื่อทดสอบซ้ำได้ในเวลาที่กำหนด ตาม step ในรูปที่ 3.13

deAnd.jpg

**รูปที่ 3.13** step ในการหน่วงเวลาการทดสอบ

โดยรูปที่ 3.13 นั้นมีวิธีการทำงานคือ เป็น step ที่ทำการตั้งค่าให้ซอฟต์แวร์ว่าจะ ทำการอ่านไฟล์ จากระบบมาทำการตรวจสอบภายในทุกกี่วินาที และถ้าครบเวลาภายในกี่วินาทีให้ทำการยกเลิกการตรวจสอบ และแสดงข้อความว่าการตรวจสอบผิดพลาด เนื่องจากเกินเวลาที่กำหนด เนื่องจากระบบ MicroData ยังไม่ทำการสร้าง processing log หรือสร้าง log ที่ไม่ถูกต้องลงในไฟล์ แต่ถ้ากระบวนการตรวจสอบข้อความที่ถูกบันทึกตรงตามที่คาดไว้สำเร็จ ระบบจะแสดงข้อความว่าตรวจสอบถูกต้อง และจะทำการข้ามไปทำการตรวจสอบขั้นตอนต่อไปทันทีโดยไม่ต้องรอให้ ครบเวลาที่กำหนด ซึ่ง step ในการตรวจสอบไฟล์บันทึกการทำงานของระบบนั้น จะเป็นดังรูปที่ 3.14



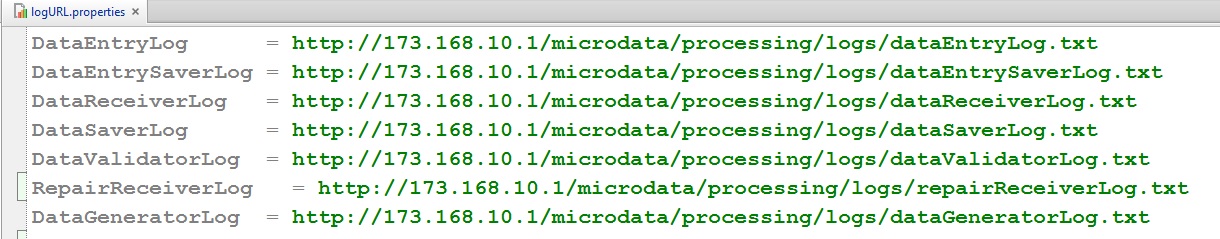
**รูปที่ 3.14** step ในการตรวจสอบ Data entry log ของ Data entry validation

โดย step นี้จะทำหน้าที่ในการตรวจสอบความถูกต้องในการทำงานของระบบ ด้วยการเปรียบเทียบระหว่าง processing log ที่ถูกสร้างขึ้นจริงระหว่างการทำงานในแต่ละโปรเซสของระบบ MicroData กับเซตของ log ที่เตรียมเอาไว้ภายในตารางที่อยู่ใน feature file โดยเซตของ log นี้จะเป็นข้อความที่ผู้ใช้ใส่กำหนดเองและใส่เข้าไปในตารางข้อมูล ซึ่งข้อความนี้คือข้อความที่ควรจะเกิดขึ้น ถ้าหากระบบทำงานได้ถูกต้องตามปกติ โดยได้มาจากการลองทำ manual testing แล้วนำเอาข้อความที่เกิดขึ้นมา หรือทำการศึกษาเอกสาร process workflow ของระบบ การทำงานของ step นี้ก็คือจะนำเอาชื่อของ log ที่เขียนไว้ใน feature ซึ่งในทีนี้คือ “DataEntryLog” ซึ่งคือ log ของโปรเซส Data entry ไปทำการหา url ที่เก็บไว้ในไฟล์ logURL.properties ดังภาพที่ 3.15



**รูปที่ 3.15** ที่เก็บไฟล์ log url

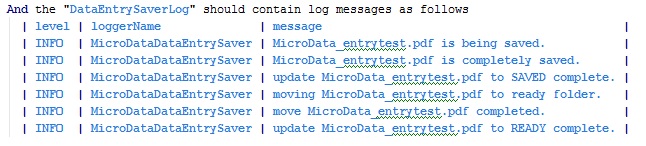
ซึ่งในไฟล์ logURL.properties นั้นจะทำการเก็บ url ที่ใช้ในการเข้าถึงไฟล์ processing log จริงในระบบ ในรูปแบบของ key value ดังรูปที่ 3.15



**รูปที่ 3.15** ไฟล์ logURL.properties

เมื่อ step ได้ url ที่ต้องการมาแล้วก็จะทำการเข้าไปอ่านไฟล์ log จาก url ที่ได้ และทำการ เปรียบเทียบกับเซตของ log ใน feature ซึ่งถ้าเซตของ log ที่เราเตรียมไว้ ผ่านการตรวจสอบสำเร็จทั้งหมดภายในเวลาที่กำหนด ก็จะมีการแสดงข้อความว่าได้ตรวจสอบ log นี้แล้ว และเริ่มทำ step ต่อไป. โดยข้อความต่างๆใน step ในรูปที่ 3.14 นั้นจะทำหน้าที่ตรวจสอบว่าโปรเซส Data entry ได้พบไฟล์ที่เราทำการ upload ขึ้นสู่ระบบ และเป็นไฟล์ที่ระบบรองรับจริง และไฟล์นั้นได้ทำการเข้าสู่สถานะ ENTERED เรียบร้อยแล้ว หาก processing log จริงที่ได้จากการอ่านแสดงข้อความว่าระบบทำงานในโปรเซสถูกต้อง ทำงานได้เป็นปกติแล้ว ตัวซอฟต์แวร์ก็จะทำการรัน step ต่อไป.

โดย step ต่อไปซึ่งคือ step ในรูปที่ 3.16 จะทำหน้าที่ในการ ไปตรวจสอบ log ของ process Data entry saver ซึ่งทำหน้าที่ในการบันทึกไฟล์ของเราลงในระบบ ว่าได้ทำการบันทึกไฟล์เรียบร้อย และทำการย้ายไฟล์ไปยัง ready folder ที่เตรียมไว้สำหรับรอ third party มาดึงไฟล์ไปทำการ Data transcription ได้สำเร็จหรือไม่.



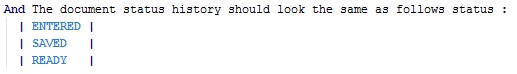
**รูปที่ 3.16** step ในการตรวจสอบ Data entry saver log

ส่วนต่อมาคือ 2 step สุดท้ายของ scenario นี้ ซึ่งเกี่ยวข้องกับการตรวจสอบข้อมูลจริงที่ถูกบันทึกลงไปในฐานข้อมูลของระบบ MicroData โดย step แรกได้แก่ step ในรูปที่ 3.17

deAndDB1.jpg

**รูปที่ 3.17** step ในการตรวจสอบสถานะ READY ของไฟล์ใน database

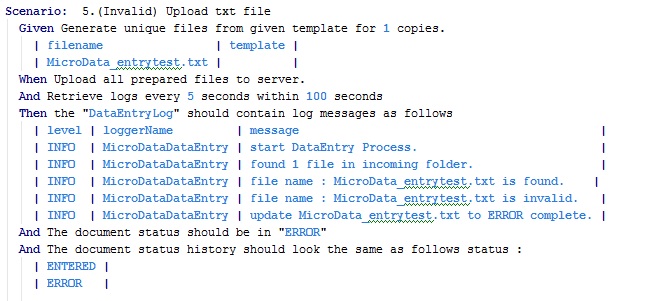
step นี้จะทำหน้าที่ในการดึงข้อมูลของไฟล์ภาพ ที่ถูกบันทึกลงไปในฐานข้อมูล ในตาราง ไฟล์ภาพ (Image Table) ขึ้นมา แล้วทำการเอาค่าในฟิลด์ “STATUS” ซึ่งเป็นฟิลด์หนึ่งในตาราง โดยฟิลด์นี้ มีไว้สำหรับเก็บ สถานะปัจจุบันของไฟล์ว่า ขณะนี้ไฟล์ของเรากำลังอยู่ในสถานะใด เพื่อทำให้ทราบว่า ตอนนี้ไฟล์ของเราอยู่ที่ไหน กำลังอยู่ในโปรเซสใด เพื่อง่ายต่อการติดตาม เมื่อได้ค่าของฟิลด์ STATUS มาแล้วก็จะนำมาเปรียบเทียบว่า ค่าของฟิลด์ STATUS ในฐานข้อมูลที่ดึงขึ้นมานั้นว่ามีค่าเดียวกันกับค่าที่เรากำหนดไว้ใน feature file หรือไม่ ซึ่งในที่นี้ค่า STATUS ที่เราทำการ กำหนด ไว้ คือ “READY” ซึ่งหมายถึง ไฟล์พร้อมที่จะถูกนำไปทำการ transcription แล้ว เพราะฉะนั้น step นี้จึงเป็นการตรวจสอบว่า ไฟล์ของเราเมื่อผ่านโปรเซส Data entry และ Data entry saver แล้วสถานะของไฟล์นั้น จะต้องมีค่าเป็น READY ถ้าหากมีค่าเป็นอย่างอื่น test ก็จะ fail และเราจะทำให้เรารู้ได้ว่า ระบบมีข้อผิดพลาดภายในโปรเซสเหล่านี้. ต่อมาคือส่วนของ step สุดท้ายดังรูปที่ 3.18



**รูปที่ 3.18** step ในการตรวจสอบประวัติสถานะของไฟล์ใน database

step นี้จะมีการทำงานที่คล้ายกับ step ก่อนหน้าในส่วนของการดึงข้อมูลใรฐานข้อมูลขึ้นมาตรวจสอบ โดยขั้นตอนแรกการทำงานของ step นี้จะเริ่มจาก ซอฟต์แวร์จะทำการดึงข้อมูลขึ้นมาจากฐานข้อมูล จากตารางชื่อ ประวัติสถานะของไฟล์ภาพ (Image status history table) ซึ่งเป็นตารางที่ทำหน้าที่ในการบันทึกว่า ไฟล์ของเราตั้งแต่เข้ามาในระบบ จนถึงเวลาปัจจุบัน เคยผ่านการ เปลี่ยนแปลงสถานะ เป็นอะไรมาบ้าง เพื่อเป็นการเก็บประวัติและแสดงถึง work flow ของไฟล์นั้น โดย step จะทำการดึงข้อมูลส่วนนี้ขึ้นมา และทำการเปรียบเทียบกับสถานะที่กำหนดไว้ใน feature ซึ่งมีจุดสำคัญคือ ลำดับของสถานะของไฟล์จะต้องเรียงลำดับกัน ตามสถานะที่เตรียมเอาไว้ใน feature file ทุกประการ เพื่อเป็นการ check ว่าระบบทำงานตาม work flow ได้ถูกต้อง หากเรียงกันผิดพลาด หรือสถานะไม่ตรงกับที่กำหนดเอาไว้ test ก็จะ fail และจะทำให้เรารู้ว่าระบบทำงานผิด work flow ที่ใดโดยดูจาก สถานะที่ไม่ตรงกับที่คาดไว้.

สำหรับ scenario ข้างต้นที่ผ่านมานั้นเป็นกรณีที่ทำการอัพโหลดไฟล์นามสกุล pdf ซึ่งระบบรองรับและใน scenario ที่ 2 และ 3 จะทำงานเช่นเดียวกันกับ scenario แรก คืออัพโหลดไฟล์นามสกุล jpg และ png ซึ่งระบบรองรับทั้งคู่ การตรวจสอบจึงมีความคล้ายคลึงกัน แตกต่างกันที่ข้อความในบางจุด ทางผู้เขียนจึงขอข้ามการอธิบายในรายละเอียดของทั้ง 2 scenarios ส่วนในกรณีที่ อัพโหลดไฟล์นามสกุลที่ระบบไม่รองรับ เช่น tiff และ txt จะถูกเขียนไว้ใน scenario ที่ 4 และ 5 ตามลำดับ. สำหรับการทำงานของการตรวจสอบในกรณีที่อัพโหลดไฟล์ที่ระบบไม่รองรับนั้น จะยกตัวอย่างด้วย scenario ที่ 5 ดังรูปที่ 3.19



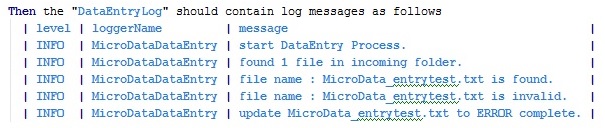
**รูปที่ 3.19** ตัวอย่าง feature กรณีไฟล์ที่ระบบไม่รองรับ

ใน scenario นี้จะเริ่มโดยการทำการสร้างไฟล์นามสกุล .txt ซึ่งระบบไม่รองรับด้วย step Given ตามรูปที่ 3.20

given5.jpg

**รูปที่ 3.20** step ในการสร้างไฟล์นามสกุล txt

หลักจากนั้นจะทำการอัพโหลดไฟล์ที่สร้างขึ้นไปบนเซิฟเวอร์ตามปกติ โดยในส่วนของการตรวจสอบ processing log จะมีความแตกต่างจากกรณีไฟล์ที่ระบบรองรับ คือ ข้อความที่ตรวจสอบ จะเป็นข้อความที่แสดงว่า ไฟล์มีข้อผิดพลาด และจะตรวจสอบเพียงแค่ processing log ของโปรเซส Data entry เท่านั้น เนื่องจากโปรเซส Data entry saver นั้นจะไม่มีการทำงานเกิดขึ้น เนื่องจากไฟล์จะถูกเปลี่ยนเป็นไฟล์ที่มีข้อผิดพลาด และเปลี่ยนสถานะเป็น ERROR และไม่ส่งต่อไปดำเนินการอื่นๆในขั้นตอนการตรวจสอบของโปรเซส Data entry. โดยข้อความที่กำหนดไว้จะแสดงดังรูปที่ 3.21



**รูปที่ 3.21** step ตรวจสอบ processing log สำหรับไฟล์ที่ระบบไม่รองรับ

หลังจากทำการตรวจสอบ processing log เสร็จเรียบร้อย step ต่อไป จะทำงานเหมือนกับ การตรวจสอบไฟล์ที่ระบบรองรับคือ การตรวจสอบสถานะของไฟล์ และ ประวัติสถานะของไฟล์ โดยใน scenario นี้จะทำการตรวจสอบสถานะของไฟล์ว่าเป็น “ERROR” หรือไม่จาก step ดังรูปที่ 3.22

And15.jpg

**รูปที่ 3.22** step ในการตรวจสอบสถานะ error

และหลังจากนั้นจะทำการตรวจสอบประวัติของสถานะ โดยเริ่มต้น เมื่อไฟล์เข้าสู่โปรเซส Data entry จะมีสถานะเป็น ENTERED และหลังจากตรวจสอบแล้วพบว่าระบบไม่รองรับไฟล์นั้น สถานะของไฟล์จะถูกเปลี่ยนเป็น ERROR และจบการทำงาน. ดังนั้นใน scenario นี้ ประวัติสถานะของไฟล์ที่ควรจะเป็น จะต้องเริ่มต้นด้วย ENTERED และจบลงด้วย ERROR เพียง 2 สถานะเรียงตามนี้เท่านั้น โดย step ในการตรวจสอบจะเป็นดังรูปที่ 3.23

and25.jpg

**รูปที่ 3.23** step ในการตรวจสอบประวัติสถานะกรณีไฟล์ไม่รองรับ

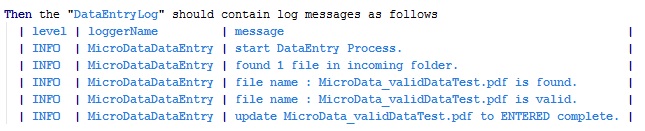
2. Valid data validation feature เป็น feature file ที่ใช้ตรวจสอบการทำงานโดยรวม ของระบบ MicroData ซึ่งต่างจาก feature Data entry validation ตรงที่ จุดประสงค์ของ feature Data entry validation นั้นมีไว้เพื่อตรวจสอบเฉพาะโปรเซสส่วนต้นของ MicroData ซึ่งคือ Data entry และ Data entry saver เท่านั้น แต่ Valid data validation feature จะทำการตรวจสอบตั้งแต่โปรเซส Data entry ผ่านโปรเซส validation จนถึง Output generator แล้วรวมไปถึงการตรวจสอบไฟล์ output ที่ถูกสร้างในออกมา. จึงสามารถเรียกได้ว่า feature file นี้เป็นการตรวจสอบตลอดทุกโปรเซสในระบบ หรือ end to end testing. โดยวิธีการทำงานแต่ละขั้นของ feature นี้ จะอธิบายดังต่อไปนี้.

ขั้นตอนแรกของ Valid data validation feature คือการสร้างไฟล์เพื่อใช้ในการส่งขึ้นไปบนเซิฟเวอร์ เหมือนกับการทดสอบใน feature ก่อนหน้า เพียงแต่ใน feature นี้เราจะสร้างเฉพาะไฟล์ นามสกุลที่ระบบรองรับเท่านั้น เพื่อให้ระบบสามารถทำงานในโปรเซสต่อๆไปได้ ในที่นี้จะใช้ ไฟล์ pdf

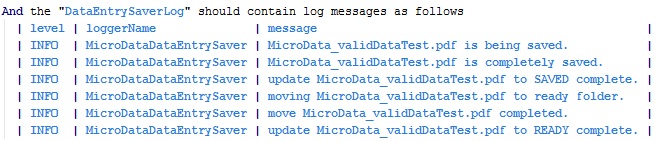
given.jpg

**รูปที่ 3.24** Step ในการสร้างไฟล์ pdf สำหรับ Valid data feature

จากรูปที่ 3.24 เป็นการสร้างไฟล์ชื่อ MicroData\_validDataTest.pdf ด้วย default template สำหรับใช้ในการทดสอบใน scenario นี้ หลังจากสร้างไฟล์แล้วก็จะทำการอัพโหลดไฟล์สู่ระบบ ด้วย step ในการอัพโหลดไฟล์สู่ระบบ step เดียวกับรูปที่ 3.12 หลังจากนั้นก็ทำการตรวจสอบ processing log ของทั้ง 2 โปรเซสเดิมคือ Data entry และ Data entry saver ว่ามี processing log ของไฟล์ MicroData\_validDataTest.pdf ที่แสดงข้อความว่าไฟล์บันทึกถูกต้อง ดังรูปที่ 3.25 และ 3.26



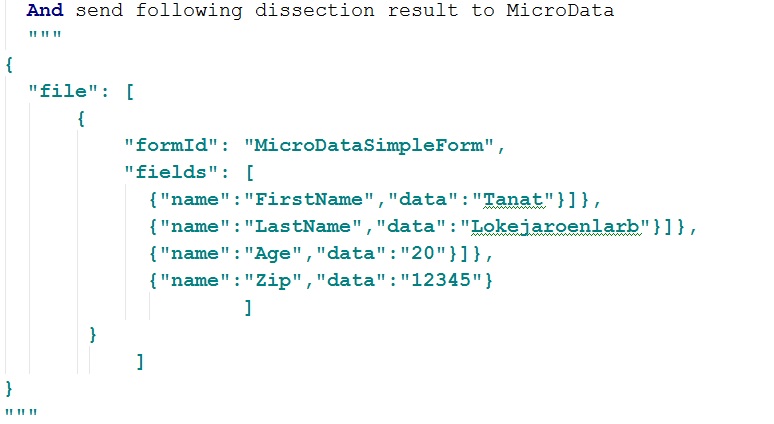
**รูปที่ 3.25** step ตรวจสอบ Data entry log ของ Valid data feature



**รูปที่ 3.26** step ตรวจสอบ Data entry saver log ของ Valid data feature

หลังจากตรวจสอบ processing log ของโปรเซส Data entry เสร็จเรียบร้อยจะทำให้เรารู้ได้ว่า ขณะนี้ไฟล์ของเราอยู่สถานะ READY ซึ่งคือสถานะที่พร้อมในการรอให้ third party มาดึงไฟล์ไปทำโปรเซส data transcription ซึ่งในการทำงานในระบบจริงนั้นจะต้องรอให้มีคนทำ โปรเซส data transcription แล้วทำการส่งข้อมูลที่เกิดหลังจาก การแปลงไฟล์ภาพเสร็จเรียบร้อยแล้วกลับมาให้ ระบบ MicroData ผ่าน RESTful web service แต่ว่า เนื่องจากซอฟต์แวร์ของเราเป็นซอฟต์แวร์ทดสอบอัตโนมัติ ทำให้เราจะไม่รอโปรเซสเหล่านี้ถูกทำด้วยมือ โดยเราจะทำการจำลอง ให้ตัวซอฟต์แวร์ของเราเองนั้น ทำหน้าที่ในการส่ง ข้อมูลที่ได้จากการแปลงไฟล์ (transcribed data) กลับมาที่ระบบ เนื่องจากไฟล์ที่ใช้ในการทดสอบเป็นไฟล์ของเราเองดังนั้น เราจึงสามารถที่จะรู้ได้ว่า ข้อมูลที่ควรจะได้รับกลับมาจากโปรเซส data transcription จะมีหน้าตาอย่างไร มีค่าเป็นอย่างไรบ้าง ซึ่งจะช่วยประหยัดเวลาในการทดสอบ ทำให้ไม่ต้องรอการทำ transcription จริง ซึ่งใช้เวลามากในการทำ. โดยข้อมูลที่จะทำการส่งกลับมานั้น ในปกติแล้วจะส่งกลับมาอยู่ในรูปของ JSON (JavaScript Object Notation). ดังนั้น step ที่เราจะใช้ในการจำลองการส่งข้อมูลนั้นจะเป็น

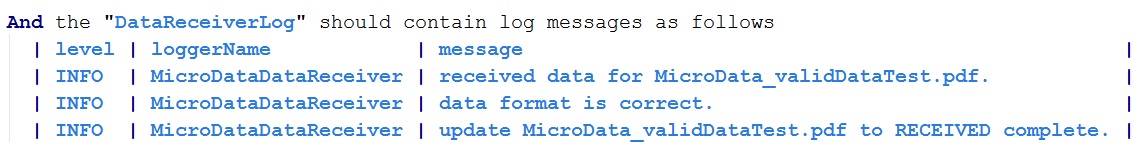
ดังรูปที่ 3.27



**รูปที่ 3.27** step ในการจำลองการส่งข้อมูล valid data

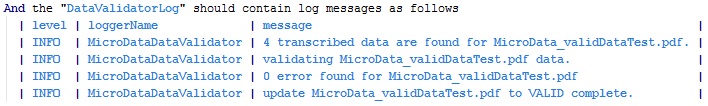
สำหรับการทำงานของ step นี้ มีหน้าที่ในการนำเอาข้อมูลในรูปแบบ JSON ที่ใส่เอาไว้ที่ด้านล่างของ step แล้วทำการส่งข้อมูลนี้ผ่าน RESTful web service ของระบบ MicroData ไปที่โปรเซส Data receiver. เมื่อทำการส่งข้อมูลของเราไปที่ MicroData แล้วขั้นตอนต่อไปก็คือขั้นตอน

การตรวจสอบ processing log ของโปรเซสต่อไป ซึ่งก็คือ Data receiver. โดยเราจะตรวจสอบว่า Data receiver ได้รับข้อมูลที่ส่งกลับมาหรือไม่ โดยข้อความที่ใช้ในการตรวจสอบ เป็นไปดังรูปที่ 3.28



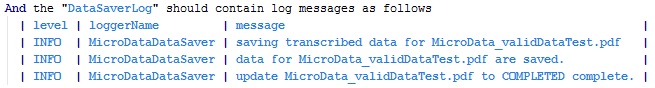
**รูปที่ 3.28** step ตรวจสอบ Data receiver log ของ Valid data feature

โดยหลักๆแล้วการตรวจสอบจะดูว่า ข้อมูลนั้นถูกส่งเข้ามาถึงระบบแล้วหรือไม่ และรูปแบบของข้อมูล JSON นั้นถูกฟอร์แมทหรือไม่ และในขั้นสุดท้ายเราจะตรวจสอบว่าสถานะของไฟล์เปลี่ยนเป็น RECEIVED หรือไม่หลังจากได้รับข้อมูลแล้ว. เมื่อผ่านการตรวจสอบ processing log ของโปรเซส Data receiver แล้วแสดงว่าข้อมูลของเราได้ผ่านการตรวจสอบโดยเบื้องต้นแล้ว ข้อมูลก็จะถูกส่งต่อไปที่โปรเซส Data validation เพื่อทำการตรวจสอบความถูกต้องของเนื้อหาของข้อมูล โดย step ในการตรวจสอบ processing log ของ โปรเซส Data validation จะเป็นไปดังรูปที่ 3.29



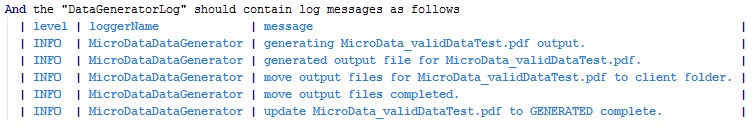
**3.29** step การตรวจสอบ Data validation log ของ Valid data feature

โดยการตรวจสอบการทำงานของโปรเซส Data validation นั้นเราจะทำการตรวจสอบข้อความว่า Data validation ได้ข้อมูลไปเท่ากับจำนวนข้อมูลที่เราส่งมาด้วย JSON หรือไม่ หลังจากนั้นจะทำการตรวจสอบว่า มีข้อผิดพลาดเกิดขึ้นหรือไม่. ซึ่งข้อผิดพลาดนี้จะเกิดขึ้นได้จากการที่ JSON ไฟล์ที่เราส่งกลับมานั้น ขัดกับข้อจำกัดที่มีการตั้งค่าไว้ โดยข้อกำหนดเหล่านี้ จะต้องทำการตั้งค่าด้วยมือเองไว้ก่อนเช่น การตั้งข้อกำหนดไว้ว่า ข้อมูลที่จะได้รับกลับมานั้นจะต้องมีฟิลด์กี่ฟิลด์ และชื่อฟิลด์มีอะไรบ้าง ซึ่งถ้าข้อมูลทีได้รับกลับมาของไฟล์ที่เราได้ตั้งข้อกำหนดนั้นไม่เป็นไปตามที่ตั้งไว้ ในขั้นตอนการตรวจสอบจะตรวจพบว่าข้อมูลไม่ถูกต้อง และทำการเปลี่ยนสถานะของไฟล์ว่าต้องการการซ่อมแซม แล้วทำการส่งข้อมูลไปซ่อมโดย Third party ในอีกโปรเซสหนึ่ง ซึ่งในที่นี้ผู้เขียนได้ทำการตั้งค่าข้อกำหนดเบื้องต้นไว้ว่า JSON ที่ส่งมาจะต้องมี 4 ฟิลด์ และแต่ละฟิลด์นั้นมีชื่อว่าอะไรบ้าง เพราะฉะนั้น ในการสร้าง JSON ที่เราได้ทำการส่งมาใน step ก่อนหน้านั้น เนื่องจากเราเป็นคนกำหนดข้อกำหนดเองดังนั้น JSON นั้นก็จะถูกสร้างขึ้นอย่างเจตนาโดยที่ไม่ทำให้เกิดข้อผิดพลาดใดๆ ซึ่งข้อความที่ควรจะเกิดขึ้นใน processing log เมื่อได้รับข้อมูล JSON นั้นก็คือ “0 error found” และ “4 transcribed data are found” สำหรับข้อมูลของเรา และในขั้นตอนสุดท้ายคือการตรวจสอบว่า มีข้อความที่แสดงว่า ไฟล์ของเราเปลี่ยนสถานะเป็น VALID เรียบร้อย. เมื่อข้อมูลของเราผ่านโปรเซสการตรวจสอบเรียบร้อยแล้ว โปรเซสต่อไป ก็คือการบันทึกข้อมูล transcribed data สำหรับไฟล์ของเรา ลงไปในฐานข้อมูลจริงของระบบ MicroData ด้วยโปรเซส Data saver โดย step ในการตรวจสอบโปรเซส Data saver มีลักษณะดังรูปที่ 3.30



**รูปที่ 3.30** step ตรวจสอบ Data saver log ของ Valid data feature

โดยการตรวจสอบข้อความของ Data saver log นั้นจะเน้นไปที่การตรวจสอบว่า มีข้อความที่แสดงถึงการบันทึกข้อมูลลงในระบบ และการบันทึกนั้นสำเร็จ ถูกต้องหรือไม่ และหลังจากนั้นก็จะเป็นการตรวจสอบในขั้นสุดท้ายว่า สถานะของไฟล์เราได้เปลี่ยนไปเป็น COMPLETED ซึ่งหมายถึงว่าไฟล์ของเราได้รับข้อมูลที่ผ่านการ transcription เรียบร้อยแล้ว. และหลังจากนั้น เมื่อระบบทำการบันทึกข้อมูลที่ผ่านการตรวจสอบแล้วเสร็จเรียบร้อย ข้อมูลของเราก็จะถูกนำเข้าสู่โปรเซสตัวสุดท้ายของ ระบบ MicroData คือโปรเซสที่เรียกว่า Data generator ซึ่งเป็นโปรเซสที่ทำหน้าที่ในการสร้างไฟล์ผลลัพธ์ออกมาให้กับลูกค้า โดยอาศัยข้อมูล transcribed data ที่ได้รับและผ่านการตรวจสอบแล้วมา. โดย step ที่ใช้ในการตรวจสอบโปรเซสนี้มีข้อความต่างๆดังรูปที่ 3.31



**รูปที่ 3.31** step ตรวจสอบ Data generator log ของ Valid data feature

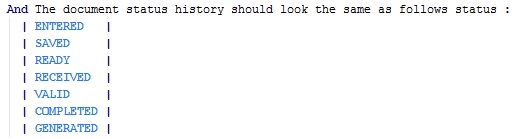
โดยข้อความหลักๆของโปรเซส Data generator ที่เราจะทำการตรวจสอบนั้น จะเริ่มจาก ข้อความที่แสดงว่า โปรเซส Data generator กำลังสร้างไฟล์ผลลัพธ์ออกมาสำหรับไฟล์ของเรา และข้อความอีกส่วนที่จะตรวจสอบคือ ข้อความที่แสดงว่าการสร้างไฟล์ผลลัพธ์จากข้อมูลนั้นสำเร็จ และ ไฟล์ผลลัพธ์ของเราที่ถูกสร้างขึ้นแล้วได้ถูกย้ายไปยังโฟลเดอร์ของลูกค้าสำเร็จแล้วหรือไม่ ส่วนในขั้นตอนสุดท้ายก็คือ เราจะทำการตรวจสอบว่า มีการเปลี่ยนสถานะของไฟล์เป็น GENERATED ซึ่งเป็นสถานะสุดท้ายของระบบแล้วหรือไม่ โดยสถานะ GENRATED จะเป็นตัวบ่งบอกว่า ไฟล์ของเราได้ถูกสร้างไฟล์ผลลัพธ์และ ส่งไปที่โฟลเดอร์ของลูกค้าสำเร็จเรียบร้อยแล้ว.

หลังจากตรวจสอบ processing log ทั้งหมดของทุกโปรเซสเป็นที่เรียบร้อยแล้ว เพื่อเป็นการ เพิ่มความแม่นยำในการตรวจสอบ ขั้นตอนต่อไปของ scenario นี้คือการตรวจสอบข้อมูลจริงในฐาน-ข้อมูล ซึ่งจะมีลักษณะการทำงานเหมือนกันกับ feature Data entry validation ทุกประการ ส่วนที่มีความแตกต่างกันคือ ค่าของข้อมูลที่ตรวจสอบนั้นจะต้องสอดคล้องกับข้อมูลใน scenario นี้ โดยขั้นตอนการตรวจสอบฐานข้อมูลขั้นตอนแรก คือ การตรวจสอบว่า สถานะของไฟล์ ซึ่งผ่านโปรเซสทุกอย่างมาแล้วจนเสร็จสิ้น จะต้องมีสถานะเป็น GENERATED โดย step จะมีลักษณะดังรูปที่ 3.32

statusGen.jpg

**รูปที่ 3.32** step ในการตรวจสอบสถานะ GENERATED ของไฟล์ใน database

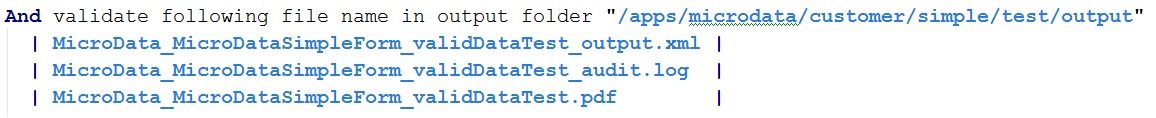
และขั้นตอนต่อไปในการตรวจสอบข้อมูลในฐานข้อมูลก็คือ การตรวจสอบประวัติของสถานะที่เคยเกิดขึ้นกับไฟล์ของเรา โดยสถานะทั้งหมดที่ควรจะต้องเกิดขึ้นกับไฟล์ใน scenario นี้ ควรจะต้องเกิดขึ้น ครบถ้วนและ เรียงตามลำดับ ดังรูปที่ 3.33



**รูปที่ 3.33** step ในการตรวจสอบประวัติสถานะของ Valid data feature

จากรูปคือสถานะทั้งหมดที่ควรจะเกิดขึ้นถ้าการทำงานของระบบ MicroData ใน scenario นี้ทำงานได้ถูกต้อง ทุกโปรเซสที่ควรจะต้องมีการทำงาน ทำงานเรียงกันอย่างถูกต้อง ดังต่อไปนี้ ENTERED -> SAVED -> READY -> RECEIVED -> VALID -> COMPLETED -> GENERATED

และสำหรับขั้นตอนสุดท้ายในการทดสอบระบบของ Valid data validation feature คือขั้นตอนการตรวจสอบถึงไฟล์ผลลัพธ์จริงๆในโฟลเดอร์ ที่มีไว้สำหรับเก็บไฟล์ผลลัพธ์ของลูกค้า ว่าภายในโฟลเดอร์นั้น มีไฟล์ผลลัพธ์ ที่เกิดขึ้นจากข้อมูลทดสอบของเราหรือไม่ โดย step ที่ใช้ในการตรวจสอบไฟล์นั้น จะมีลักษณะดังรูปที่ 3.33

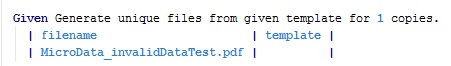


**รูปที่ 3.33** step ในการตรวจไฟล์ผลลัพธ์ของ valid data feature

โดยการทำงานหลักๆของ step นี้ จะเริ่มจากทำการเชื่อมต่อกับเซิฟเวอร์ของระบบผ่านทาง SFTP protocol โดยเมื่อทำการเชื่อมต่อสำเร็จแล้ว ก็จะทำการเปลี่ยนตำแหน่ง path ไปที่ path ที่กำหนดไว้ใน step ของเรา ซึ่งในที่นี้คือ “/apps/microdata/customer/simple/test/output” ซึ่งเป็นตำแหน่งไฟล์ path ที่เอาไว้เก็บไฟล์ผลลัพธ์ของไฟล์ของเรา. ซึ่งการที่ต้องมาใส่ path ของไฟล์ ใน step เนื่องจากว่า ไฟล์ผลลัพธ์ของลูกค้าแต่ละคนจะ ถูกตั้งค่าตำแหน่งโฟลเดอร์ สำหรับเก็บไฟล์ผลลัพธ์เอาไว้ต่างที่กัน ดังนั้นไฟล์ผลลัพธ์ที่ถูกสร้างขึ้นจะไปอยู่ในตำแหน่งของโฟลเดอร์เฉพาะ ซึ่งเราจะกำหนดตำแหน่งนั้นผ่านทางตัวแปรใน step นี้. เมื่อเราเปลี่ยน path ไปอยู่ที่ตำแหน่งโฟลเดอร์ที่เก็บไฟล์ผลลัพธ์แล้ว step ก็จะทำการอ่านไฟล์ทั้งหมดในโฟลเดอร์ เพื่อนำมาเปรียบเทียบกับชื่อไฟล์ที่เรากำหนดเอาไว้ที่ด้านล่างภายใน step ของเรา ว่ามีไฟล์ผลลัพธ์ที่เราต้องการ อยู่ในโฟลเดอร์นั้นหรือไม่ ถ้าไม่มีแปลว่า ขั้นตอนการสร้างไฟล์ผลลัพธ์มีข้อผิดพลาด โดยชื่อไฟล์ที่ถูกสร้างจะเกิดขึ้นจากการนำเอาชื่อแบบฟอร์มรวมกับชื่อไฟล์ที่อัพโหลดขึ้นไป. เมื่อผ่านการตรวจสอบทั้งหมดทุกขั้นตอนข้างต้น จะช่วยยืนยันได้ว่า ขั้นตอนการทำงานโดย รวมของระบบ MicroData นั้นทำงานได้ถูกต้อง ในกรณีที่ ข้อมูลที่อัพโหลดรองรับโดยระบบ และข้อมูลที่ส่งกลับมาถูกต้อง ไม่ขัดกับข้อกำหนดที่ตั้งไว้.

3. RepairProcessValidation feature เป็น feature สุดท้ายของการทดสอบ โดย feature นี้เป็น feature ที่สร้างขึ้นมาเพื่อใช้ในการทำงานของระบบ MicroData ในกรณีที่มีการทำงานเกี่ยวกับการซ่อมข้อมูลที่ไม่ถูกต้อง เนื่องจากข้อมูลมีความเป็นไปได้ที่จะเกิดการผิดข้อกำหนดในขั้นตอนการตรวจสอบข้อมูลที่โปรเซส Data validation โดย feature นี้จะมีหน้าที่ในการตรวจสอบว่า เมื่อมีการส่งข้อมูลที่ผิดข้อกำหนดมา โปรเซสในการตรวจสอบจะต้องตรวจพบได้ว่า ข้อมูลที่ส่งมามีข้อผิดพลาด และจะต้อง ส่งข้อมูลที่ผิดพลาดเหล่านั้นไปยังตัว Repairing third party ซึ่งทำหน้าที่ ในการแก้ไขข้อมูล เหล่านั้นให้ถูกต้อง และหลังจากนั้นก็จะทำการตรวจสอบว่า เมื่อเสร็จสิ้นการซ่อมข้อมูลเรียบร้อยแล้ว ข้อมูลที่กลับมาจากการซ่อมจะต้องผ่านการตรวจสอบอย่างถูกต้อง และระบบ MicroData สามารถ สร้างไฟล์ผลลัพธ์ออกมาได้ในขั้นตอนสุดท้าย. โดยใน feature นี้จะแบ่งออกเป็น 2 scenario แต่ละ scenario จะมีข้อแตกต่างกันเล็กน้อยในส่วนของ ข้อมูล JSON ที่สร้างขึ้นโดยตั้งใจ เพื่อเป็นการทำการทดสอบด้วยข้อมูลที่แตกต่าง สำหรับ scenario แรกจะอธิบายดังต่อไปนี้

โดย scenario แรกนั้น เราจะจำลอง ทำการส่งข้อมูล JSON ที่มีส่วนที่ขัดกับข้อกำหนดคือ เราจะสร้าง ให้ข้อมูลนั้นมีเพียงแค่ 2 ฟิลด์ ดังนั้นจึงมีฟิลด์ 2 ฟิลด์ที่หายไป จากข้อกำหนดที่เราตั้งไว้คือ 4 ฟิลด์ และเราจะทดสอบว่า processing log ของ การ validation ตรวจพบข้อผิดพลาดนี้หรือไม่ และหลังจากนั้นเราจะทำการ ซ่อมข้อมูลนั้น โดยการจำลองการทำงานเป็น third party ในการทำการซ่อมข้อมูล ตรวจสอบว่ากรซ่อมสำเร็จหรือไม่ และสุดท้ายก็จะ ตรวจสอบความถูกต้องการทำงานในส่วนต่างๆ ตามที่เคยได้ทำมา และตรวจสอบว่าไฟล์ผลลัพธ์สามารถถูกสร้างขึ้นได้ ในกรณีที่ข้อมูลเคยเกิดการผิดข้อกำหนดหรือไม่. โดย step แรกของ scenario นี้ก็ จะเริ่มต้นเหมือนกับ scenario อื่นๆนั่นคือ จะทำการสร้างไฟล์เพื่อใช้ในการทดสอบ โดย step ดังรูปที่ 3.34

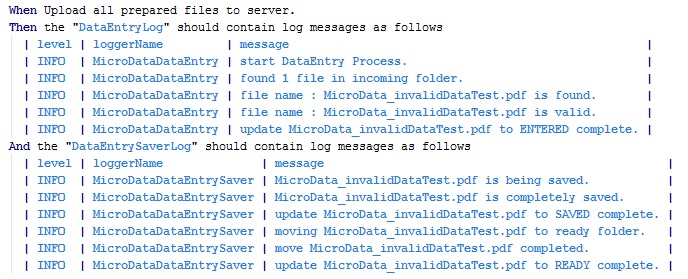


**รูปที่ 3.34** step ในการสร้างไฟล์ pdf สำหรับ Repair feature

โดยไฟล์ที่สร้าง จะสร้างจาก default template และมีชื่อไฟล์ว่า MicroData\_invalidData

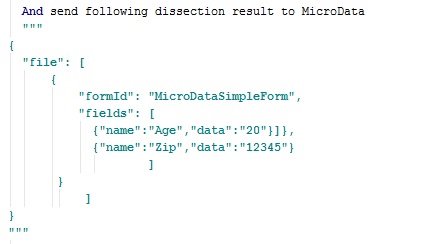
Test.pdf. หลังจากสร้างไฟล์ที่ต้องการเสร็จเรียบร้อยแล้ว step ต่อไปจากนี้คือการอัพโหลดไฟล์และ

ตรวจสอบ processing log ของโปรเซส Data entry และโปรเซส Data entry saver แบบเดียวกันกับ step ใน feature valid data เนื่องจากว่าไฟล์เป็นไฟล์ที่ระบบรองรับ โดยจะแสดง step ทั้งสองในรูปที่ 3.35



**รูปที่ 3.35** step อัพโหลดและตรวจสอบขั้นต้นของ Repair feature

เมื่อจบขั้นตอนข้างต้นแล้ว ขั้นตอนต่อไปของ scenario คือการจำลองการทำงานของ third party ในการส่งข้อมูล transcribed data กลับมายังระบบ MicroData ซึ่งข้อมูลในรูปแบบของ JSON ที่เราจะส่งกลับมานั้น เราจะจงใจสร้างให้ข้อมูลมีฟิลด์หายไป 2 ฟิลด์ จากข้อกำหนดที่เราตั้งไว้ว่า ข้อมูลที่กลับมาจะต้องมีฟิลด์ 4 ฟิลด์ โดยในกรณีนี้ฟิลด์ที่เราจะไม่ส่งกลับมาให้นั้นคือ ฟิลด์ FirstName และฟิลด์ LastName โดย step ในการส่งข้อมูลกลับมา และหน้าตาของ ข้อมูลที่อยู่ในรูป JSON ที่เราจะส่งกลับมานั้น จะแสดง ดังรูปที่ 3.36



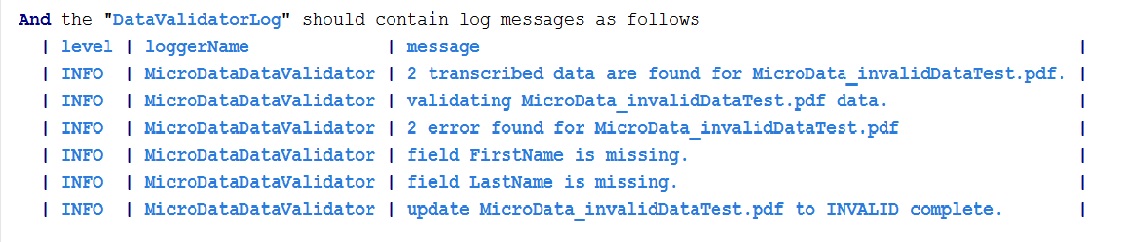
**รูปที่ 3.36** ข้อมูล JSON ที่ฟิลด์หายไป 2 ฟิลด์

หลังจากส่งข้อมูลที่มีข้อผิดพลาดกลับมาแล้ว step ต่อไปคือการตรวจสอบ processing log ของโปรเซส Data receiver ซึ่งในกรณีนี้ถึงข้อมูลที่เราส่งกลับมาจะเป็นข้อมูลที่จงใจให้ผิดพลาด แต่ในโปรเซส Data receiver นั้นจะมีหน้าที่ในการตรวจสอบถึงรูปแบบและฟอร์แมทของข้อมูล JSON เท่านั้นว่าเป็น ไฟล์ JSON ที่ถูกหรือผิดฟอร์แมท ไม่ใช่การตรวจสอบถึงการผิดข้อกำหนดของระบบเอง เพราะฉะนั้นโปรเซส Data receiver จะทำงานได้ตามปกติ ไม่มีข้อผิดพลาด ดังนั้น processing log ของโปรเซส Data receiver จะมีข้อความที่แสดงว่าดังรูปที่ 3.37



**รูปที่ 3.37** step ตรวจสอบ Data receiver log ของ Repair feature

ต่อมาในส่วนของโปรเซส Data validator นั้น ใน scenario นี้ หากการทำงานของระบบถูกต้องตามปกติ การทำงานที่ควรจะเกิดขึ้นคือ โปรเซส data validator จะต้องตรวจพบว่าข้อมูลที่ ได้รับมานั้น มีข้อผิดพลาดโดย มีฟิลด์ 2 ฟิลด์หายไปจากที่ควรจะมี 4 ฟิลด์ และทำการบอกว่าฟิลด์ที่หายไปนั้นคือฟิลด์ FirstName และ LastName ตามที่เราตั้งใจ จากนั้นต้องทำการอัพเดทสถานะของไฟล์เป็น INVALID โดยข้อความที่เกิดขึ้นนั้นควรจะเป็นไปดังรูปที่ 3.38



**3.38** step การตรวจสอบ Data validation log ของ Repair feature

หลังจากไฟล์ของเราอยู่ในสถานะ INVALID แล้วขั้นตอนต่อไปในการทำงานตามปกติก็คือ ระบบจะทำการเรียกใช้บริการของ Repairing third party ที่มีส่วนติดต่อแอพพลิเคชั่นแบบ SOAP web service เพื่อส่งข้อมูลที่มีข้อผิดพลาดไปทำการซ่อม เมื่อซ่อมเสร็จแล้ว third party จะทำการส่งข้อมูลที่ซ่อมเสร็จแล้วกลับมาให้กับระบบ MicroData ด้วยการเรียกใช้บริการของ MicroData ที่เป็น SOAP web service เช่นเดียวกัน ดังนั้นในการทำงานของซอฟต์แวร์ทดสอบอัตโนมัติของเรานั้น เราจะทำการจำลองการทำงาน 2 ขั้นตอนคือ ขั้นแรกจะทำการสร้าง request ที่ใช้ในการขอซ่อมไฟล์ ในรูปแบบของ SOAP message แล้วทำการส่ง message นั้นไปหา third party และหลังจากนั้นโดยปกติแล้วเราจะต้องรอ third party ในการทำการซ่อมไฟล์ และส่งข้อมูลที่สมบูรณ์กลับมาให้เราผ่าน service แต่ในที่นี้ เราจะทำการจำลองการทำงานของ Repairing third party ด้วยการสร้าง SOAP message ที่มีข้อมูลที่สมบูรณ์เหมือนผ่านการซ่อมแล้วอย่างถูกต้อง พร้อมที่จะส่งกลับมาให้กับ MicroData แล้วนำ message นั้นไปวางไว้ที่ด้านล่างของ step ใน feature file ของเรา เพื่อให้ สามารถส่งข้อมูลที่ซ่อมแล้วกลับมาแบบอัตโนมัติ ไม่ต้องรอให้ตัว third party ทำงานจริง.

โดย step ที่ใช้ในการทำขั้นตอนข้างต้น และหน้าตาของ SOAP message ที่ใช้ในการซ่อมข้อมูล จะแสดงดังรูปที่ 3.39



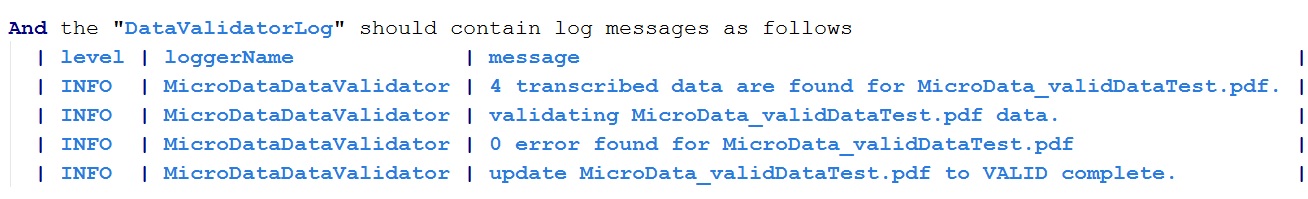
**รูปที่ 3.39** step ในการซ่อมข้อมูลของ scenario ที่ฟิลด์หายไป 2 ฟิลด์

หลังจากทำการส่งข้อมูลที่ได้รับการซ่อมโดย Repairing third party แล้วกลับมาที่ระบบ MicroData. ปกติข้อมูลที่กลับมาจาก Transcription third party จะถูกส่งไปที่ Data receiver process แต่ในที่นี้ข้อมูลของเราถูกส่งกลับมาโดย Repairing third party ดังนั้นข้อมูลจะถูกส่งกลับมาที่โปรเซส Repair receiver ที่ทำหน้าที่คล้ายกับ Data receiver แต่ Repair receiver จะทำหน้าที่ตรวจฟอร์แมทของ SOAP และ JSON เช่นกัน เมื่อตรวจสอบว่าฟอร์แมทของข้อมูลที่ส่งกลับมานั้นถูกต้อง โปรเซส Repair receiver จะทำการเปลี่ยนสถานะของไฟล์จาก INVALID ไปเป็น REPAIRED แล้วจึงทำการส่งต่อไปที่โปรเซส Data validation ต่อไป โดยข้อความของการตรวจสอบโปรเซส Repair receiver จะเป็นไปดังรูปที่ 3.40 ดังนี้



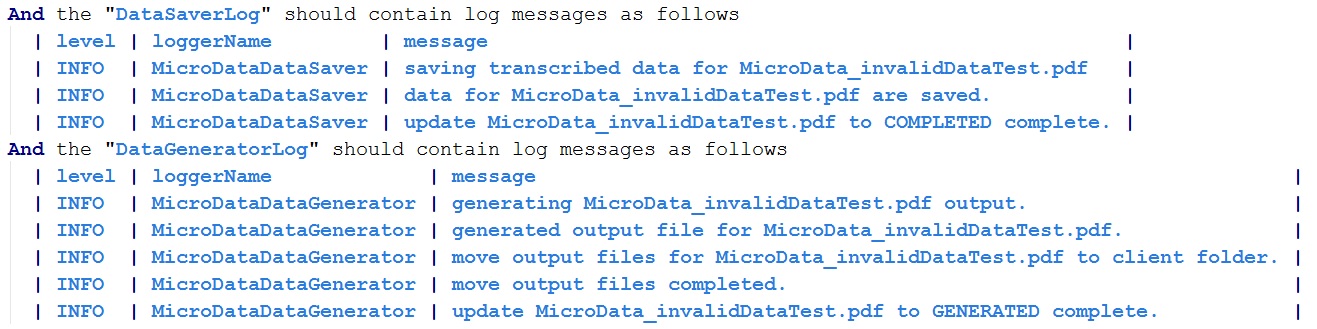
**รูปที่ 3.40** step การตรวจสอบ Repair receiver log ของ Repair feature

หลังจากตรวจสอบ Repair receiver processing log แล้ว โปรเซสต่อไปที่เราจะทำการตรวจสอบต่อไปนั้นคือ โปรเซส Data validation เนื่องจากระบบ MicroData จะทำการตรวจสอบข้อมูลที่ถูกซ่อมกลับมาแล้วอีกครั้ง โดยข้อมูลที่ผ่านการ validation ครั้งนี้จะต้องไม่มีข้อผิดพลาดแล้ว ดังนั้นข้อความของ processing log ในรอบนี้ของ Data validation จะต้องเป็นไปดังรูปที่ 3.41



**รูปที่ 3.41** step การตรวจสอบ Data validator log ของ Repair feature

โดย processing log ที่ถูกต้องจะต้องแสดงว่าพบฟิลด์ครบทั้ง 4 ฟิลด์ และทั้ง 4 ฟิลด์นั้นมีชื่อตรงตามกับข้อกำหนด และเปลี่ยนสถานะของไฟล์เป็น VALID. หลังจากผ่านการตรวจสอบจากการ ทำ validation แล้ว อีก 2 ขั้นตอนสุดท้ายคือโปรเซส Data saver และ Data generator จะทำงานเหมือนกับกับกรณีที่ข้อมูลมีความถูกต้องทุกประการ โดยข้อความของทั้ง 2 โปรเซส จะแสดงรวมกันในรูปที่ 3.42

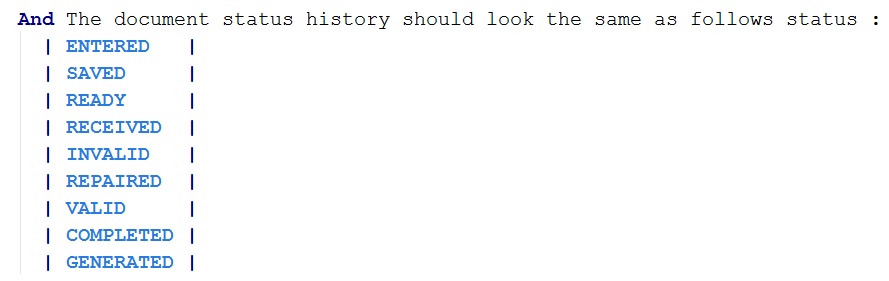


**รูปที่ 3.42** step การตรวจสอบ Data saver และ output generator ของ Repair feature

หลังจากตรวจสอบโปรเซส Data saver และ Data generator สำเร็จแล้ว ขั้นตอนสุดท้ายในการตรวจสอบความถูกต้องในการทำงาน ที่เกี่ยวข้องกับการซ่อมข้อมูลใน feature นี้ก็คือการตรวจสอบข้อมูลในฐานข้อมูลเหมือนที่ผ่านมา โดยในท้ายที่สุดแล้ว ข้อมูลของเราจะต้องถูกสร้างไฟล์ผลลัพธ์ออกมา เพราะฉะนั้นสถานะของไฟล์ของเราหลังจากจบทุกโปรเซส จะต้องมีสถานะเป็น GENERATED โดย step นั้นจะใช้ step เดียวกันกับในรูปที่ 3.32 และในส่วนของประวัติของสถานะนั้น ใน scenario นี้ ข้อมูลของเรามีการผ่านไปที่โปรเซสการซ่อม และการรับกลับมาต่างๆ โดย สถานะทั้งหมดที่ควรจะเกิดขึ้นกับไฟล์ควรจะเกิดขึ้นเรียงตามลำดับดังนี้

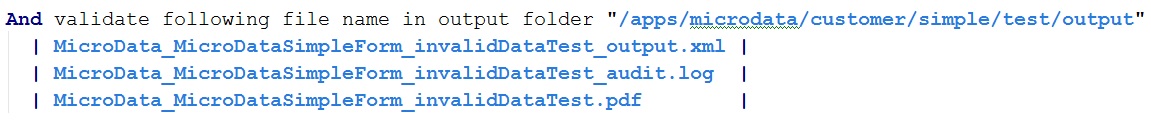
ENTERED -> SAVED -> READY -> RECEIVED -> INVALID -> REPAIRED -> VALID -> COMPLTED -> GENERATED

โดย step ในการตรวจสอบประวัติสถานะของ scenario นี้จะแสดงดังรูปที่ 3.43



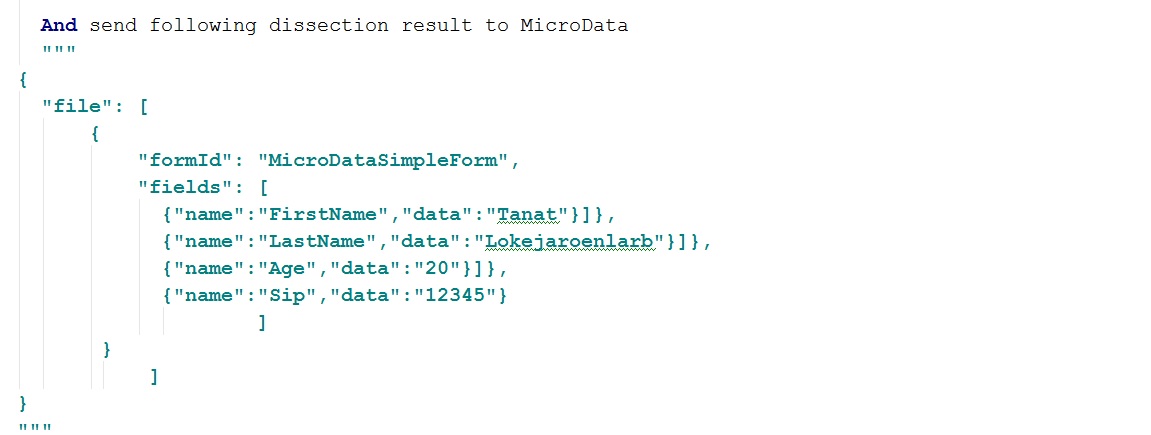
**รูปที่ 3.43** step การตรวจสอบประวัติสถานะของ Repair feature

และในขั้นตอนสุดท้ายของ scenario คือการตรวจสอบไฟล์ผลลัพธ์ด้วย step ในรูปที่ 3.44



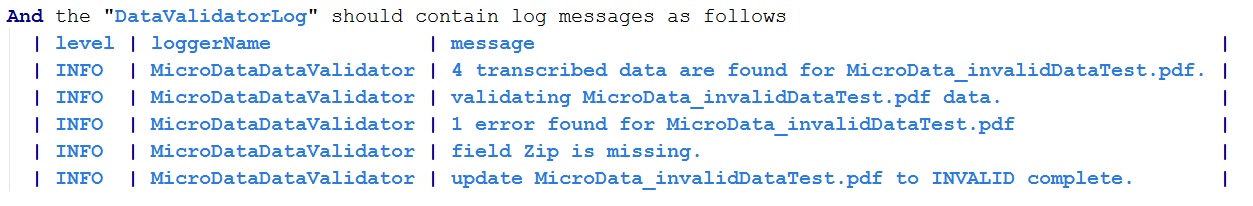
**รูปที่ 3.44** step การตรวจสอบไฟล์ผลลัพธ์ของ Repair feature

ส่วน scenario ที่สองนั้น จะเป็นการทดสอบในกรณีต่างกับกรณีแรกตรงที่ ข้อมูลที่ผิดพลาดนั้น ผิดพลาดที่ชื่อของฟิลด์ใน JSON แทนที่จะเป็นการที่มีฟิลด์หายไป โดย step และวิธีการทดสอบใน scenario นี้เกือบทั้งหมดนั้นจะเหมือนกันกับ scenario ก่อนหน้า ดังนั้นผู้เขียนจึงขอยกตัวอย่างเฉพาะ step ที่มีความแตกต่างกัน โดยเริ่มจาก ข้อมูล transcribed data ที่ทำการส่งกลับมานั้น จะมีฟิลด์ที่ชื่อไม่ตรงกันกับข้อกำหนด โดยในที่นี้จะทำการทดสอบโดยการเปลี่ยนชื่อ จากข้อกำหนดที่ต้องการให้มีฟิลด์ที่ชื่อว่า Zip ที่เอาไว้เก็บค่าของรหัสไปรษณีย์ โดยเราจะทำการเปลี่ยนเป็น Sip เพื่อให้เกิดข้อผิดพลาดขึ้น โดย step ในการส่ง data จะมีลักษณะดังรูปที่ 3.45



**รูปที่ 3.45** step ในการส่งไฟล์ที่ชื่อฟิลด์ผิดข้อกำหนดของ Repair feature

จากรูปจะเห็นได้ว่าฟิลด์สุดท้ายจะมีชื่อฟิลด์เป็น Sip แทนที่จะเป็น Zip ดังนั้นในโปรเซสการทำ Data validation นั้นจะต้องตรวจพบและแสดงข้อความว่าฟิลด์ Zip หายไป โดย step การตรวจในส่วน validation จะเป็นไปดังรูปที่ 3.46



**รูปที่ 3.46** step ในการตรวจสอบ data validator log ของ Repair feature กรณี field Zip

หลังจากที่ไฟล์ของเราผิดพลาดจากการที่มีชื่อฟิลด์ผิดพลาด ไฟล์ของเราก็จะถูกเปลี่ยนไปอยู่ที่สถานะ INVALID โดย step ในการส่ง message กลับมาซ่อมข้อมูลนั้นจะส่งฟิลด์ Zip ที่ถูกต้องกลับมาให้ดังรูปที่ 3.47



**รูปที่ 3.47** step ในการซ่อมข้อมูลฟิลด์ Zip ของ Repair feature

และเมื่อซ่อมข้อมูลแล้ว step ต่างๆในการตรวจสอบข้อมูลทั้งหมด จะเหมือนกับ scenario ก่อนหน้าที่มีฟิลด์หายไป 2 ฟิลด์

**3.4 การพัฒนาซอฟต์แวร์**

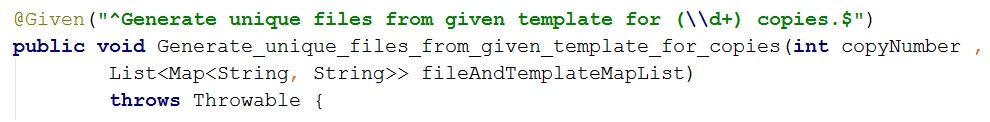
หลังจากได้ทำการออกแบบการพัฒนาซอฟต์แวร์ทดสอบอัตโนมัติ ในส่วนของ feature file หรือ test script และอธิบายถึงจุดประสงค์ของการทดสอบในแต่ละ feature file รวมถึงการทำงานคร่าวๆ ของแต่ละ step ไปในหัวข้อที่แล้วนั้น โดยในหัวข้อนี้ จะอธิบายถึงการพัฒนาซอร์สโค้ดที่ใช้ในการ implement step แต่ละ step ภายใน feature file ว่าการทำงานในด้านหลัง ที่เขียนด้วยภาษา Java นั้น ทำงานอย่างไร

โดยจะอธิบายแต่ละ step ภายในแต่ละ feature ดังต่อไปนี้

1. **Step การสร้างไฟล์จาก template**

เริ่มต้นที่ step แรกคือ step ที่ใช้ในการสร้างไฟล์ขึ้นมาเพื่อใช้ในการอัพโหลดสู่ระบบ

โดยส่วนหัวของเมทอดที่ทำงานในด้านหลังของ step นี้จะแสดงดังรูปที่ 3.48

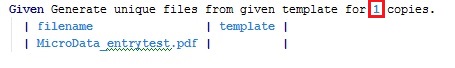


**รูปที่ 3.48** ส่วน header ของ method การสร้างไฟล์

จากรูปที่ 3.48 จะเห็นว่ามีส่วนของ Annotation @Given ในด้านบนของเมทอด ซึ่ง

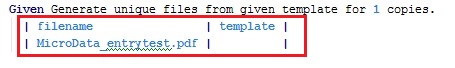
ส่วนนี้ทำหน้าที่ในการเชื่อมโยงซึ่งทำโดยอัตโนมัติด้วย Cucumber framework โดยการเชื่อมโยงนั้นมีไว้เพื่อบอกว่าหากภายใน scenario มีการเรียกใช้ step ที่มีข้อความตรงกับ regular expression ซึ่งอยู่ภายในข้อความหลัง @Given นี้ เมทอดที่อยู่ใต้ @Given ที่เชื่อมกันนี้จะถูกเรียกให้ทำงาน

โดยจากรูปจะสังเกตุเห็นได้ว่า เมทอดมีพารามิเตอร์จำนวน 2 พารามิเตอร์ ซึ่งหลักการของการกำหนดพารามิเตอร์นั้นจะเป็นไปตาม framework ของ Cucumber คือ ในเมทอดนั้น พารามิเตอร์จะถูกกำหนดตามจำนวนของค่าที่เปลี่ยนแปลงได้ใน regular expression ของเรา ซึ่งค่าที่เปลี่ยนแปลงได้ในชื่อ step นั้นหลักๆแล้วจะประกอบไปด้วย 2 กรณีคือ หากในประโยคของ step เป็นตัวเลข จะถูกแทนที่ด้วย regular expression ([\\d](file:///\\d)+) ซึ่งค่าของตัวเลขใดๆที่ใส่มานั้น จะถูกแมพเข้ามาสู่พารามิเตอร์ที่รับด้วยประเภทของข้อมูล Integer ส่วนกรณีที่ 2 คือหากภายในข้อความของ step มีเครื่องหมายฟันหนู (“”) จะถูกแทนที่ด้วย regular expression "([^"]\*)" ซึ่งค่าของข้อความภายในเครื่องหมายฟันหนูจะถูกแมพเข้าสู่พารามิเตอร์ที่รับด้วยประเภทข้อมูล String. โดยหากใน step มีค่าที่เปลี่ยนแปลงได้มากกว่า 1 ค่า พารามิเตอร์ในเมทอดนั้นจะเชื่อมโยงกับข้อความภายใน step โดยเรียงจากทางด้านซ้ายไปขวา พารามาเตอร์ตัวแรกจะเชื่อมกับข้อความที่เปลี่ยนแปลงได้ค่าแรกที่พบ และเรียงลำดับไปเรื่อยๆ. ในที่นี้ step แรกของเรานั้นหากดูจากรูปที่ 3.49 จะเห็นได้ว่ามีค่าที่เปลี่ยนแปลงได้จำนวน 1 ค่าในกรอบสีแดง โดยจะเห็นว่าข้อความที่เปลี่ยนแปลงได้นั้นจะมีสีเป็นสีฟ้า ซึ่งจากรูป ค่า 1 ที่อยู่ในกรอบสีแดง จะถูกส่งไปให้กับพารามิเตอร์ copyNumber ในเมทอดในรูปที่ 3.48 ซึ่งหากใน step มีค่าเป็นเลขอื่นๆ ค่าของ copyNumber ก็จะมีค่าต่างๆกันไปตามที่ได้ใส่ไว้ใน step.



**รูปที่ 3.49** ค่าที่เปลี่ยนแปลงได้ใน step ประเภท Integer

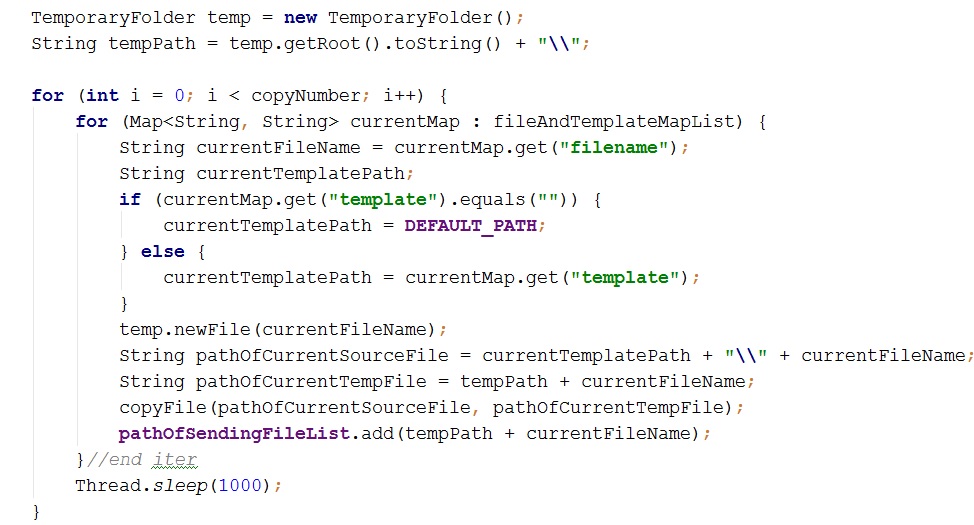
จากรูปที่ 3.49 จะเห็นได้ว่าข้อความที่เปลี่ยนแปลงได้ใน step นั้นมีเพียงแค่ 1 จุดแต่ว่าพารามิเตอร์ของเราในเมทอดในรูปที่ 3.48 นั้นมี 2 ค่า เป็นเพราะว่าหลังจาก Cucumber ทำการเชื่อมโยงค่าของข้อความที่เปลี่ยนแปลงได้หมดแล้ว พารามิเตอร์ช่องอื่นๆที่ถัดไปจากการเชื่อมโยงข้อความที่เปลี่ยนแปลงได้นั้น จะเป็นช่องพารามิเตอร์ที่เอาไว้เชื่อมโยงเอาค่าในตารางข้อมูลที่อยู่ด้านล่างของแต่ละ step เข้ามาในเมทอด. ตารางข้อมูลจะแสดงในกรอบสีแดงดังรูปที่ 3.50



**รูปที่ 3.50** ตัวอย่างตารางข้อมูลด้านล่างของ step

โดยตารางในรูปที่ 3.50 นั้นจะถูกแปลงเป็นพารามิเตอร์ที่มีประเภทเป็น List ของ Map<String,String> คือ list ที่ประกอบไปด้วยคู่ของ key , value ที่มีประเภทข้อมูลเป็น String ทั้งคู่ โดยแต่ละ Map จะมี key 2 key คือ 1. Key ชื่อ filename และ 2. Key ชื่อ template. ซึ่งจากตารางดังรูป list จะมี map 1 map ซึ่งถ้าเราดึงค่าจาก key fileName ของ map นั้น เราจะได้ value ออกมาเป็น MicroData\_entrytest.pdf นั่นเอง.

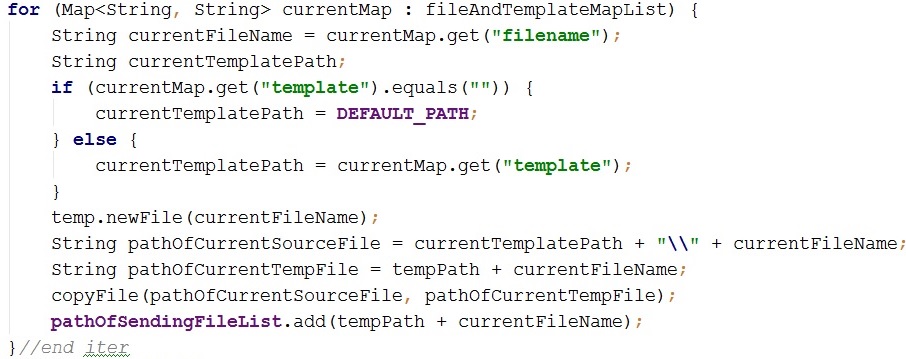
ต่อมาหลังจากทำความเข้าใจกับการเชื่อมโยงค่าต่างๆจาก step เข้ามาสู่เมทอดของเราแล้วนั้น ขั้นตอนต่อไปคือ การนำเอาข้อมูลต่างๆเหล่านั้นไปทำงาน ตามคำอธิบายของ step. โดยซอร์สโค้ดของ step นี้จะมีการทำงานดังรูปที่ 3.51



**รูปที่ 3.51** ซอร์สโค้ดของ step การสร้างไฟล์

เริ่มจาก 2 บรรทัดแรกดังรูปที่ 3.51 คือ คำสั่งที่ใช้ในการสร้าง temporary folder ขึ้นมาในระบบคอมพิวเตอร์ จากนั้นทำการนำเอาตำแหน่งของ temporary folder ที่ถูกสร้างขึ้นมานั้น เก็บเอาไว้ในตัวแปรชื่อว่า tempPath เพื่อที่จะนำไปใช้ในการสร้างไฟล์และส่งไฟล์ต่อไป

ส่วนขั้นตอนต่อไปนั้น จาก for ที่เห็นคือจะวนลูปตามจำนวนรอบตามที่กำหนดเอาไว้ใน step ซึ่งแต่ละรอบนั้นจะทำการสร้างไฟล์ขึ้นมาด้วยซอร์สโค้ดที่จะอธิบายดังรูปที่ 3.52



**รูปที่ 3.52** ซอร์สโค้ดการทำงานแต่ละรอบของการสร้างไฟล์

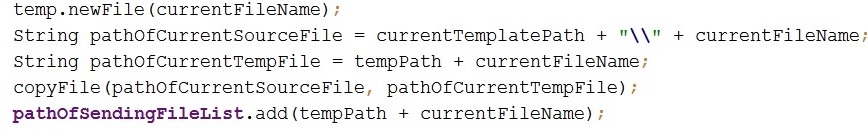
โดยแต่ละรอบของ copyNumber นั้นเราจะทำการวนรอบด้วยแต่ละ Map ใน พารามิเตอร์ fileAndTemplateMapList ซึ่งเก็บค่าของตารางข้อมูลจาก step เอาไว้ หรือถ้าอธิบายง่ายๆก็คือการวนลูปด้านในนี้ เป็นการวนลูปของแต่ละไฟล์ที่เขียนในตารางข้อมูลจาก step ซึ่งในตัวอย่างนี้จะมีเพียงแค่ map เดียว. โดยจะทำการเก็บค่าของ fileName เอาไว้ในตัวแปรชื่อว่า currentFileName. ส่วนทางด้านของ template จะมีตัวแปรที่ชื่อว่า currentTemplatePath เอาไว้เก็บค่า. ซึ่งจากที่เคยอธิบายไปในหัวข้อการออกแบบ คือหากผู้เขียน scenario ไม่ได้ทำการใส่ค่าของ template มานั้นจะถือว่า ให้ใช้ template default ส่วนถ้าใส่ค่า template มาก็จะนำ path ของ template file เก็บไว้ในตัวแปร currentTemplatePath ส่วน path ของ default template นั้นกำหนดไว้ดังรูปที่ 3.53



**รูปที่ 3.53** ตัวแปร DEFAULT\_PATH ของ template

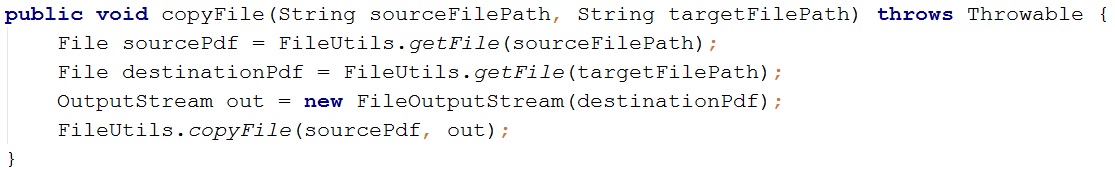
โดยตัวแปร DEFAULT\_PATH นั้นที่เอาไว้เก็บตำแหน่งของไฟล์ template ต่างๆนั้นจะประกาศไว้เป็น global เพื่อให้ทุก เมทอดเข้าถึงได้ โดยตำแหน่งของที่เก็บ template file นั้นจะอยู่ที่ src/test/resources/template ดังรูป.

เมื่อได้ค่าของชื่อไฟล์ และตำแหน่งของ template มาแล้วขั้นตอนต่อไปจะแสดงดังรูปที่ 3.54



**รูปที่ 3.54** ซอร์สโค้ดการสร้าง คัดลอกและเตรียมไฟล์

โดยคำอธิบายการทำงานของซอร์สโค้ดในรูปที่ 3.54 คือ เริ่มจากบรรทัดแรกจะทำการสร้างไฟล์ขึ้นด้วยชื่อไฟล์จากตารางข้อมูลใน step เอาไว้ใน temporary folder ซึ่งไฟล์นี้จะเป็นไฟล์เปล่าๆ หลังจากนั้นบรรทัดที่ 2 คือการนำเอากำหนดตำแหน่งของไฟล์ template เก็บเอาไว้ในตัวแปรชื่อ pathOfCurrentSourceFile และเอาตำแหน่งของไฟล์ปัจจุบัน ซึ่งก็คือ path ของ temporary folder ต่อด้วยชื่อไฟล์ของเรา จากนั้นบรรทัดที่ 3 จะเป็นการเรียกเมทอดซึ่งทำหน้าที่ในการคัดลอกเนื้อหาของไฟล์จาก source file ซึ่งเป็นอาร์กิวเมนต์ตัวที่หนึ่ง ไปยัง target file ซึ่งเป็นอาร์กิวเมนต์ตัวที่ 2 ของเมทอด. ซึ่งจะเห็นได้ว่า source file หรือ template file นั้นคือการนำเอาชื่อไฟล์ของเราไปต่อท้ายตำแหน่งของที่เก็บ template file เพราะฉะนั้นหากในโฟลเดอร์ template ของเราไม่มีไฟล์ที่ชื่อเหมือนไฟล์ที่เราใส่เอาไว้ใน step ก็จะเกิด error ขึ้นได้. โดยเมทอดที่ทำหน้าที่ในการคัดลอกไฟล์นั้นจะแสดงดังรูปที่ 3.55



**รูปที่ 3.55** เมทอดในการคัดลอกไฟล์

โดยเมทอดนี้จะทำการเปิด output stream ของไฟล์ target แล้วทำการคัดลอกไฟล์ต้นทางไปให้กับปลายทาง โดยกระบวนการทำงานต่างๆเหล่านี้จะใช้งานด้วย Class FileUtils ซึ่งทำหน้าที่อำนวยความสะดวกในการทำงานกับ Input/Output Stream ต่างๆของ Java โดย FileUtils เป็น utility class ตัวนึงของ library ที่ชื่อว่า apache commons.

หลังจากคัดลอกไฟล์แล้วขั้นตอนสุดท้ายของ step นี้คือการนำเอาตำแหน่งของไฟล์ที่เราต้องการจะส่งทั้งหมดนั้นไปเก็บไว้ที่ List ตัวนึงเพื่อนำไปใช้ในการส่งไฟล์ต่อไป โดย List ตัวนี้จะเป็นตัวแปรแบบ global ประกาศเอาไว้ด้านนอกของเมทอด ดังรูปที่ 3.56

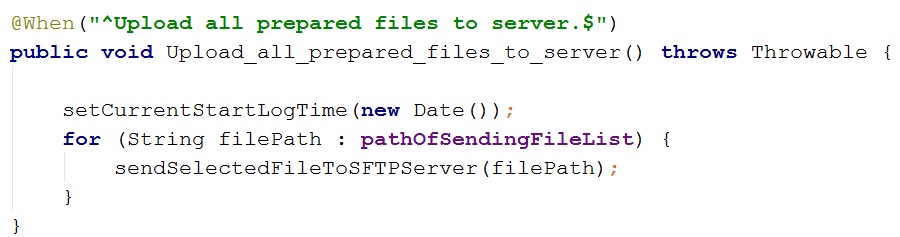


**รูปที่ 3.56** list สำหรับเก็บตำแหน่งของไฟล์ที่สร้างแล้วทั้งหมด

โดยเมื่อเราทำการสร้างและคัดลอกไฟล์ทั้งหมดที่ต้องการสำเร็จหมดเรียบร้อยแล้วก็จะใช้คำสั่งที่อยู่บรรทัดสุดท้ายในรูปที่ 3.54 ในการเก็บตำแหน่งของไฟล์ทั้งหมดเอาไว้

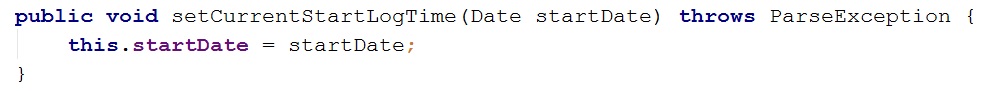
1. **Step การอัพโหลดไฟล์ขึ้นสู่เซิฟเวอร์**

หลังจาก step ก่อนหน้าที่ได้ทำการสร้างไฟล์และคัดลอกไฟล์ และทำการเก็บตำแหน่งของไฟล์ที่ต้องการจะส่งทั้งหมดเอาไว้ใน list ที่เตรียมเอาไว้แล้วนั้น. ใน step นี้จะทำการส่งไฟล์ทั้งหมดขึ้นไปสู่ระบบ. สำหรับซอร์สโค้ดในเมทอดของ step นี้จะแสดงดังรูปที่ 3.57



**รูปที่ 3.57** ซอร์สโค้ดเมทอดการส่งไฟล์

โดย step นี้จะเริ่มจากการทำเมทอดที่ชื่อว่า setCurrentStartLogTime() โดยส่งค่า ของ object ของ Class Date ที่พึ่งสร้างใหม่เข้าไปเป็นอาร์กิวเมนต์ของเมทอด ซึ่ง object ที่สร้างใหม่ของ Class Date นั้นจะเป็นตัวเก็บค่าของเวลาปัจจุบัน.โดยซอร์สโค้ดของเมทอด setCurrentStartLogTime จะมีดังรูปที่ 3.58



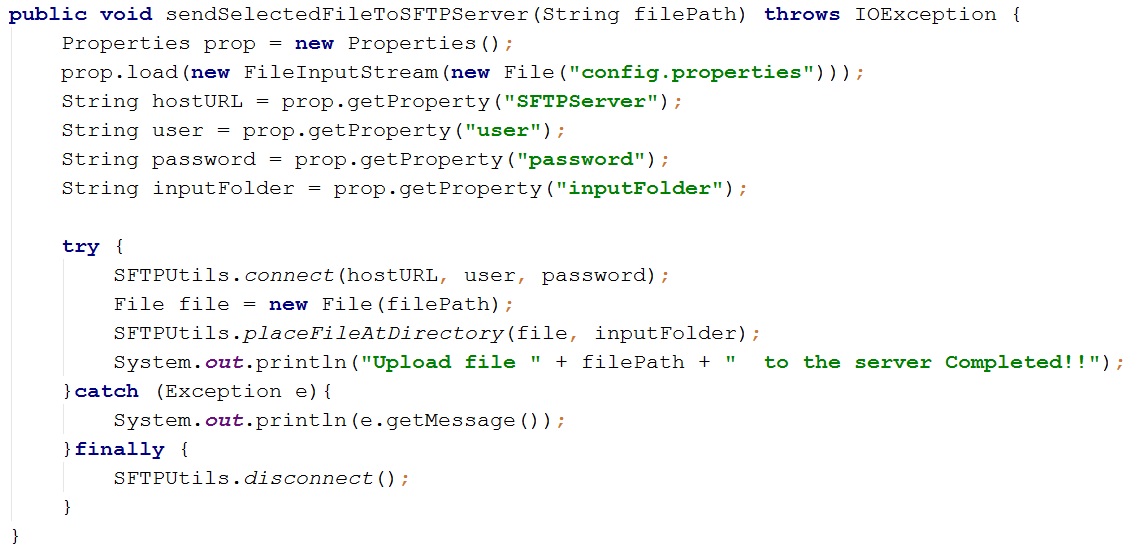
**รูปที่ 3.58** ซอร์สโค้ดของเมทอด setCurrentStartLogTime

ซึ่งจุดประสงค์ของเมทอดนี้ก็คือ เนื่องจากว่า ใน processing log ที่เราจะทำการตรวจสอบนั้นจะถูกบันทึกลง ภายในช่วงเวลาต่างๆ การที่เราทำการดึงมาตรวจสอบทั้งหมด จะทำให้การตรวจสอบอาจมีการคลาดเคลื่อนได้ ดังนั้นการตรวจสอบ processing log จึงจะทำการดึงเอาเฉพาะ processing log ที่อยู่ในช่วงเวลาหนึ่งๆ ที่มีนัยะสำคัญต่อการตรวจสอบเท่านั้น. เมทอดนี้จึงทำหน้าที่ในการตั้งค่าเวลาเริ่มต้นของการตรวจสอบ หรือก็คือ การดึงเอา processing log มาจะเอาเฉพาะ log ที่เริ่มบันทึกหลังจากเวลาที่ตั้งในเมทอดนี้เท่านั้น เนื่องจากว่า เราจะทำคำสั่งนี้ก่อนที่เราจะทำการอัพโหลดไฟล์ขึ้นสู่ระบบ เพื่อเป็นการยืนยันได้ว่า log ที่จะเกิดขึ้นทั้งหมดจะเกิดขึ้นหลังจากเวลานี้. โดยเราจะมีตัวแปรที่ทำการเก็บเวลาที่จะเริ่มทำการดึง log และเวลาสุดท้ายที่จะทำการดึงเช่นกัน เอาไว้เป็นตัวแปร global ดังรูปที่ 3.59



**รูปที่ 3.59** ตัวแปรช่วยในการเก็บเวลา

หลังจากทำการตั้งเวลาเรียบร้อยแล้ว ต่อมาเมทอดจะทำงานโดยการวนลูปเอาค่า ตำแหน่งของไฟล์ที่เราต้องการจะส่งออกมาจาก List pathOfSendingFileList ที่ได้เตรียมไว้ ออกมาเก็บที่ตัวแปร filePath. หลังจากนั้นก็จะทำการส่งตำแหน่งของไฟล์แต่ละไฟล์ที่ต้องการไปยังระบบด้วยเมทอดที่ชื่อว่า sendSelectedFileToSFTPServer ซึ่งรับพารามิเตอร์คือ filePath ซึ่งมีประเภทของข้อมูลเป็น String โดยในส่วนของการทำงานของเมทอดในการอัพโหลดไฟล์นั้นจะอธิบายจากรูปที่ 3.60

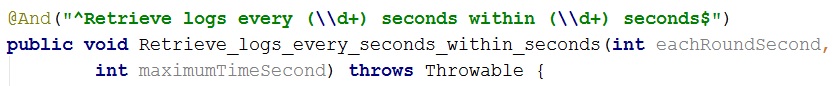


**รูปที่ 3.60** ซอร์สโค้ดเมทอดการส่งไฟล์

โดยการทำงานของเมทอดนี้จะเริ่มจากการสร้าง object ของ Class Properties ขึ้นมาแล้วทำการโหลดค่าของ property ต่างๆที่อยู่ในไฟล์ที่ชื่อว่า config.properties มาเก็บเอาไว้ในตัวแปร prop. โดยค่าต่างๆที่อยู่ในไฟล์จะเป็นข้อมูลที่จำเป็นในการเชื่อมต่อกับเซิฟเวอร์ของระบบ. โดยจะทำการเก็บค่า user password url ของเซิฟเวอร์ และตำแหน่งของโฟลเดอร์บนเซิฟเวอร์ที่ต้องการจะนำไฟล์ไปวาง จาก properties file เอาไว้ในตัวแปร. ต่อมาจะใช้ SFTPUtils ซึ่งเป็น Class นึงใน library ที่เขียนขึ้นในการส่งไฟล์ขึ้นไปบนระบบ ซึ่งสร้างขึ้นโดยอาศัย library Jsch ซึ่งเป็น library ที่ใช้ในการทำงานต่างๆเกี่ยวกับการรับ-ส่งข้อมูลผ่าน protocol เช่น FTP , SFTP. โดยการส่งไฟล์ด้วย SFTPUtils จะเริ่มจากการใช้คำสั่ง connect ในการเชื่อมต่อกับเซิฟเวอร์ผ่าน protocol SFTP ด้วยข้อมูลที่เราตั้งไว้ในไฟล์ config.properties หากการเชื่อมต่อไม่สำเร็จจะมีการ Throw Exception ออกมา. เมื่อทำการเชื่อมต่อสำเร็จแล้ว ก็จะทำการสร้าง object ของไฟล์ขึ้นมาจากตำแหน่งของ filePath และทำการอัพโหลดไฟล์ด้วยเมทอด placeFileAtDirectory โดยรับพารามิเตอร์เป็น object ของไฟล์และตำแหน่งที่ต้องการจะนำขึ้นไปวาง หากการอัพโหลดไม่สำเร็จ จะมีการ Throw Exception ออกมา. เมื่อทำการอัพโหลดเสร็จแล้วจะทำการแสดงข้อความบอกถึงผลการอัพโหลด และเข้าไปทำงานในส่วนของ finally ทำการปิดการเชื่อมต่อจากระบบ.

1. **Step การหน่วงเวลาการทดสอบ**

สำหรับ step ในการหน่วงเวลาเพื่อรอให้ระบบทำงานนั้น เป็น step ที่มีหน้าที่ในการตั้งค่าเบื้องต้นสำหรับ step ต่อไปในการตรวจสอบ processing log โดยเมทอดที่เป็นตัวเชื่อมกับ step นี้จะแสดงดังรูปที่ 3.61



**รูปที่ 3.61** ส่วน header ของเมทอดการหน่วงเวลา

โดยเมทอดของ step นี้จะมีพารามิเตอร์ 2 ตัวที่เป็นข้อความที่เปลี่ยนค่าได้ และไม่มีพารามิเตอร์ที่รับข้อมูลจากตารางข้อมูล เนื่องจากไม่มีตารางข้อมูลใน step นี้. พารามิเตอร์ตัวแรกคือ eachRoundSecond มีไว้เพื่อตั้งว่าในการตรวจสอบ log แต่ละรอบจะให้มีเวลาเท่าไหร่ เนื่องจากว่าในจังหวะที่เราทำการดึง processing log มาตรวจสอบนั้น อาจจะยังไม่มีการบันทึก log ที่เรากำลังสนใจ. เวลาใน eachRoundSecond จะมีไว้เพื่อทำการหยุดรอก่อนที่จะทำการดึง processing log รอบต่อไป. ส่วนพารามิเตอร์ตัวที่ 2 นั้นมีชื่อตัวแปรว่า

maximumTimeSecond เอาไว้เก็บเวลาที่มากที่สุดที่จะใช้ในการตรวจสอบ log ใดๆ กล่าวคือ ในการตรวจสอบ log แต่ละตัว ถ้าหากเกินเวลาที่ตั้งเอาไว้ในเวลารอสูงสุดนี้ ก็จะให้ระบบทำการแสดงรายงานว่าผิดพลาดเลย เพื่อที่จะไม่ต้องทำการตรวจสอบไปเรื่อยๆ. โดยการตั้งค่าจะทำการตั้งค่าให้กับตัวแปรที่ประกาศไว้เป็น global ดังรูปที่ 3.62



**รูปที่ 3.62** ตัวแปรในการตั้งค่ารอบเวลาการตรวจสอบ log

ส่วนการทำงานของ step คือการรับค่าจาก step มาตั้งค่าให้กับตัวแปร global ตรงๆ ดังรูปที่ 3.63

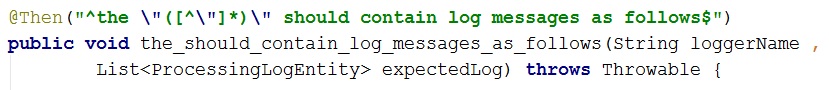


**รูปที่ 3.63** ซอร์สโค้ดการทำงานของของเมทอดการหน่วงเวลา

1. **Step การตรวจสอบ processing log**

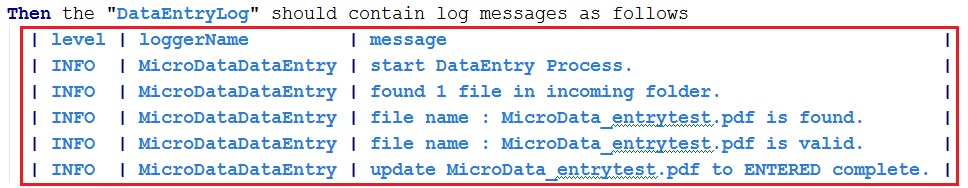
หลังจากทำการอัพโหลดไฟล์ และทำการตั้งค่าต่างๆเสร็จหมดเรียบร้อยแล้วใน step ก่อนๆ ใน step นี้จะเป็นการทำการตรวจสอบ processing log ที่เกิดขึ้นจากการทำงานของระบบ เทียบกับ processing log ที่เราเตรียมไว้.

เริ่มจากส่วน header ของเมทอดที่ทำหน้าที่ในการเชื่อมโยงดังรูปที่ 3.64



**รูปที่ 3.64** ส่วน header ของเมทอดการตรวจสอบ processing log

โดย step นี้จะมีค่าที่เปลี่ยนแปลงได้ในชื่อ step อยู่ 1 ค่าซึ่งเป็นประเภท String ซึ่งจะถูกแมพกับพารามิเตอร์ loggerName ซึ่งทำหน้าที่รับชื่อของโปรเซสที่เราต้องการจะทำการตรวจสอบ. ส่วน พารามิเตอร์ตัวที่ 2 นั้นจะเป็นพารามิเตอร์ที่ทำหน้าที่ในการเชื่อมโยงกับค่าในตารางข้อมูลดังรูปที่ 3.65



**รูปที่ 3.65** ตารางข้อมูลที่เก็บข้อมูล expected log

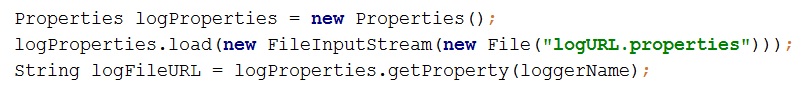
โดยในตารางข้อมูลนั้นจะประกอบไปด้วย 3 ส่วนคือ level, loggerName และ message ของ processing log ดังนั้นพารามิเตอร์ที่เราจะนำมารับนั้น จะใช้เป็นประเภท map ไม่ได้เนื่องจากมี 3 ส่วน ดังนั้นเราจึงจะต้องสร้าง Entity ตัวนึงเอาไว้ใช้สำหรับการเชื่อมโยงตารางข้อมูลนี้มาในโปรแกรมของเราโดย Class ที่เราจะนำมาเป็นตัวเชื่อมนั้นจะมีชื่อว่า ProcessingLogEntity โดยมีฟิลด์ 3 ฟิลด์ที่ชื่อเหมือนกับในตารางข้อมูลเพื่อให้ Cucumber สามารถทำการ auto mapping ให้เราได้ โดย Class จะแสดงดังรูปที่ 3.66



**รูปที่ 3.66** ProcessingLogEntity mapping class

เพราะฉะนั้นในการเชื่อมข้อมูล 1 แถวของ processing log จากตารางข้อมูลจะถูกแทนที่ด้วย object ของ ProcessingLogEntity 1 ตัว. ดังนั้นเนื่องจากในตารางข้อมูลจะมี processing log จำนวนหลายๆชุด พารามิเตอร์ในเมทอดของเราจึงเชื่อมค่ากลับมาเป็น List ของ ProcessingLogEntity ดังรูปที่ 3.64

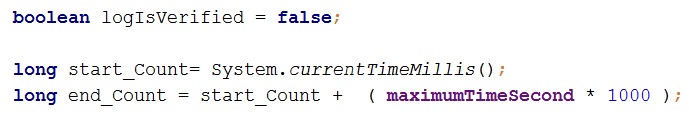
สำหรับการทำงานขั้นต่อไปหลังจากทำการเชื่อมโยงข้อมูลจาก step มาสู่เมทอดแล้วก็คือ การทำการเปรียบเทียบ processing log. ในส่วนแรกของเมทอดนี้จะทำหน้าที่ในการดึงเอา url ซึ่งเป็นที่อยู่ของ log บนเซิฟเวอร์ออกมาจากไฟล์ properties ที่ได้ทำการกำหนดเอาไว้แล้วว่า loggerName ที่ใส่มาเป็นพารามิเตอร์นั้น loggerName ตัวไหนจะมี url อยู่ที่ไหน ดังที่เคยอธิบายเอาไว้ในส่วนของการออกแบบ. โดยขั้นตอนการทำงานจะอธิบายดังรูปที่ 3.67



**รูปที่ 3.67** ซอร์สโค้ดการดึงเอา url จาก logURL.properties

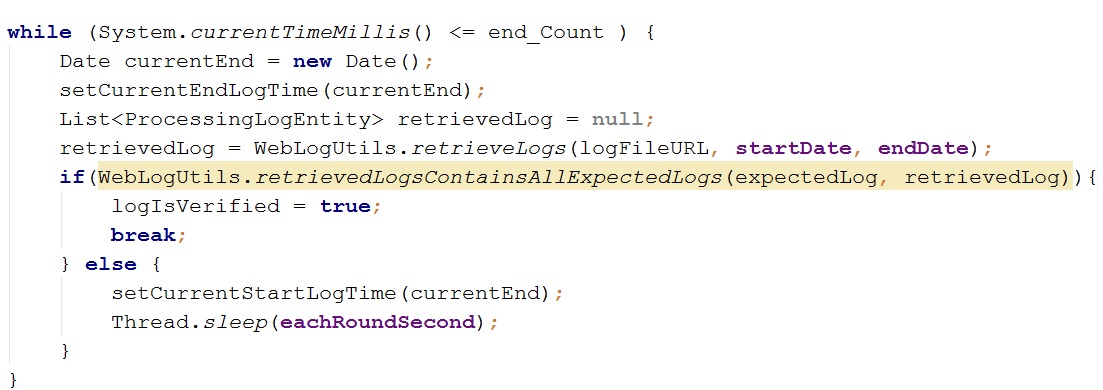
วิธีการทำงานคือ ทำการสร้าง object properties มาจากไฟล์ logURL หลังจากนั้นจึง นำค่า loggerName เป็น key ไปหาค่า url ในไฟล์ แล้วนำมาเก็บไว้ที่ตัวแปรlogFileURL หากหาไม่เจอโปรแกรมจะ throw Exception ออกมา.

หลังจากได้ url ของไฟล์ log มาแล้วขั้นตอนต่อไปคือการเตรียมข้อมูลเงื่อนไขที่จำเป็นสำหรับการตรวจสอบ log โดยมีวิธีการดังรูปที่ 3.68



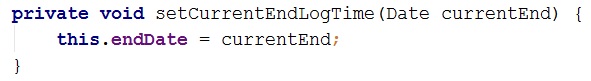
**รูปที่ 3.68** ซอร์สโค้ดการเตรียมเงื่อนไขสำหรับการทดสอบ processing log

โดยเริ่มจากประกาศตัวแปร logIsVerified เอาไว้โดยมีค่าเริ่มต้นเป็น false ส่วนรายละเอียดจะอธิบายในส่วนต่อไป. ต่อมาทำการประกาศตัวแปร start\_Count ที่มีประเภทข้อมูลเป็น long ที่เอาไว้เก็บทศนิยม เนื่องจากว่า คำสั่ง System.currentTimeMillis(); นั้นจะ return ค่าออกมาเป็นหน่วย มิลลิวินาที ซึ่งแสดงค่าของเวลา ณ ปัจจุบัน. โดยตัวแปร start\_Count มีหน้าที่เก็บเวลาเริ่มต้นในการตรวจสอบ log. ส่วนตัวแปร end\_Count จะมีค่า คือเอาเวลาเริ่มต้น หรือ start\_Count ไป + กับ maximumTimeSecond ซึ่งได้อธิบายไว้ใน step ที่แล้วว่า เป็นเวลาสูงสุดที่จะรอ ดังนั้นค่าของ end\_Count ก็คือเวลาสุดท้ายก่อนที่จะยกเลิกการตรวจสอบ โดยที่ต้องนำ maximumSecond \*1000 เพราะต้องการแปลงหน่วยเวลาของ maximumSecond จากวินาทีเป็น มิลลิวินาที. หลังจากเตรียมตัวแปรที่จำเป็นไว้แล้ว ต่อมาคือขั้นตอนการตรวจสอบ log โดยจะแสดงดังรูปที่ 3.69



**รูปที่ 3.xx** ซอร์สโค้ดการตรวจสอบ processing log

โดยขั้นตอนการตรวจสอบนั้นอยู่ใน ลูป While ที่มีเงื่อนไขคือ เวลา ณ ปัจจุบันจะต้องไม่มากกว่าค่าของตัวแปร end\_Count หรือหมายความอีกอย่างก็คือ ลูปการตรวจสอบนี้จะหยุดการทำงาน เมื่อเวลาที่ตรวจสอบเกินเวลาที่กำหนดไว้. โดยเมื่อเข้ามาในรอบของการตรวจสอบ จะเริ่มจากการจับเวลา ณ ปัจจุบัน และตั้งค่าให้กับตัวแปร global endDate ซึ่งมีไว้บอกถึงเวลาสุดท้ายในการดึง log. โดยเมทอดที่ใช้มีวิธีการทำงานดังรูปที่ 3.70

 **รูปที่ 3.70** ซอร์สโค้ดของเมทอด setCurrentEndLogTime

เมื่อทำการตั้งเวลาสุดท้ายในการดึง processing log จากระบบแล้ว ขั้นตอนต่อไปคือขั้นตอนการดึง processing log ของระบบจาก url ที่ได้มาจากไฟล์ logURL.properties

โดยจะใช้ Class ชื่อว่า WebLogUtils ซึ่งเป็น Class ที่ทำหน้าที่เกี่ยวกับการตรวจสอบ processing log โดยตรง โดยการดึง log จากระบบ จะใช้เมทอดของ Class WebLogUtils

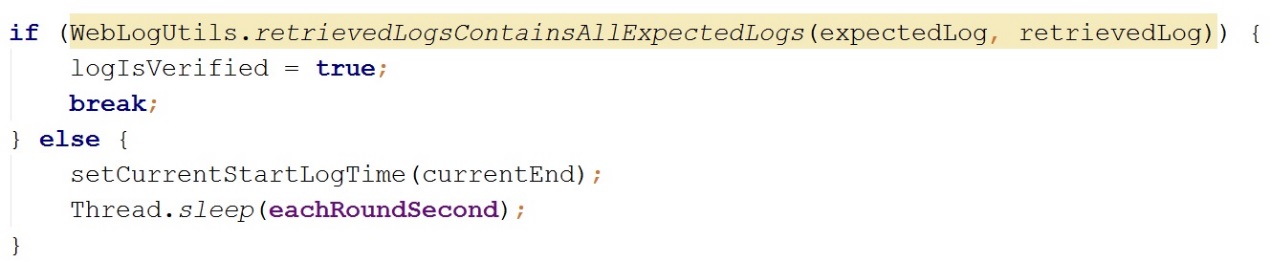
ชื่อ retrievedLogs ซึ่งรับอาร์กิวเมนต์ทั้งหมด 3 ตัวเรียงตามลำดับ ได้แก่ 1.url ของไฟล์ log 2. เวลาของ processing log ที่ต้องการเริ่มดึง และ 3.คือ เวลาสุดท้ายของ processing log ดังรูปที่ 3.71



**รูปที่ 3.71** ซอร์สโค้ดการใช้เมทอด retrieveLogs

ดังนั้นเมทอดนี้จะทำการดึง processing log ที่มีช่วงเวลาการบันทึกอยู่ระหว่าง startDate คือเวลาอัพโหลด และ endDate คือเวลาปัจจุบันจากไฟล์ log บนระบบที่ url ที่กำหนด. โดยเมทอดนี้จะคืนค่าออกมาเป็น List ของ ProcessingLogEntity ซึ่งเราจะนำไปเก็บไว้ในตัวแปรชื่อ retrievedLog. หลังจากได้ log จากระบบมาแล้ว ขั้นตอนต่อไปคือการตรวจสอบ log ซึ่งเราจะนำเอา list ของ log ที่ได้มาจากระบบ มาทำการตรวจสอบว่า ภายใน list ของ log ที่ได้มานั้นมี log ทุกตัวใน list ของ log ที่เราจะตรวจสอบอยู่หรือไม่ โดยการตรวจสอบนั้น เราจะใช้ เมทอดของ WebLogUtils อีกเช่นกัน โดยเมทอดจะมีชื่อว่า

retrievedLogsContainsAllExpectedLogs() ซึ่งเมทอดนี้จะรับพารามิเตอร์ 2 ตัวด้วยกัน คือ 1. List ของ ProcessingLogEntity ที่เรากำหนดว่าต้องมี และ 2. คือ List ของ ProcessingLogEntity ทั้งหมดที่ได้มาจากการอ่านไฟล์ processing log จากระบบ. ซึ่งเมทอดการตรวจสอบนี้จะมีการคืนค่าออกมาเป็นประเภท Boolean โดยเป็น true หากตรวจสอบพบทุก expectedLog ใน retrievedLog และ false ถ้าไม่พบ expectedLog ทั้งหมดใน retrievedLog. โดยการใช้เมทอดจะแสดงดังรูปที่ 3.72



**รูปที่ 3.72** ซอร์สโค้ดเงื่อนไขการตรวจสอบ log

โดยจากรูปจะเห็นได้ว่าการใช้เมทอดตรวจสอบจะอยู่ในเงื่อนไข if-else คือ ถ้าหากตรวจสอบ processing log พบ expected log ทุกตัวที่เราคาดหวังไว้ ก็จะทำการเปลี่ยนค่า boolean ที่ประกาศเอาไว้เป็น true เพื่อบอกว่าการตรวจสอบถูกต้องและ break; การตรวจสอบเลย เพื่อไม่ต้องทำการรอตามเวลา timeout ที่เราได้ตั้งไว้. ส่วนถ้าหากตรวจสอบไม่พบ จะทำการนำเอาเวลาสุดท้ายในการดึง log ของรอบนี้ไปเป็นเวลาเริ่มต้นในการดึงของรอบหน้า แล้วทำการหยุดรอให้ระบบทำงาน เป็นเวลาตามที่ได้กำหนดเอาไว้ด้วยการใช้คำสั่ง Thread.sleep() แล้วส่งค่าเวลา eachRoundSecond เป็นอาร์กิวเมนต์. หากเข้าสู่เงื่อนไขที่ตรวจสอบ processing log ไม่พบนั้น ซอฟต์แวร์ก็จะทำการวนขึ้นไปทำการตรวจสอบว่า เกินเวลา maximum ที่การตรวจสอบ while หรือไม่ หากยังไม่เกินก็จะทำการตรวจสอบต่อไปเรื่อยๆ. โดยหลังจากหมดเวลาการตรวจสอบหรือ ออกจากเงื่อนไข while แล้วนั้นขั้นตอนต่อไปก็คือการตรวจสอบผล โดยการตรวจสอบจะใช้เมทอดดังรูปที่ 3.73

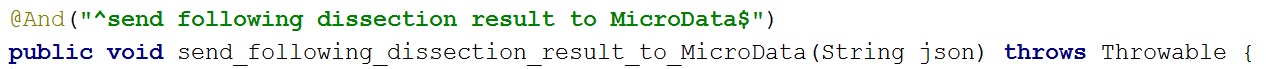


**รูปที่ 3.73** ซอร์สโค้ดสำหรับตรวจสอบผลของการตรวจสอบ log

โดยซอร์สโค้ดจากรูปที่ 3.73 นั้นเป็นการใช้เมทอดของ Class ชื่อว่า Assert ซึ่งเป็น Class สำหรับใช้ช่วยในการตรวจสอบผลของการทำงานต่างๆ โดยเมทอดที่ใช้คือเมทอด assertTrue() ซึ่งรับพารามิเตอร์ 2 ตัวคือ String ข้อความ ซึ่งคือข้อความที่จะถูกแสดงออกมา หากการตรวจสอบผิดพลาด ส่วนพารามิเตอร์ตัวที่ 2 คือค่าของ boolean ที่ใช้ในการตรวจสอบ ซึ่งอาจจะใส่เป็น statement ที่คืนค่าออกมาเป็น boolean ได้ โดยเมทอด asserTrue นั้นมีหลักการทำงานคือ ต้องการจะตรวจสอบว่าค่าของพารามิเตอร์ตัวที่ 2 นั้นเป็น จริง (true) หรือไม่ หากเป็น true การตรวจสอบของ step นี้ก็จะผ่าน ส่วนถ้าเงื่อนไข หรือค่าที่ส่งให้พารามิเตอร์ตัวที่ 2 เป็นเท็จ (false) ก็จะหยุดการตรวจสอบ แสดงผลว่าการตรวจสอบผิดพลาด และแสดงข้อความในพารามิเตอร์ตัวที่ 1 ออกมาให้กับผู้ใช้. โดยค่าของพารามิเตอร์ตัวที่ 2 เราจะส่งตัวแปร logIsVerified เข้าไป ซึ่งเป็น true หากการตรวจสอบ processing log ของเราถูกต้อง และถ้าเวลาเกิน timeout แล้วยังตรวจสอบไม่พบก็จะมีค่าเป็น false.

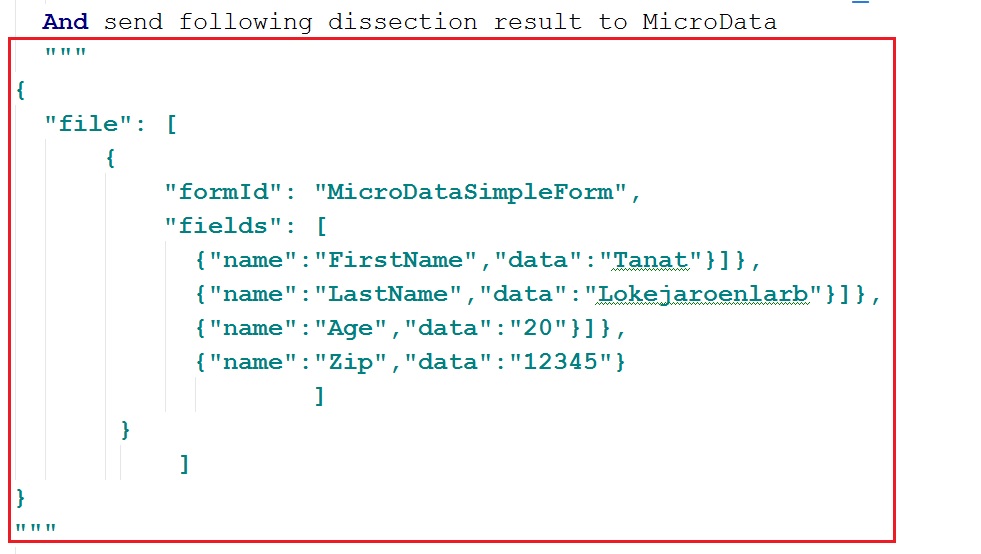
1. **Step การจำลองการส่งข้อมูลกลับสู่ระบบ**

Step นี้คือ step ที่ทำหน้าที่จำลองการทำงานของ third party เพื่อส่งข้อมูลtranscribed data กลับมายังระบบ. โดยการส่งข้อมูลมาที่ระบบนั้น จะส่งข้อมูลผ่านทาง API (Application programming interface) ที่ทางระบบ MicroData เตรียมเอาไว้ ซึ่งเป็น RESTFul web service. ดังนั้น step นี้จะนำเอาข้อมูล JSON ที่เตรียมเอาไว้ส่งไปที่ service ดังกล่าว โดยส่วน header ของ step นี้จะแสดงดังรูปที่ 3.74



**รูปที่ 3.74** ส่วน header ของเมทอดส่งข้อมูล

โดยจากรูปจะเห็นว่ามีพารามิเตอร์ตัวเดียวที่ชื่อว่า json ซึ่งมีประเภทข้อมูลเป็น String โดยพารามิเตอร์ตัวนี้จะรับเอาค่าของข้อมูล json จากด้านล่างของ step แสดงในกรอบสีแดง ดังรูปที่ 3.75



**รูปที่ 3.75** ข้อมูล json ด้านล่างของ step การส่งข้อมูล

หลังจากทำการเชื่อมโยงค่าใน step มาไว้ที่ตัวแปร json แล้วขั้นตอนต่อไปจะแสดงขั้นตอนการทำงานของเมทอด ซึ่งจะอธิบายจากรูปที่ 3.76



**รูปที่ 3.76** ซอร์สโค้ดการทำงานของเมทอดส่งข้อมูล

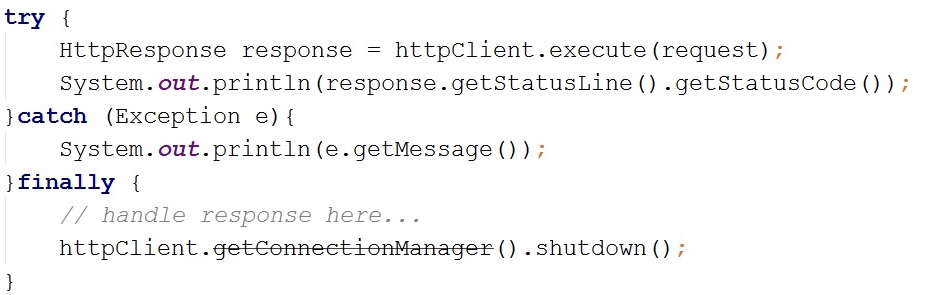
จากรูปที่ 3.76 นั้นการทำงานในส่วนแรกคือการสร้าง object ของ Class DefaultHttpClient ซึ่งเป็น Class ที่ใช้ช่วยในการส่งข้อความผ่าน HTTP (HyperText Transfer Protocol) ซึ่งอยู่ใน library org.apache ที่ใช้ช่วยในการทำงานกับ HTTP protocol เมื่อสร้าง object เก็บไว้ในตัวแปรชื่อว่า httpClientแล้ว ส่วนต่อไปแสดงดังภาพที่ 3.77 จะเป็นส่วนของการสร้าง payload สำหรับการส่งข้อความเพื่อใช้ RESTFul service เนื่องจากว่า RESTFul service นั้นอาศัยเทคโนโลยี HTTP ในการรับ-ส่งข้อมูล



**รูปที่ 3.77** ซอร์สโค้ดการสร้างและตั้งค่า HttpPut object

โดยการทำงานของรูปที่ 3.77 จะเริ่มจากการนำเอา url ตำแหน่งที่อยู่ของ api ของ MicroData เก็บไว้ในตัวแปรชื่อว่า microDataServiceURL จากนั้นทำการสร้าง object ของ Class HttpPut ซึ่งรับพารามิเตอร์เป็นที่อยู่ url ที่ต้องการส่งข้อความ HttpMessage ด้วย method PUT ไปหา เนื่องจากว่า service ของ MicroData ตัวนี้มีหน้าที่ในการให้ผู้ใช้ส่งข้อความเข้ามาที่ระบบ ดังนั้นจึงใช้เมทอด PUT ของ HTTP protocol

จากนั้นทำการตั้งค่า header ของ message เพื่อบ่งบอกว่าข้อความที่เรากำลังจะส่งไปให้นั้นเป็นประเภท json และทำการใช้เมทอด setEntity ในการนำเอา json จาก step ใส่ลงไปใน payload ของ message. เมื่อได้ object ของ HttpPut ที่พร้อมสำหรับการส่งแล้ว ขั้นตอนต่อไปก็คือการส่งข้อความไปที่ service ของระบบของ MicroData โดยขั้นตอนจะแสดงดังรูปที่ 3.78

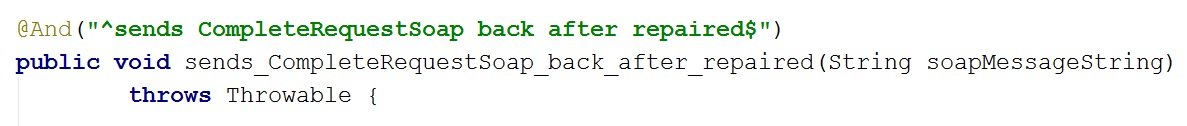


**รูปที่ 3.78** ซอร์สโค้ดการส่ง Http message ด้วย HttpClient

โดยการทำงานทั้งหมดในการส่ง message ไปที่ service นั้นจะอยู่ภายใต้ช่วง try-catch-finally เนื่องจากว่า การทำงานเกี่ยวกับ Http protocol มีสิทธิ์ที่จะเกิดข้อผิดพลาดเกี่ยวกับระบบ และ throw Exception ต่างๆเช่น IOExceotion, ClientProtocolException. คำสั่งที่ใช้ในการส่ง message คือ การใช้เมทอด execute ของ httpClient object โดยเมทอดนี้จะทำการส่ง message ไปที่ที่เราทำการตั้งค่าเอาไว้ที่ request โดยเมทอดนี้จะ return ค่าเป็น object ของ HttpResponse ซึ่งเราจะทำการแสดงข้อความออกมาให้ผู้ใช้เป็น สถานะของการส่งว่าสำเร็จหรือไม่ โดยปกติหากการส่งสำเร็จจะได้ค่ากลัวมาเป็น 200 และถ้าผิดพลาดจะเป็น 400 หรือ 500 ขึ้นอยู่กับชนิดของข้อผิดพลาด. เมื่อทำการส่งข้อมูล Json และแสดงผลลัพธ์แล้ว ซอฟต์แวร์จะเข้าสู่ Finally ไม่ว่าจะส่งสำเร็จหรือไม่ และทำการปิดการเชื่อมต่อกับ httpClient เป็นการจบเมทอด.

1. **Step การจำลองการส่งข้อมูลที่ได้รับการซ่อมแซมกลับสู่ระบบ**

ใน step นี้จะทำหน้าที่ในการส่งข้อมูล Json data ที่ถูกต้อง เป็นข้อมูลที่ได้รับการแก้ไขให้ถูกต้องตามข้อกำหนดแล้ว กลับสู่ระบบผ่านทาง SOAP web service ของระบบ MicroData เพื่อทำการทดสอบโปรเซสการซ่อมไฟล์ที่มีข้อมูลผิดพลาดของระบบ. โดยส่วน header ของ step นี้จะแสดงดังรูปที่ 3.79



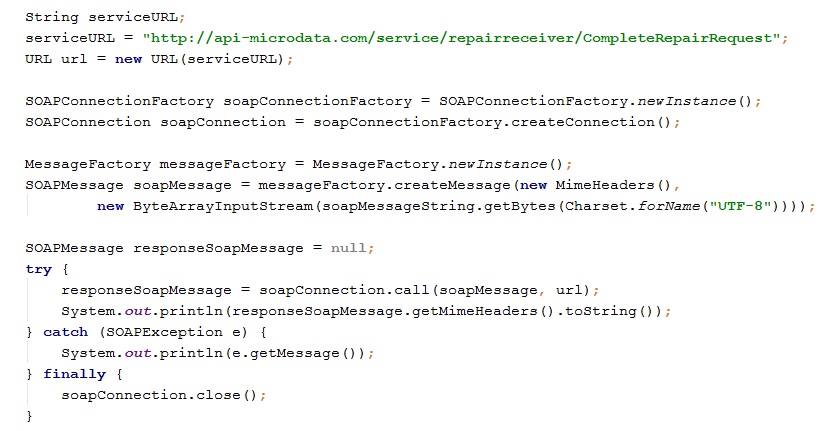
**รูปที่ 3.79** ส่วน header ของเมทอดจำลองการซ่อมไฟล์

จากรูป เมทอดนี้จะมีพารามิเตอร์ 1 ตัวคือ soapMessageString ซึ่งมีประเภทเป็น String โดยเป็นตัวแปรที่เอาไว้เก็บข้อความที่เป็น SOAP message ที่เชื่อมโยงมาจากด้านล่างของ step โดยตัวอย่างข้อความ SOAP message ที่ step จะแสดงดังข้อความในกรอบสีแดง ในรูปที่ 3.80



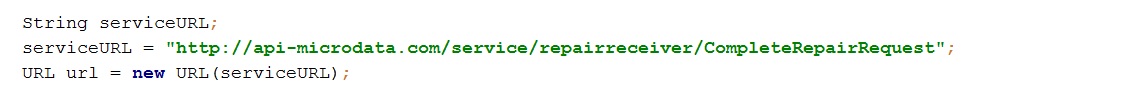
**รูปที่ 3.80** ตัวอย่างข้อความ SOAP message ในการซ่อมข้อมูล

โดยหลังจากเชื่อมเอาข้อความ SOAP message มาเก็บไว้ที่พารามิเตอร์ในเมทอดแล้ว. สำหรับขั้นตอนในการทำงานของเมทอดการส่งข้อมูล SOAP message ไปให้กับ service ของระบบ MicroData เพื่อทำการซ่อมข้อมูล โดยใช้ SOAP protocol นั้นจะมีการทำงานดังรูปที่ 3.81



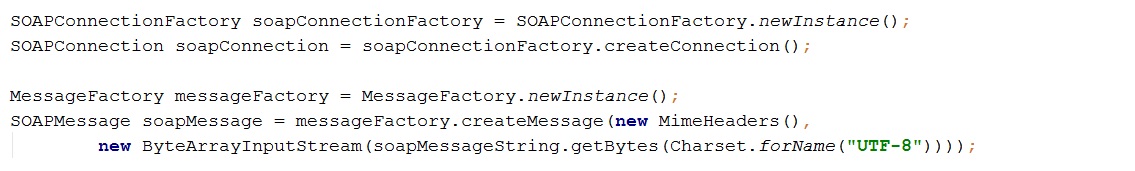
**รูปที่ 3.81** ตัวอย่างข้อความ SOAP message ในการซ่อมข้อมูล

โดยการอธิบายการทำงานของเมทอดจะแบ่งออกซอร์สโค้ดออกเป็นส่วนๆเพื่อให้ง่ายต่อการทำความเข้าใจ สำหรับส่วนแรกนั้น จะทำหน้าที่ในการสร้าง object ของ Class URL เพื่อนำไปใช้ในการส่งข้อความด้วย SOAP service. โดย URL นี้เป็นที่อยู่ของ SOAP service ของระบบ MicroData การทำงานส่วนนี้ทำงานด้วยคำสั่งในรูปที่ 3.82



**รูปที่ 3.82** ซอร์สโค้ดการสร้าง URL ของ SOAP service

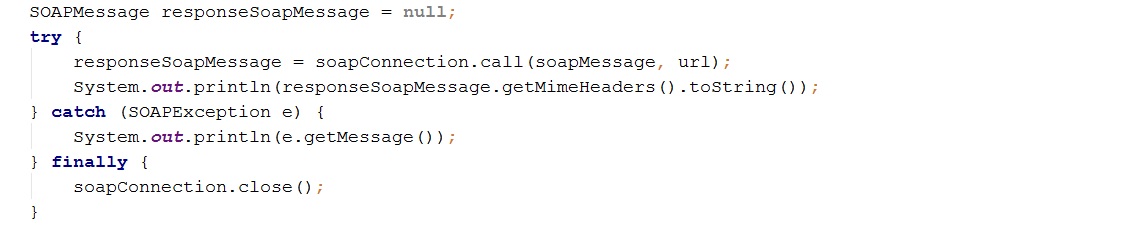
เมื่อทำการสร้าง object ของ URL ไว้แล้ว ส่วนต่อไปคือการสร้าง object ต่างๆและทำการตั้งค่า สำหรับ Class ที่อยู่ใน library javax.xml.soap ซึ่งเป็น built-in library ของ Java เองอยู่แล้ว. โดยการทำงานจะแสดงดังรูปที่ 3.83



**รูปที่ 3.82** ซอร์สโค้ดการสร้าง SOAP message

จากรูปด้านบน ซอร์สโค้ดบรรทัดแรกสุดคือ ขั้นตอนการสร้าง object ของ SOAPConnectionFactory Class โดยใช้ static เมทอดของ SOAPConnecionFactory ด้วยเมทอด newInstance() จากนั้นเราจะนำ object ที่ได้ไปทำการสร้าง connection ด้วยการเรียกเมทอด createConnection() แล้วเก็บเอาไว้ในตัวแปร soapConnection หลังจากนั้นในส่วนต่อมาจะทำการสร้าง object messageFactory ขึ้นมาซึ่งทำหน้าที่เป็น เครื่องมือในการสร้าง SOAP message ที่เราต้องการ โดยการสร้าง SOAP message นั้นเราจะใช้ เมทอดของ messageFactory ชื่อ createMessage ซึ่งเราจะทำการส่งอาร์กิวเมนต์เข้าไปเป็น InputStream ซึ่ง InputStream ที่เราจะส่งเข้าไปนั้น เราจะทำการเปิด InputStream ของ String soapMessageString จาก step ของเรา.

หลังจากได้ soapMessage มาแล้วนั้น ขั้นตอนต่อไปก็คือขั้นตอนการส่ง message ไปที่ URL ที่อยู่ของ service ของระบบ MicroData โดยซอร์สโค้ดจะมีขั้นตอนการทำงานดังรูปที่ 3.83



**รูปที่ 3.83** ซอร์สโค้ดการส่ง SOAP message

โดยการทำงานจะอยู่ในช่วง try-catch-finally เช่นเดียวกันกับการส่งข้อมูลด้วย Http ในขั้นตอนก่อนหน้า. เนื่องจากว่าในการส่ง message ด้วย SOAP นั้นมีโอกาศที่โปรแกรมจะ throw SOAPException ได้. โดยการส่ง message นั้นจะทำงานโดยการเรียกเมทอด call() ของ object soapConnection โดยรับพารามิเตอร์ 2 ตัวคือ soapMessage และ url ของ service ซึ่งหลังจากทำการดำเนินการเมทอดนี้แล้วนั้นจะเป็นการส่ง SOAP message ไปที่ url ที่กำหนด และเมทอดจะคืนค่ากลับมาเป็น SOAP message เช่นเดียวกัน แต่จะเป็น SOAP message ซึ่งแสดงผลของการส่งข้อความว่าสำเร็จหรือไม่ อย่างไร. เมื่อได้ response message กลับมา ก้จะทำการแสดงผลออกมาให้กับผู้ใช้. และเมื่อทำงานจบไม่ว่าจะสำเร็จหรือไม่ ก็จะเข้าสู่ finally และทำการปิด soap connection ด้วยการเรียกคำสั่ง soapConnection.close()

1. **Step การตรวจสอบสถานะของไฟล์**

Step การตรวจสอบสถานะของไฟล์นี้เป็นหนึ่งใน step ที่ใช้ในการตรวจสอบความถูกต้องในการทำงานของระบบ ด้วยการตรวจสอบจากข้อมูลจริงในฐานข้อมูล. โดยในส่วน header ของ step นี้จะแสดงดังรูปที่ 3.84



**รูปที่ 3.84** ส่วน header ของเมทอดการตรวจสอบสถานะ

โดยพารามิเตอร์ที่เชื่อมมานั้นจะใช้ชื่อว่า expectedStatus มีประเภทเป็น String ซึ่งพารามิเตอร์ตัวนี้เป็นพารามิเตอร์ที่ทำหน้าที่ในการเชื่อมเอาค่าของ สถานะที่คาดหวังจากใน feature เข้ามาสู่โปรแกรม. โดยข้อความใน feature ที่ถูกเชื่อมเข้ามาจะแสดงในกรอบสีแดงดังรูปที่ 3.85



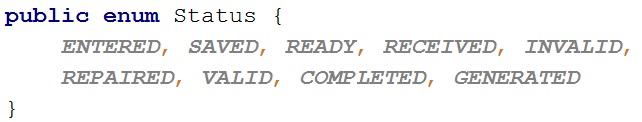
**รูปที่ 3.85** ข้อความแสดงสถานะที่ต้องการตรวจสอบใน feature

ต่อมาคือ ขั้นตอนในการตรวจสอบสถานะ. โดยการดึงข้อมูลจากฐานข้อมูลขึ้นมาตรวจสอบนั้น ซอฟต์แวร์ของเราจะใช้งาน library ที่ช่วยในการทำ ORM (Object relation mapping) คือ JPA (Java Persistence API) และ library ในการช่วยทำการเขียน query ในระดับสูงด้วย Spring Data library แทนการดึงข้อมูลจากฐานข้อมูลโดยตรงด้วยการเขียนคำสั่ง SQL (Structured Query Language) โดยตรง. ซึ่งใน step นี้ สิ่งที่เราสนใจในการตรวจสอบคือ การตรวจสอบสถานะของไฟล์ ดังนั้นเราจึงจำเป็นต้องการดึงข้อมูลของตาราง Image ขึ้นมาทำการตรวจสอบ. โดย Class ที่ทำหน้าที่เป็นตัวแทนของตารางในฐานข้อมูลจะถูกเรียกว่า Entity Class. โดย Entity Class จะถูกเขียนขึ้นโดยใช้ syntax ของ library JPA ในการเชื่อมโยงตาราง และค่าของแต่ละ field. สำหรับ Class Entity ของตาราง Image นั้นจะมีลักษณะดังรูปที่ 3.86



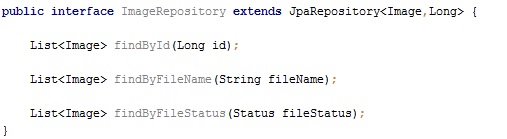
**รูปที่ 3.86** Image JPA Entity Class

จากรูปที่ 3.86 แสดงถึง Class ที่ทำหน้าที่เป็นตัวแทนของตารางชื่อว่า IMAGE ภายในฐานข้อมูล ซึ่ง object ของ class นี้จะใช้ในการเก็บข้อมูลของ 1 record ในตาราง IMAGE. โดยตาราง IMAGE นั้นจะมี field จำนวน 4 field คือ 1.IMAGE\_ID เป็นคีย์หลัก 2.FILE\_NAME คือชื่อของไฟล์ที่ถูกอัพโหลดขึ้นสู่ระบบ 3.CREATED\_TIME คือเวลาที่ถูกสร้าง และ 4.FILE\_STATUS คือสถานะของไฟล์ ณ ปัจจุบัน ซึ่งทั้งหมดถูกเชื่อมโยงกับ class Image ด้วย attribute ใน class โดยใช้ annotation @Column. โดย field สถานะนั้นจะมีประเภทของข้อมูลเป็น Status ซึ่งเป็น Class Enumerable ที่ได้ทำการสร้างเอาไว้. Class Enumerable มีไว้สำหรับการกำหนดค่าของข้อมูลที่จำกัด ไม่เปลี่ยนแปลง ซึ่งในที่นี้ คือ Status หรือสถานะต่างๆของไฟล์ที่เป็นไปได้ในระบบ โดย Class Status จะแสดงดังรูปที่ 3.87



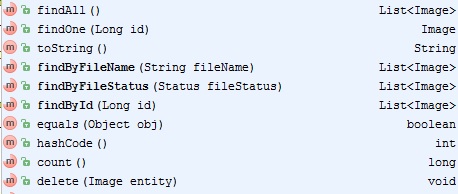
**รูปที่ 3.87** Status Enumerable Class

ซึ่ง Enumerable Status จะทำการเก็บค่าของสถานะต่างๆที่เป็นไปได้ทั้งหมดในระบบ และจะช่วยให้ไม่มีการใส่สถานะที่นอกเหนือจากที่มีในระบบลงในฐานข้อมูล. เมื่อเรามี Class ที่เป็นตัวแทนของตาราง IMAGE แล้วขั้นตอนต่อไปคือ การดึงข้อมูลของตารางขึ้นมาจากฐานข้อมูล. ซึ่งการทำการดึงข้อมูลขึ้นมานั้น เราจะใช้ API ของ library Spring Data เป็นตัวช่วยในการทำงานแทนการเขียนคำสั่ง SQL โดยตรง. โดยเราจะต้องทำการสร้าง class repository สำหรับ class Image ขึ้นมา. ซึ่ง class repository นั้นเป็น class ที่ทำหน้าที่ช่วยในการเตรียม API สำหรับการจัดการต่างๆกับข้อมูล. Class repository ของ Class Image จะมีลักษณะดังรูปที่ 3.88



**รูปที่ 3.88** Image Spring Data repository class

โดย Class ImageRepository จะเป็น interface class ซึ่ง extends interface ชื่อว่า JpaRepository<Image,Long> โดยภายในเครื่องหมายกำหนด generic นั้น คือชื่อ Class Entity และประเภทข้อมูลของคีย์หลักของตาราง. โดยเมทอดที่ประกาศเอาไว้ใน repository class นั้นจะถูก Spring Data แปลงเป็นคำสั่ง SQL ในเบื้องหลังโดยอัตโนมัติ โดยใช้หลักการการตั้งชื่อเมทอดคือ คำว่า findBy เทียบเท่ากับคำสั่ง where ส่วนประโยคหลังจาก findBy จะเป็นชื่อของ field ของตารางเช่น เมทอด findById ซึ่งรับพารามิเตอร์เป็นประเภท Long เก็บไว้ในตัวแปร id นั้น จะมีค่าเทียบเท่ากับการพิมพ์คำสั่ง SQL ว่า select \* from IMAGE where Image.id = id. ซึ่งเมทอดจะคืนค่าเป็น List ของ object Image ซึ่งคือ ทุก record ที่ได้มาจากการ query คำสั่งข้างต้น. ส่วนเมทอดอื่นๆ ก็จะเป็นการ query จาก field FILE\_NAME และ FILE\_STATUS. นอกจากเมทอดที่ประกาศเอาไว้ใน class repository ที่สร้างขึ้นนั้น class ImageRepository ยังสืบทอดเมทอดพื้นฐานอื่นๆจาก interface JpaRepository มาอีกด้วย. ซึ่งเมทอดพื้นฐานที่ทาง Spring Data ได้จัดเตรียมไว้ให้นั้น จะเป็นเมทอดทั่วไปที่ใช้ในการจัดการกับข้อมูล เช่น เมทอดการลบข้อมูล การค้นหาทั้งหมด โดยตัวอย่างจะแสดงดังรูปที่ 3.89



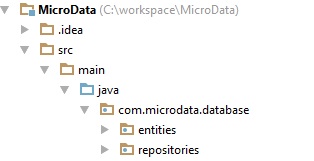
**รูปที่ 3.89** ตัวอย่างเมทอดพื้นฐานในการจัดการข้อมูล

จากรูป เช่น เมทอด findAll() ซึ่งจะคืนค่าเป็นทุก record ในตาราง Image หรือเมทอด count() ที่ใช้นับจำนวน record ภายในตาราง Image เป็นต้น. โดยในการตรวจสอบนั้นเราจะใช้งาน repository นี้ในการช่วยดึงข้อมูลจากฐานข้อมูลขึ้นมาตรวจสอบ. โดยการทำงานของ step นี้จะเริ่มจากการนำเอา object ของ ImageRepository class เข้ามาในโปรแกรม. เนื่องจาก ImageRepository เป็น interface เราจึงไม่สามารถ สร้าง object ขึ้นมาตรงๆได้ เราจึงต้องใช้ Annotation ชื่อว่า @Autowired ของ Spring framework ซึ่งใช้ช่วยในการทำ dependency injection ให้ทำการ inject object ของ interface ImageRepository เข้ามาในโปรแกรมโดยไม่ต้องทำการสร้าง Object ขึ้นมาเอง โดย Spring จะเป็นตัวจัดการในการสร้าง object และส่ง object เข้ามาให้. โดยการที่จะทำให้ Spring framework สร้าง object ของ repository ของเราให้ได้นั้น จำเป็นที่จะต้องมีการตั้งค่าด้วยการเพิ่ม tag พิเศษเพิ่มเติมลงไปในไฟล์ application-config.xml ในโปรเจค. โดย application-config.xml เป็นไฟล์ที่ถูกสร้างให้โดย default เมื่อทำการสร้างโปรเจคสำหรับ Spring framework. ซึ่งไฟล์นี้ใช้ช่วยในการตั้งค่าโปรเจคของ Spring framework โดย tagที่ต้องเพิ่มลงไปเพื่อให้ Spring framework สร้าง object ของ repository ให้นั้นคือ tag ในรูปที่ 3.90



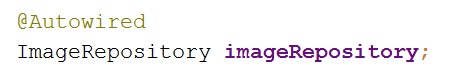
**รูปที่ 3.90** ตัวอย่างเมทอดพื้นฐานในการจัดการข้อมูล

โดยความหมายของ tag jpa:repositories คือ เป็นการบอกให้ Spring เข้าไปทำการค้นหาทุก Class ที่เป็น sub package ของ package com.microdata.database. ที่มี Annotation ที่เกี่ยวข้องกับ Spring data และ JPA เช่น @Entity หรือ class repository ต่างๆ เพื่อทำการสร้าง object ให้กับ Class เหล่านั้นแล้วเก็บเอาไว้ใน context ของระบบเพื่อที่จะสามารถ Inject เข้าไปให้กับส่วนต่างๆของโปรแกรม. โดยภายใต้ package com.microdata.database จะมี package ต่างๆที่ทำหน้าที่รวบรวมทั้ง entity และ repository class ต่างๆสำหรับระบบ ดังรูปที่ 3.91



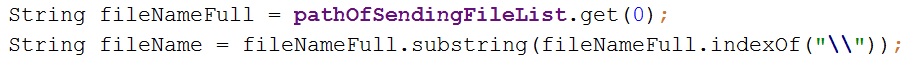
**รูปที่ 3.91** package com.microdata.database

โดย Spring จะทำการเชื่อมโยง Class ที่ Annotate ด้วย Annotation Entity เข้ากับตารางจริงในฐานข้อมูลซึ่งต้องทำการเชื่อมต่อกับ datasource เอาไว้ก่อน เช่น Class Image เป็นต้น และทำการสร้าง object ให้กับ imageRepository ด้วยเช่นกัน. หลังจากทำการสร้าง object ต่างๆเอาไว้ภายใน context แล้วก็จะทำการ inject ค่าของ imageRepository. โดยตัวแปรที่รับค่าที่ inject เข้ามาจะแสดงดังรูปที่ 3.92



**รูปที่ 3.92** ตัวแปร imageRepository

หลังจากได้ตัวแปร imageRepository มาแล้วเราจะทำการ query เพื่อหา record ที่เป็นตัวแทนของไฟล์ของเราที่ต้องการตรวจสอบจากตาราง Image. โดยวิธีการ query นั้นเราจะทำการค้นหาด้วยการนำเอาชื่อไฟล์ที่เราอัพโหลดขึ้นสู่ระบบในการค้นหา.โดยวิธีการค้นหาจะเริ่มจากรูปที่ 3.93



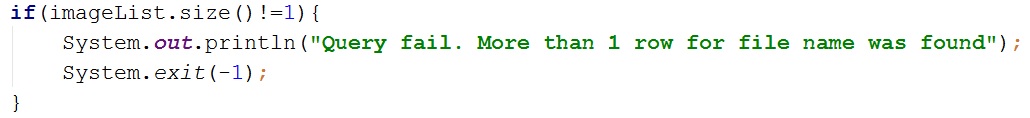
**รูปที่ 3.93** ซอร์สโค้ดการดึงชื่อไฟล์

ซอร์สโค้ดในส่วนแรกจะเริ่มจากการดึงเอาชื่อไฟล์เต็มๆจาก ชื่อ path ที่ใช้ในการอัพโหลดไฟล์ออกมาเก็บไว้ในตัวแปร fileNameFull หลังจากนั้นในบรรทัดที่ 2 จะเป็นการตัดเอาเฉพาะชื่อไฟล์จริงๆออกมาจากตัวแปร fileNameFull เนื่องจากว่าตัวแปร จะทำการเก็บที่อยู่ของ path ด้วย. เราจึงตัดเอาเฉพาะข้อความหลังเครื่องหมาย // ด้วยการใช้เมทอด subString() ตั้งแต่ตำแหน่งของ // ซึ่งหาได้จากเมทอด indexOf() ซึ่งค่าที่ได้จะเป็นชื่อไฟล์จริงๆที่ไม่มี path ปนอยู่ด้วย จากนั้นจึงนำไปเก็บไว้ในตัวแปร String fileName. เมื่อได้ชื่อไฟล์มาแล้วขั้นตอนต่อไปจะแสดงดังรูปที่ 3.94



**รูปที่ 3.94** ซอร์สโค้ดการหา record Image จากชื่อไฟล์

โดยคำสั่งในรูปที่ 3.94 นั้นเป็นคำสั่งที่ใช้ในการค้นหา record ของ Image ที่มีชื่อ file ตรงกับตัวแปร fileName. ซึ่งคำสั่งนี้จะคืนค่ากลับมาเป็น List ของ object ของ Class Image ซึ่ง object ทุกตัวภายใน List คือ object ของ Class Image ที่ field FILE\_NAME มีค่าเท่ากับชื่อไฟล์ที่ทำการอัพโหลด. ต่อมาเมื่อได้ List ของ record มาแล้วก็จะทำการตรวจสอบ List ดังกล่าวดังรูปที่ 3.95



**รูปที่ 3.95** ซอร์สโค้ดการตรวจสอบ List ของ Image

โดยซอร์สโค้ดในส่วนนี้นั้นจะเป็นการตรวจสอบว่า List ที่ได้รับมานั้นมีขนาดไม่เท่ากับ 1 หรือไม่ เนื่องจากว่า หากการค้นข้อมูลดำเนินไปอย่างถูกต้อง record ที่ได้รับกลับมาจะต้องมีเพียงแค่ record เดียว เนื่องจากชื่อไฟล์จะไม่มีการอนุญาติให้ซ้ำกัน. หากขนาดของ List ไม่เท่ากับ 1 จะแสดงข้อความออกมาว่าการค้นหาข้อมูลผิดพลาดและ ทำการออกจากโปรแกรมด้วยรหัส -1. ส่วนถ้าหาก List มีขนาดเป็น 1 ถูกต้องก็จะทำงานต่อไปดังรูปที่ 3.96



**รูปที่ 3.96** ซอร์สโค้ดการเก็บ object Image จาก List

โดยซอร์สโค้ดบรรทัดนี้จะทำการดึงเอา record ที่ได้รับออกมาเก็บไว้ในตัวแปรชื่อชื่อว่า retrievedImage ซึ่งมีประเภทเป็น Image. หลังจากเราได้ object ซึ่งเป็นตัวแทนของไฟล์ของเราแล้วนั้น ขั้นตอนสุดท้ายของ step คือการตรวจสอบว่าสถานะของไฟล์นั้นตรงกับที่เราต้องการหรือไม่ โดยการทำงานจะแสดงดังรูปที่ 3.97



**รูปที่ 3.97** ซอร์สโค้ดการตรวสอบสถานะของไฟล์

โดยการตรวจสอบนั้นจะใช้เมทอด assertEquals() ซึ่งใช้ในการตรวจสอบว่า object 2 ตัวมีค่าเท่ากันหรือไม่ของ Class Assert. โดยเมทอด assertEquals() นั้นรับพารามิเตอร์ 3 ตัวคือ 1. เป็นข้อความที่จะแสดงออกมาเมื่อการตรวจสอบมีข้อผิดพลาด object 2 ตัวมีค่าไม่เท่ากัน 2. เป็น object ที่เราคาดหวัง ในที่นี้ก็คือ สถานะของไฟล์ที่เราต้องการตรวจสอบ ส่วนพารามิเตอร์ตัวสุดท้ายคือ object ที่เราต้องการตรวจสอบ ซึ่งเราจะทำการดึงเอาค่าของ สถานะของไฟล์ออกมาจาก object image ที่เราหามาได้ ซึ่งจะได้ออกมาเป็น object ของ Class enumerable Status เราจึงจำเป็นต้องเรียกใช้เมทอด name() เพื่อแปลงเป็น String. หากตรวจสอบพบว่าสถานะเท่ากัน ก็จะผ่านการตรวจสอบ หากไม่ผ่านระบบจะแสดงข้อความที่เตรียมเอาไว้ และหยุดการตรวจสอบลงพร้อมบอกว่าการตรวจสอบในส่วนนี้ผิดพลาด.

1. **Step การตรวจสอบประวัติสถานะของไฟล์**

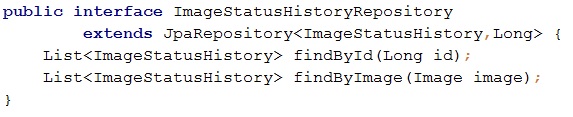
สำหรับ step นี้จะเป็น step ที่ทำการตรวจสอบต่อจากการตรวจสอบสถานะของไฟล์ก่อนหน้า โดยใน step นี้การตรวจสอบจะตรวจสอบตาราง IMAGE\_STATUS\_HISTORY ซึ่งทำหน้าที่ในการเก็บประวัติว่า ไฟล์ของเราตั้งแต่ถูกอัพโหลดเข้ามาสู่ระบบ MicroData นั้นผ่านการทำงานกับโปรเซสไหนมาบ้าง ซึ่งจะช่วยทำให้เรารู้ได้ว่า ถ้ามีข้อผิดพลาดเกิดขึ้นในระบบ ข้อผิดพลาดนั้นเกิดขึ้นที่โปรเซสใด เนื่องจากสถานะของไฟล์จะมีจุดที่ผิดปกติเกิดขึ้น จากประวัติสถานะของไฟล์ในกรณีที่ระบบทำงานถูกต้องทุกอย่างตามผังงาน โดย Class Entity ที่ทำหน้าที่เชื่อมสำหรับตาราง IMAGE\_STATUS\_HISTORY นั้นจะอยู่ใน package com.microdata.database.entities ที่เดียวกันกับ Class Image โดยจะมีลักษณะดังรูปที่ 3.98



**รูปที่ 3.98** ImageStatusHistory JPA Entity Class

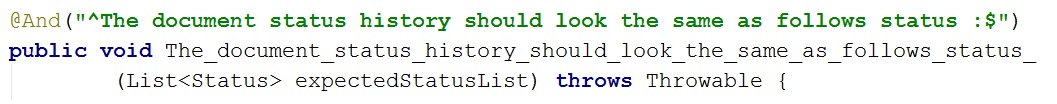
โดยตาราง IMAGE\_STATUS\_HISTORY ประกอบไปด้วยทั้งหมด 5 field คือ 1.IMAGE\_STATUS\_HISTORY\_ID คือคีย์หลักของตาราง 2. CREATED\_TIME คือเวลาที่ถูกสร้าง 3. FILE\_STATUS คือสถานะของไฟล์ที่ถูกบันทึก 4. MESSAGE\_TX คือ field ที่ใช้เก็บข้อความที่ถูกระบบบันทึกขณะ กำลังเปลี่ยนสถานะของไฟล์ และ 5.คือ field ที่เอาไว้เก็บคีย์นอกของตาราง IMAGE โดยใน JPA จะสามารถเชื่อมคีย์นอกได้ในรูปของ Object โดยใช้ Annotation @ManyToOne สำหรับความสัมพันธ์แบบ many to one โดยประเภทข้อมูลของ field นี้จะต้องเป็น Class ที่เป็นตัวแทนของตารางหลักของคีย์นอก ในที่นี้คือตาราง IMAGE. โดยจากการที่เชื่อมโยงกันแบบ many to one นั้นหมายความว่า record ของ IMAGE 1 record หรือ image 1 image นั้นมี image status history ได้หลาย record.

ส่วน repository class ของ class ImageStatusHistory นั้นจะอยู่ใต้ package com.microdata.database.repositories เช่นเดียวกันกับ ImageRepositories โดยจะมีลักษณะดังรูปที่ 3.99



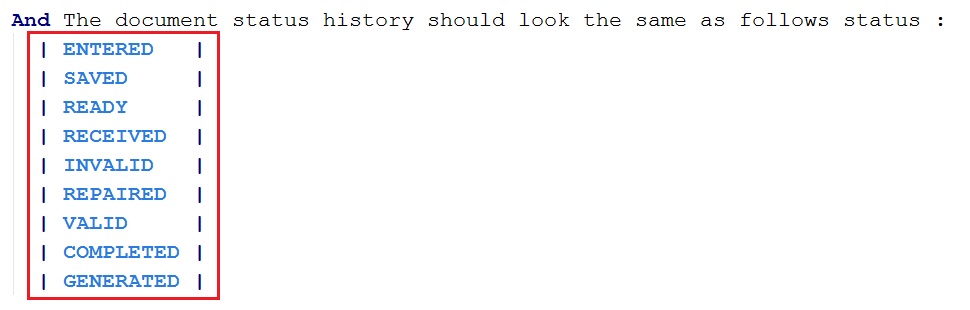
**รูปที่ 3.99** ImageStatusHistory Spring Data repository class

โดยเมทอดที่มีความสำคัญใน interface นี้คือ findByImage ซึ่งเป็นเมทอดที่จะทำการคืนค่าออกมาเป็น List ของ ImageStatushistory ที่มีคีย์นอกตรงกับ record Image ที่ถูกส่งเข้ามาทาง parameter ซึ่งเราจะใช้เมทอดนี้เป็นหลักในการดึงข้อมูลขึ้นมาตรวจสอบ. หลังจากประกาศ Class ต่างๆเกี่ยวกับฐานข้อมูลไปแล้ว ต่อไปในส่วนของ header ของ step นี้ จะแสดงดังรูปที่ 3.100

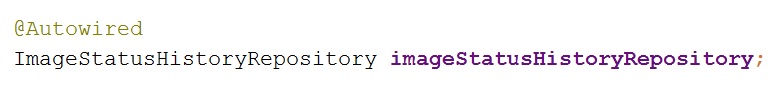


**รูปที่ 3.100** header ของเมทอดการตรวจสอบประวัติสถานะ

สำหรับ step นี้จะรับพารามิเตอร์ 1 ตัวชื่อว่า expectedStatusList ซึ่งมีประเภทข้อมูลเป็น List ของ object จาก Class Status โดยเป็นพารามิเตอร์ที่เชื่อมโยงกับข้อมูลจากตารางภายใน feature ซึ่งข้อมูลในตารางจะแสดงในกรอบสีแดงดังรูปที่ 3.101

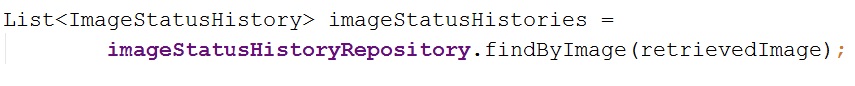
 **รูปที่ 3.101** ข้อมูลในตารางของ step การตรวจสอบประวัติสถานะ

โดยข้อมูลในตารางนั้นจะเป็นข้อความสถานะ ซึ่งสถานะทั้งหมดที่ใส่ไว้ในตารางจะต้องมีข้อมูลอยู่ใน Enumerable Status เนื่องจากเมื่อโปรแกรมทำการเชื่อมโยงข้อมูล เข้าไปสู่พารามิเตอร์ซึ่งรับเป็น List ของ Status หากข้อความในตารางไม่สามารถเชื่อมกับค่าที่มีใน Enumberable Status ได้จะเกิด error ขึ้นในระหว่างการเชื่อมโยง. หลังจากได้ข้อมูลสถานะที่ต้องการทั้งหมดเข้ามาสู่เมทอดแล้วนั้น ต่อไปจะเป็นขั้นตอนในการตรวจสอบประวัติของสถานะ โดยในส่วนแรกนั้นจะทำการค้นหา Object ของ Image ด้วยวิธีการเดียวกันกับรูปที่ 3.93 – 3.96 โดยจะได้ Object ของ Class Image ที่เป็นไฟล์ที่เราต้องการจะตรวจสอบมา หลังจากนั้นเราจะทำการตรวจสอบด้วยการใช้ เมทอดของ interface ImageStatusHistoryRepository ดังนั้นเราจึงต้องทำการ inject object เข้ามา ดังรูปที่ 3.102



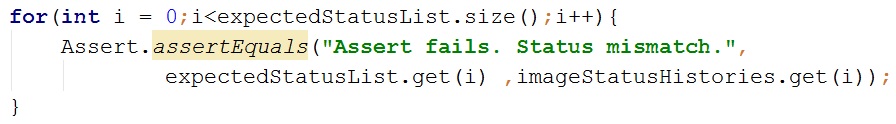
**รูปที่ 3.102** ตัวแปร imageStatusHistoryRepository

หลังจากได้ object ของ imageStatusHistoryRepository แล้วก็จะทำการใช้เม-ทอด findByImage() ที่ได้ทำการประกาศเอาไว้ใน interface ในรูปที่ 3.99 เพื่อทำการดึงเอาประวัติสถานะทั้งหมดของ Image ที่ต้องการตรวจสอบขึ้นมา ดังรูปที่ 3.103



**รูปที่ 3.103** ซอร์สโค้ดการดึงประวัติสถานะจาก Image ที่จะตรวจสอบ

โดยเมทอดนี้จะคืนค่าออกมาเป็น List ของ object ที่แสดงถึง record ในตาราง IMAGE\_STATUS\_HISTORY โดยเรียงตามลำดับการถูกบันทึก ก่อน-หลัง นับตั้งแต่ไฟล์ถูกอัพโหลดสู่ระบบ และผ่านโปรเซสต่างๆ จะไม่มีการสลับสถานะเกิดขึ้นในการดึงขึ้น record ทั้งหมดขึ้นมาด้วยเมทอด findByImage(). ซึ่งเราจะใช้ประโยชน์จากการที่ record ที่ถูกดึงขึ้นมาเป็นแบบเรียงตามลำดับในการตรวจสอบความถูกต้องของลำดับสถานะที่ควรจะเกิด. โดยการตรวจสอบนั้นจะใช้วิธีดังรูปที่ 3.104



**รูปที่ 3.104** ซอร์สโค้ดการตรวจสอบประวัติของสถานะ

โดยวิธีการทำการตรวจสอบของซอร์สโค้ดชุดนี้นั้นจะเริ่มจากการวนลูปด้วย ลูปแบบ for-loop โดยจำนวนรอบเท่ากับขนาดของสถานะที่ต้องการตรวจสอบ ซึ่งเป็นการช่วยยืนยันในเบื้องต้นด้วย เพราะว่า ขนาดของ List ประวัติสถานะที่ได้จากการค้นจากฐานข้อมูล จะต้องมีขนาดเท่ากับ List ของประวัติสถานะที่เรากำหนดไว้ที่ feature. โดยการตรวจสอบนั้นจะใช้เมทอด assertEquals() ซึ่งใช้ในการตรวจสอบว่า object 2 ตัวมีค่าเท่ากันหรือไม่ของ Class Assert เช่นเดียวกันกับ step ก่อนหน้า. โดยการตรวจสอบจะตรวจสอบกันที่ตำแหน่งต่อตำแหน่งเลยว่า ประวัติสถานะภายใน List ทั้ง 2 จะต้องมีค่าเดียวกันในทุกตำแหน่ง ถ้ามีตำแหน่งใดที่ไม่ถูกต้อง หมายความว่า ไฟล์ที่ถูกอัพโหลดไม่ได้ทำงานตามผังงานของระบบที่กำหนดไว้อย่างถูกต้อง ซอฟต์แวร์จะหยุดทำงาน และแสดงข้อความว่า status mismatch เพื่อบอกผู้ใช้ว่า สถานะไม่ตรงกับที่คาดหวัง และผู้ใช้จะรู้ได้ว่าระบบมีข้อผิดพลาดที่จุดใด.

1. **Step การตรวจสอบไฟล์ผลลัพธ์ในโฟลเดอร์ที่กำหนด**

สำหรับใน Step สุดท้ายนี้เป็น step ที่ทำงานในการตรวจสอบว่า สำหรับไฟล์ที่ถูกอัพโหลดสู่ระบบ และผ่านการทำงานจากโปรเซสทุกๆโปรเซสอย่างถูกต้องเรียบร้อยแล้ว ซึ่งจะอยู่ในสถานะ “PUBLISHED” นั้น ไฟล์ดังกล่าวจะต้องถูกสร้างไฟล์ผลลัพธ์ที่ได้จากการแปลงไฟล์เป็นข้อมูลอิเล็กทรอนิกอย่างสมบูรณ์. โดย step จะเชื่อมต่อไปสู่เซิฟเวอร์ของระบบ และจากนั้นทำการเข้าไปตรวจสอบภายในโฟลเดอร์ที่กำหนด ซึ่งเป็นโฟลเดอร์ที่เอาไว้เก็บไฟล์ผลลัพธ์ของลูกค้ารายที่เราได้ทำการอัพโหลดไฟล์ขึ้นไป.แล้วทำการตรวจสอบ โดยนำเอาชื่อไฟล์ที่กำหนดเอาไว้ภายในตารางของ feature file เข้าไปค้นหาในโฟลเดอร์ที่เก็บไฟล์ผลลัพธ์ เพื่อดูว่า มีไฟล์ทั้งหมดที่ควรจะถูกสร้างอยู่จริงภายในโฟลเดอร์หรือไม่. โดยในส่วนของ header ของ เมทอดของ step นี้จะแสดงดังรูปที่ 3.105