ระบบติดตามการขนส่งพร้อมการวางแผนเส้นทางอัจฉริยะ

SMART LOGISTICS AND TRACKING SYSTEM WITH ROUTE PLANNING

โดย

นายรัชภัทร เศรษฐธรกุล

นายรามพัฒน์ ถนัดทาง

นายวโรดม ภูมิประเสริฐ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2567

ระบบติดตามการขนส่งพร้อมการวางแผนเส้นทางอัจฉริยะ

SMART LOGISTICS AND TRACKING SYSTEM WITH ROUTE PLANNING

โดย

นายรัชภัทร เศรษฐธรกุล 64011253

นายรามพัฒน์ ถนัดทาง 64011257

นายวโรดม ภูมิประเสริฐ 64011270

อาจารย์ที่ปรึกษา

ผศ.ดร.กฤษณ์ วงรุจิระ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2567

ปริญญานิพนธ์ปีการศึกษา 2567

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบติดตามการขนส่งพร้อมการวางแผนเส้นทางอัจฉริยะ

SMART LOGISTICS AND TRACKING SYSTEM WITH ROUTE PLANNING

ผู้จัดทำ

1. นายรัชภัทร เศรษฐธรกุล 64011253
2. นายรามพัฒน์ ถนัดทาง 64011257
3. นายวโรดม ภูมิประเสริฐ 64011270

..... **อาจารย์ที่ปรึกษา**

(ผศ.ดร.กฤษณ์ วงรุจิระ)

กิตติกรรมประกาศ

การดำเนินปริญญานิพนธ์เรื่อง “ระบบติดตามการขนส่งพร้อมการวางแผนเส้นทางอัจฉริยะ” สามารถดำเนินการจนสำเร็จลุล่วงไปได้ด้วยดี ด้วยความกรุณา และความช่วยเหลือจากอาจารย์ที่ปรึกษา และผู้ที่เกี่ยวข้องในส่วนต่าง ๆ ทางผู้จัดทำขอขอบคุณ ผศ.ดร.กฤษณ์ วงรุจิระ ที่ได้กรุณาให้คำแนะนำ ความรู้ ชี้แนะแนวคิดและนำแนวทางที่ดีที่เป็นประโยชน์ต่อการแก้ไขปัญหาในการทำงาน รวมถึงสอดแทรกทักษะประสบการณ์ต่าง ๆ ให้กับผู้จัดทำ ทางผู้จัดทำขอกราบขอบพระคุณเป็นอย่างสูงไว้ ณ โอกาศนี้

ขอขอบคุณคณาจารย์และเจ้าหน้าที่ประจำภาควิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังทุกท่าน ที่ได้อบรมสั่งสอน ประสิทธิ์ประสาทวิชา ความรู้ และประบการณ์ให้แก่ผู้จัดทำ

สุดท้ายนี้ขอกราบขอบพระคุณบิดา มารดา และครอบครัวที่ให้การสนับสนุน ความรัก ความห่วงใย รรมถึงกำลังใจที่สำคัญเสมอมา และที่สำคัญคือสนับสนุนให้โอกาสทางด้านการศึกษาอันมีค่ายิ่งแก่ผู้จัดทำ

นายรัชภัทร เศรษฐธรกุล
นายรามพัฒน์ ถนัดทาง
นายวโรดม ภูมิประเสริฐ
ผู้จัดทำ

ระบบติดตามการขนส่งพร้อมการวางแผนเส้นทางอัจฉริยะ
SMART LOGISTICS AND TRACKING SYSTEM WITH ROUTE PLANNING

โดย นายรัชภัทร เศรษฐธรกุล 64011253
นายรามพัฒน์ ถนัดทาง 64011257
นายวโรดม ภูมิประเสริฐ 64011270

อาจารย์ที่ปรึกษา ผศ.ดร.กฤษณ์ วงรุจิระ

บทคัดย่อ

ปริญญานิพนธ์นี้นำเสนอการพัฒนาระบบติดตามการขนส่งด้วย IoT โดยใช้เซ็นเซอร์วัดอุณหภูมิ ความชื้น และโมดูล GPS ในการเก็บข้อมูลสิ่งแวดล้อมระหว่างการขนส่ง ข้อมูลถูกส่งไปยังฐานข้อมูลทุกๆ 3 นาทีผ่านโครงข่าย LTE ใช้พลังงานจากแบตเตอรี่ 18650 นอกจากนี้ ระบบยังได้นำเทคนิค K-means Clustering มาช่วยในการแบ่งกลุ่มพิกัด และประยุกต์ใช้ OSRM (Open Source Routing Machine) ในการคำนวณเส้นทางที่เหมาะสมที่สุดผ่านการสร้างตารางระยะทาง (Distance Matrix) เพื่อให้ได้ลำดับเส้นทางที่เหมาะสม

ABSTRACT

This thesis presents the development of a transportation tracking system using IoT technology, incorporating temperature and humidity sensors, and a GPS module to collect environmental data during transport. Data is transmitted to a database every 3 minutes via an LTE network and is powered by a 18650 battery. Additionally, the system employs K-means Clustering to assist in grouping coordinates and utilizes OSRM (Open Source Routing Machine) to calculate the optimal route by generating a Distance Matrix, enabling efficient route sequencing.

สารบัญ

สารบัญ (ต่อ)

สารบัญรูป

หน้า

รูปที่

สารบัญรูป (ต่อ)

หน้า

รูปที่

สารบัญรูป (ต่อ)

หน้า

รูปที่

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

หน้า

รูปที่

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

หน้า

รูปที่

สารบัญตาราง

หน้า

ตารางที่

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันระบบการขนส่งมีบทบาทสำคัญในการเชื่อมโยงผู้ผลิตกับผู้บริโภค โดยเฉพาะในอุตสาหกรรมที่ต้องควบคุมอุณหภูมิในการขนส่ง เช่น อาหารแช่แข็งหรือสินค้าที่ต้องการการรักษาคุณภาพ ซึ่งในการขนส่งจำเป็นต้องมีการควบคุมอุณหภูมิและความชื้น เพื่อให้มั่นใจว่าสินค้าจะไม่เสียหายระหว่างการขนส่ง ปัญหาที่พบได้บ่อยคือผู้ประกอบการไม่สามารถดูข้อมูลได้แบบเรียลไทม์ นอกจากนี้การวางแผนเส้นทางในการขนส่งก็เป็นปัจจัยสำคัญ การเลือกเส้นทางที่ไม่เหมาะสมทำให้เกิดความล่าช้าและเพิ่มค่าใช้จ่ายในการขนส่ง ซึ่งเป็นปัญหาที่กระทบต่อการรักษาคุณภาพของสินค้าและต้นทุน

ปริญญานิพนธ์จัดทำขึ้นเพื่อให้ผู้ใช้งานสามารถตรวจสอบอุณหภูมิ ความชื้น และตำแหน่ง GPS ของรถขนส่งได้แบบเรียลไทม์ เพื่อให้สามารถปรับปรุงการขนส่งและลดความเสี่ยงในการเสียหายของสินค้า พร้อมทั้งวางแผนเส้นทางในการขนส่ง เพื่อเพิ่มความรวดเร็ว และลดต้นทุนในการขนส่ง

1.2 วัตถุประสงค์

1) เพื่อศึกษาวิธีการติดตามตำแหน่งผ่านการใช้งาน GPS
2) เพื่อศึกษาการสร้าง Database ในการเก็บข้อมูลที่ได้จาก NodeMCU
3) เพื่อศึกษาโครงสร้าง API เพื่อใช้ในการรับส่งข้อมูลระหว่าง Web Server
4) เพื่อศึกษาระบบ Root planner สำหรับการส่งพิกัดหลายจุด

1.3 ขอบเขตของปริญญานิพนธ์

ทำการออกแบบระบบติดตามการขนส่งและวางแผนเส้นทางอัจฉริยะสำหรับรถมอเตอร์ไซค์ขนส่งแบบเย็น โดยระบบจะใช้ Node MCU ในการรวบรวมข้อมูลจาก เซนเซอร์อุณหภูมิและความชื้น และ โมดูล GPS เพื่อส่งข้อมูลผ่านเครือข่าย LTE ไปยัง Database โดยใช้โปรโตคอล MQTT ซึ่งจะนำข้อมูลไปแสดงที่เว็บไซต์เพื่อให้ผู้ใช้งานสามารถตรวจสอบค่าอุณหภูมิ ความชื้น และตำแหน่ง GPS ของรถขนส่งแบบเรียลไทม์ และย้อนหลังได้ นอกจากนี้ ผู้ใช้ยังสามารถระบุพิกัดจุดส่งสินค้าผ่านหน้าเว็บ ซึ่งจะใช้ API จาก Google Maps ในการคำนวณและวางแผนเส้นทางที่สั้นที่สุดสำหรับการขนส่ง

บทที่ 2

ทฤษฎีและหลักการที่เกี่ยวข้อง

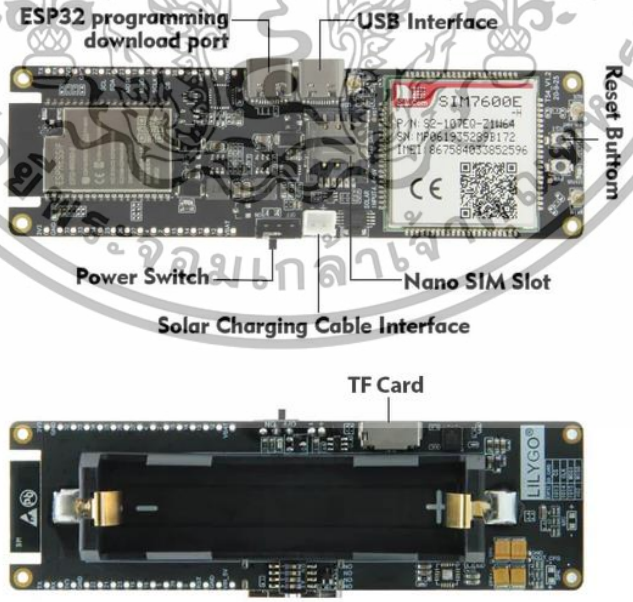
2.1 การทำงานของไมโครคอนโทรลเลอร์และโมดูลเซนเซอร์

2.1.1 บอร์ด Lilygo T-SIM7670E

ไมโครคอนโทรลเลอร์ LILYGO TTGO T-SIM-A7670E R2 เป็นอุปกรณ์ที่ล้ำสมัยซึ่งใช้ชิป ESP32 และผสานรวมกับโมดูล SIM-A7670E มันมาพร้อมกับความสามารถในการสื่อสารไร้สายที่หลากหลาย เช่น GNSS (GPS), Wi-Fi, Bluetooth และ LTE ทำให้เป็นเครื่องมือที่มีความหลากหลายและนวัตกรรม เหมาะสำหรับนักพัฒนาที่ต้องการความอเนกประสงค์และประสิทธิภาพ

ด้วยการเชื่อมต่อ USB-C สำหรับการเขียนโปรแกรมและที่ใส่แบตเตอรี่ 18650 ในตัวอุปกรณ์นี้พร้อมที่จะเป็นส่วนหนึ่งของโครงการต่าง ๆ อินเทอร์เฟซการสื่อสารที่กว้างขวางช่วยให้สามารถเชื่อมต่อและควบคุมอุปกรณ์ต่อพ่วงได้หลากหลาย ทำให้สามารถปรับใช้กับการใช้งานที่แตกต่างกันได้มากมายในด้านการสื่อสารไร้สาย

นอกจากนี้ มันยังมีประสิทธิภาพโดดเด่นด้วยการรองรับระบบ GNSS, Wi-Fi, Bluetooth และเครือข่ายโทรคมนาคม 2G และ 4G LTE CAT 1 ทำให้เป็นตัวเลือกที่สำคัญสำหรับแอปพลิเคชัน IoT และการส่งข้อมูลระยะไกล แสดงดังรูปที่ 2.1



รูปที่ 2.1 บอร์ด ESP32 Lilygo T-SIM7670E

ตารางที่ 2.1 ข้อมูลจำเพาะของบอร์ด Lilygo T-SIM7670E

ข้อมูล	รายละเอียด
ไมโครคอนโทรลเลอร์	ESP32-WROVER-E
การสื่อสาร LTE และ GSM	T-SIM7070G
การรองรับ	Nano SIM Card, TF Card
การเชื่อมต่อ USB ไปยัง TTL	CH9102
Clock Speed	240Mhz
หน่วยความจำ PSRAM	8MB
หน่วยความจำแฟลช	4MB
โปรโตคอล	TCP/ UDP/ HTTP/ HTTPS/ FTP/TLS/ DTLS/ PING/ LWM2M/ COAP/MQTT
ระบบระบุตำแหน่ง GNSS	Beidou, GPS, GLONASS
อินเทอร์เฟซ	USB, SD Card, Solar Input interface
การเชื่อมต่อแบบไร้สาย	Wi-Fi: 802.11 b/g/n Bluetooth: v4.2 BR/EDR and BLE
อินเทอร์เฟซอุปกรณ์ต่อพ่วง	I2C, SPI, UART, SDIO

2.1.2 เซนเซอร์ Mi Temperature And Humidity Monitor 2

Mi Temperature and Humidity Monitor 2 เป็นอุปกรณ์ตรวจวัดอุณหภูมิและความชื้นจาก Xiaomi ที่ออกแบบมาเพื่อใช้งานในบ้านหรือพื้นที่ที่ต้องการการควบคุมสภาพอากาศอย่างแม่นยำ โดยมาพร้อมกับขนาดที่กะทัดรัดเพียง 43 x 43 x 12.5 มม. และมีหน้าจอ LCD แสดงผลอุณหภูมิและความชื้นอย่างชัดเจน อุปกรณ์นี้ใช้เซนเซอร์ที่มีความแม่นยำสูง สามารถวัดอุณหภูมิในช่วง -9.9°C ถึง 60°C และความชื้นในช่วง 0% ถึง 99% RH ซึ่งมีความแม่นยำถึง ±0.1°C สำหรับอุณหภูมิ และ ±1% RH สำหรับความชื้น นอกจากนี้ ยังสามารถเชื่อมต่อผ่านเทคโนโลยี Bluetooth Low Energy (BLE) เพื่อส่งข้อมูลไปยังแอป Mi Home บนสมาร์ทโฟน ทำให้สามารถตรวจสอบข้อมูลแบบเรียลไทม์ได้ แสดงได้ดังรูปที่ 2.2

รูปที่ 2.2 Mi Temperature And Humidity Monitor 2

ตารางที่ 2.2 ข้อมูลจำเพาะของ Mi Temperature And Humidity Monitor 2

ข้อมูล	รายละเอียด
วัสดุของผลิตภัณฑ์	ABS, PMMA
รุ่นแบตเตอรี่	CR2032
ช่วงการวัดอุณหภูมิ	0°C– 60°C
ช่วงการวัดความชื้น	0%–99% RH
ขนาดผลิตภัณฑ์	43*43*12.5 มม.
น้ำหนัก	21 กรัม
แรงดันไฟในการทำงาน	DC2.5V–3V
การเชื่อมต่อไร้สาย	Bluetooth 4.2 BLE (ระยะการเชื่อมต่อ 10 เมตร)
ความละเอียดการแสดงผลอุณหภูมิ	0.1°C

2.1.3 GY-NEO-6M GPS Module

GPS Module คือเป็นชิ้นส่วนอุปกรณ์รับสัญญาณของฮาร์ดแวร์ ที่สามารถเพิ่มเข้ากับชิ้นส่วนอื่นๆของฮาร์ดแวร์ต่างๆได้ เพื่อให้สามารถรับข้อมูลจากดาวเทียม GPS ได้ ซึ่งประกอบไปด้วย เสาอากาศเป็นตัวรับสัญญาณหลายช่องสัญญาณและการคำนวนในการรับส่งข้อมูลระยะทาง, เวลาที่ส่งไป แล้วถอดรหัสข้อมูลเหล่านั้นออกมาเป็นพิกัด ที่ส่งจากดาวเทียม และโปรเซสเซอร์ที่อยู่ใน GPS Module จะจัดการกับข้อมูลเหล่านี้และรายงานออกมาเป็น ตำแหน่ง พิกัด ความเร็ว และข้อมูลสำคัญต่างๆ โมดูล GY-NEO6M GPS สำหรับระบุตำแหน่งต่างๆบนโลกเป็นค่า ละติจูด

ลองจิจูด มี Library พร้อมใช้งาน รองรับไฟ 3-5V การใช้งานโมดูล GPS ครั้งแรก (Cold Start) จะต้องรอล็อคสัญญาณประมาณ 15 นาที ระหว่างล็อคสัญญาณครั้งแรกต้องเอาโมดูลภายนอกอาคาร หากล็อกสัญญาณได้แล้วไฟ Led จะกระพริบ แสดงได้ดังรูปที่ 2.3

รูปที่ 2.3 GY-NEO-6M GPS Module

ตารางที่ 2.3 ข้อมูลจำเพาะของ NEO-6M GPS Module

ข้อมูล	รายละเอียด
ชื่อผลิตภัณฑ์	GY-GPS6MV2
รุ่นแบตเตอรี่	CR2032
ขนาดของโมดูล	25 x 25 มม.
ขนาดของสายอากาศ	25 x 35 มม.

2.1.4 ภาษา C++ สำหรับ ARDUINO

C++ คือ ภาษา C programming language รุ่นใหม่ เป็นภาษาในการเขียนโปรแกรม ถูกพัฒนาโดย Dr.Bjarne Stroustrup ซึ่งเป็นนักวิจัยอยู่ที่ห้องปฏิบัติการ Bell Labs ประเทศสหรัฐอเมริกาในระหว่างปี พ.ศ. 2525-2528 ภาษา C++ เกิดจากแนวคิดในการเพิ่มประสิทธิภาพภาษา CC โดยได้นำความสามารถของ ภาษา C มาพัฒนา ให้เป็นโปรแกรมภาษาที่มีความเป็น Object Oriented Programming (โปรแกรมเชิงวัตถุ) และนี้เองคือที่มาของภาษา C++ จากการพัฒนานี้ทำให้ทุกสิ่งที่ภาษา C ทำได้ ภาษา C++ ก็จะสามารถทำได้ แต่สิ่งที่ภาษา C++ ทำได้ ภาษา C อาจจะทำไม่ได้

2.1.4.1 Preprocessor directives

Preprocessor directives เป็นส่วนที่เขียนไว้บนสุดของโปรแกรม และจะขึ้นต้นด้วย # ซึ่ง Code ในส่วนตรงนี้ จะทำงานก่อนที่จะ Compile Code ส่วนใหญ่จะไว้ใช้กำหนดค่าคงที่ หรือ Import library

2.1.4.2 ส่วนของการกำหนดค่า (Global declarations)

เป็นส่วนของการประกาศตัวแปรภายนอก Function หรือประกาศ Function ต่างๆ ซึ่งจะเป็นการประกาศแบบ Global หมายความว่าทุกๆ Function จะสามารถเรียกใช้ตัวแปร หรือ Function ที่ประกาศแบบนี้ได้

2.1.4.3 ฟังก์ชัน setup() และ ฟังก์ชัน loop()

ฟังก์ชัน setup() และฟังก์ชัน loop() เป็นคำสั่งที่ Arduino บังคับต้องให้มีในทุกโปรแกรม โดยทั้งสอง Function นี้ Arduino กำหนดให้มีหน้าที่ดังนี้

2.1.4.1 Function setup()

จะถูกเรียกใช้ทุกครั้งที่โปรแกรมเริ่มทำงาน ส่วนใหญ่จะเอาไว้ใช้กำหนดค่าตัวแปรเริ่มต้น หรือเริ่มต้นใช้งาน library ต่างๆ

2.1.4.2 Function loop() Function

จะถูกเรียกใช้หลังจาก Function setup() และจะทำงานแบบวนลูปไม่รู้จบ หมายความว่าเมื่อ loop() ทำงานเสร็จ ก็จะวนกลับมาทำงาน loop() อีกครั้ง วนแบบนี้ไปเรื่อยๆ

2.1.4.3 การสร้างฟังก์ชันและการใช้งานฟังก์ชัน (Users-defined function)

นอกจาก Function setup() และ loop() แล้วเรายังสามารถสร้าง Function ขึ้นมาใช้งานเองได้

2.1.5 การสื่อสารแบบ I2C

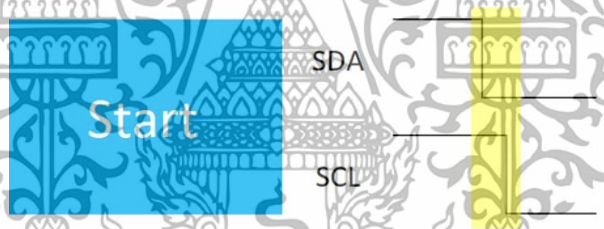
I²C นั้นย่อมาจาก Inter-Integrated Circuit - IIC หรือในรูป I²C (อ่านว่า "ไอ-สแคว-ซี" หรือจะอ่านว่า "ไอ-ทู-ซี" หรือ "ไอ-ไอ-ซี") เป็นการสื่อสารแบบ Serial รูปแบบหนึ่ง ถูกคิดค้นขึ้นใน ค.ศ.1982 โดย Philip semiconductor (ปัจจุบันคือ NXP Semiconductors) โดยมีจุดประสงค์เพื่อใช้รับส่งข้อมูลความเร็วต่ำระหว่างอุปกรณ์ มีจุดเด่นคือเชื่อมต่อแบบบัสสามารถเชื่อมต่ออุปกรณ์ได้เป็นจำนวนมาก โดยใช้สายสัญญาณเพียง 2 เส้นเท่านั้น จึงช่วยลดปริมาณของสายสัญญาณ และอุปกรณ์มีขนาดเล็กลง โดยสายสัญญาณ 2 เส้นนั้น คือ

1) SCL (Serial Clock Line) มีหน้าที่ส่งสัญญาณนาฬิกาเพื่อใช้สำหรับควบคุมการรับส่งข้อมูล ซึ่งความเร็วนี้ตามมาตรฐานคือ 100kHz และมีโหมดอื่นคือ Fast Mode มีความเร็วสูงสุด 400kHz, Hi-Speed Mode มีความเร็วสูงสุด 3.4MHz และ Ultra Fast Mode มีความเร็วสูงสุดที่ 5MHz โดยไมโครคอนโทรลเลอร์แต่ละตัวก็จะมีความเร็วของสัญญาณนาฬิกาต่างกัน (บอร์ด Arduino หรือ STM ที่เราใช้กันทั่ว ๆ ไปนี้ส่วนมากจะรองรับความเร็ว Standard และ Fast Mode เท่านั้น)

2) SDA (Serial Data Line) เป็นสายที่ใช้รับส่งข้อมูลที่ต้องการจะสื่อสารและเนื่องจากการรับส่งข้อมูลของสายสัญญาณทั้งสองเป็นแบบ Open-Drain จึงจำเป็นต้องต่อ Pull-Up Resistor ที่สายสัญญาณทั้งสอง และต้องอาศัยไฟเลี้ยงด้วย

2.1.5.1 การเริ่มต้นส่งข้อมูล (Start)

ในสถานะว่าง (Idle) ไม่มีการรับ-ส่งข้อมูล ทั้ง SDA และ SCL มีสถานะเป็น 1 ค้างทั้งคู่ เมื่อเริ่มมีการสื่อสารเกิดขึ้น ขา SDA จะเปลี่ยนลอจิกจาก 1 เป็น 0 ในขณะที่ SCL ยังมีสถานะเป็น 1 ค้างอยู่ สามารถแสดงได้ดังรูปที่ 2.4

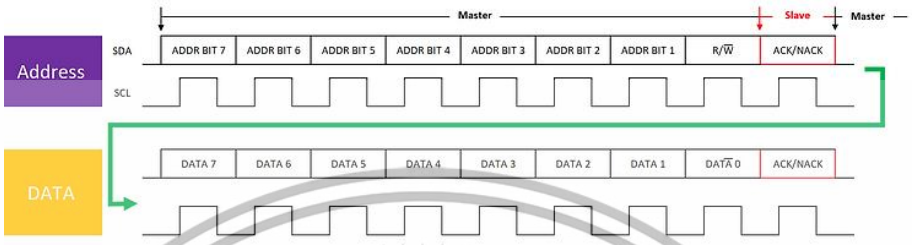


รูปที่ 2.4 การส่งสัญญาณเริ่มต้นส่งข้อมูล (Start) ของ I2C

2.1.5.2 การรับ-ส่งข้อมูล

ใน 1 เฟรม จะประกอบด้วยข้อมูล 8 บิต และ ACK/NACK อีก 1 บิต รวมการรับ-ส่งข้อมูล 1 เฟรม มี 9 บิต หลักจาก Master ส่งสัญญาณเริ่มต้น (Start) ออกไป เฟรมแรกสุดจะต้องเป็นหมายเลขของ Slave ที่จะสื่อสารในบิตที่ 7 ถึงบิตที่ 1 และทิศทางการรับส่งข้อมูลในบิตที่ 0 (R/W bit : Read / Write bit) หากในรอบนั้น Master ต้องการส่งข้อมูลไป Slave ในบิตที่ 0 จะกำหนดเป็น 0 (W) หากในรอบนั้น Master ต้องการรับข้อมูลจาก Slave ในบิตที่ 0 จะกำหนดเป็น 1 (R) จากนั้น Master จึงปล่อยลอยสาย SDA หาก Slave หมายเลขที่กำหนดมีอยู่จริง ตัว Slave จะเข้าควบคุมสาย SDA แทน โดยดึงสัญญาณสาย SDA ให้เป็น 0 เพื่อบอก Master ว่าพร้อมรับ-ส่งข้อมูลแล้ว เมื่อ Master ส่งสัญญาณที่ขา SCL จาก 0 -> 1 -> 0 ทางฝั่ง Slave จะ

ปล่อยลอยสาย SDA แล้ว Master จะกลับมาควบคุมสาย SDA อีกครั้ง หลังจากนั้นหากมีการรับ-ส่งข้อมูล ก็จะรับ-ส่งกันต่อไป หากไม่มีมีการรับ-ส่งข้อมูล ก็จะส่งสัญญาณสิ้นสุด (Stop) ออกไป สามารถแสดงได้ดังรูปที่ 2.5



รูปที่ 2.5 การรับ-ส่งข้อมูลของ I2C

2.1.5.3 การสิ้นสุดการส่งข้อมูล (Stop)

หากรับ-ส่งข้อมูลเรียบร้อยแล้ว ฝั่ง Master จะปล่อยสาย SCL ให้ลอยก่อน จากนั้นจึงปล่อยสาย SDA ให้ลอยตาม เพื่อเป็นการส่งสัญญาณ Stop บอกฝั่ง Slave ทุกตัวว่าจบการรับ-ส่งข้อมูลในรอบนั้นแล้ว สามารถแสดงได้ดังรูปที่ 2.6

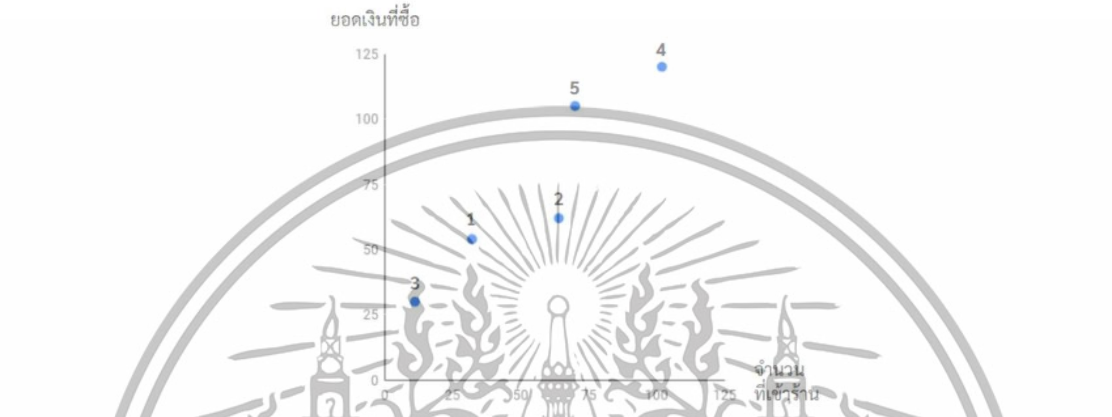
รูปที่ 2.6 การสิ้นสุดการส่งข้อมูล (Stop) ของ I2C

2.2 การจัดกลุ่มข้อมูลด้วยวิธี K-means (K-means Clustering)

การแบ่งกลุ่มข้อมูลแบบ K-means (K-means Clustering) เป็นเทคนิคการเรียนรู้แบบ การเรียนรู้แบบไม่มีผู้สอน (Unsupervised Machine Learning) ที่ใช้ในการแบ่งกลุ่มข้อมูลโดยไม่ต้องมีตัวอย่างหรือคำสั่งจากผู้สอน โดยมันทำหน้าที่ค้นหาความคล้ายคลึงกันหรือรูปแบบที่ซ่อนอยู่ในข้อมูล เพื่อช่วยให้เราเห็นโครงสร้างและความสัมพันธ์ของข้อมูลได้ชัดเจนยิ่งขึ้น ทำให้การสำรวจข้อมูลและการวิเคราะห์ในขั้นตอนต่อไปง่ายขึ้น

เราใช้ K-Means Clustering เพื่อจัดกลุ่มข้อมูลให้เป็นกลุ่มย่อย ๆ ตามความคล้ายคลึงกัน โดยไม่ต้องมีตัวอย่างหรือคำสั่งจากผู้สอน K ในที่นี้หมายถึงจำนวนกลุ่มที่เราต้องการแบ่งข้อมูล

ออกลองนึกถึงข้อมูลเป็นจุดต่าง ๆ ในพื้นที่ที่มีหลายมิติ (หรือหลายคุณสมบัติ) เช่น ข้อมูลแต่ละจุดอาจจะมีคุณสมบัติและค่าเฉพาะของมัน อัลกอริทึมจะจัดกลุ่มข้อมูลตามความคล้ายคลึง โดยวัดจากระยะห่างระหว่างจุด ซึ่งมักใช้ ระยะทางแบบยูคลิเดี้ยน (Euclidean distance) ในการวัด แสดงได้ดังรูปที่ 2.7

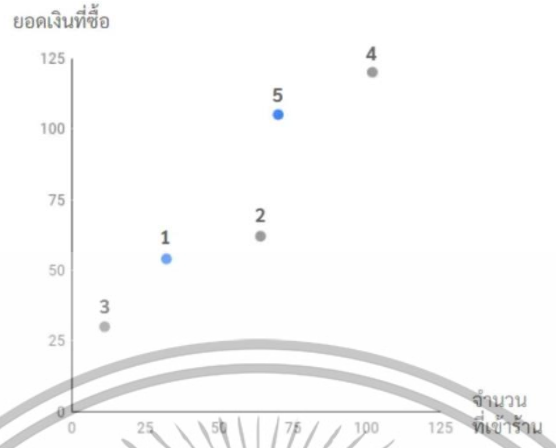


รูปที่ 2.7 ตัวอย่างข้อมูลที่ใช้ในขั้นตอนการทำงาน

1) กำหนดจำนวนกลุ่มที่ต้องการจัดกลุ่มของข้อมูล แทนค่าด้วย K ต้องการแบ่งกลุ่มข้อมูลตัวอย่างเป็น 2 กลุ่ม ดังนั้น ค่า K = 2

2) สุ่มจุดศูนย์กลาง (Centroid) จำนวน K จุด ในข้อมูลที่เราต้องการจัดกลุ่ม จุดนี้คือจุดเริ่มต้นที่จะใช้เป็นศูนย์กลางของแต่ละกลุ่ม

จากข้อมูลตัวอย่าง สมมุติให้จุด 1 และจุด 5 เป็น Centroid โดยจุด 1 เป็นกลุ่มที่ 1 และจุด 5 เป็นกลุ่มที่ 2 แสดงได้ดังรูปที่ 2.8



รูปที่ 2.8 สมมุติให้จุด 1 และจุด 5 เป็น Centroid

3) คำนวณระยะทางจากจุดข้อมูลต่างๆ กับจุดศูนย์กลาง ข้อมูลจะถูกจัดกลุ่ม (Cluster) กับจุดศูนย์กลางที่ใกล้ที่สุด

คำนวณระยะทางจากจุดข้อมูลกับ Centroid โดยได้ดังสมการที่ 2.1

$$distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (2.1)$$

ตารางที่ 2.4 ตารางแสดงการจัดกลุ่ม (cluster) ของข้อมูลตัวอย่าง

จุดที่	(x, y)	ระยะห่าง Centroid 1 (32,54)	ระยะห่าง Centroid 5 (70,105)	กลุ่ม(cluster)
1	(32,54)	0.00	63.60	1
2	(64,62)	32.98	43.42	1
3	(11,30)	31.89	95.43	1
4	(102,120)	96.21	35.34	2
5	(70,105)	63.60	0.00	2

4) หาค่าเฉลี่ยของแต่ละกลุ่มเพื่อกำหนดจุดศูนย์กลางใหม่ แล้วทำขั้นตอนที่ 3 ซ้ำจนกว่าค่าเฉลี่ยหรือจุดศูนย์กลางไม่เปลี่ยนแปลง

การหาค่าศูนย์กลางใหม่ของแต่ละกลุ่ม สามารถคำนวณได้ดังสมการที่ 2.2

$$x_{\{centroid\}} = \frac{x_1 + x_2 + \ldots + x_n}{n} \quad (2.2)$$

$$y_{\{centroid\}} = \frac{y_1 + y_2 + \ldots + y_n}{n} \quad (2.3)$$

จากสมการที่ (2.2) และ (2.3) หาค่าเฉลี่ยของแต่ละกลุ่มหรือจุดศูนย์กลางใหม่ได้ดังนี้

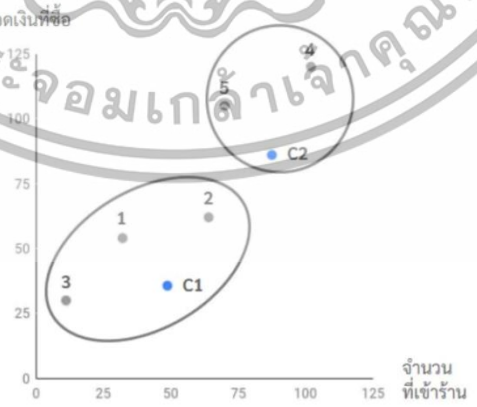
$$\text{cluster } 1 = \left(\frac{32+64+11}{3}, \frac{54+62+30}{3}\right) = (35.67, 48.67)$$

$$\text{cluster } 2 = \left(\frac{102+70}{2}, \frac{70+105}{2}\right) = (86, 87.5)$$

ตารางที่ 2.5 ตารางแสดงการคำนวณระยะทางจุดข้อมูลกับจุด centroid

จุดที่	(x, y)	ระยะห่าง Cluster 1 (35.67,48.67)	ระยะห่าง Cluster 2 (86,87.5)	กลุ่ม (cluster)
1	(32, 54)	6.47	63.55	1
2	(64, 62)	31.31	33.68	1
3	(11, 30)	30.94	94.51	1
4	(102, 120)	97.4	36.22	2
5	(70, 105)	65.97	23.71	2

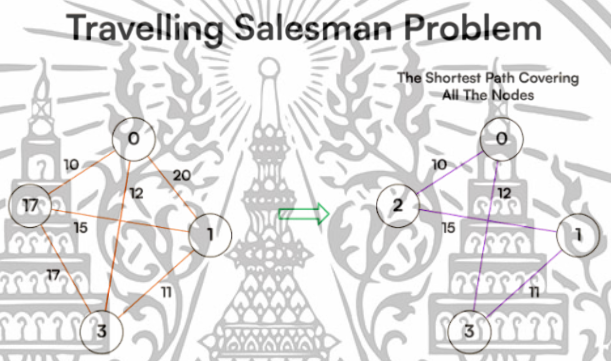
จะเห็นว่า เมื่อคำนวณค่าเฉลี่ยหรือจุดศูนย์กลางใหม่ในครั้งที่ 2 ค่าเฉลี่ยของแต่ละกลุ่มไม่เปลี่ยนแปลงแล้ว จึงสามารถจบการแบ่งจัดกลุ่มข้อมูลด้วย K-Mean Algorithm ได้ ซึ่งแสดงได้ดังรูปที่ 2.9



รูปที่ 2.9 ผลลัพธ์การแบ่งกลุ่มด้วยวิธีแบบ K-means (K-means Clustering)

2.3 ปัญหาการหาลำดับเส้นทางที่สั้นที่สุด (Traveling Salesman Problem)

ปัญหาการจัดลำดับเส้นทางที่สั้นที่สุดหรือ Traveling Salesman Problem (TSP) ซึ่งเป็นปัญหาที่เกี่ยวข้องกับการหาเส้นทางที่สั้นที่สุดในการเดินทางผ่านเมืองหรือจุดต่าง ๆ โดยผ่านแต่ละจุดเพียงครั้งเดียวและกลับมาที่จุดเริ่มต้น โดยมีเงื่อนไขว่าต้องการให้ระยะทางทั้งหมดของเส้นทางนั้นสั้นที่สุด Traveling Salesman Problem (TSP) ถือเป็นปัญหาที่มีความซับซ้อนสูงและจัดอยู่ในประเภท NP-hard เนื่องจากจำนวนเส้นทางที่เป็นไปได้จะเพิ่มขึ้นแบบทวีคูณเมื่อจำนวนจุดเพิ่มขึ้น ทำให้ไม่สามารถหาคำตอบที่ดีที่สุดได้อย่างรวดเร็วโดยใช้วิธีการค้นหาทั้งหมดซึ่งแสดงตัวอย่างของปัญหาดังรูปที่ 2.0



รูปที่ 2.10 ตัวอย่างการจัดลำดับเส้นทางที่สั้นที่สุด (Traveling Salesman Problem)

2.4 การเขียนโปรแกรมเพื่อแก้ปัญหาด้วยวิธีการลองทุกความเป็นไปได้ (Brute Force)

วิธีแบบทดลองความเป็นไปได้ทั้งหมด (Brute Force) ในการแก้ปัญหา Traveling Salesman Problem (TSP) คือการตรวจสอบทุกความเป็นไปได้ที่เป็นไปได้ทั้งหมดของเส้นทางการเดินทาง เพื่อหาคำตอบที่ดีที่สุด ซึ่งหมายความว่าเราจะต้องสร้างรายการเส้นทางทุกเส้นทางที่เป็นไปได้ที่ผ่านทุกจุด 1 ครั้งและกลับมายังเมืองเริ่มต้น แล้วคำนวณระยะทางของแต่ละเส้นทาง จากนั้นเลือกเส้นทางที่สั้นที่สุด วิธี Brute Force จะใช้งานได้ดีในกรณีที่จำนวนจุดที่น้อย แต่ถ้ามีจุดมีจำนวนมาก (เช่น 10 จุดขึ้นไป) การคำนวณจะช้ามากจนไม่สามารถใช้งานจริงได้ โดยทั่วไปแล้วการแก้ปัญหา TSP ที่มีจุดจำนวนมากจะทำให้เกิดความซับซ้อนในการคำนวณโดยมี Time Complexity เป็น O(n!) โดย n คือจำนวนจุดที่ต้องเดินทางผ่าน เช่นมีจำนวนจุดเริ่มต้น 1 จุด และ

จำนวนจุดที่ต้องแวะผ่าน 3 จุด n เป็น 3 ซึ่งขั้นตอนในแก้ปัญหาด้วยวิธีทดลองความเป็นไปได้ทั้งหมด (Brute Force) ประกอบด้วย

1) ระบุจุดเริ่มต้นที่ต้องการ จากนั้นสร้างชุดการเดินทางผ่านจุดอื่นๆ ทั้งหมด

2) สร้างเส้นทางที่สามารถเป็นไปได้ทั้งหมดซึ่งผ่านจุดแต่ละจุดเพียงครั้งเดียว

3) หาผลรวมของระยะทางในแต่ละเส้นทางที่เกิดขึ้น

4) จากผลลัพธ์ที่ได้ เลือกเส้นทางที่มีระยะทางสั้นที่สุดเป็นคำตอบ

ตัวอย่างในการคำนวณ

สมมติให้มีจุด 4 จุด A , B , C , D โดยกำหนดให้ A เป็นจุดเริ่มต้น (กม.)

ตารางที่ 2.6 ตารางแสดง Distance Matrix ระหว่างจุด A , B , C , D

จุดเริ่มต้น/จุดสิ้นสุด	A	B	C	D
A	0	5	9	6
B	5	0	3	7
C	9	3	0	4
D	6	7	4	0

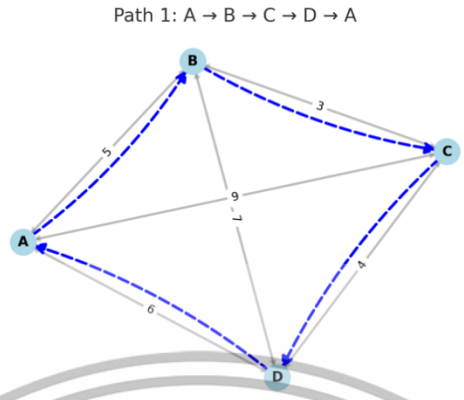
การคำนวณจะทำการทดลองเส้นทางที่เป็นไปได้และคำนวณระยะทางรวมของแต่ละเส้นทาง เพื่อหาเส้นทางที่ใช้ระยะทางน้อยที่สุด

ตารางที่ 2.7 ตารางแสดง การคำนวนระยะทางรวมของแต่ละเส้นทาง

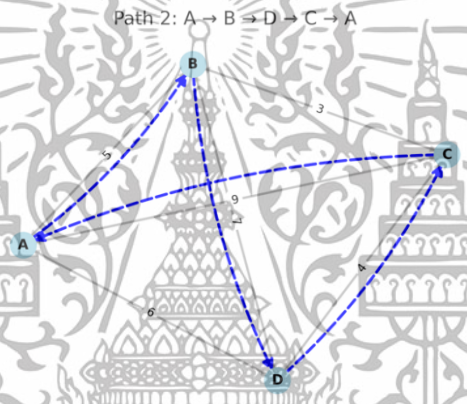
เส้นทาง	ระยะทางรวม (กม.)
A ⟶ B ⟶ C ⟶ D ⟶ A	18
A ⟶ B ⟶ D ⟶ C ⟶ A	25
A ⟶ C ⟶ B ⟶ D ⟶ A	25
A ⟶ C ⟶ D ⟶ B ⟶ A	25
A ⟶ D ⟶ B ⟶ C ⟶ A	25
A ⟶ D ⟶ C ⟶ B ⟶ A	18

ซึ่งจะเห็นได้ว่าต้องใช้การคำนวณ O(n!) = O(3!) = 6 ครั้ง

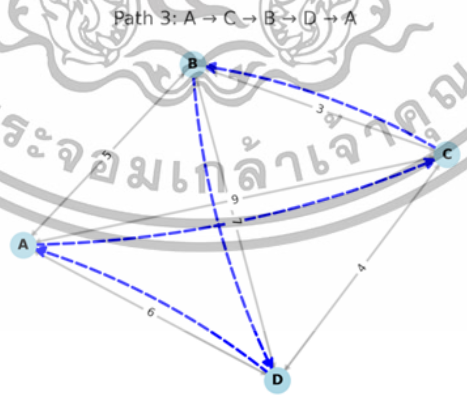
แสดงกราฟตัวอย่างเส้นทางที่เป็นไปได้ทั้งหมด ซึ่งแสดงดังรูปที่ 2.11 – 2.16



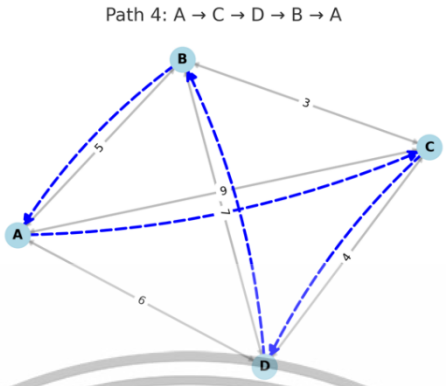
รูปที่ 2.11 เส้นทางที่เป็นไปได้ เส้นทางที่ 1



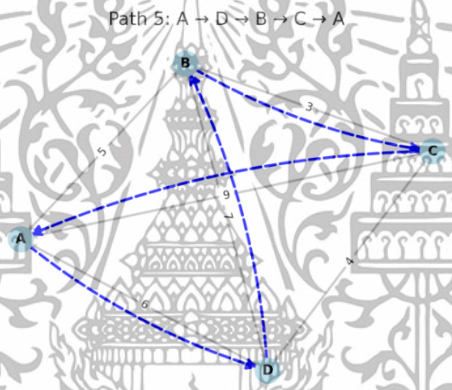
รูปที่ 2.12 เส้นทางที่เป็นไปได้ เส้นทางที่ 2



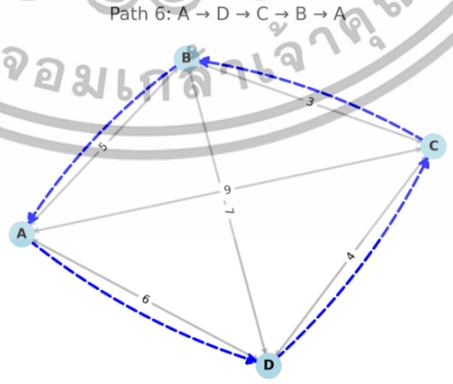
รูปที่ 2.13 เส้นทางที่เป็นไปได้ เส้นทางที่ 3



รูปที่ 2.14 เส้นทางที่เป็นไปได้ เส้นทางที่ 4



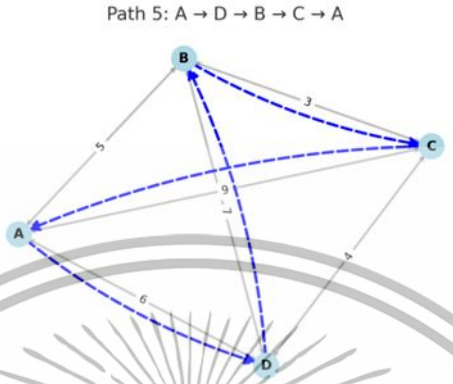
รูปที่ 2.15 เส้นทางที่เป็นไปได้ เส้นทางที่ 5



รูปที่ 2.16 เส้นทางที่เป็นไปได้ เส้นทางที่ 6

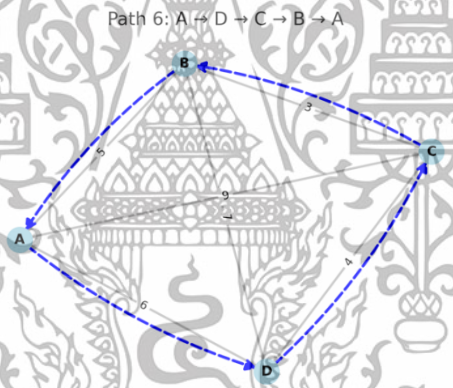
ดังนั้นเส้นทางที่สั้นที่สุดในกรณีนี้มีทั้งหมดสองเส้นทางคือ

1) A ⟶ B ⟶ C ⟶ D ⟶ A มีระยะทางรวม 18 กม. แสดงดังรูปที่ 2.17



รูปที่ 2.17 เส้นทางที่สั้นที่สุด เส้นทางที่ 1

2) A ⟶ D ⟶ C ⟶ B ⟶ A มีระยะทางรวม 18 กม. แสดงดังรูปที่ 2.16



รูปที่ 2.18 เส้นทางที่สั้นที่สุด เส้นทางที่ 2

2.5 การเขียนโปรแกรมเพื่อแก้ปัญหาแบบโปรแกรมเชิงพลวัต (Dynamic Programming)

การหาคำตอบที่เหมาะสมที่สุดของปัญหาที่มีความซับซ้อนเชิงผสมมักจะใช้แนวคิดของ Dynamic Programming ในการแก้ไข โดยการแบ่งปัญหาออกเป็นปัญหาย่อย ๆ ที่เล็กกว่าและใช้ผลลัพธ์ที่ได้จากปัญหาย่อยในการสร้างคำตอบของปัญหาใหญ่ หนึ่งในอัลกอริทึมที่ใช้ Dynamic Programming ในการแก้ปัญหาคืออัลกอริทึม Held-Karp ซึ่งถูกพัฒนาขึ้นมาเพื่อ

แก้ปัญหาการหาจัดลำดับเส้นทางที่สั้นที่สุด หรือ Traveling Salesman Problem (TSP) โดยสามารถลด Time Complexity จาก O(n!) ให้เหลือ $O(2^n.n^2)$

สมการที่ใช้สำหรับคำนวณต้นทุนเมื่อเรามีเซตย่อยที่ประกอบด้วยเพียงจุดเดียวเท่านั้น

$$cost(x_i, \{x_i\}) = D_{1i} \tag{2.4}$$

สำหรับเซตย่อยที่มีจุดมากกว่า 1 จุด เราจะใช้สมการที่คำนวณจากต้นทุนของการเดินทางที่ผ่านจุดต่าง ๆ ในเซตย่อยนั้น ๆ โดยฟังก์ชันต้นทุนจะหาค่าที่น้อยที่สุดของการเดินทางจากเซตย่อยที่เล็กลงมา

โดยให้ S เป็นเซตย่อยที่ประกอบด้วยจุดหลายจุด (เช่น S = {B , C}) และต้องการคำนวนระยะทางจากจุดใดๆ (เช่น x_i = B) ให้เดินผ่านจุดในเซตย่อยทั้งหมดและสิ้นสุดที่ x_i

$$cost(x_i, S) = \min\left(cost\left(x_j, S\right)\{x_i\}) + D_{ji}\right) \tag{2.5}$$

$$cost(x_j \setminus \{x_i\})$$

คือต้นทุนของการเดินทางจากจุดเริ่มต้นผ่านจุดต่าง ๆ ในเซตย่อยยกเว้น X_i และสิ้นสุดที่จุด X_j

$$D_{ji}$$

คือระยะทางจากจุด X_j มายัง X_i

ขั้นตอนสุดท้ายคือการคำนวณระยะทางรวมของวง Hamiltonian ที่สั้นที่สุดที่เริ่มต้นจาก Aผ่านทุกจุดเพียงครั้งเดียวแล้วกลับมายัง A โดยจะคำนวณจากฟังก์ชันต้นทุนของเซตย่อยขนาดใหญ่สุด (ที่ครอบคลุมทุกจุด) บวกกับระยะทางในการกลับมายังจุดเริ่มต้น

$$Shortest\ Hamiltonian\ Cycle = \min\left(cost\left(x_i, V\{A\}\right) + D_{iA}\right) \tag{2.6}$$

คำอธิบาย

$$cost(x_i, V\{A\})$$

คือต้นทุนของการเดินทางจากจุดเริ่มต้น A ผ่านทุกจุดและสิ้นสุดที่ x_j

$$D_{jA}$$

คือระยะทางจากจุด x_i กลับมายังจุดเริ่มต้น A

ตัวอย่างขั้นตอนในการแก้ไขปัญหาสำหรับการจัดลำดับเส้นทางที่สั้นที่สุด

1) สร้าง Distance Matrix

ตารางที่ 2.8 ตาราง Distance Matrix

จุดเริ่มต้น/จุดสิ้นสุด	**A**	**B**	**C**	**D**
A	0	5	9	6
B	5	0	3	7
C	9	3	0	4
D	6	7	4	0

2) พิจารณากลุ่มย่อยที่มี S (Subset) ขนาด 1

ตาราง 2.9 กระบวนการทำงานสำหรับสับเซต S ขนาด 1

สับเซต S	**ฟังก์ชันระยะทาง**	**ผลลัพธ์**
$S = \{x_B\}$	$cost(\{x_B, \{x_B\}) = D_{AB}$	5
$S = \{x_C\}$	$cost(\{x_C, \{x_C\}) = D_{AC}$	9
$S = \{x_D\}$	$cost(\{x_D, \{x_D\}) = D_{AD}$	6

3) พิจารณากลุ่มย่อยที่มี S (Subset) ขนาด 2

ตารางที่ 2.10 กระบวนการทำงานสำหรับสับเซต S ขนาด 2

สับเซต S	**ฟังก์ชันระยะทาง**	**ผลลัพธ์**
$S = \{x_B, x_C\}$	$cost(x_B, \{x_B, x_C\}) = min(cost(x_C, \{x_C\}) + D_{\{CB\}}) = 9 + 3$	12
	$cost(x_C, \{x_B, x_C\}) = \min(cost(x_B, \{x_B\}) + D_{\{BC\}}) = 5 + 3$	8
$S = \{x_B, x_D\}$	$cost(x_B, \{x_B, x_D\}) = min(cost(x_D, \{x_D\}) + D_{\{DB\}}) = 6 + 7$	13
	$cost(x_D, \{x_B, x_D\}) = min(cost(x_B, \{x_B\}) + D_{\{BD\}}) = 5 + 7$	12
$S = \{x_C, x_D\}$	$cost(x_C, \{x_C, x_D\}) = min(cost(x_D, \{x_D\}) + D_{\{DC\}}) = 6 + 4$	10
	$cost(x_D, \{x_C, x_D\}) = min(cost(x_C, \{x_C\}) + D_{\{CD\}}) = 9 + 4$	13

4) พิจารณากลุ่มย่อยที่มี S (Subset) ขนาด 3

ตารางที่ 2.11 กระบวนการทำงานสำหรับสับเซต S ขนาด 2

สับเซต S	ฟังก์ชันระยะทาง	ผลลัพธ์
$S = \{B, C, D\}$	$cost(x_B, \{x_B, x_C, x_D\}) = \min(cost(x_C, \{x_C, x_D\}) + D_{\{CB\}}, cost(x_D, \{x_C, x_D\}) + D_{\{DB\}}) = \min(10 + 3, 13 + 7)$	13
	$cost(x_C, \{x_B, x_C, x_D\}) = \min(cost(x_B, \{x_B, x_D\}) + D_{\{BC\}}, cost(x_D, \{x_B, x_D\}) + D_{\{DC\}}) = \min(13 + 3, 12 + 4)$	16
	$cost(x_D, \{x_B, x_C, x_D\}) = \min(cost(x_B, \{x_B, x_C\}) + D_{\{BD\}}, cost(x_C, \{x_B, x_C\}) + D_{\{CD\}}) = \min(12 + 7, 8 + 4)$	12

5) ขั้นตอนสุดท้ายคำนวณระยะทางที่สั้นที่สุด

$$Shortest\ Hamiltonian\ type = \min\{\cos t\{x_B, \{x_B, x_c, x_D\}\} + D_{BA}, cost\{x_c, \{x_B, x_c, x_D\}\} + D_{CA}, cost\{x_D, \{x_B, x_c, x_D\} + D_{DA}\} = \min\{13 + 5, 16 + 9, 12 + 6\} = \min\{18, 25, 18\}$$

ดังนั้นเราได้ค่าเส้นทางที่สั้นที่สุดสองเส้นทาง

6) ทำการย้อนรอยเพื่อหาลำดับเส้นทาง (Backtracking)

หาเส้นทางแรกโดย ระยะทางที่สั้นที่สุด จาก A ← B

ต่อมาเราจะดูว่าจาก B ← อะไรสั้นที่สุด ผลคือจาก B ← C

ดังนั้นเส้นทางที่เหลือจะเป็น A ← B ← C ← D ← A

ดังนั้นเส้นทางแรกที่ได้คือ A → D → C → B → A

จากหลักการเดียวกันจะได้ว่าเส้นทางที่สองคือ A ← D ← C ← B ← A

ดังนั้นเส้นทางที่สองคือ A → B → C → D → A

2.6 เครื่องมือและซอฟต์แวร์ที่ใช้ในปริญญานิพนธ์

2.6.1 Arduino IDE

Arduino IDE (Integrated Development Environment) แสดงได้ดังรูปที่ 2.19 เป็นซอฟต์แวร์สำหรับการพัฒนาและทดสอบโปรแกรมที่ใช้ในการควบคุมไมโครคอนโทรลเลอร์ Arduino ซึ่งเป็นแพลตฟอร์มที่นิยมใช้ในงานสร้างโปรเจคอิเล็กทรอนิกส์และ Internet of Things (IoT) โดยเฉพาะในกลุ่มผู้ที่เริ่มต้นและผู้ที่สนใจในการพัฒนาอุปกรณ์ฮาร์ดแวร์

Arduino IDE รองรับการทำงานบนระบบปฏิบัติการที่หลากหลาย เช่น Windows, macOS และ Linux ซึ่งสามารถติดตั้งและใช้งานได้สะดวก เนื่องจากมีฟังก์ชันการทำงานที่ครอบคลุมในการเขียนโค้ด การตรวจสอบข้อผิดพลาด และการอัปโหลดโปรแกรมไปยังบอร์ด Arduino

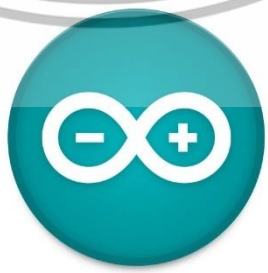
2.6.1.1 หลักการทำงานของ Arduino IDE

1) การคอมไพล์ (Compile) เมื่อเขียนโค้ดเสร็จสมบูรณ์แล้ว ผู้ใช้สามารถคอมไพล์โค้ดเพื่อตรวจสอบและแปลงโค้ดให้เป็นคำสั่งที่บอร์ด Arduino เข้าใจได้

2) การอัปโหลด (Upload) เมื่อคอมไพล์โค้ดสำเร็จแล้ว ผู้ใช้สามารถอัปโหลดโค้ดที่ผ่านการแปลงแล้วไปยังบอร์ด Arduino เพื่อให้บอร์ดทำงานตามคำสั่งที่กำหนด

3) การตรวจสอบข้อผิดพลาด (Debugging) – Arduino IDE มีระบบตรวจสอบข้อผิดพลาดที่ช่วยบอกตำแหน่งที่เกิดข้อผิดพลาดในโค้ด ทำให้นักพัฒนาสามารถแก้ไขโค้ดได้อย่างสะดวก

4) การรองรับไลบรารีที่หลากหลาย Arduino IDE มีระบบจัดการไลบรารีที่ช่วยให้ผู้ใช้สามารถติดตั้งไลบรารีเสริมได้อย่างง่ายดาย เพื่อเพิ่มความสามารถให้กับบอร์ด Arduino สำหรับการใช้งานในด้านต่าง ๆ เช่น การเชื่อมต่อกับเซ็นเซอร์ โมดูลการสื่อสาร หรืออุปกรณ์เสริมอื่น ๆ



รูปที่ 2.19 โลโก้ของ Arduino DIE

2.6.2 MQTT

MQTT (Message Queuing Telemetry Transport) แสดงได้ดังรูปที่ 2.20 คือ โปรโตคอลการสื่อสารที่ออกแบบมาเพื่อส่งข้อมูลในระบบที่มีข้อจำกัดเรื่องพลังงานและการเชื่อมต่อ เช่น อุปกรณ์ IoT (Internet of Things) โดยเฉพาะ ซึ่ง MQTT ใช้หลักการการส่งข้อความแบบ "Publish/Subscribe" ทำให้มีความเร็วสูงและใช้งานทรัพยากรน้อย เหมาะกับการส่งข้อมูลในระบบที่มีความหน่วงต่ำ

2.6.2.1 หลักการทำงานของ MQTT

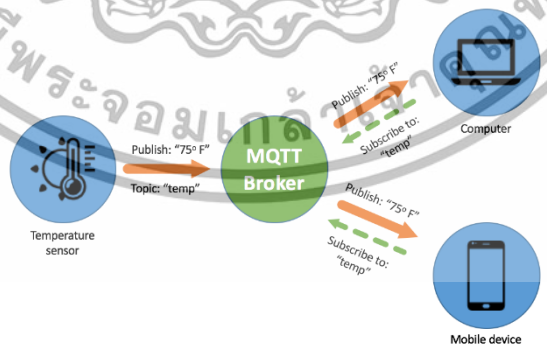
MQTT ใช้โมเดล Publish/Subscribe ประกอบไปด้วยองค์ประกอบสำคัญ 3 ส่วน

1) Publisher ผู้ส่งข้อมูล เป็นผู้ส่งข้อความไปยัง "Topic" ที่กำหนด เช่น อุปกรณ์เซ็นเซอร์ที่ส่งข้อมูลอุณหภูมิ

2) Subscriber ผู้รับข้อมูล จะทำการ "Subscribe" (สมัครรับ) ข้อมูลจาก Topic ที่ต้องการ เมื่อมีข้อความใหม่ถูกส่งใน Topic นี้ Subscriber ก็จะได้รับข้อมูลนั้น

3) Broker ตัวกลางในการจัดการการส่งข้อมูลจาก Publisher ไปยัง Subscriber โดย Broker จะเก็บและจัดการการส่งข้อมูลไปยังผู้ที่สมัครรับตาม Topic ที่ต้องการ

การทำงานของ Topic เป็นหัวข้อที่กำหนดในระบบ MQTT โดย Publisher จะส่งข้อความไปยัง Topic และ Subscriber จะรับข้อความจาก Topic ที่ตรงกัน ตัวอย่างเช่น หากมี Topic ชื่อ "home/temperature" และมีอุปกรณ์เซ็นเซอร์ที่ Publish ข้อมูลไปยัง Topic นี้ ผู้ที่ Subscribe Topic "home/temperature" ก็จะได้รับข้อมูลที่ส่งไปเสมอ



รูปที่ 2.20 การทำงานของ MQTT

2.6.3 ภาษา Python

Python เป็นภาษาโปรแกรมเชิงวัตถุ (Object-Oriented Programming) แสดงได้ดังรูปที่ 2.21 และเชิงสคริปต์ที่ถูกพัฒนาโดย Guido van Rossum ในช่วงปี 1990 จุดเด่นของภาษา Python คือความง่ายต่อการอ่าน เขียน และเรียนรู้ ทำให้เหมาะสำหรับนักพัฒนาทุกระดับ ทั้งผู้เริ่มต้นและผู้มีประสบการณ์

2.6.3.1 คุณสมบัติและลักษณะเด่นของ Python

1) Syntax ที่เรียบง่าย การเขียนโค้ดใน Python ไม่ต้องใช้เครื่องหมายที่ซับซ้อน ทำให้อ่านง่ายและเหมาะสำหรับผู้เริ่มต้น

2) การจัดการหน่วยความจำอัตโนมัติ Python จัดการหน่วยความจำให้เอง ทำให้ผู้พัฒนาไม่ต้องกังวลเกี่ยวกับการจัดสรรหรือเคลียร์หน่วยความจำ

3) ไลบรารีและโมดูลจำนวนมาก Python มีไลบรารีมาตรฐานและไลบรารีจากภายนอก (Third-party Libraries) ที่ช่วยเพิ่มความสามารถ เช่น การจัดการข้อมูล, การวิเคราะห์ข้อมูล, การเรียนรู้ของเครื่อง (Machine Learning), การทำงานกับเว็บ และอื่น ๆ

4) การใช้งานแบบ Multi-paradigm – รองรับหลายรูปแบบ เช่น การเขียนเชิงฟังก์ชัน (Functional Programming), เชิงวัตถุ (Object-Oriented Programming), และเชิงโครงสร้าง (Procedural Programming)

รูปที่ 2.21 โลโก้ของภาษา Python

2.6.4 โปรแกรม Postman

Postman เป็นเครื่องมือที่ใช้ในการทดสอบและพัฒนา API (Application Programming Interface) แสดงได้ดังรูปที่ 2.22 เป็นที่นิยมใช้กันอย่างแพร่หลายโดยนักพัฒนาและนักทดสอบซอฟต์แวร์ โดยเฉพาะอย่างยิ่งในงานพัฒนาซอฟต์แวร์ที่ต้องการเชื่อมต่อหรือส่งข้อมูลระหว่างระบบผ่าน HTTP เช่น REST API หรือ GraphQL

Postman ช่วยให้ผู้ใช้สามารถส่งคำขอ (Requests) ไปยังเซิร์ฟเวอร์และรับคำตอบ (Responses) กลับมาได้ ทำให้สามารถทดสอบการทำงานของ API อย่างละเอียดได้ คุณสมบัติเด่นของ Postman ประกอบไปด้วย

1) การส่งคำขอ HTTP รองรับการส่งคำขอแบบ HTTP (GET, POST, PUT, DELETE ฯลฯ) ไปยัง API ซึ่งผู้ใช้สามารถกำหนดพารามิเตอร์ เฮดเดอร์ (Headers) และเนื้อหา (Body) ของคำขอได้อย่างละเอียด

2) การทดสอบ API อัตโนมัติ สามารถสร้างการทดสอบอัตโนมัติ (Automated Testing) โดยใช้คำสั่ง JavaScript ในการตรวจสอบคำตอบที่ได้รับ เพื่อให้มั่นใจว่า API ทำงานได้ตามที่คาดหวัง

3) การจัดการคอลเลคชัน (Collections) ผู้ใช้สามารถจัดเก็บคำขอและพารามิเตอร์ต่างๆ ไว้ในรูปแบบคอลเลคชัน เพื่อความสะดวกในการทดสอบหลายคำขอในคราวเดียว

4) การใช้งานร่วมกับระบบ CI/CD Postman รองรับการเชื่อมต่อกับระบบ Continuous Integration และ Continuous Delivery (CI/CD) เช่น Jenkins และ GitLab เพื่อทำการทดสอบ API อัตโนมัติในกระบวนการพัฒนาซอฟต์แวร์

5) การสร้างและแบ่งปันเอกสาร API Postman สามารถสร้างเอกสาร API ที่อ่านง่ายและแชร์ได้ ทำให้การสื่อสารข้อมูล API ภายในทีมงานง่ายขึ้น

6) Mock Server – สามารถสร้าง Mock Server เพื่อทดสอบ API ในสภาพแวดล้อมที่ยังไม่มีเซิร์ฟเวอร์จริง

รูปที่ 2.22 โลโก้ของโปรแกรม Postman

2.6.5 ReactJS

ReactJS เป็นไลบรารี JavaScript แสดงได้ดังรูปที่ 2.23 แบบโอเพนซอร์สที่พัฒนาโดย Facebook สำหรับการสร้างส่วนติดต่อผู้ใช้งาน (User Interface หรือ UI) โดยเฉพาะในรูปแบบของแอปพลิเคชันเว็บเชิงโต้ตอบ (Single Page Application - SPA) React ถูกออกแบบมาเพื่อให้การพัฒนา UI ที่ซับซ้อนกลายเป็นเรื่องง่ายและมีประสิทธิภาพ ด้วยการใช้แนวคิดของคอมโพเนนต์ (Component) ซึ่งช่วยให้การเขียนและการจัดการโค้ดเป็นไปอย่างเป็นระบบ และนำกลับมาใช้ใหม่ได้ง่าย

2.6.5.1 คุณสมบัติและการทำงานของ ReactJS

1) การทำงานแบบคอมโพเนนต์ (Component-Based) React ใช้แนวคิดของคอมโพเนนต์ ซึ่งเป็นชิ้นส่วนเล็กๆ ที่สามารถสร้างและจัดการ UI ได้แต่ละส่วน โดยคอมโพเนนต์แต่ละตัวจะมีโค้ดที่ประกอบด้วยโครงสร้าง HTML, CSS และ JavaScript อยู่ด้วยกัน ทำให้สามารถนำคอมโพเนนต์นี้ไปใช้ซ้ำได้ในส่วนต่างๆ ของแอปพลิเคชัน

2) Virtual DOM – React ใช้เทคนิค Virtual DOM เพื่อเพิ่มประสิทธิภาพในการอัปเดต UI โดยจะสร้างสำเนาของ DOM ที่อยู่ในหน่วยความจำ และเมื่อมีการเปลี่ยนแปลงเกิดขึ้น React จะคำนวณและเปรียบเทียบการเปลี่ยนแปลงนั้นใน Virtual DOM ก่อนทำการอัปเดตไปยัง DOM จริง วิธีนี้ช่วยลดการทำงานหนักของเบราว์เซอร์ ทำให้การแสดงผลเร็วขึ้น

3) การจัดการสถานะ (State Management) React มีระบบจัดการสถานะ (State) ที่ช่วยเก็บข้อมูลของคอมโพเนนต์แต่ละตัว การเปลี่ยนแปลงสถานะทำให้ UI สามารถอัปเดตตามข้อมูลที่เปลี่ยนแปลงได้อย่างราบรื่น

4) การทำงานร่วมกับไลบรารีและเฟรมเวิร์คอื่นๆ React สามารถใช้ร่วมกับไลบรารีอื่น ๆ เช่น Redux สำหรับจัดการสถานะของแอปพลิเคชัน หรือใช้ร่วมกับ Next.js สำหรับการพัฒนาเว็บไซต์เชิงโต้ตอบที่ต้องการ SEO ได้ดี

5) Reusable Components คอมโพเนนต์ใน React สามารถนำมาใช้ซ้ำได้ ทำให้การพัฒนา UI ของแอปพลิเคชันมีความรวดเร็ว และสามารถลดโค้ดที่ซ้ำซ้อน

รูปที่ 2.23 โลโก้ของ ReactJS

2.6.6 Django

Django เป็นเฟรมเวิร์คสำหรับพัฒนาเว็บแอปพลิเคชันที่เขียนด้วยภาษา Python ซึ่งออกแบบมาเพื่อช่วยให้นักพัฒนาสามารถสร้างเว็บแอปพลิเคชันได้อย่างรวดเร็วและมีประสิทธิภาพ โดย Django มีจุดเด่นในการจัดการโครงสร้างของโปรเจกต์และเน้นความปลอดภัย มีการออกแบบให้เหมาะกับการทำงานที่มีขนาดใหญ่หรือระบบที่ต้องการประสิทธิภาพสูง

2.6.6.1 คุณสมบัติและการทำงานของ Django

1) โครงสร้างที่เป็นระเบียบและชัดเจน (MTV Pattern) – Django ใช้โครงสร้างการพัฒนาแบบ Model-Template-View (MTV) ซึ่งเป็นรูปแบบคล้ายกับ Model-View-Controller (MVC) ทำให้การพัฒนาและการจัดการโค้ดมีความเป็นระบบ โดยแบ่งออกเป็น

1.1) Model – จัดการข้อมูลและโครงสร้างของฐานข้อมูล

1.2) Template – การจัดการหน้าเว็บหรือส่วนของการแสดงผล

1.3) View – ฟังก์ชันที่ทำหน้าที่เชื่อมต่อข้อมูลระหว่าง Model และ Template

2) ORM (Object-Relational Mapping) Django มีระบบ ORM ที่ช่วยให้การจัดการกับฐานข้อมูลง่ายขึ้น โดยสามารถเขียนโค้ด Python เพื่อจัดการฐานข้อมูลแทนการใช้คำสั่ง SQL ซึ่งช่วยลดโอกาสที่จะเกิดข้อผิดพลาดในการเขียน SQL และเพิ่มความสะดวกสบายในการพัฒนา

3) ความปลอดภัยสูง Django มีการตั้งค่าความปลอดภัยที่ดี เช่น การป้องกัน SQL Injection, Cross-Site Scripting (XSS), และ Cross-Site Request Forgery (CSRF) ซึ่งช่วยป้องกันการโจมตีทางเว็บหลายประเภท

4) Admin Interface Django มีระบบแอดมินอินเทอร์เฟซที่สร้างขึ้นอัตโนมัติเพื่อจัดการข้อมูลในฐานข้อมูล ทำให้นักพัฒนาสามารถจัดการข้อมูลได้ง่ายโดยไม่ต้องเขียนโค้ดเพิ่มเติม

5) การขยายและการนำไปใช้ (Scalability) Django สามารถนำไปใช้งานในโปรเจกต์ที่มีขนาดใหญ่ได้อย่างมีประสิทธิภาพ เนื่องจากมีการจัดการทรัพยากรและการโหลดได้ดี

6) รองรับการทำงานแบบหลายแอปพลิเคชัน Django ช่วยให้สามารถแบ่งการทำงานเป็นแอปพลิเคชันย่อยๆ ในโปรเจกต์เดียวกันได้ ทำให้การพัฒนาและการจัดการโค้ดมีความเป็นระเบียบและง่ายขึ้น

2.6.7 OpenStreetMap

OpenStreetMap (OSM) แสดงได้ดังรูปที่ 2.24 เป็นโครงการแผนที่ออนไลน์แบบโอเพนซอร์ส ที่ให้ข้อมูลแผนที่ทั่วโลกอย่างละเอียดซึ่งผู้ใช้สามารถเข้าถึงได้ฟรี ข้อมูลทั้งหมดใน OpenStreetMap เกิดจากการรวบรวมข้อมูลของผู้ใช้งานทั่วโลก โดยทุกคนสามารถเพิ่ม แก้ไข และปรับปรุงข้อมูลแผนที่ได้อย่างอิสระ โครงการนี้ก่อตั้งขึ้นในปี 2004 โดย Steve Coast โดยมีวัตถุประสงค์เพื่อตอบสนองความต้องการแผนที่ที่สามารถเข้าถึงได้ฟรี ซึ่งไม่ถูกจำกัดโดยลิขสิทธิ์หรือข้อกำหนดจากผู้ให้บริการแผนที่เชิงพาณิชย์

2.6.7.1 คุณสมบัติและการทำงานของ OpenStreetMap

1) การใช้งานข้อมูลแบบโอเพนซอร์ส ข้อมูลแผนที่ใน OSM เปิดให้ทุกคนสามารถใช้งานได้ฟรีภายใต้ลิขสิทธิ์แบบ Open Database License (ODbL) ทำให้ผู้ใช้สามารถนำข้อมูลไปใช้ในเชิงพาณิชย์หรือปรับแต่งข้อมูลได้โดยไม่เสียค่าใช้จ่าย

2) การอัปเดตข้อมูลโดยผู้ใช้ ข้อมูลใน OSM ได้รับการอัปเดตโดยผู้ใช้ทั่วโลกที่ร่วมมือกันผ่านการเพิ่มข้อมูลใหม่และการแก้ไขข้อมูลที่มีอยู่ ซึ่งทำให้แผนที่มีความแม่นยำและทันสมัยมากขึ้น

3) การครอบคลุมแผนที่ที่กว้างขวาง OSM มีข้อมูลที่ครอบคลุมทั้งข้อมูลถนน สถานที่สำคัญ (Points of Interest) เส้นทางเดินเท้า สถานที่บริการต่างๆ เช่น โรงเรียน โรงพยาบาล ร้านค้า ฯลฯ ซึ่งมีความละเอียดและครอบคลุมทั่วโลก

4) การให้บริการแผนที่และ API OSM ให้บริการ API สำหรับนักพัฒนาเพื่อดึงข้อมูลแผนที่ไปใช้ในแอปพลิเคชันต่างๆ เช่น การนำทาง แอปพลิเคชันสถานที่ท่องเที่ยว หรือระบบจัดการขนส่ง ทำให้เป็นที่นิยมในกลุ่มผู้พัฒนาแอปพลิเคชันและเว็บไซต์

5) ความสามารถในการปรับแต่งข้อมูลแผนที่ – ผู้ใช้สามารถดาวน์โหลดข้อมูลแผนที่และปรับแต่งรูปแบบการแสดงผลได้ตามต้องการ เช่น การกำหนดสี ขนาด หรือประเภทของสถานที่ที่จะให้แสดงในแผนที่

รูปที่ 2.24 โลโก้ของ OpenStreetMap

2.6.8 OSRM (Open Source Routing Machine)

OSRM (Open Source Routing Machine) เป็นซอฟต์แวร์โอเพนซอร์สที่ใช้ในการคำนวณเส้นทางและระยะทางบนแผนที่ โดย OSRM ได้รับการพัฒนาโดย Mapbox และถูกออกแบบมาเพื่อใช้กับข้อมูลจาก OpenStreetMap (OSM) ซึ่งทำให้การใช้งานแผนที่และการคำนวณเส้นทางมีประสิทธิภาพสูงและสามารถทำได้ฟรี OSRM ถูกสร้างขึ้นเพื่อช่วยให้นักพัฒนาสามารถเพิ่มฟังก์ชันการนำทางในแอปพลิเคชันหรือเว็บไซต์ได้อย่างง่ายดายโดยไม่ต้องพึ่งพาบริการนำทางจากผู้ให้บริการรายใหญ่เช่น Google Maps หรือ Bing Maps

2.6.8.1 คุณสมบัติและการทำงานของ OSRM

1) การคำนวณเส้นทางที่รวดเร็ว OSRM ถูกออกแบบมาเพื่อให้สามารถคำนวณเส้นทางได้อย่างรวดเร็วมาก โดยเฉพาะอย่างยิ่งสำหรับการนำทางแบบ real-time ซึ่งเหมาะสำหรับแอปพลิเคชันที่ต้องการการตอบสนองเร็ว ยกตัวอย่างเช่น การนำทางแบบ GPS หรือการคำนวณระยะเวลาการเดินทางสำหรับการจัดการขนส่ง

2) การทำงานร่วมกับข้อมูล OpenStreetMap OSRM ใช้ข้อมูลจาก OpenStreetMap (OSM) ทำให้สามารถใช้แผนที่และเส้นทางได้ฟรี โดยสามารถนำไปใช้ได้ทั้งในเชิงพาณิชย์และโครงการส่วนตัวโดยไม่ต้องเสียค่าลิขสิทธิ์

3) รองรับการคำนวณเส้นทางหลายรูปแบบ OSRM รองรับการคำนวณเส้นทางสำหรับการเดินทางหลายประเภท เช่น เส้นทางสำหรับรถยนต์ เส้นทางสำหรับจักรยาน เส้นทางสำหรับการเดินเท้า โดยแต่ละประเภทเส้นทางสามารถปรับแต่งให้เหมาะสมกับการใช้งานจริงได้

4) API ที่ใช้งานง่าย OSRM มี API ที่สะดวกสำหรับนักพัฒนาในการเรียกใช้งาน โดยมีฟังก์ชันต่าง ๆ เช่น การคำนวณเส้นทาง (Route), การหาพิกัดที่ใกล้เคียง (Nearest), การทำแผนที่จับคู่เส้นทาง (Match), และการวิเคราะห์ระยะทางระหว่างจุดต่าง ๆ (Table)

5) รองรับการปรับแต่งและการขยายตัว OSRM เปิดโอกาสให้ผู้ใช้สามารถปรับแต่งการตั้งค่าได้หลากหลาย ไม่ว่าจะเป็นการปรับพารามิเตอร์การคำนวณเส้นทาง การกำหนดน้ำหนักของเส้นทางตามข้อจำกัดต่าง ๆ และการขยายให้รองรับการคำนวณที่ซับซ้อนขึ้น

2.6.9 Docker

Docker เป็นแพลตฟอร์มโอเพนซอร์สที่ใช้ในการสร้าง จัดการ และรันแอปพลิเคชันในรูปแบบของคอนเทนเนอร์ (Container) โดยคอนเทนเนอร์เป็นเทคโนโลยีที่ช่วยแยกแอปพลิเคชันและสภาพแวดล้อมการทำงานของแอปออกจากกัน ทำให้สามารถรันแอปพลิเคชันได้อย่างสม่ำเสมอไม่ว่าจะเป็นการรันบนคอมพิวเตอร์ของนักพัฒนา เซิร์ฟเวอร์ หรือในระบบคลาวด์

2.6.9.1 คุณสมบัติและการทำงานของ Docker

1) คอนเทนเนอร์ (Container) Docker ใช้คอนเทนเนอร์เพื่อแพ็คเกจแอปพลิเคชันพร้อมกับสภาพแวดล้อมที่แอปพลิเคชันนั้นต้องการ ทำให้แอปสามารถรันได้อย่างสม่ำเสมอในทุกที่ที่มี Docker ติดตั้งอยู่ คอนเทนเนอร์จะช่วยลดปัญหาความไม่เข้ากันระหว่างสภาพแวดล้อมของการพัฒนาและการใช้งานจริง (Production)

2) ภาพลักษณ์ (Images) Docker Image เป็นไฟล์ที่ประกอบไปด้วยโค้ดของแอปพลิเคชันและทุกสิ่งที่แอปต้องการเพื่อทำงานได้ เช่น ไลบรารีและไฟล์ระบบ นักพัฒนาสามารถสร้าง Image ของแอปและนำไปใช้งานซ้ำได้ง่าย โดยไม่ต้องติดตั้งและตั้งค่าทุกอย่างใหม่

3) ความสามารถในการใช้งานข้ามแพลตฟอร์ม Docker รองรับการใช้งานได้ทั้งบนระบบปฏิบัติการ Linux, Windows, และ macOS ทำให้แอปพลิเคชันสามารถย้ายจากเครื่องหนึ่งไปยังอีกเครื่องหนึ่งได้โดยไม่มีปัญหาเกี่ยวกับสภาพแวดล้อม

4) การปรับแต่งและขยาย Docker ทำให้การสร้างสภาพแวดล้อมที่ซับซ้อน เช่นการทำงานร่วมกับฐานข้อมูลหรือบริการอื่นๆ เป็นเรื่องง่าย เพราะสามารถใช้ Docker Compose ซึ่งเป็นเครื่องมือที่ใช้กำหนดคอนฟิกหลายคอนเทนเนอร์ในไฟล์ YAML ไฟล์เดียวได้

5) Docker Hub Docker Hub เป็นบริการคลังเก็บภาพลักษณ์ของ Docker ที่ใช้สำหรับแชร์และดึง Image ของแอปพลิเคชัน ซึ่งนักพัฒนาสามารถดึง Image ที่สร้างไว้แล้วมาใช้งานหรือแบ่งปัน Image ของตนเองให้ผู้อื่นได้อย่างสะดวก

2.6.10 MySQL

MySQL แสดงได้ดังรูปที่ 2.25 เป็นระบบจัดการฐานข้อมูลเชิงสัมพันธ์ (Relational Database Management System - RDBMS) ที่ใช้ภาษา SQL (Structured Query Language) ในการจัดการและจัดเก็บข้อมูล โดย MySQL เป็นซอฟต์แวร์โอเพนซอร์สและได้รับการพัฒนาครั้งแรกโดยบริษัท MySQL AB ในปี 1995 ซึ่งปัจจุบันถูกดูแลโดย Oracle Corporation MySQL ได้รับความนิยมอย่างกว้างขวางในหมู่นักพัฒนาและองค์กรทั่วโลกเนื่องจากมีความเสถียร ประสิทธิภาพสูง และใช้งานง่าย

2.6.10.1 คุณสมบัติและการทำงานของ MySQL

1) โครงสร้างเชิงสัมพันธ์ (Relational Structure) MySQL ใช้โครงสร้างเชิงสัมพันธ์ในการจัดเก็บข้อมูล ซึ่งข้อมูลจะถูกจัดเก็บในรูปแบบของตาราง (Tables) ที่เชื่อมโยงกันด้วยคีย์ (Keys) ทำให้ง่ายต่อการจัดการข้อมูลและรักษาความสอดคล้องของข้อมูล

2) การรองรับคำสั่ง SQL MySQL ใช้คำสั่ง SQL ในการดำเนินการต่าง ๆ เช่น การเพิ่มข้อมูล (INSERT), การอ่านข้อมูล (SELECT), การแก้ไขข้อมูล (UPDATE), และการลบข้อมูล (DELETE) รวมถึงการจัดการโครงสร้างของฐานข้อมูล

3) ความปลอดภัยและการเข้าถึงที่ควบคุมได้ MySQL มีระบบการจัดการผู้ใช้ที่ช่วยให้ผู้ดูแลสามารถกำหนดสิทธิ์การเข้าถึงข้อมูลและการกระทำต่างๆ ในฐานข้อมูลได้ เช่น กำหนดว่าใครสามารถอ่าน เขียน หรือลบข้อมูลได้

4) รองรับการใช้งานร่วมกับแอปพลิเคชันหลายประเภท MySQL สามารถใช้งานร่วมกับภาษาโปรแกรมหลากหลาย ยกตัวอย่างเช่น PHP, Python, Java, และ Node.js ทำให้เหมาะสำหรับการพัฒนาเว็บแอปพลิเคชันและระบบที่ต้องการการจัดการข้อมูล

รูปที่ 2.25 โลโก้ของ MySQL

2.6.11 Google Maps API

Google Maps API เป็นชุดเครื่องมือที่พัฒนาโดย Google เพื่อให้บริการแผนที่และฟังก์ชันการนำทางแบบเชิงโต้ตอบสำหรับการพัฒนาแอปพลิเคชันและเว็บไซต์ Google Maps API ช่วยให้นักพัฒนาสามารถฝังแผนที่และข้อมูลภูมิศาสตร์ลงในแอปพลิเคชันหรือเว็บไซต์ของตนเอง รวมถึงสามารถเข้าถึงฟังก์ชันต่างๆ เช่น การคำนวณเส้นทาง การหาสถานที่ การค้นหาพิกัด และการแสดงข้อมูลจราจรแบบเรียลไทม์

2.6.11.1 คุณสมบัติและการทำงานของ Google Maps API

1) การฝังแผนที่ Google Maps API ช่วยให้ผู้ใช้สามารถฝังแผนที่ที่ปรับแต่งได้ลงในเว็บไซต์หรือแอปพลิเคชัน ทำให้ผู้ใช้สามารถดูแผนที่ เลือกแสดงข้อมูลเฉพาะ และย่อ/ขยายแผนที่ได้ตามความต้องการ

2) Geocoding และ Reverse Geocoding Google Maps API รองรับการแปลงที่อยู่ให้เป็นพิกัด (Geocoding) และแปลงพิกัดกลับเป็นที่อยู่ (Reverse Geocoding) ซึ่งช่วยให้การหาตำแหน่งของสถานที่หรือแปลงพิกัดให้เป็นที่อยู่สามารถทำได้อย่างสะดวก

3) Directions API รองรับการคำนวณเส้นทางระหว่างจุดต่างๆ โดยสามารถระบุประเภทการเดินทางได้ เช่น เดินเท้า ขี่จักรยาน ขับรถยนต์ หรือใช้ระบบขนส่งสาธารณะ รวมถึงสามารถคำนวณระยะเวลาการเดินทางและเสนอเส้นทางที่เหมาะสมได้

4) Distance Matrix API ช่วยในการคำนวณระยะทางและเวลาในการเดินทางระหว่างหลายตำแหน่ง ทำให้การคำนวณระยะทางระหว่างจุดต่างๆ ง่ายและสะดวก โดยเฉพาะในการสร้างระบบที่ต้องการการคำนวณระยะทางในเชิงโลจิสติกส์

5) Places API ใช้สำหรับค้นหาข้อมูลสถานที่ เช่น ร้านค้า ร้านอาหาร โรงแรม ปั๊มน้ำมัน และสถานที่สำคัญอื่นๆ รวมถึงสามารถกรองข้อมูลตามประเภทหรือระยะทางจากตำแหน่งที่กำหนด

6) Street View API ช่วยให้ผู้ใช้สามารถดูภาพ Street View ที่เป็นภาพถ่าย 360 องศาในมุมมองระดับถนน ซึ่งช่วยเพิ่มประสบการณ์ในการดูแผนที่ให้เสมือนจริงมากยิ่งขึ้น

7) Traffic Layer และ Real-time Data Google Maps API ให้ข้อมูลจราจรแบบเรียลไทม์บนแผนที่ ช่วยให้ผู้ใช้สามารถดูสภาพการจราจรในปัจจุบัน รวมถึงใช้ในการคำนวณเส้นทางที่เหมาะสมตามสภาพการจราจรได้

2.6.12 VM Instance บน Google Cloud Platform

Google Cloud Platform (GCP) แสดงได้ดังรูปที่ 2.26 เป็นแพลตฟอร์มคลาวด์ที่ให้บริการโครงสร้างพื้นฐานเสมือน (Infrastructure as a Service: IaaS) ผ่าน Compute Engine ซึ่งช่วยให้สามารถสร้าง Virtual Machine (VM) Instance สำหรับรันแอปพลิเคชันหรือบริการต่างๆ ได้

รูปที่ 2.26 Google Cloud Platform

ตารางที่ 2.12 ข้อมูลจำเพาะของ VM Instance บน Google Cloud Platform

ข้อมูล	รายละเอียด
Machine Type	e2-medium (2 vCPU, 4 GB RAM)
Operating System	Ubuntu 20.04 LTS
Disk Type	Standard Persistent Disk
Disk Size	20GB
Network Configuration	External IP, Firewall Rules
Additional Features	การใช้ Auto-scaling, Load Balancer, หรือการกำหนดค่าให้รองรับ Containerized Application (Docker, Kubernetes)

บทที่ 3

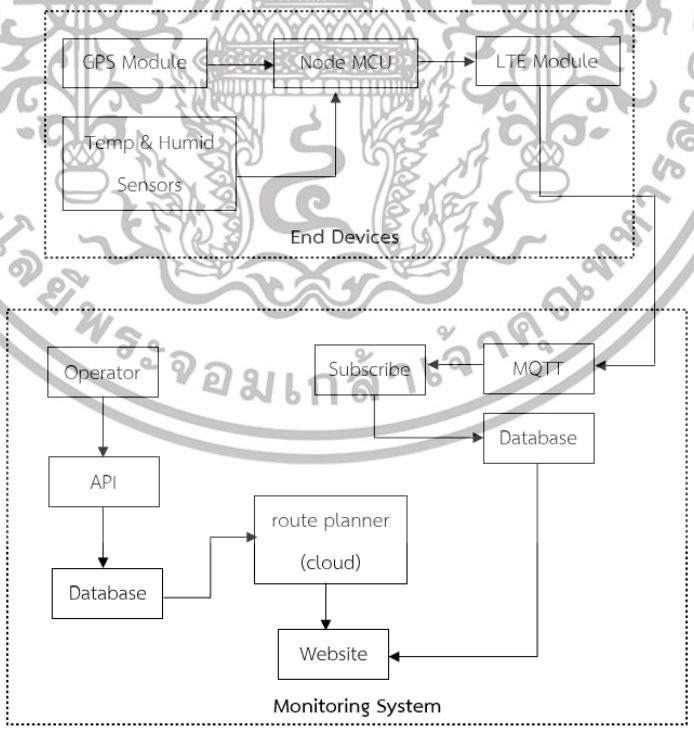
การออกแบบและการจัดทำปริญญานิพนธ์

3.1 การออกแบบ

ระบบติดตามการขนส่งและการวางแผนเส้นทางอัจฉริยะนี้ถูกออกแบบมาเพื่อติดตามข้อมูลสภาพแวดล้อมในระหว่างการขนส่ง เช่น อุณหภูมิ, ความชื้น และตำแหน่ง โดยอุปกรณ์ปลายทาง (End Devices) จะรวบรวมข้อมูลเหล่านี้ผ่านเซ็นเซอร์และโมดูล GPS จากนั้นจะส่งข้อมูลผ่านเครือข่าย LTE ไปยัง Database ซึ่งประกอบด้วยสองส่วน ได้แก่

1) End Devices ประกอบด้วยโมดูล GPS, Node MCU, โมดูล LTE และเซ็นเซอร์วัดอุณหภูมิและความชื้น ซึ่งทำหน้าที่ในการเก็บและส่งข้อมูลสภาพแวดล้อม

2) Monitoring System เป็นระบบรวบรวมข้อมูลและวางแผนเส้นทาง โดยใช้การเชื่อมต่อกับ MQTT Broker เพื่อรับข้อมูลเซ็นเซอร์และบันทึกลง Database จากนั้นระบบจะใช้การวิเคราะห์เส้นทาง (Route Planner) บนคลาวด์เพื่อคำนวณเส้นทางที่สั้นที่สุดระหว่างต้นทางและปลายทาง แสดงได้ดังรูปที่ 3.1

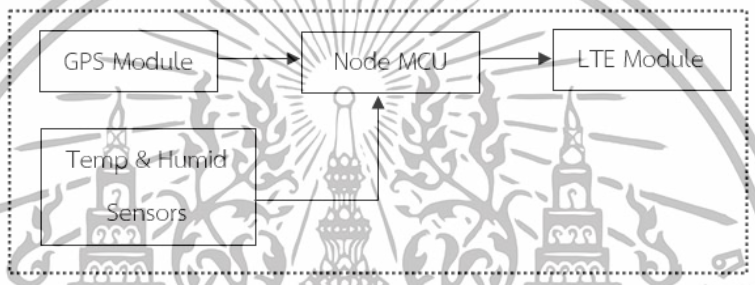


รูปที่ 3.1 บล็อกไดอะแกรมการทำงานของระบบทั้งหมด

3.1.1 การออกแบบการทำงานของระบบติดตามการขนส่ง

3.1.1.1 การออกแบบการทำงานในส่วนฮาร์ดแวร์

ในการทำงานของระบบติดตามการขนส่งในส่วนฮาร์ดแวร์จะใช้ GPS Module , เซนเซอร์วัดอุณหภูมิ และเซนเซอร์วัดความชื้น โดยที่ เซนเซอร์วัดอุณหภูมิ และวัดความชื้น ทำการส่งข้อมูลโดยใช้บลูทูธพลังงานต่ำ (Bluetooth low energy) ส่งข้อมูลไปยัง Node MCU จากนั้นจะส่งข้อมูลไปยัง Database โดยผ่าน MQTT Broker ด้วยเครือข่าย LTE แสดงได้ดังรูปที่ 3.2



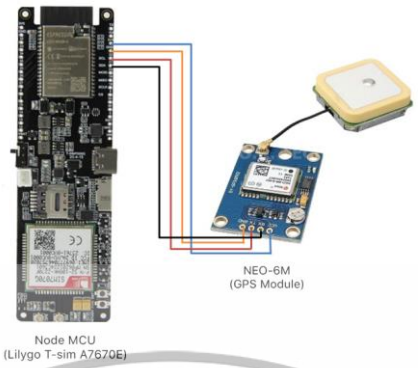
รูปที่ 3.2 บล็อกไดอะแกรมการทำงานของระบบติดตามการขนส่งในส่วนฮาร์ดแวร์

1) การออกแบบการทดสอบการทำงานของ GPS Module

ในการทดสอบนี้จะเป็นการทดสอบการใช้งานของ GPS Module ร่วมกับ Node MCU ในการทดสอบ โดยทำการเขียนโปรแกรมผ่านโปรแกรม Arduino IDE และใช้ Library TinyGPS++ เพื่อแสดงพิกัดที่อยู่ปัจจุบัน แสดงขาการเชื่อมต่อดังตารางที่ 3.1 และการต่อวงจรดังรูปที่ 3.3

ตารางที่ 3.1 ขาการเชื่อมต่อ Node MCU และ GPS Module

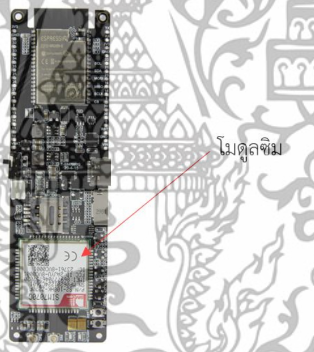
Node MCU (Lilygo A7670E)	GPS Module
3V3	VCC
GND	GND
IO22	RX
IO21	TX



รูปที่ 3.3 การต่อวงจรของ GPS Module กับ Node MCU

2) การออกแบบการทดสอบการทำงานของ Node MCU และโมดูลซิม 7070G

ในการทดสอบนี้เป็นการตรวจสอบการทำงานร่วมกันระหว่างโมดูลซิม A7670E และ Node MCU LilyGO A7670E โดยจำเป็นต้องกำหนดขาการเชื่อมต่ออย่างถูกต้อง เนื่องจากโมดูลรวมอยู่กับ Node MCU แสดงโมดูลซิมดังรูปที่ 3.4 และขาการเชื่อมต่อดังตารางที่ 3.2



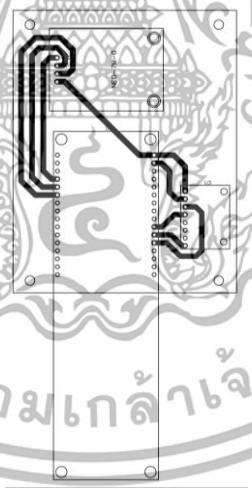
รูปที่ 3.4 โมดูลซิม A7670E

ตารางที่ 3.2 ขาการเชื่อมต่อ Node MCU และ LTE A7670E Module

Node MCU (Lilygo T-Sim7070G)	LTE A7670E Module
25	DTR
33	RI
26	TX
27	RX

3) การออกแบบแผงวงจรพิมพ์ (PCB) และกล่องอุปกรณ์ด้วยการพิมพ์สามมิติ (3D Printing)

การออกแบบแผงวงจรพิมพ์ (PCB) ใช้ ซอฟต์แวร์ EasyEDA ซึ่งเป็นเครื่องมือที่ช่วยให้สามารถออกแบบวงจรอิเล็กทรอนิกส์และสร้างต้นแบบแผงวงจรได้อย่างแม่นยำ แผงวงจรนี้รวมอุปกรณ์หลัก เช่น ESP32 LilyGO T-SIM7670E, GPS Module และโมดูลซิม A7670E เพื่อให้ทำงานร่วมกันได้อย่างเสถียร นอกจากนี้ ได้มีการกำหนดตำแหน่งขั้วต่อไฟฟ้าและจุดเชื่อมต่อสัญญาณให้เหมาะสม เพื่อลดสัญญาณรบกวนและเพิ่มประสิทธิภาพของระบบ แสดงได้ดังรูปที่ 3.5

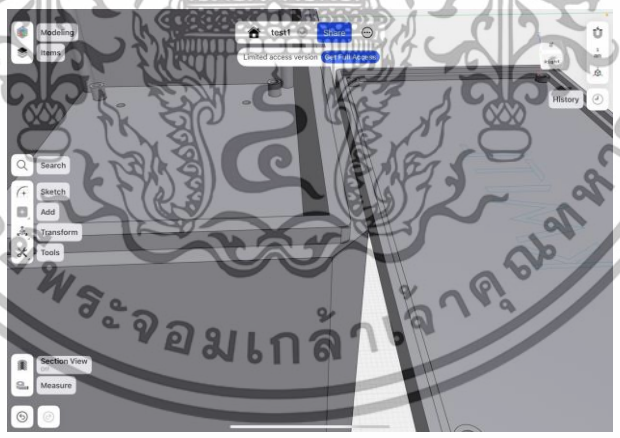


รูปที่ 3.5 การออกแบบแผงวงจรพิมพ์ (PCB)

การออกแบบกล่องอุปกรณ์ด้วยการพิมพ์สามมิติ (3D Printing)

กล่องอุปกรณ์ได้รับการออกแบบโดยใช้ ซอฟต์แวร์ Shapr3D ซึ่งช่วยให้สามารถสร้างแบบจำลองสามมิติของกล่องได้อย่างละเอียด ก่อนนำไปผลิตด้วยเครื่องพิมพ์สามมิติ (3D Printer) ในการออกแบบกล่องได้คำนึงถึงปัจจัยสำคัญ ได้แก่ การจัดวางอุปกรณ์ภายใน และ ความแข็งแรงและความทนทาน แสดงได้ดังรูปที่ 3.6 และรูปที่ 3.7

รูปที่ 3.6 การออกแบบกล่องอุปกรณ์ด้วยการพิมพ์สามมิติ (3D Printing)

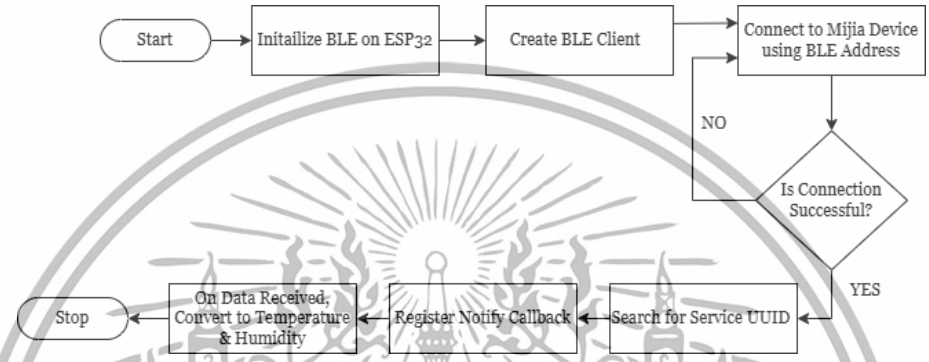


รูปที่ 3.7 การออกแบบกล่องอุปกรณ์ด้วยการพิมพ์สามมิติ (3D Printing)

3.1.1.2 การออกแบบการทำงานในส่วนซอฟต์แวร์

1) การออกแบบการทดสอบการเชื่อมต่อของ Mi Temperature And Humidity Monitor 2 และ Node MCU (Lilygo A7670E)

ในการทดสอบการรับค่าจาก Mi Temperature And Humidity Monitor 2 จะทำการสื่อสารผ่านทาง Bluetooth Low Energy (BLE) โดยใช้ไลบรารี่ BLEDevice.h ซึ่งในการเชื่อมต่อจะต้องระบุ Mac address เพื่อเป็นการเจาะจงตัวอุปกรณ์ ในการรับค่าอุณหภูมิและความชื้น จะต้องระบุค่า Service ID ซึ่งสามารถค้นหาได้จากแอปพลิเคชัน BLE Device ในการทดสอบการทำงาน แสดงได้ดังรูปที่ 3.8

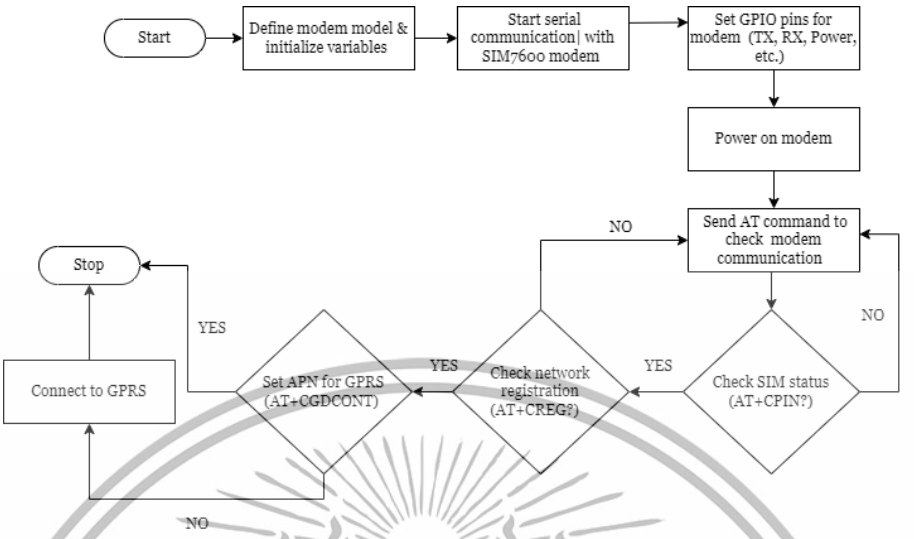


รูปที่ 3.8 แผนผังการทำงานของการเชื่อมต่อ Mi Temperature And Humidity Monitor 2

2) การออกแบบการทดสอบการทำงานของโมดูลซิม A7076E

การเชื่อมต่อโมดูลซิม A7070G กับ Node MCU (Lilygo T-Sim7070G) ซึ่งเริ่มต้นด้วยการกำหนดโมเดลของโมเด็มและตั้งค่าตัวแปร จากนั้นจึงเริ่มการสื่อสารผ่านพอร์ตซีเรียลกับโมเด็ม SIM7600 พร้อมตั้งค่า GPIO ที่จำเป็นและเปิดการทำงานของโมเด็ม

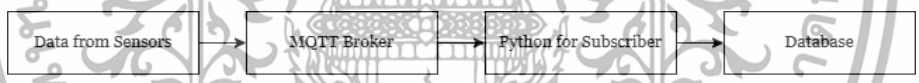
ระบบจะส่งคำสั่ง AT เพื่อทดสอบการสื่อสารกับโมเด็ม หากการสื่อสารไม่สำเร็จ จะวนกลับไปทดสอบใหม่ เมื่อเช็คสถานะซิมการ์ดและการลงทะเบียนเครือข่ายเรียบร้อยแล้ว จะกำหนดค่า APN เพื่อเชื่อมต่อ GPRS และเข้าสู่สถานะพร้อมใช้งาน แสดงได้ดังรูปที่ 3.9



รูปที่ 3.9 แผนผังการทำงานของโมดูลซิม A7670E กับ Node MCU (Lilygo T-Sim7670E)

3) การออกแบบการทดสอบของการส่งข้อมูลผ่าน MQTT Broker

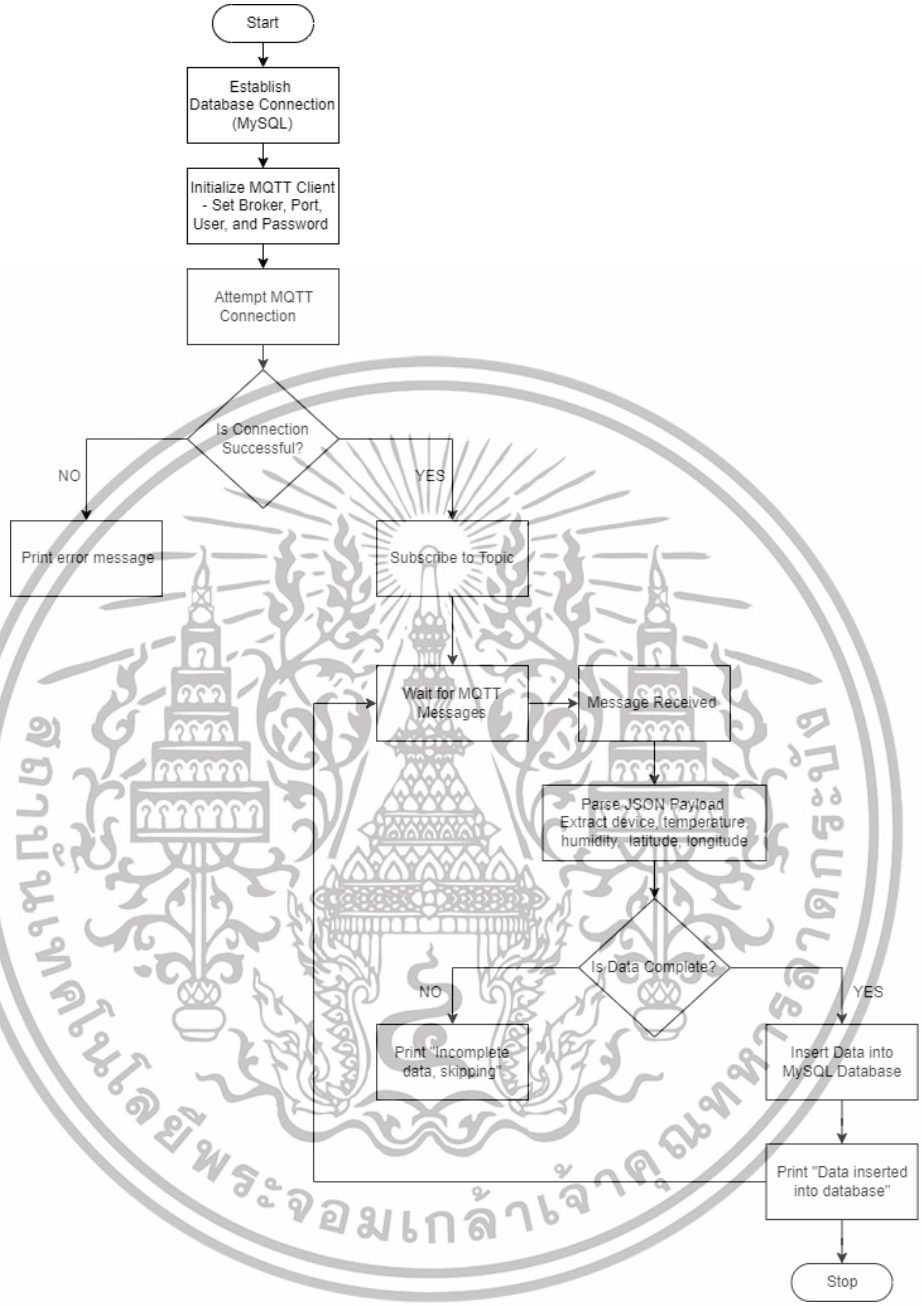
ในขั้นตอนการส่งข้อมูลจาก MQTT Broker โดยเริ่มจากเซ็นเซอร์ส่งข้อมูลไปยัง MQTT Broker จากนั้นมีสคริปต์ Python ทำหน้าที่เป็น Subscriber เพื่อรับข้อมูลจาก Broker ก่อนที่จะบันทึกลงในฐานข้อมูล แสดงได้ดังรูปที่ 3.10



รูปที่ 3.10 การส่งข้อมูลผ่าน MQTT Broker

4) การออกแบบการทดสอบโค้ด Python ที่รับข้อมูลมาจาก MQTT Broker และบันทึกข้อมูลไปยัง Database

ในขั้นตอนนี้จะใช้การเขียนโปรแกรมภาษา Python เพื่อรับข้อมูลที่ถูกส่งเข้าสู่ MQTT Broker โดยข้อมูลประกอบด้วยค่าต่าง ๆ ได้แก่ อุณหภูมิ, ความชื้น, และพิกัด หลังจากที่ Python รับข้อมูลเหล่านี้จาก MQTT Broker แล้ว จะทำการประมวลผลและบันทึกลงในฐานข้อมูล แสดงตัวอย่างได้ดังรูปที่ 3.11

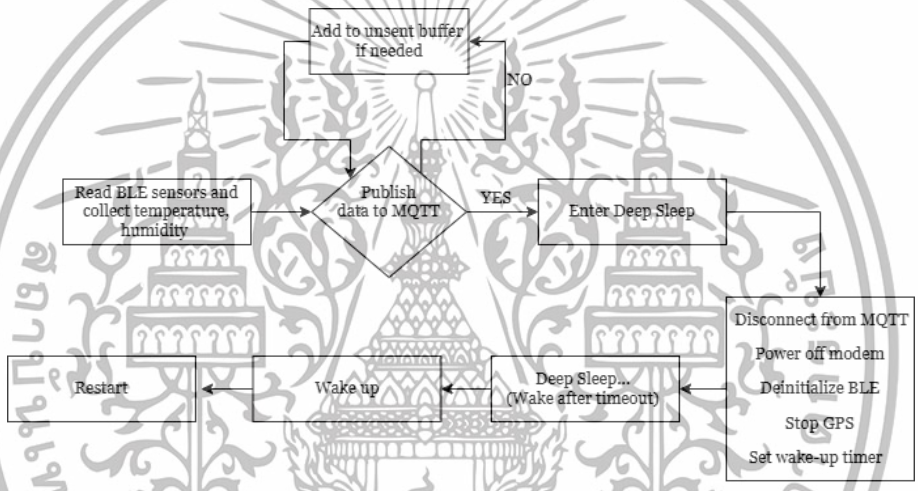


รูปที่ 3.11 แผนผังการทำงานของสคริปต์ Python

5) การออกแบบการทดสอบ Sleepmode ของ Node MCU (Lilygo A7670E)

ในการทดสอบนี้ออกแบบมาเพื่อประหยัดพลังงาน โดยหลังจากการส่งข้อมูลไปยัง MQTT Broker หากมีข้อมูลที่ยังไม่ได้ส่ง ระบบจะเก็บไว้ในบัฟเฟอร์ก่อน เมื่อการส่งข้อมูลสำเร็จแล้ว จะเข้าสู่โหมด Deep Sleep เพื่อประหยัดพลังงาน

ในกระบวนการเข้าสู่โหมด Deep Sleep จะมีการตัดการเชื่อมต่อและหยุดการทำงานของส่วนต่าง ๆ เช่น ตัดการเชื่อมต่อจาก MQTT, ตัดการทำงานของโมดูล ,หยุดการทำงานของ GPS แสดงได้ดังรูปที่ 3.12

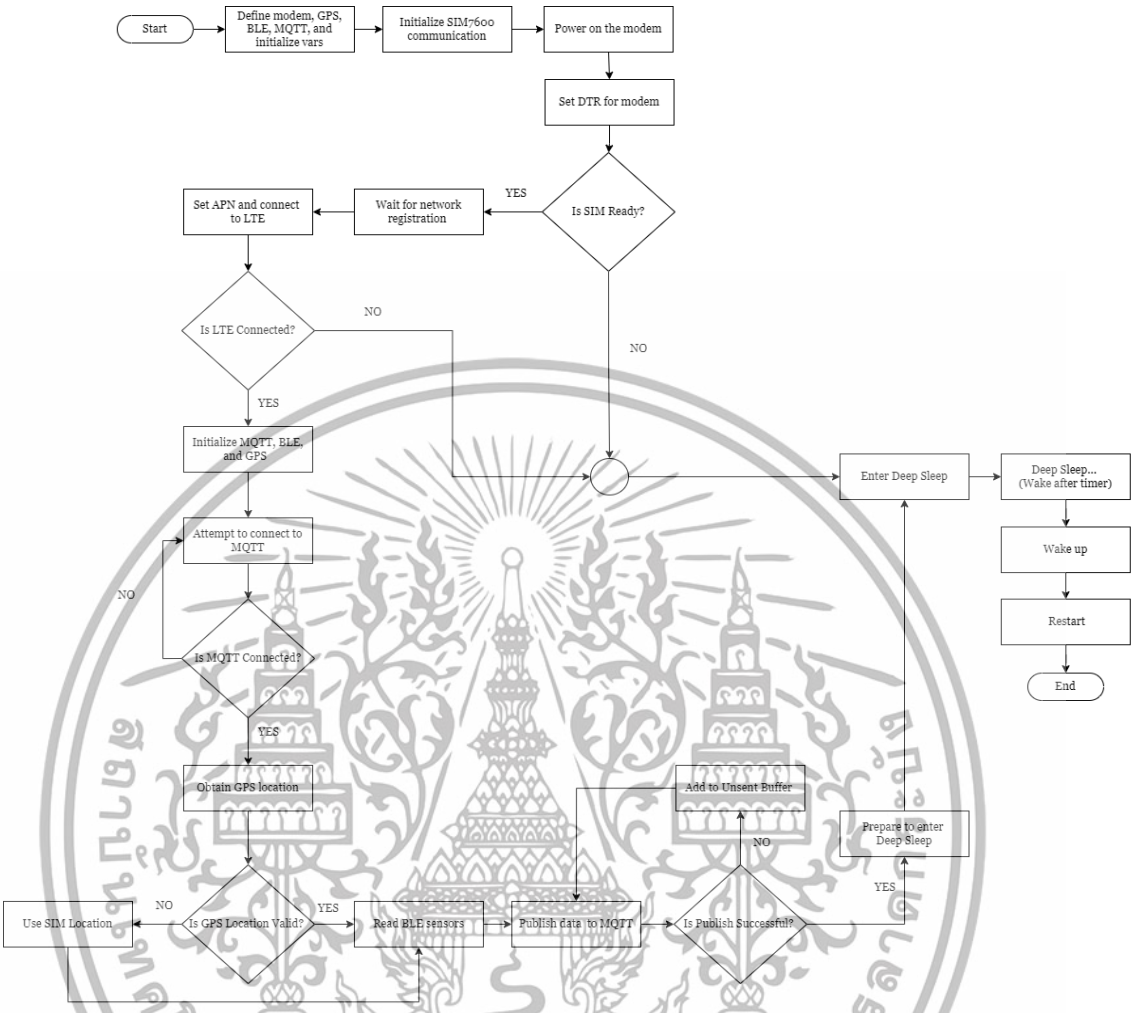


รูปที่ 3.12 แผนผังการทำงาน Sleepmode ของ Node MCU

6) การออกแบบการทำงานของระบบติดตามการขนส่ง

ในการออกแบบการทำงานของระบบติดตามการขนส่งเริ่มจากการตั้งค่าอุปกรณ์ โดยเริ่มจากเปิดใช้งานโมเด็ม, GPS, BLE จากนั้นเชื่อมต่อกับโมเด็มผ่านพอร์ตซีเรียลพร้อมกำหนดค่า DTR เมื่อโมเด็มทำงาน ระบบจะตรวจสอบสถานะซิม หากซิมพร้อม ระบบจะตั้งค่า APN และเชื่อมต่อกับ LTE พร้อมรอการลงทะเบียนเครือข่าย

หากเชื่อมต่อ LTE ได้สำเร็จ ระบบจะเริ่มใช้งาน MQTT, BLE, และ GPS จากนั้นจะพยายามเชื่อมต่อกับ MQTT Broker เมื่อเชื่อมต่อสำเร็จ จะตรวจสอบตำแหน่ง GPS และอ่านค่าจากเซ็นเซอร์ BLE ก่อนส่งข้อมูลไปยัง MQTT หากส่งข้อมูลสำเร็จ ระบบจะเข้าสู่โหมด Deep Sleep แต่หากการส่งล้มเหลว ข้อมูลจะถูกเก็บในบัฟเฟอร์ก่อนเข้าสู่ Deep sleep และส่งใหม่ในรอบถัดไป แสดงได้ดังรูปที่ 3.13

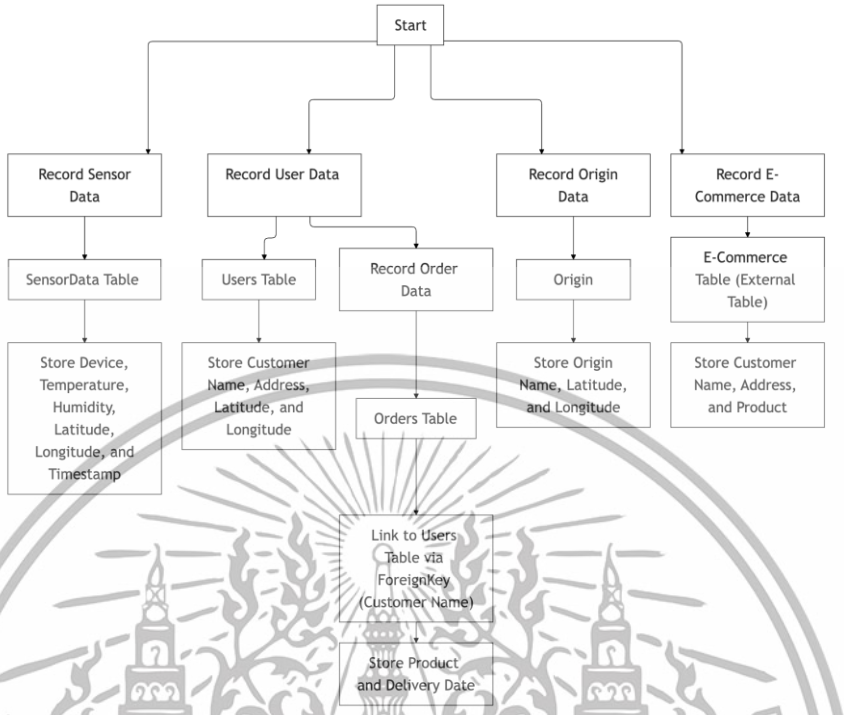


รูปที่ 3.13 แผนผังการทำงานของระบบติดตามการขนส่ง

3.1.2 ออกแบบการสร้างตารางฐานข้อมูลเพื่อให้สามารถใช้งานข้อมูลได้ในส่วนระบบหลังบ้าน

3.1.2.1 การออกแบบตารางฐานข้อมูล MySQL

การออกแบบการทำงานของระบบนี้เน้นไปที่การพัฒนาระบบให้มีความเป็นระเบียบตามแบบแผนโดยจะทำการออกแบบข้อมูลตาราง MySQL ร่วมกับการใช้ Tool ใน Django ในการทำ make migration และ migrate ซึ่งแสดงตามแผนผังการออกแบบตารางฐานข้อมูลดังรูปที่ 3.14



รูปที่ 3.14 แผนผังการออกแบบระบบฐานข้อมูล

3.1.3 การออกแบบการทำงานของระบบค้นหาเส้นทาง

3.1.3.1 การออกแบบการทำงานในส่วนซอฟต์แวร์

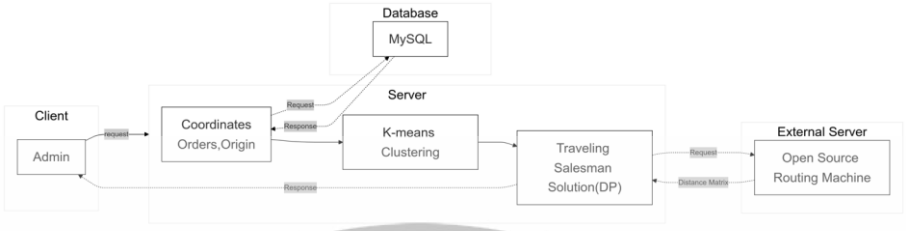
ในการออกแบบการทำงานของระบบค้นหาเส้นทางอัจฉริยะในส่วนซอฟต์แวร์ระบบได้รับการพัฒนาให้รองรับการจัดลำดับเส้นทางที่สั้นที่สุดสำหรับการจัดส่งโดย Django เป็น Backend API ที่ทำหน้าที่ในการคำนวณเส้นทางและจัดลำดับเส้นทางอัตโนมัติเพื่อช่วยให้พนักงานขนสามารถส่งสินค้าได้มีประสิทธิภาพและเป็นระบบแบบแผนโดยผ่านกระบวนการและเทคนิคต่าง ๆ ซึ่งประกอบด้วย

K-means Cluster เพื่อใช้ในการแบ่งกลุ่มของรายการสินค้าออกเป็นกลุ่มต่างๆเพื่อเตรียมความพร้อมในการจัดลำดับเส้นทาง

Traveling Salesman Problem (TSP) โดยทดสอบด้วยเทคนิคแบบ Brute Force และ Dynamic Programming (DP) Algorithm เพื่อใช้ในการจัดลำดับเส้นที่สั้นที่สุดจากDistance Matrix ระหว่างพิกัดที่อยู่ของรายการสินค้าที่ต้องส่ง

OSRM (Open-Source Routing Machine) ใช้สำหรับการส่งคำขอเพื่อดึงข้อมูลระยะทางในรูปแบบ Distance Matrix จากพิกัดของลูกค้าแต่ละราย

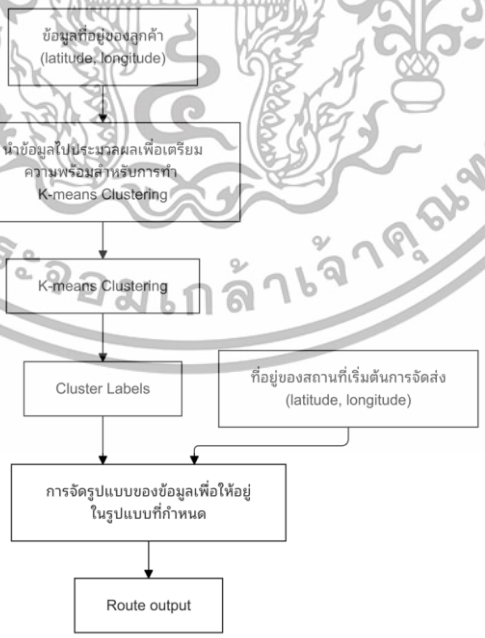
MySQL เพื่อใช้ในการจัดเก็บข้อมูลพิกัดที่ตั้งของลูกค้าแต่ละราย รวมถึงจัดเก็บข้อมูลรายการคำสั่งซื้อและวันที่กำหนดในการจัดส่ง แสดงได้ดังรูปที่ 3.15



รูปที่ 3.15 แผนผังการทำงานของระบบค้นหาเส้นทางอัจฉริยะในส่วนหลังบ้าน

1) การออกแบบระบบทดสอบประสิทธิภาพของการแบ่งกลุ่มด้วยวิธีแบบ K-means Clustering

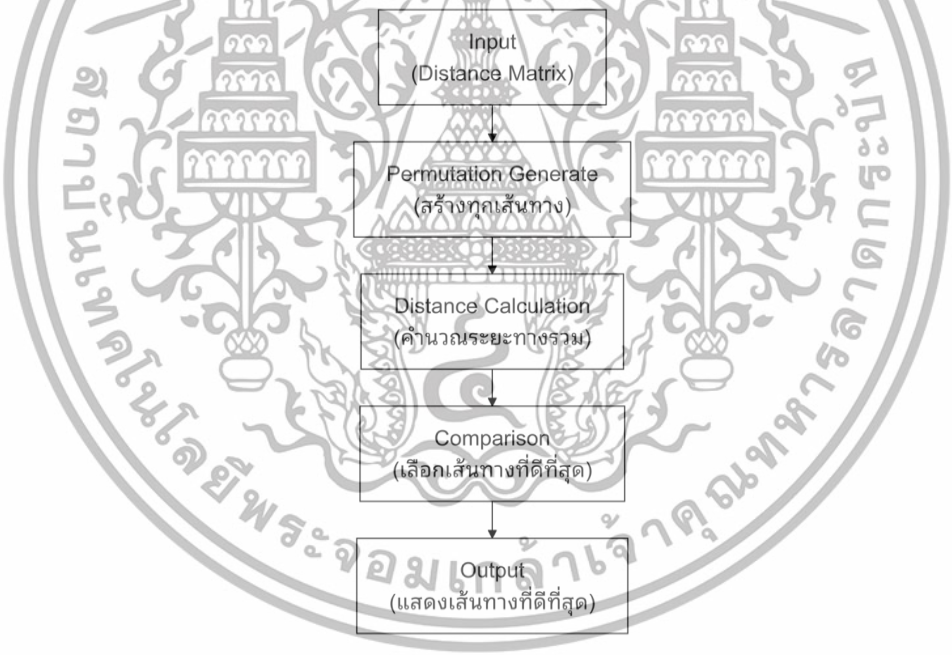
ในการทดสอบนี้เป็นการทดสอบประสิทธิภาพของการแบ่งกลุ่มข้อมูลด้วยวิธี K-means Clustering ซึ่งเป็นวิธีการที่นิยมใช้ในงานวิเคราะห์ข้อมูลเพื่อการจัดกลุ่ม (clustering) ข้อมูลที่มีลักษณะคล้ายคลึงกัน โดยจะทำการสมมติค่าของ Distance Matrix และจุดเริ่มต้น (origin) จากนั้นจะทำการแบ่งกลุ่มด้วยวิธีแบบ K-means Clustering ซึ่งจะทำการเขียนโปรแกรม Python โดยใช้ Library scikit-learn เพื่อใช้ในการแบ่งกลุ่มข้อมูลให้อยู่ในลักษณะที่ใกล้เคียงกันโดยมีขั้นตอนการทำงานดังรูปที่ 3.16



รูปที่ 3.16 แผนผังการทำงานของการแบ่งกลุ่มด้วยวิธีแบบ K-means Cluster

2) การออกแบบระบบทดสอบประสิทธิภาพของการจัดลำดับเส้นทางที่สั้นที่สุดด้วยวิธีแบบ Brute Force

ในส่วนนี้จะออกแบบระบบทดสอบประสิทธิภาพของการจัดลำดับเส้นทางที่สั้นที่สุดโดยใช้วิธี Brute Force ซึ่งเป็นวิธีการที่ตรวจสอบทุกเส้นทางที่เป็นไปได้เพื่อหาทางเลือกที่มีระยะทางรวมสั้นที่สุด วิธีนี้เหมาะสมสำหรับจำนวนจุดที่ไม่มากนัก เนื่องจากมีความซับซ้อนในเชิงเวลา (time complexity) ที่สูงโดยการทดสอบจะนำข้อมูลที่ได้จากการทำการแบ่งกลุ่มด้วยวิธีแบบ K-means Clustering มาทำการจัดลำดับเส้นทางที่สั้นที่สุดในแต่ละกลุ่มด้วยภาษา Python โดยจะใช้ Library itertools เพื่อสร้างลำดับการเดินทางที่เป็นไปได้ทั้งหมดสำหรับจุดหมายในกลุ่มนั้น ๆ จากนั้นจะคำนวณระยะทางรวมของแต่ละลำดับ และเปรียบเทียบเพื่อตรวจหาลำดับที่มีระยะทางสั้นที่สุด ในการทดสอบนี้จะบันทึกค่าต่าง ๆ ได้แก่ เวลาที่ใช้ในการคำนวณ ลำดับเส้นทางที่ได้ และระยะทางรวม ซึ่งจะแสดงแผนผังของระบบการทำงานดังรูปที่ 3.17



รูปที่ 3.17 แผนผังการทำงานของจัดลำดับเส้นทางด้วยวิธีแบบ Brute Force

3) การออกแบบการทดสอบประสิทธิภาพของการจัดลำดับเส้นทางที่สั้นที่สุดด้วยวิธีแบบ Dynamic Programming

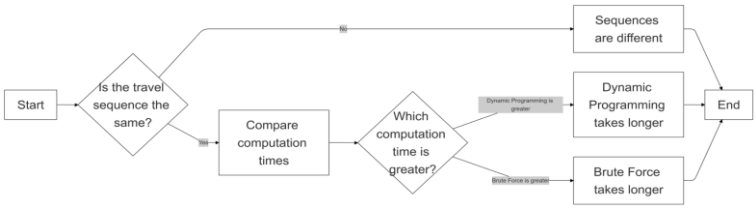
ในการทดสอบนี้ เป็นการประเมินประสิทธิภาพของการจัดลำดับเส้นทางที่สั้นที่สุดโดยใช้วิธี Dynamic Programming ซึ่งช่วยลดความซับซ้อนในเชิงเวลาเมื่อเปรียบเทียบกับวิธี Brute Force โดยเฉพาะอย่างยิ่งในกรณีที่จำนวนจุดที่ต้องการจัดลำดับมีมากขึ้น Dynamic Programming สามารถลดจำนวนการคำนวณซ้ำที่ไม่จำเป็น โดยบันทึกผลลัพธ์ของการคำนวณในแต่ละขั้นตอนเพื่อใช้ซ้ำได้ ซึ่งทำให้การคำนวณมีประสิทธิภาพมากขึ้น

การทดสอบนี้จะนำข้อมูลจากการแบ่งกลุ่มด้วยวิธี K-means Clustering มาทำการจัดลำดับเส้นทางที่สั้นที่สุดในแต่ละกลุ่มด้วยภาษา Python ซึ่งใช้เทคนิคการแก้ปัญหา Traveling Salesman Problem โดยใช้วิธีแบบ Dynamic programming โดยในการทดสอบนี้จะบันทึกค่าต่าง ๆ ซึ่งประกอบไปด้วย เวลาที่ใช้ในการคำนวณ ลำดับเส้นทางที่ได้ ระยะทางรวม ซึ่งจะบันทึกระยะทางรวมของเส้นทางที่สั้นที่สุดเพื่อนำมาเปรียบเทียบกับวิธีการอื่น ๆ ในการทดสอบแสดงแผนผังของระบบการทำงานดังรูปที่ 3.18

รูปที่ 3.18 แผนผังการทำงานของจัดลำดับเส้นทางด้วยวิธีแบบ Dynamic programming

4) การออกแบบการทดสอบเปรียบเทียบประสิทธิภาพวิธีการจัดลำดับเส้นทางที่สั้นที่สุดระหว่าง Brute Force และ Dynamic Programming

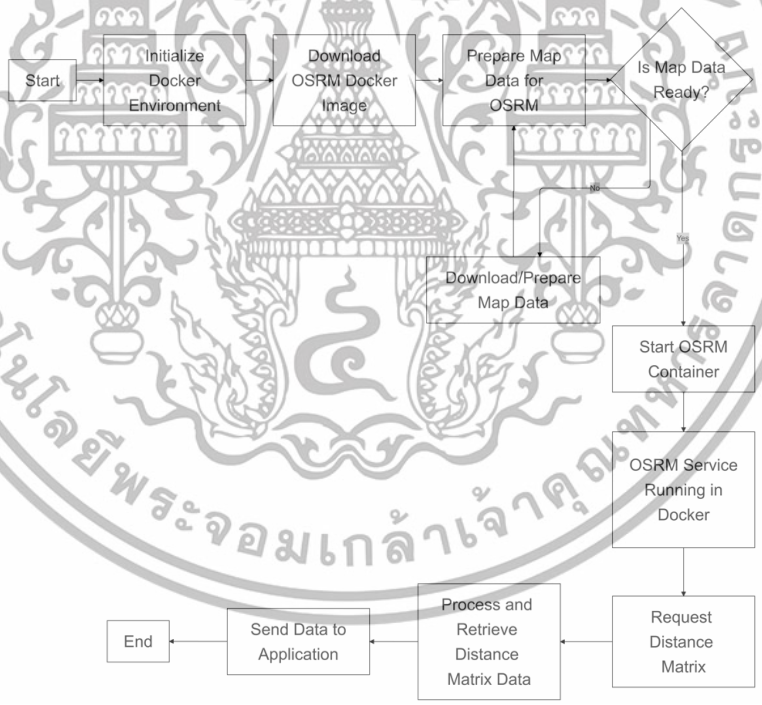
ในการออกแบบการทดสอบนี้ เราต้องการเปรียบเทียบประสิทธิภาพระหว่างวิธี Brute Force และ Dynamic Programming ในการหาลำดับเส้นทางที่สั้นที่สุดสำหรับปัญหา Traveling Salesman Problem (TSP) โดยเฉพาะเมื่อจำนวนจุดที่ต้องเดินทางมีจำนวนเพิ่มขึ้นเรื่อย ๆ โดยจะทำการเปรียบเทียบผลของเส้นทางและระยะทางทั้งหมดในการเดินทางว่าเท่ากันหรือไม่และทำการเปรียบเทียบเวลาที่ใช้ว่าประสิทธิภาพเวลาในการคำนวนแตกต่างกันมากน้อยเพียงใดดังรูปที่ 3.19



รูปที่ 3.19 แผนผังการเปรียบเทียบระหว่างวิธีแบบ Brute Force และ Dynamic Programming

5) การออกแบบระบบการติดตั้ง OSRM (Open Source Routing Machine) ใน Docker เพื่อใช้การขอข้อมูล Distance Matrix

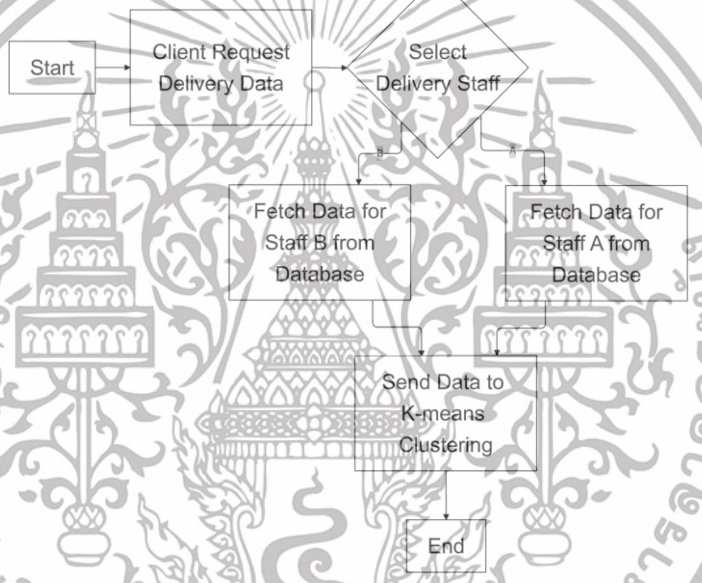
ในการออกแบบระบบการติดตั้ง OSRM (Open Source Routing Machine) ใน Docker นี้สำหรับใช้ในการขอข้อมูล Distance Matrix ในขั้นตอนการทำ Traveling Salesman Solution เพื่อให้การคำนวณเส้นทางเป็นไปตามเส้นทางบนถนนจริงๆ โดยมีแผนผังขั้นตอนการติดตั้งตามรูปที่ 3.20



รูปที่ 3.20 แผนผังระบบการติดตั้ง OSRM ใน Docker

6) การออกแบบระบบเพื่อทำการดึงข้อมูลจากฐานข้อมูลเพื่อใช้ในการคำนวนลำดับเส้นทางจากพิกัดจุดเริ่มต้นจริงและพิกัดที่อยู่ของแต่ละจุดหมายที่พนักงานขอส่งต้องทำการขนส่งจริง

ในการออกแบบระบบเพื่อทำการดึงข้อมูลจากฐานข้อมูลเพื่อใช้ในการคำนวนลำดับเส้นทางจากพิกัดจุดเริ่มต้นจริงและพิกัดที่อยู่ของแต่ละจุดหมายที่พนักงานขอส่งต้องทำการขนส่งจริงจะใช้ Django Backend ในการขอข้อมูลจากฐานข้อมูล MySQL โดยแสดงแผนผังการทำงานดังรูปที่ 3.21



รูปที่ 3.21 แผนผังระบบการดึงข้อมูลจากฐานข้อมูล MySQL

ในการออกแบบการรวมการทำงานของระบบย่อยต่างๆ เพื่อให้เป็นระบบค้นหาเส้นทางอัจฉริยะที่ใช้ข้อมูลเส้นทางจริงจะเป็นการรวมการทำงานของการออกแบบเพื่อทำการทดสอบการทำงานของแต่ละส่วนให้สามารถใช้งานร่วมกันได้ โดยประกอบไปด้วย การ Client ทำการขอเส้นทางของพนักงานขนส่ง A หรือ B และวันที่ที่ต้องจัดส่งจากนั้นระบบจะทำการดึงข้อมูลจากฐานข้อมูล MySQL เพื่อไปใช้ในแบ่งกลุ่มของรายการสินค้าที่ต้องจัดส่งเป็น 2 กลุ่ม เมื่อแบ่งกลุ่มเป็นสองกลุ่มแล้วจะทำการคำนวนลำดับเส้นทางและระยะทางที่ใช้ในการเดินทางโดยทำการส่งพิกัดของแต่ละกลุ่มไปขอ Distance Matrix จาก OSRM (Open Source Routing Machine) เพื่อใช้ในการแก้ปัญหา Traveling Salesman Problem ด้วยวิธีที่มีประสิทธิภาพ

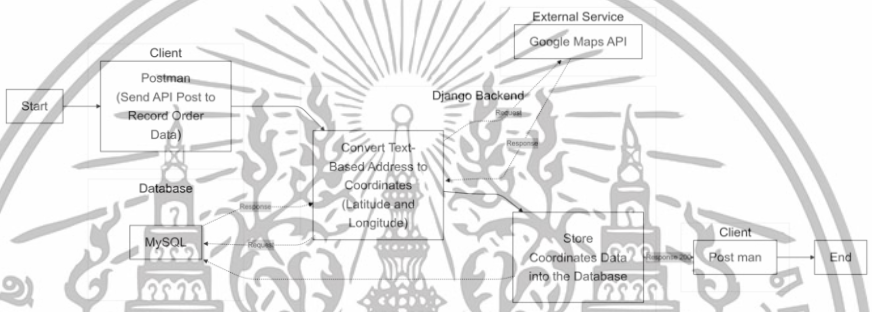
มากกว่าจากการทำการทดสอบวิธี Brute Force และ Dynamic Programming จากนั้นทำการเปรียบเทียบค่าความต่างของระยะทางของพนักงานขนส่งทั้งสองว่าต่างกันมากน้อยเพียงใด ถ้าเกิดมากเกินกว่าที่กำหนดจะทำการย้ายรายการสินค้าของพนักงานที่มีระยะทางขนส่งที่มากกว่าไปให้พนักงานที่มีระยะทางน้อยกว่าเพื่อไม่ให้เกิดความได้เปรียบเสียเปรียบมากจนเกินไป Response ตอบกลับไปที่ Client ซึ่งจะใช้ Postman ในการส่ง API ซึ่งเปรียบเสมือน Postman เป็น Client ซึ่งแสดงแผนผังการทำงานของระบบดังรูปที่ 3.22

รูปที่ 3.22 แผนผังการทำงานของระบบการคำนวณเส้นทาง

3.1.4 ออกแบบการทำงานของระบบการแปลงข้อมูลที่อยู่ให้อยู่ในรูปเป็นพิกัด

3.1.4.1 การออกแบบการทำงานในส่วนซอฟต์แวร์

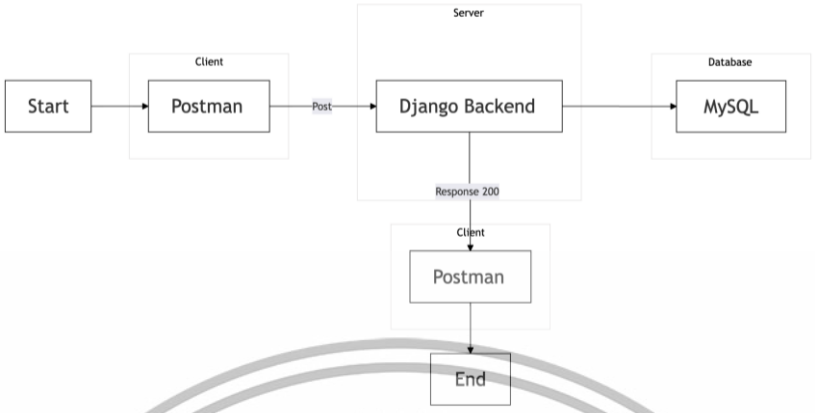
การออกแบบการทำงานของระบบนี้เน้นไปที่การพัฒนาซอฟต์แวร์เพื่อแปลงข้อมูลที่อยู่ในรูปภาพ เช่น ข้อความที่อยู่หรือสถานที่บนแผนที่ ให้กลายเป็นพิกัดละติจูดและลองจิจูดที่สามารถนำไปใช้งานในระบบอื่น ๆ ได้ โดยการทำงานของซอฟต์แวร์นี้จะประกอบไปด้วยการส่ง API เพื่อทำการบันทึกข้อมูลรายการสินค้าที่ทำการสั่งซื้อ และ การแปลงข้อมูลที่อยู่ในรูปแบบข้อความให้เป็นพิกัดโดยทำการเรียกใช้ API จาก Google maps API แสดงได้ดังรูปที่ 3.23



รูปที่ 3.23 แผนผังการทำงานของระบบการคำนวณเส้นทาง

1) การออกแบบระบบทดสอบการเพิ่มข้อมูลคำสั่งซื้อผ่านการเรียกใช้ API

ในการออกแบบระบบทดสอบการเพิ่มข้อมูลคำสั่งซื้อผ่านการเรียกใช้ API นี้มีไว้เพื่อทดสอบการเพิ่มข้อมูลคำสั่งซื้อผ่านการ POST API เข้ามาที่ระบบหลังบ้านและทำการบันทึกข้อมูลลงในฐานข้อมูล MySQL โดยจะทำการทดสอบส่ง POST request เข้ามาที่ Django Backend เพื่อให้ Django Backend บันทึกคำสั่งซื้อใหม่ลงไปในฐานข้อมูลซึ่งแสดงแผนผังการทำงานดังรูปที่ 3.24

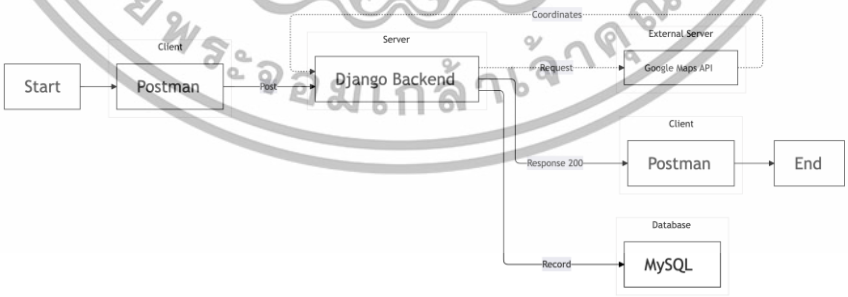


รูปที่ 3.24 แผนผังการทำงานของระบบทดสอบการเพิ่มข้อมูลคำสั่งซื้อผ่านการเรียกใช้ API

2) การออกแบบระบบทดสอบการแปลงข้อมูลที่อยู่ในรูปแบบข้อความให้อยู่ในรูปของพิกัด

ในการออกแบบระบบทดสอบการแปลงข้อมูลที่อยู่ในรูปแบบข้อความให้อยู่ในรูปแบบของพิกัดมีไว้เพื่อเตรียมความพร้อมของข้อมูลในการใช้งานในส่วนของการแบ่งกลุ่มด้วยวิธีแบบ K-means Clustering ที่จะนำไปใช้ในระบบการค้นหาเส้นทางอัจฉริยะต่อไป

โดยขั้นตอนในการออกแบบประกอบด้วย การตรวจสอบว่าในคำสั่งซื้อใหม่นั้นมีข้อมูลรายชื่อผู้ใช้งาน (User) และข้อมูลพิกัดบันทึกไว้ก่อนหน้าหรือไม่ถ้ายังไม่มีระบบจะทำการส่งคำขอผ่าน API ไปที่ Google maps API เพื่อทำการขอพิกัดจากที่อยู่ในรูปข้อความของผู้ใช้ซึ่งแสดงแผนผังการทำงานดังรูปที่ 3.25

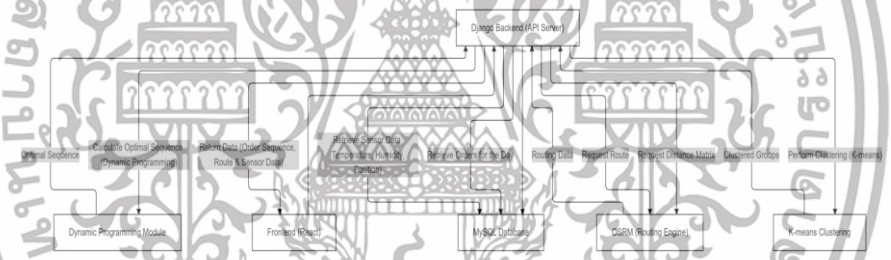


รูปที่ 3.25 แผนผังการทำงานของระบบการแปลงข้อมูลที่อยู่ในรูปแบบข้อความให้เป็นค่าพิกัด

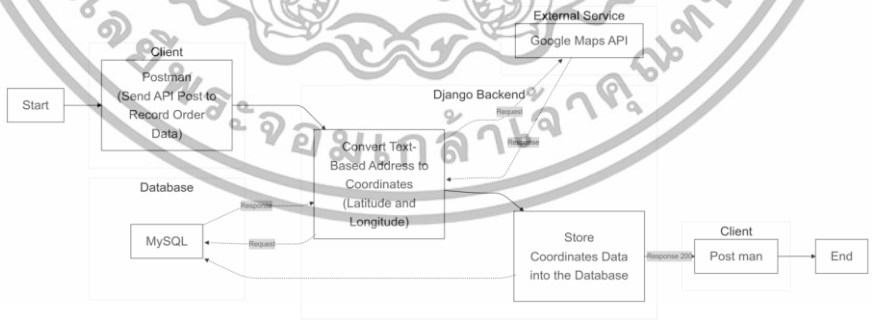
3.1.5 ออกแบบการทำงานของระบบหลังบ้าน

3.1.5.1 การออกแบบการทำงานในส่วนซอฟต์แวร์ของระบบหลังบ้าน

การออกแบบการทำงานในส่วนของระบบหลังบ้านนี้มีไว้เพื่อใช้ในการรองรับคำขอข้อมูลและคำสั่งต่างๆจากฝั่งของ Client และทำหน้าที่เป็นสื่อกลางในการสื่อสารกับฐานข้อมูลเพื่อส่งข้อมูลกลับไปยัง Client ตามที่ต้องการซึ่งจะต้องครอบคลุมทุกคำขอโดยใช้ API ซึ่งจะใช้สถาปัตยกรรม Backend (Django) ในการทำหน้าที่เป็น API Server และเชื่อมต่อกับ External Service ต่างๆและส่งคืนค่ากลับไปยังฝั่ง Client ด้วย API ซึ่งการทดสอบจะประกอบด้วย การขอข้อมูลของเซนเซอร์อุณหภูมิ ความชื้นและข้อมูลพิกัดของอุปกรณ์จากฐานข้อมูล การขอข้อมูลของรายการคำสั่งซื้อพร้อมวันที่จัดส่ง การขอข้อมูลของลำดับเส้นทางที่สั้นที่สุดของสินค้าที่ต้องได้รับการส่งในวันนั้นๆตามชื่อพนักงานขนส่งที่ทำการเรียกขอ ซึ่งแสดงรูปแบบการทำงานของระบบหลังบ้านในส่วนหลักดังรูปที่ 3.26 และแสดงรูปแบบการทำงานของระบบหลังบ้านสำหรับการแปลงที่อยู่เป็นพิกัดในส่วนย่อยดังรูปที่ 3.27



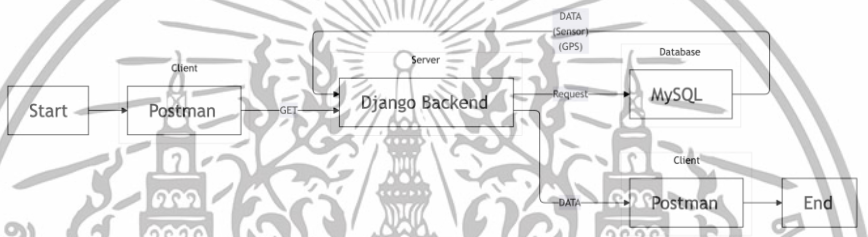
รูปที่ 3.26 แผนผังการทำงานของระบบหลังบ้าน



รูปที่ 3.27 แผนผังการทำงานของระบบแปลงข้อมูลที่อยู่เป็นพิกัดละติจูดและลองจิจูด

1) การออกแบบระบบทดสอบการส่งข้อมูลของเซนเซอร์อุณหภูมิ ความชื้นและข้อมูลพิกัดของอุปกรณ์จากฐานข้อมูลผ่าน API

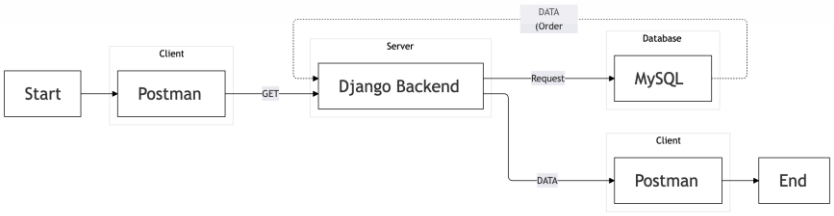
ในการออกแบบระบบทดสอบการส่งข้อมูลของเซนเซอร์อุณหภูมิ ความชื้นและข้อมูลพิกัดของอุปกรณ์จากฐานข้อมูลผ่าน API มีไว้เพื่อใช้ในการทดสอบ การ GET request เพื่อทำการดึงข้อมูลจากฝั่งของ Backend ซึ่งทดสอบโดยการใช้ Postman GET request ไปที่ Server (Django Backend) เพื่อดูการตอบสนองของ Server (Django Backend) จาก Postman โดยแสดงแผนผังการทำงานของระบบทดสอบการส่งข้อมูลของเซนเซอร์อุณหภูมิ ความชื้นและข้อมูลพิกัดของอุปกรณ์จากฐานข้อมูลผ่าน API แสดงดังรูปที่ 3.28



รูปที่ 3.28 แผนผังการทำงานของเซนเซอร์อุณหภูมิ ความชื้นและข้อมูลพิกัดของอุปกรณ์ Database ผ่าน API

2) การออกแบบระบบทดสอบการส่งข้อมูลของรายการคำสั่งซื้อพร้อมวันที่จัดส่งจากฐานข้อมูลผ่าน API

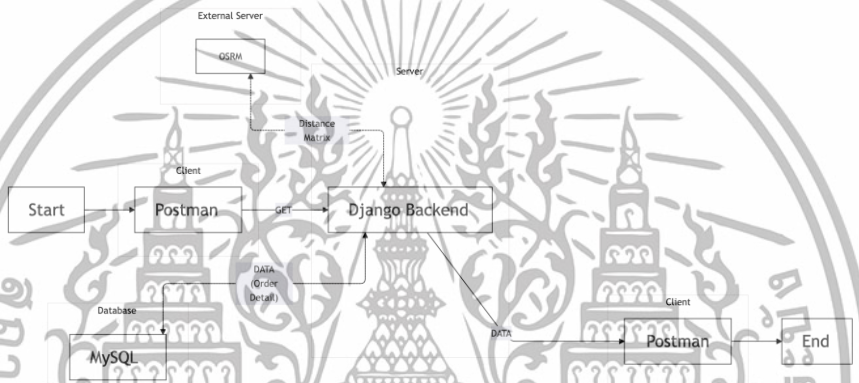
ในการออกแบบระบบทดสอบการส่งข้อมูลของรายการคำสั่งซื้อพร้อมวันที่จัดส่งจากฐานข้อมูลผ่าน API มีไว้เพื่อใช้ในการทดสอบ การ GET request เพื่อทำการดึงข้อมูลจากฝั่งของ Backend ซึ่งทดสอบโดยการใช้ Postman GET request ไปที่ Server (Django Backend) เพื่อดูการตอบสนองของ Server (Django Backend) จาก Postman โดยแสดงแผนผังการทำงานดังรูปที่ 3.29



รูปที่ 3.29 แผนผังการทำงานของรายการคำสั่งซื้อจากฐานข้อมูลผ่าน API

3) การออกแบบระบบทดสอบการส่งข้อมูลลำดับเส้นทางที่สั้นที่สุดของแต่ละพนักงานขนส่งในวันที่ระบุผ่าน API

ในการออกแบบระบบทดสอบการส่งข้อมูลลำดับเส้นทางที่สั้นที่สุดตามพนักงานขนส่งในวันที่ระบุผ่าน API มีไว้เพื่อใช้ในการทดสอบ การ GET request เพื่อทำการดึงข้อมูลจากฝั่งของ Backend ซึ่งทดสอบโดยการใช้ Postman GET request ไปที่ Server (Django Backend) เพื่อดูการตอบสนองของ Server (Django Backend) จาก Postman โดยแสดงแผนผังการทำงานดังรูปที่ 3.30



รูปที่ 3.30 แผนผังการส่งข้อมูลลำดับเส้นทางที่สั้นที่สุดตามพนักงานขนส่งในวันที่ระบุผ่าน API

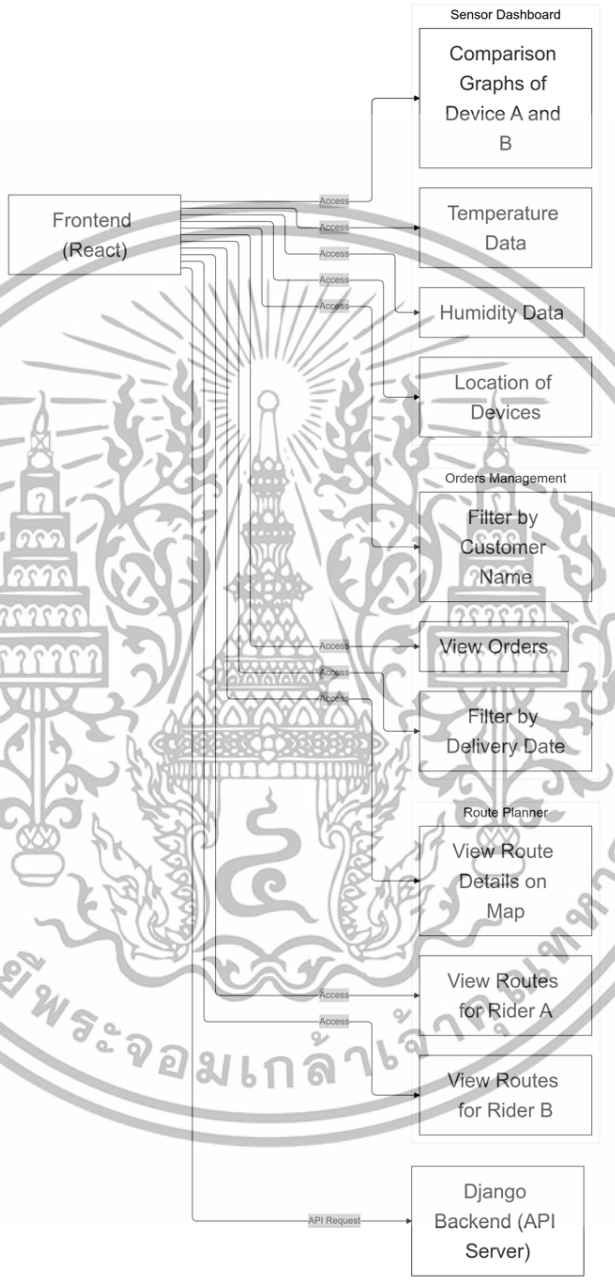
3.1.6 ออกแบบการทำงานของระบบหน้าบ้าน

3.1.6.1 การออกแบบการทำงานในส่วนซอฟต์แวร์ของระบบหน้าบ้าน

การออกแบบซอฟต์แวร์ของระบบหน้าบ้านมีวัตถุประสงค์เพื่อสร้าง UX และ UI โดยเน้นที่ความสามารถในการตอบสนองต่อความต้องการของผู้ใช้งานอย่างรวดเร็ว ในการออกแบบเน้นให้สามารถรองรับการแสดงผลได้บนอุปกรณ์หลายประเภท เช่นคอมพิวเตอร์ หรือสมาร์ทโฟน ผ่านการพัฒนาหน้าเว็บให้เป็นแบบ responsive

นอกจากนี้ยังคำนึงถึงการจัดการข้อมูลที่แสดงผลให้ตรงกับความต้องการของผู้ใช้งาน รวมถึงการแสดงผลข้อมูลสำคัญ เช่น ข้อมูลจากเซนเซอร์ คำสั่งซื้อ และเส้นทางที่แสดง

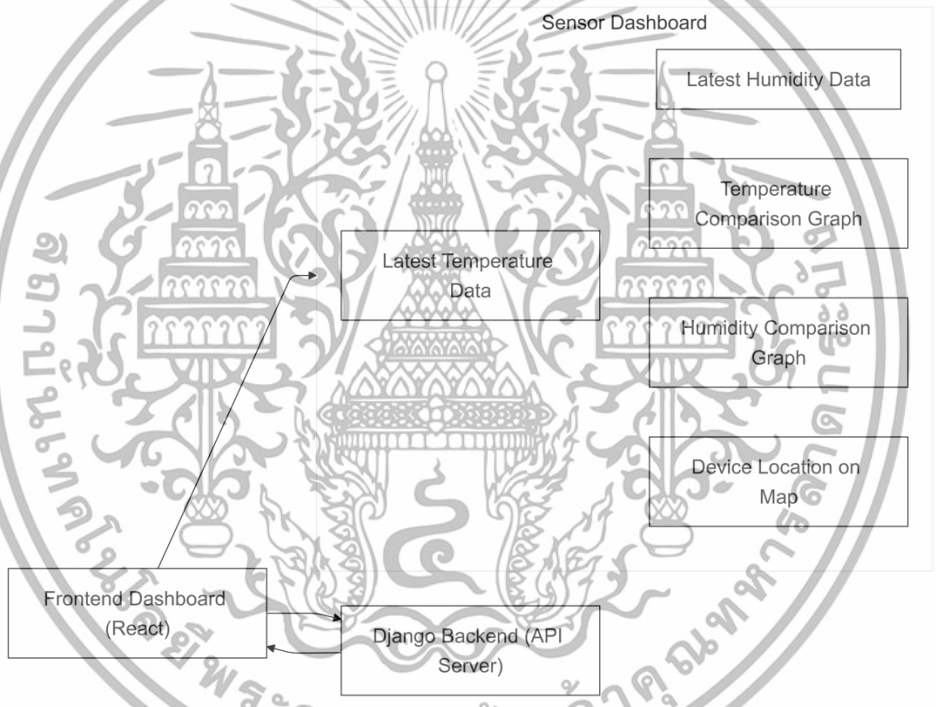
บนแผนที่ โดยใช้ Frontend (React) ในการพัฒนาระบบ Web Application ส่วนหน้าบ้านและทำการสื่อสารกับ Backend (Django) โดยใช้ API แสดงแผนผังการทำงานได้ดังรูปที่ 3.31



รูปที่ 3.31 แผนผังการทำงานในส่วนซอฟต์แวร์ของระบบหน้าบ้าน

1) การออกแบบระบบทดสอบการแสดงผลข้อมูลเซนเซอร์อุณหภูมิ ความชื้นและข้อมูลพิกัดของอุปกรณ์บนหน้า Dashboard

การออกแบบระบบทดสอบนี้มีไว้เพื่อทำการทดสอบการดึงข้อมูลเซนเซอร์อุณหภูมิ ความชื้น และข้อมูลพิกัดของอุปกรณ์ผ่าน API จากนั้นนำมาแสดงผลบนหน้า Dashboard อย่างถูกต้องและมีประสิทธิภาพเพื่อให้ผู้ใช้งานสามารถติดตามสถานะของอุปกรณ์ได้ โดยใช้เครื่องมืออย่าง Tailwind และ Material@Tailwind เพื่อให้การออกแบบหน้า UI เป็นระเบียบเรียบร้อยง่ายต่อการพัฒนา โดยจะแสดงผล อุณหภูมิล่าสุด , ความชื้นล่าสุด , กราฟอุณหภูมิ , กราฟความชื้น และ ตำแหน่งของอุปกรณ์ ซึ่งแสดงแผนผังดังรูปที่ 3.32

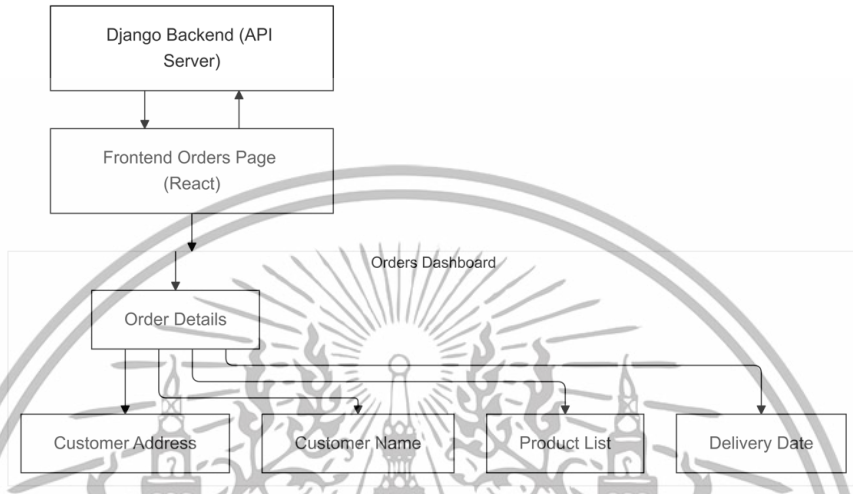


รูปที่ 3.32 แผนผังการทำงานของข้อมูลเซนเซอร์อุณหภูมิ ความชื้นและข้อมูลพิกัดของอุปกรณ์บนหน้า Dashboard

2) การออกแบบระบบทดสอบการแสดงผลข้อมูลรายการคำสั่งซื้อพร้อมวันที่จัดส่งบนหน้า Orders

การออกแบบระบบทดสอบนี้มีไว้เพื่อทดสอบการดึงข้อมูลรายการคำสั่งซื้อ พร้อมวันที่จัดส่งผ่าน API จากนั้นนำมาแสดงผลบนหน้า Orders อย่างถูกต้องและครบถ้วน โดยใช้เครื่องมืออย่าง Tailwind และ Material@Tailwind เพื่อให้การออกแบบหน้า UI

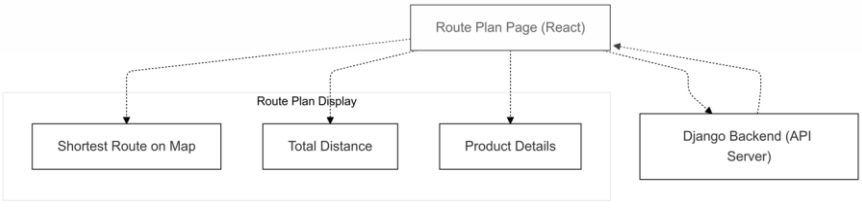
เป็นระเบียบเรียบร้อยง่ายต่อการพัฒนาซึ่งจะแสดง รายละเอียดคำสั่งซื้อ เช่น ชื่อผู้สั่งซื้อ รายการสินค้า จำนวนสินค้า ที่อยู่ผู้สั่งซื้อและวันที่จัดส่ง เพื่อให้ผู้ดูแลระบบสามารถติดตามสถานะการสั่งซื้อได้อย่างง่ายดาย ซึ่งแสดงแผนผังการทำงานดังรูปที่ 3.33



รูปที่ 3.33 แผนผังการทำงานขอการแสดงผลข้อมูลรายการคำสั่งซื้อพร้อมวันที่จัดส่งแสดงผลบนหน้า Orders

3) การออกแบบระบบทดสอบการแสดงผลลำดับเส้นทางที่สั้นที่สุด

การออกแบบระบบทดสอบนี้มีไว้เพื่อทดสอบการดึงข้อมูลลำดับเส้นทางและรายละเอียดที่เกี่ยวข้องผ่าน API จากนั้นนำมาแสดงผลบนหน้า Route Plan อย่างถูกต้องและครบถ้วน โดยใช้เครื่องมืออย่าง Tailwind และ Material@Tailwind เพื่อให้การออกแบบหน้า UI เป็นระเบียบเรียบร้อยง่ายต่อการพัฒนาซึ่งจะแสดง รายละเอียดเกี่ยวกับข้อมูลเส้นทางบนแผนที่ , ระยะทางการเดินทางทั้งหมด ข้อมูลสินค้าซึ่งแสดงแผนผังการทำงานดังรูปที่ 3.34



รูปที่ 3.34 แผนผังการทำงานการแสดงผลลำดับเส้นทางที่สั้นที่สุด

3.2 เครื่องมือที่ใช้ในการทดลอง

3.2.1 Lilygo T-A7670E

ใช้สำหรับเขียนโปรแกรมในการส่งค่าอุณหภูมิ ความชื้น และพิกัด เพื่อส่งข้อมูลเข้าสู่ MQTT Broker

3.2.2 GPS module NEO-6M

ใช้สำหรับบอกพิกัด โดยเขียนโปรแกรมผ่านทาง Node MCU

3.2.3 Mi Temperature And Humidity Monitor 2

ใช้สำหรับอ่านค่าอุณหภูมิและความชื้น และส่งข้อมูลผ่านบลูทูธ

3.2.4 MySQL

ใช้สำหรับการเก็บและจัดการข้อมูลในรูปแบบของตาราง

3.2.5 ReactJS

ใช้สำหรับการทำ frontend Web Application

3.2.6 Python3

ใช้สำหรับการพัฒนาระบบหลังบ้าน

3.2.7 Postman

ใช้สำหรับการทำการทดสอบ API

3.2.8 Django & Django rest framework

เป็น backend framework สำหรับการจัดการ API และทำ Logics สำหรับการค้นหาเส้นทางอัจฉริยะ

3.2.9 OSRM (open source routing machine)

ใช้สำหรับการทำ service ผ่านใน localhost เพื่อทำการสร้าง Distance matrix จากพิกัดแผนที่ในประเทศไทยและสร้างเส้นทางสำหรับเดินทางไปยังพิกัดต่างๆ

3.2.10 Docker

ใช้สำหรับการสร้าง,การนำไปใช้ และการจัดการแอปพลิเคชันที่อยู่ในรูปแบบของคอนเทนเนอร์ Docker

3.2.11 Google map API

ใช้สำหรับการแปลงที่อยู่ในรูปแบบข้อความให้เป็นพิกัด

3.3 การจัดเก็บผลการทดลอง

3.3.1 การทดสอบการทำงานของระบบติดตามการขนส่ง

เป็นการทดสอบ Mi Temperature And Humidity Monitor 2 และ GPS Module กับ Node MCU รวมถึงการส่งข้อมูลไปยัง MQTT Broker นอกจากนี้ ยังมีการทดสอบการใช้งานจริงบนยานพาหนะสองประเภท ได้แก่ รถจักรยานยนต์ และ รถไฟ เพื่อประเมินประสิทธิภาพของระบบในสภาพแวดล้อมที่แตกต่างกัน

3.3.2 การทดสอบการสร้างตารางฐานข้อมูลสำหรับใช้ในระบบหลังบ้านทั้งหมด

เป็นการทดลองการสร้างตารางเพื่อใช้สำหรับดึงข้อมูลหรือเพิ่มข้อมูลโดยใช้ Django เป็นตัวจัดการ

3.3.3 การทดสอบการทำงานของระบบค้นหาเส้นทาง

เป็นการทดลองการแบ่งกลุ่มพิกัดแบบ K-means Clustering และการทดลองการจัดเส้นทางที่สั้นที่สุดด้วยวิธี Brute Force และ Dynamic Programming

3.3.4 การทดสอบการทำงานของการแปลงข้อมูลที่อยู่เป็นพิกัดละติจูด ลองจิจูด

เป็นการทดลองการ POST Request เพื่อเพิ่มคำสั่งซื้อในตาราง Database โดยเรียกใช้บริการจาก Google API

3.3.5 การทดสอบการทำงานของระบบหลังบ้าน

เป็นการทดลองส่งข้อมูลของเซนเซอร์อุณหภูมิ ความชื้น ข้อมูลพิกัด จาก Database ผ่านทาง API โดยใช้ Postman และทดลองส่งข้อมูลลำดับเส้นทางที่สั้นที่สุด พร้อมลำดับพิกัด โดยใช้ Postman และ Django

3.3.6 การทดสอบการทำงานของระบบหน้าบ้าน

เป็นการทดลองเพื่อแสดงผลข้อมูลในส่วนของ User Interface การดึงข้อมูลจาก Database และ Server ผ่าน API โดยใช้ ReactJS

บทที่ 4

ผลการทดลอง

4.1 ผลการทดสอบการทำงานของระบบติดตามการขนส่ง

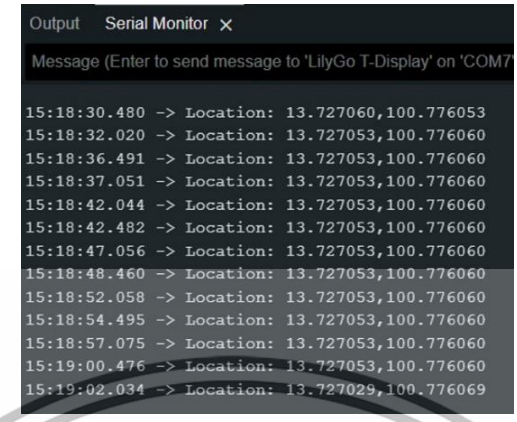
4.1.1 การทดสอบการทำงานในส่วนฮาร์ดแวร์

4.1.1.1 การทดสอบการทำงานของ GPS Module

ในการทดสอบการเชื่อมต่อ Node MCU กับ GPS Module ใช้สายไฟเพื่อต่อระหว่างขาอินพุต/เอาต์พุตของ Node MCU และ GPS Module ผ่านแผงวงจรเบรดบอร์ด มีการจ่ายไฟผ่านสาย USB เพื่อให้ Node MCU สามารถสื่อสารและรับข้อมูลตำแหน่งจาก GPS Module ได้ แสดงได้ดังรูปที่ 4.1

รูปที่ 4.1 Node MCU ที่เชื่อมต่อกับ GPS Module

ผลที่ได้จากการทดสอบแสดงให้เห็นถึงข้อมูลตำแหน่งที่ได้รับจาก GPS Module ผ่าน Serial Monitor โดยมีการแสดงผลตำแหน่งในรูปแบบละติจูดและลองจิจูดในแต่ละช่วงเวลา แสดงได้ดังรูปที่ 4.2



รูปที่ 4.2 ค่าพิกัดที่ได้จาก GPS Module แสดงผ่าน Serial Monitor

4.1.1.2 การทดสอบการทำงานร่วมกันของ Node MCU และโมดูลซิม A7670E

การทดสอบการเชื่อมต่อระหว่าง Node MCU กับโมดูล SIM A7670E ที่ติดตั้งสายอากาศ LTE เพื่อเพิ่มประสิทธิภาพในการรับสัญญาณ โดย Node MCU ได้รับการจ่ายไฟผ่านสาย USB เพื่อใช้งานโมดูล แสดงได้ดังรูปที่ 4.3

รูปที่ 4.3 Node MCU ที่เชื่อมต่อกับโมดูล SIM A7076E พร้อมสายอากาศ

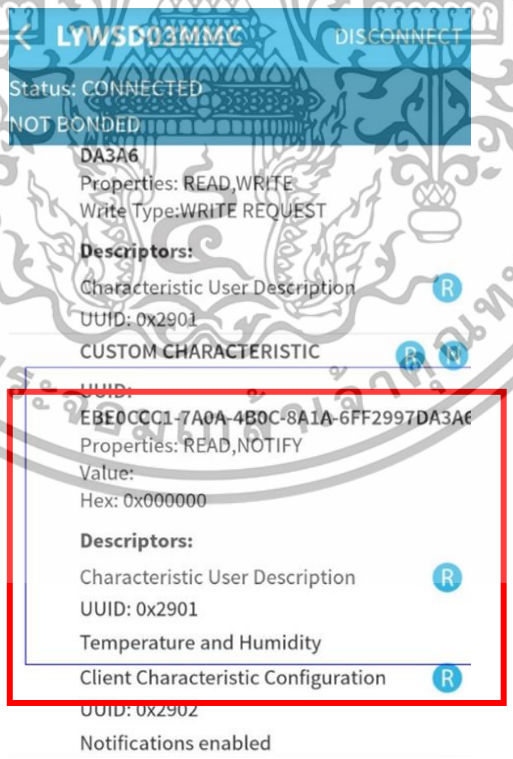
4.1.2 การทดสอบการทำงานในส่วนซอฟต์แวร์

4.1.2.1 การทดสอบการเชื่อมต่อของ Mi Temperature & Humidity Monitor 2 และ Node MCU

ตรวจสอบค่า Mac Address ของ Mi Temperature & Humidity Sensor 2 ผ่านแอปพลิเคชัน Mi home แสดงได้ดังรูป 4.4

รูปที่ 4.4 Mac Address ของ Mi Temperature & Humidity Sensor 2 ผ่านแอป Mi home

เชื่อมต่อผ่านแอปพลิเคชัน BLE Scanner และทำการค้นหา UUID ของค่าอุณหภูมิและความชื้น แสดงได้ดังรูป 4.5



รูปที่ 4.5 UUID ของค่าอุณหภูมิและความชื้น

ทำการเขียนโปรแกรมโดยใช้ Arduino IDE เพื่อสื่อสารระหว่าง Node MCU กับ Mi Temperature & Humidity Sensor 2 โดยต้องระบุ Mac Address และ UUID ของค่าอุณหภูมิและความชื้น แสดงได้ดังรูป 4.6

```
#define MJ_ADDR "A4:C1:38:05:08:C4"
static BLEAddress MJAddress(MJ_ADDR);
static boolean connected = false;
static BLERemoteCharacteristic* pRemoteCharacteristic;
uint8_t* pData;
float temp = 0;
float humi =0;
float volt =0;

// The remote service we wish to connect to.
static BLEUUID serviceUUID("ebe0ccb0-7a0a-4b0c-8a1a-6ff2997da3a6"); //
// The characteristic of the remote service we are interested in.
static BLEUUID    charUUID("ebe0ccc1-7a0a-4b0c-8a1a-6ff2997da3a6"); //
```

รูปที่ 4.6 การเขียนโปรแกรมที่ใช้สำหรับการสื่อสารระหว่าง Node MCU กับ Mi Temp & Humidity Sensor 2

จาก Raw data: 98 C 47 3D A ซึ่งประกอบด้วย 5 ไบต์ (98, C, 47, 3D, A) โดยแต่ละไบต์เป็นเลขฐาน 16 เป็นการเก็บข้อมูลเฉพาะ ได้แก่ อุณหภูมิ ความชื้น และแรงดันไฟฟ้าของแบตเตอรี่

ไบต์ที่ 1 (98) และ ไบต์ที่ 2 (C) ใช้สำหรับเก็บค่าอุณหภูมิ (Temperature) ในหน่วยเซลเซียส โดยในการแปลงค่าจะเรียงจาก ไบต์ที่ 2 ต่อด้วย ไบต์ที่ 1 แปลงเป็นเลขฐาน 2 แล้วหารด้วย 100

ไบต์ที่ 3 (47) ใช้สำหรับเก็บค่าความชื้น (Humidity) ซึ่งสามารถแปลงจากเลขฐาน 16 เป็นเลขฐาน 2 ซึ่งมีหน่วยเปอร์เซ็นต์ความชื้น (%)

ไบต์ที่ 4 (3D) และ ไบต์ที่ 5 (A) ใช้สำหรับเก็บค่าแรงดันไฟฟ้า โดยมีสมการการแปลงดังนี้ volt = [(ไบต์ที่ 5 * 256) + ไบต์ที่ 4)] / 1000 หน่วยเป็น V แสดงได้ดังรูปที่ 4.7

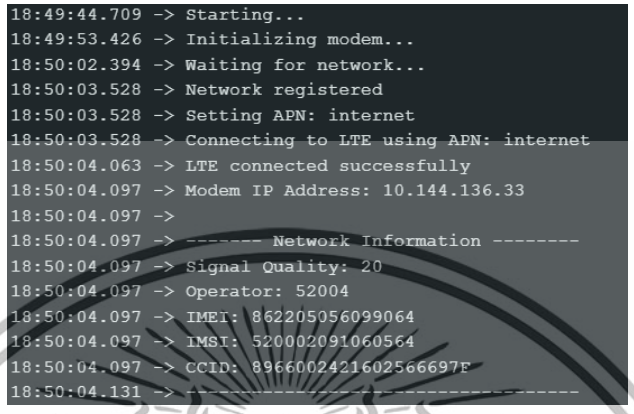
```
Raw Data: 98 C 47 3D A
Temperature (C): 32.24 (Raw: 3224)
Humidity (%): 71.00 (Raw: 71)
```

รูปที่ 4.7 ข้อมูลจาก Mi Temp & Humidity Sensor 2

4.1.2.2 การทดสอบการทำงานของโมดูลซิม A7670E

ในการทดสอบการทำงานของโมดูลซิม A7670E ผ่าน Serial Monitor เริ่มจากการเปิดใช้งานโมเด็มและรอการลงทะเบียนเครือข่าย เมื่อเชื่อมต่อสำเร็จ ระบบจะตั้งค่า APN

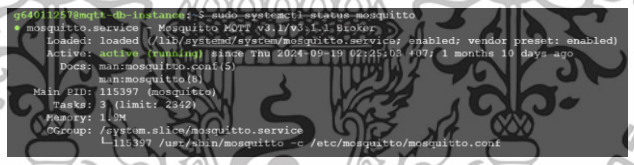
และเชื่อมต่อกับ LTE โดยข้อมูลที่ได้รับประกอบด้วยคุณภาพสัญญาณ, รหัสผู้ให้บริการ, IMEI ของอุปกรณ์, รวมถึงหมายเลข IMSI และ CCID ของซิม แสดงได้ดังรูปที่ 4.8



รูปที่ 4.8 ข้อมูลจากโมดูลซิม A7076E ผ่าน Serial Monitor

4.1.2.3 การทดสอบการรับข้อมูลผ่าน MQTT Broker

ในขั้นตอนนี้จะเป็นการตรวจสอบสถานะของ MQTT Broker ซึ่งพบว่ากำลังทำงานในสถานะ active (running) พร้อมรายละเอียดเกี่ยวกับเวอร์ชันที่ใช้ (v3.1/v3.1.1), เวลาที่เริ่มต้นทำงาน, หน่วยความจำที่ใช้, และ PID หลักของโปรเซส แสดงได้ดังรูปที่ 4.9



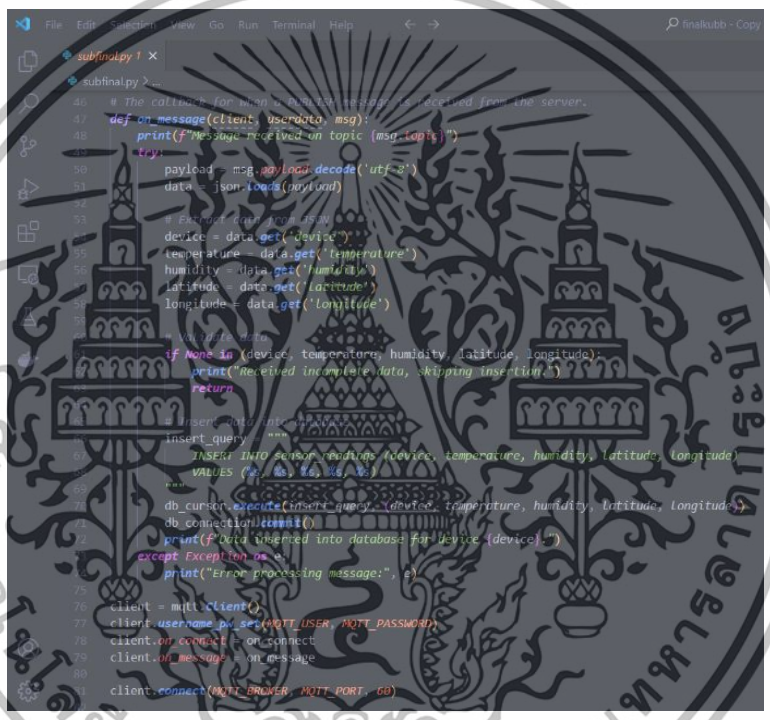
รูปที่ 4.9 สถานะของ MQTT Broker

ขั้นตอนถัดมาเป็นการทดสอบการรับข้อมูลจาก MQTT Broker โดยใช้คำสั่ง mosquitto_sub เพื่อตรวจสอบการ subscribe ไปยัง Topic "test" โดยข้อมูลที่ได้รับประกอบด้วยข้อความ เช่น "I'm Pokpong", "pokpong", และ "testtest" แสดงได้ดังรูปที่ 4.10

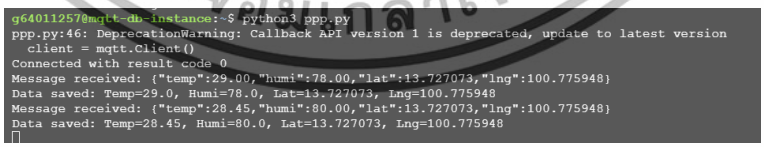
รูปที่ 4.10 การทดสอบการรับข้อมูลจาก MQTT Broker

4.1.2.4 การทดสอบโค้ด Python ที่รับข้อมูลมาจาก MQTT Broker และบันทึกข้อมูลไปยัง Database

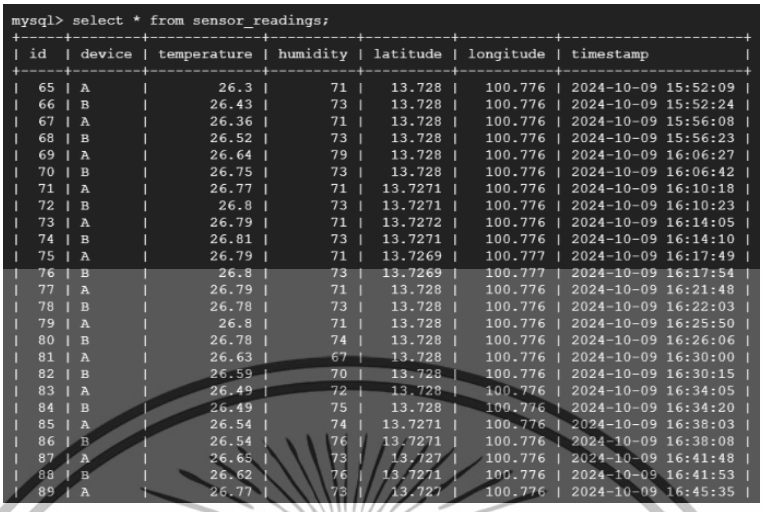
ในขั้นตอนนี้โค้ด Python จะรับข้อมูลจาก MQTT Broker และบันทึกข้อมูลลงใน Database โดยใช้ไลบรารี paho-mqtt เพื่อ Subscribe ไปยัง Topic ของ MQTT เมื่อมีข้อความใหม่เข้ามา โดยฟังก์ชัน on_message จะแปลงข้อมูลจาก JSON และแยกค่าต่าง ๆ เช่น อุปกรณ์, อุณหภูมิ, ความชื้น, ละติจูด, และลองจิจูด หลังจากนั้นจะใช้คำสั่ง SQL เพื่อบันทึกลงฐานข้อมูล แสดงได้ดังรูปที่ 4.11 รูปที่ 4.12 และรูปที่ 4.13



รูปที่ 4.11 โค้ด Python ที่รับข้อมูลมาจาก MQTT Broker และบันทึกข้อมูลไปยัง Database



รูปที่ 4.12 ข้อมูลที่ได้รับจาก MQTT Broker และบันทึกข้อมูลไปยัง Database



```
mysql> select * from sensor_readings;
+-----+--------+-------------+----------+----------+-----------+---------------------+
| id  | device | temperature | humidity | latitude | longitude | timestamp           |
+-----+--------+-------------+----------+----------+-----------+---------------------+
|  65 | A      |        26.3 |       71 |   13.728 |   100.776 | 2024-10-09 15:52:09 |
|  66 | B      |       26.43 |       73 |   13.728 |   100.776 | 2024-10-09 15:52:24 |
|  67 | A      |       26.36 |       71 |   13.728 |   100.776 | 2024-10-09 15:56:08 |
|  68 | B      |       26.52 |       73 |   13.728 |   100.776 | 2024-10-09 15:56:23 |
|  69 | A      |       26.64 |       79 |   13.728 |   100.776 | 2024-10-09 16:06:27 |
|  70 | B      |       26.75 |       73 |   13.728 |   100.776 | 2024-10-09 16:06:42 |
|  71 | A      |       26.77 |       71 |  13.7271 |   100.776 | 2024-10-09 16:10:18 |
|  72 | B      |       26.8  |       73 |  13.7271 |   100.776 | 2024-10-09 16:10:23 |
|  73 | A      |       26.79 |       71 |  13.7272 |   100.776 | 2024-10-09 16:14:05 |
|  74 | B      |       26.81 |       73 |  13.7271 |   100.776 | 2024-10-09 16:14:10 |
|  75 | A      |       26.79 |       71 |  13.7269 |   100.777 | 2024-10-09 16:17:49 |
|  76 | B      |       26.8  |       73 |  13.7269 |   100.777 | 2024-10-09 16:17:54 |
|  77 | A      |       26.79 |       71 |   13.728 |   100.776 | 2024-10-09 16:21:48 |
|  78 | B      |       26.78 |       73 |   13.728 |   100.776 | 2024-10-09 16:22:03 |
|  79 | A      |       26.8  |       71 |   13.728 |   100.776 | 2024-10-09 16:25:50 |
|  80 | B      |       26.78 |       74 |   13.728 |   100.776 | 2024-10-09 16:26:06 |
|  81 | A      |       26.63 |       67 |   13.728 |   100.776 | 2024-10-09 16:30:00 |
|  82 | B      |       26.59 |       70 |   13.728 |   100.776 | 2024-10-09 16:30:15 |
|  83 | A      |       26.49 |       72 |   13.728 |   100.776 | 2024-10-09 16:34:05 |
|  84 | B      |       26.49 |       75 |   13.728 |   100.776 | 2024-10-09 16:34:20 |
|  85 | A      |       26.54 |       74 |  13.7271 |   100.776 | 2024-10-09 16:38:03 |
|  86 | B      |       26.54 |       76 |  13.7271 |   100.776 | 2024-10-09 16:38:08 |
|  87 | A      |       26.65 |       73 |   13.727 |   100.776 | 2024-10-09 16:41:48 |
|  88 | B      |       26.62 |       76 |  13.7271 |   100.776 | 2024-10-09 16:41:53 |
|  89 | A      |       26.77 |       73 |   13.727 |   100.776 | 2024-10-09 16:45:35 |
```

รูปที่ 4.13 ข้อมูลที่ได้รับจาก MQTT Broker และบันทึกข้อมูลไปยัง Database

4.1.2.5 การทดสอบ Sleepmode ของ Node MCU

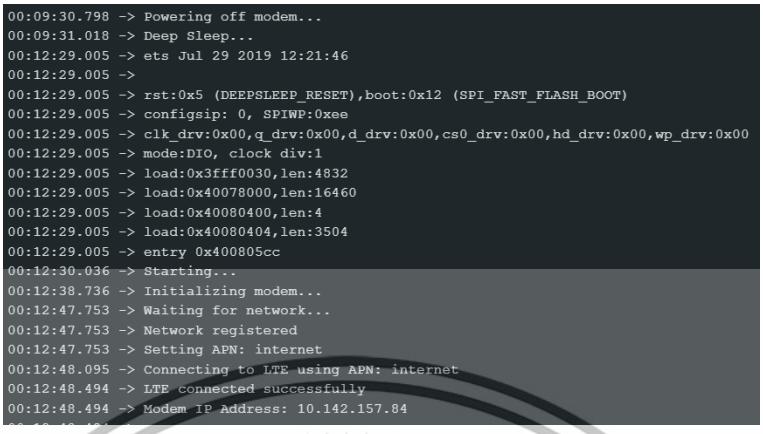
ในการทดสอบการเข้าสู่โหมด Deep Sleep ของ Node MCU เริ่มจากการเก็บข้อมูลจากอุปกรณ์ A และ B จากนั้นส่งข้อมูลไปยัง MQTT Broker ซึ่งประกอบด้วยข้อมูลอุณหภูมิ, ความชื้น, ละติจูด, และลองจิจูด

เมื่อการส่งข้อมูลสำเร็จระบบจะเตรียมเข้าสู่โหมด Deep Sleep โดยเริ่มจากการปิดโมเด็มก่อน หลังจากนั้น Node MCU จะเข้าสู่โหมด Deep Sleep ซึ่งช่วยในการประหยัดพลังงานในการทำงาน แสดงได้ดังรูปที่ 4.14

```
00:09:30.426 -> Collecting data for A...
00:09:30.426 -> Data collected for A.
00:09:30.426 -> Publishing data for Device A to MQTT...
00:09:30.426 -> Payload: {"device": "A", "temperature": 26.56, "humidity": 71.00, "latitude": 13.727807, "longitude": 100.776939}
00:09:30.472 -> Published to MQTT
00:09:30.472 -> Data for Device A published successfully.
00:09:30.472 -> Collecting data for B...
00:09:30.472 -> Data collected for B.
00:09:30.472 -> Publishing data for Device B to MQTT...
00:09:30.472 -> Payload: {"device": "B", "temperature": 26.52, "humidity": 73.00, "latitude": 13.727807, "longitude": 100.776939}
00:09:30.519 -> Published to MQTT
00:09:30.519 -> Data for Device B published successfully.
00:09:30.519 -> Preparing to enter Deep Sleep...
00:09:30.798 -> Powering off modem...
00:09:31.018 -> Deep Sleep...
```

รูปที่ 4.14 การเข้าสู่ Deep Sleep ของ Node MCU

ในขั้นตอนถัดไปเป็นการตื่นขึ้นของ Node MCU หลังจากเข้าสู่โหมด Deep Sleep ระบบเริ่มต้นการทำงานด้วยการรีเซ็ต (DEEPSLEEP_RESET) จากนั้นโหลดการตั้งค่าต่าง ๆ ของฮาร์ดแวร์และเข้าสู่โหมดบูต เมื่อบูตเสร็จแล้ว จะเริ่มการเชื่อมต่อกับโมเด็ม ลงทะเบียนเครือข่าย และตั้งค่า APN เพื่อเชื่อมต่อกับเครือข่าย LTE เมื่อการเชื่อมต่อสำเร็จ ระบบจะแสดง IP Address ที่ได้รับ แสดงได้ดังรูปที่ 4.15

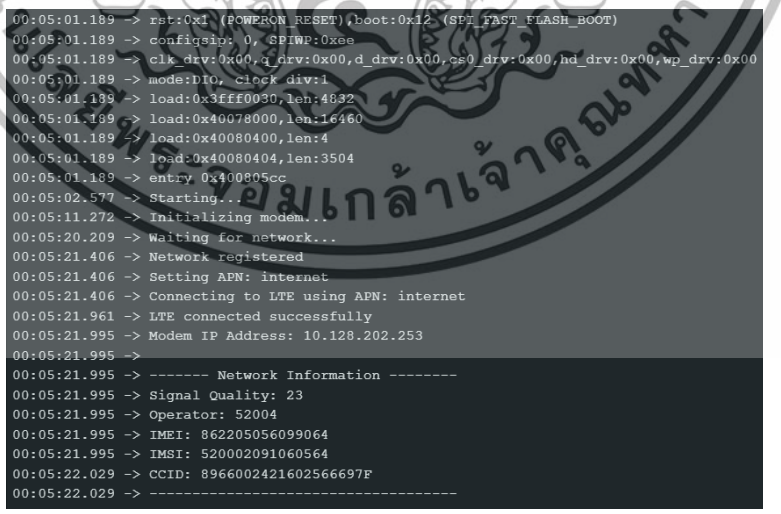


รูปที่ 4.15 การตื่นขึ้นของ Node MCU หลังจาก Deep Sleep

4.1.2.6 การทดสอบการทำงานร่วมกันของ Node MCU และโมดูลเซนเซอร์ทั้งหมด

การทดสอบการทำงานร่วมกันระหว่างไมโครคอนโทรลเลอร์และโมดูลเซนเซอร์ เริ่มจากการบูตระบบ (Power-on Reset) และการกำหนดค่าการทำงานต่าง ๆ เช่น การกำหนดโหมดการทำงานและการโหลดข้อมูล เมื่อบูตสำเร็จ ระบบจะเริ่มต้นการทำงานของโมเด็มลงทะเบียนเครือข่าย และตั้งค่า APN เพื่อเชื่อมต่อกับ LTE

เมื่อการเชื่อมต่อกับ LTE สำเร็จ จะได้รับ IP Address พร้อมแสดงข้อมูลคุณภาพสัญญาณ (Signal Quality), รหัสผู้ให้บริการ (Operator), เลข IMEI, หมายเลข IMSI และ CCID ของซิมการ์ด แสดงได้ดังรูปที่ 4.16



รูปที่ 4.16 การลงทะเบียนเครือข่ายและเชื่อมต่อ LTE

หลังจากลงทะเบียนเครือข่าย ต่อไปคือการเชื่อมต่อกับ MQTT Broker จากนั้นจะอ่านข้อมูล GPS และระบุตำแหน่งที่ได้โดยใช้ข้อมูลพิกัดจาก GPS Module โดยกำหนดค่าหน่วงเวลา 5 วินาที ถ้า GPS Module ใช้ไม่ได้ จะใช้ค่าพิกัดจากซิมการ์ด (ผ่านคำสั่ง AT+CLBS=4) เมื่อได้รับข้อมูลละติจูดและลองจิจูดแล้ว ระบบจะเชื่อมต่อกับอุปกรณ์ Mi Temp & Humi Sensors หลังจากเชื่อมต่อกับอุปกรณ์ได้แล้ว ระบบจะเริ่มเก็บข้อมูลจากอุปกรณ์ A ก่อน แล้วจึงส่งข้อมูลไปยัง MQTT Broker ซึ่งประกอบด้วยอุณหภูมิ, ความชื้น, ละติจูด, และลองจิจูด แสดงได้ดังรูปที่ 4.17

รูปที่ 4.17 รับอุณหภูมิ ความชื้น ค่าพิกัด และส่งไปที่ MQTT Broker

ต่อมาเป็นการเข้าสู่โหมด Deep Sleep โดยระบบเริ่มจากการเก็บข้อมูลจากอุปกรณ์ B และส่งข้อมูลนี้ไปยัง MQTT Broker ซึ่งข้อมูลที่ส่งประกอบด้วยอุณหภูมิ, ความชื้น, ละติจูด, และลองจิจูด หลังจากนั้นจะเตรียมเข้าสู่โหมด Deep Sleep โดยทำการปิดโมเด็มก่อนที่จะเข้าสู่โหมด Deep Sleep เพื่อลดการใช้พลังงาน แสดงดังรูปที่ 4.18

รูปที่ 4.18 ส่งข้อมูลเสร็จสิ้น และเข้าสู่โหมด Deep Sleep

ต่อไปเป็นขั้นตอนการรับข้อมูลจาก MQTT Broker โดยใช้สคริปต์ Python เพื่อรับข้อมูลที่ถูกส่งมาจากอุปกรณ์ โดยข้อมูลที่ได้รับประกอบด้วยอุณหภูมิ, ความชื้น, ละติจูด, และลองจิจูด และบันทึกข้อมูลลงใน Database แสดงดังรูปที่ 4.19

```
g64011257@mqtt-db-instance:~$ python3 ppp.py
ppp.py:46: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
  client = mqtt.Client()
Connected with result code 0
Message received: {"temp":29.00,"humi":78.00,"lat":13.727073,"lng":100.775948}
Data saved: Temp=29.0, Humi=78.0, Lat=13.727073, Lng=100.775948
Message received: {"temp":28.45,"humi":80.00,"lat":13.727073,"lng":100.775948}
Data saved: Temp=28.45, Humi=80.0, Lat=13.727073, Lng=100.775948
```

รูปที่ 4.19 การทำงานของสคริปต์ Python

เมื่อสคริปต์ Python บันทึกข้อมูลใน Database แล้ว ข้อมูลในตารางจะประกอบด้วยคอลัมน์ต่าง ๆ ได้แก่ Device ระบุอุปกรณ์ (A หรือ B ของเซ็นเซอร์วัดอุณหภูมิและความชื้น) ,temperature (ค่าของอุณหภูมิ) ,humidity (ค่าของความชื้น) latitude และ longitude (ค่าตำแหน่งละติจูดและลองจิจูด) และ timestamp (เวลาที่ข้อมูลถูกบันทึก) แสดงดังรูปที่ 4.20

```
mysql> select * from sensor_readings;
+-----+--------+-------------+----------+----------+-----------+---------------------+
| id  | device | temperature | humidity | latitude | longitude | timestamp           |
+-----+--------+-------------+----------+----------+-----------+---------------------+
|  65 | A      |        26.3 |       71 |   13.728 |   100.776 | 2024-10-09 15:52:09 |
|  66 | B      |       26.43 |       73 |   13.728 |   100.776 | 2024-10-09 15:52:24 |
|  67 | A      |       26.36 |       71 |   13.728 |   100.776 | 2024-10-09 15:56:08 |
|  68 | B      |       26.52 |       73 |   13.728 |   100.776 | 2024-10-09 15:56:23 |
|  69 | A      |       26.64 |       79 |   13.728 |   100.776 | 2024-10-09 16:06:27 |
|  70 | B      |       26.75 |       73 |   13.728 |   100.776 | 2024-10-09 16:06:42 |
|  71 | A      |       26.77 |       71 |  13.7271 |   100.776 | 2024-10-09 16:10:18 |
|  72 | B      |        26.8 |       73 |  13.7271 |   100.776 | 2024-10-09 16:10:23 |
|  73 | A      |       26.79 |       71 |  13.7272 |   100.776 | 2024-10-09 16:14:05 |
|  74 | B      |       26.81 |       73 |  13.7271 |   100.776 | 2024-10-09 16:14:10 |
|  75 | A      |       26.79 |       71 |  13.7269 |   100.777 | 2024-10-09 16:17:49 |
|  76 | B      |        26.8 |       73 |  13.7269 |   100.777 | 2024-10-09 16:17:54 |
|  77 | A      |       26.79 |       71 |   13.728 |   100.776 | 2024-10-09 16:21:48 |
|  78 | B      |       26.78 |       73 |   13.728 |   100.776 | 2024-10-09 16:22:03 |
|  79 | A      |        26.8 |       71 |   13.728 |   100.776 | 2024-10-09 16:25:50 |
|  80 | B      |       26.78 |       74 |   13.728 |   100.776 | 2024-10-09 16:26:06 |
|  81 | A      |       26.63 |       67 |   13.728 |   100.776 | 2024-10-09 16:30:00 |
|  82 | B      |       26.59 |       70 |   13.728 |   100.776 | 2024-10-09 16:30:15 |
|  83 | A      |       26.49 |       72 |   13.728 |   100.776 | 2024-10-09 16:34:05 |
|  84 | B      |       26.49 |       75 |   13.728 |   100.776 | 2024-10-09 16:34:20 |
|  85 | A      |       26.54 |       74 |  13.7271 |   100.776 | 2024-10-09 16:38:03 |
|  86 | B      |       26.54 |       76 |  13.7271 |   100.776 | 2024-10-09 16:38:08 |
|  87 | A      |       26.65 |       73 |   13.727 |   100.776 | 2024-10-09 16:41:48 |
|  88 | B      |       26.62 |       76 |  13.7271 |   100.776 | 2024-10-09 16:41:53 |
|  89 | A      |       26.77 |       73 |   13.727 |   100.776 | 2024-10-09 16:45:35 |
```

รูปที่ 4.20 Database ที่ได้รับข้อมูลจาก Node MCU

4.1.3 การทดสอบระบบติดตามการขนส่งร่วมกับยานพาหนะ

4.1.3.1 การทดสอบระบบติดตามการขนส่งร่วมกับรถจักรยานยนต์

การทดสอบนี้มีวัตถุประสงค์เพื่อตรวจสอบประสิทธิภาพของระบบติดตามการขนส่งเมื่อใช้งานร่วมกับ รถจักรยานยนต์ ซึ่งเป็นยานพาหนะที่มีการเคลื่อนที่รวดเร็วและสามารถเข้าถึงพื้นที่ที่จำกัดได้ โดยการทดลองแบ่งออกเป็นสองส่วนหลัก ได้แก่ การตรวจสอบความถูกต้องของข้อมูลที่ส่งผ่านระบบ และ การทดสอบสภาพแวดล้อมในการใช้งานจริง

ในขั้นตอนแรกจะเป็นการติดตั้งและทดสอบ ระบบติดตามการขนส่ง บนรถจักรยานยนต์ โดยกล่องสีแดงที่วางอยู่บนกระเป๋าขนส่งเป็น กล่องอุปกรณ์ที่ออกแบบและผลิตด้วยเทคโนโลยีการพิมพ์สามมิติ (3D Printing) ภายในบรรจุ Node MCU, โมดูลซิม, GPS Module

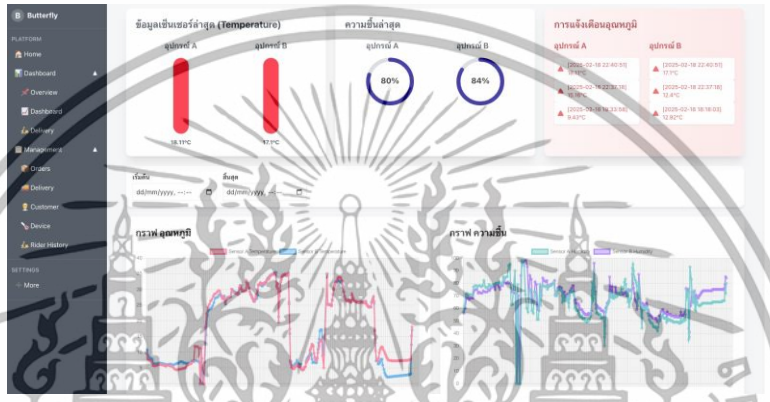
และเซ็นเซอร์วัดอุณหภูมิและความชื้น ซึ่งใช้สำหรับบันทึกและส่งข้อมูลการขนส่งไปยังระบบเซิร์ฟเวอร์ผ่านเครือข่าย LTE แสดงได้ดังรูปที่ 4.21 และรูปที่ 4.22 ตามลำดับ

รูปที่ 4.21 อุปกรณ์ติดตามการขนส่งภายในกล่องอุปกรณ์

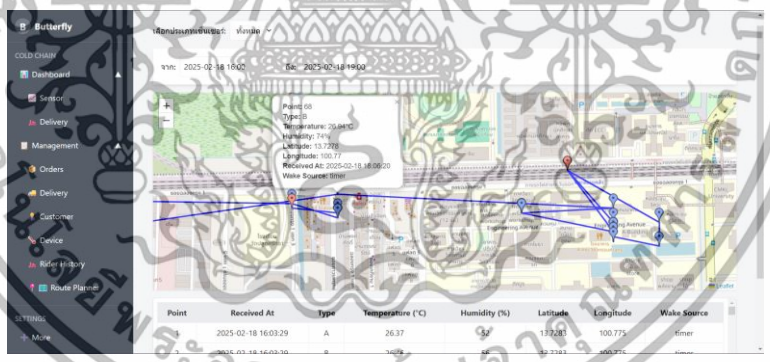
รูปที่ 4.22 การติดตั้งอุปกรณ์ติดตามการขนส่งบนรถจักรยานยนต์

ในการทดสอบพบว่าระบบสามารถติดตามและบันทึกตำแหน่งของยานพาหนะได้อย่างแม่นยำ โดยใช้ GPS Module เพื่อระบุตำแหน่งและส่งข้อมูลผ่าน เครือข่าย LTE ไปยัง MQTT Broker ก่อนนำมาแสดงผลบนแพลตฟอร์ม Butterfly Cold Chain ผ่านแผนที่ดิจิทัล ซึ่งช่วยให้สามารถดูเส้นทางการเดินทาง รวมถึงจุดเริ่มต้นและจุดหมายปลายทางได้อย่าง

ชัดเจน นอกจากนี้ ระบบยังสามารถวัดและส่งข้อมูล อุณหภูมิและความชื้น จาก Mi Temperature & Humidity Monitor 2 แบบเรียลไทม์ ทำให้สามารถติดตามการเปลี่ยนแปลงของสภาพแวดล้อมระหว่างการขนส่งได้อย่างต่อเนื่อง อีกทั้งยังมีฟังก์ชันให้เลือกช่วงเวลาที่ต้องการดูข้อมูลย้อนหลัง เพื่อช่วยในการวิเคราะห์และตรวจสอบแนวโน้มของข้อมูลได้สะดวกยิ่งขึ้น แสดงได้ดังรูปที่ 4.23 และรูปที่ 4.24



รูปที่ 4.23 แดชบอร์ดแสดงข้อมูลอุณหภูมิและความชื้นจากเซ็นเซอร์ในระบบติดตามการขนส่ง



รูปที่ 4.24 ระบบแสดงผลเส้นทางการขนส่งและข้อมูลเซ็นเซอร์ย้อนหลังของรถจักรยานยนต์

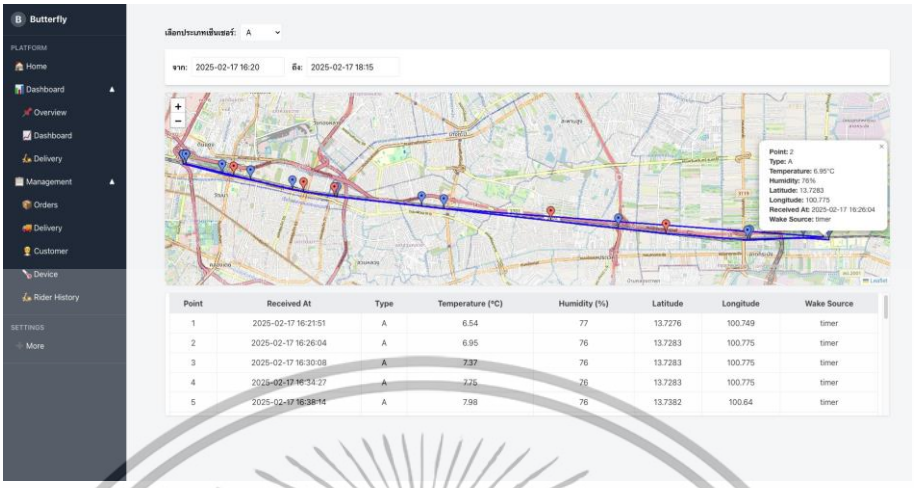
4.1.3.2 การทดสอบระบบติดตามการขนส่งร่วมกับรถไฟ

ในการทดสอบระบบติดตามการขนส่งร่วมกับรถไฟ อุปกรณ์ติดตามถูกติดตั้งไว้ในกล่องและวางบนกระเป๋าขนส่ง เพื่อจำลองการใช้งานจริงระหว่างการเดินทาง ระบบจะบันทึกค่าพิกัด GPS อุณหภูมิ และความชื้นแบบเรียลไทม์ พร้อมส่งข้อมูลผ่านเครือข่าย LTE ไปยัง MQTT Broker และแสดงผลบนแพลตฟอร์ม Butterfly Cold Chain โดยการทดสอบนี้ช่วยตรวจสอบว่าระบบสามารถติดตามตำแหน่งและส่งข้อมูลได้อย่างต่อเนื่อง แม้ในสภาพแวดล้อมของรถไฟที่มี

แรงสั่นสะเทือนและบางช่วงที่สัญญาณเครือข่ายอาจไม่เสถียร นอกจากนี้ยังช่วยวิเคราะห์ความแม่นยำของตำแหน่งที่บันทึก และตรวจสอบการเปลี่ยนแปลงของอุณหภูมิและความชื้นภายในกระเป๋าขนส่ง เพื่อให้แน่ใจว่าระบบสามารถทำงานได้อย่างเสถียรและเหมาะสมกับการใช้งานจริง แสดงได้ดังรูปที่ 4.25

รูปที่ 4.25 การทดสอบระบบติดตามการขนส่งบนรถไฟ

โดยระบบสามารถบันทึกพิกัด GPS อุณหภูมิ และความชื้นแบบเรียลไทม์ พร้อมส่งข้อมูลผ่านเครือข่าย LTE ไปยังแพลตฟอร์ม ระบบแสดงเส้นทางการเดินทางของรถไฟบนแผนที่ โดยมี จุดสีน้ำเงิน แสดงตำแหน่งที่บันทึกข้อมูล และ จุดสีแดง เป็นจุดเริ่มต้นและปลายทาง กล่องข้อมูลที่ปรากฏบนแผนที่ให้รายละเอียดของแต่ละจุด เช่น อุณหภูมิ ความชื้น เวลาที่บันทึก และพิกัดตำแหน่ง นอกจากนี้ ตารางด้านล่างยังช่วยให้สามารถตรวจสอบแนวโน้มการเปลี่ยนแปลงของข้อมูลได้อย่างง่ายดาย การทดสอบนี้ช่วยยืนยันว่าระบบสามารถติดตามเส้นทางขนส่ง ตรวจสอบสภาพแวดล้อม และบันทึกข้อมูลได้อย่างต่อเนื่อง ซึ่งมีประโยชน์ต่อการควบคุมคุณภาพสินค้าในระหว่างการขนส่ง แสดงได้ดังรูปที่ 4.26



รูปที่ 4.26 ระบบแสดงผลเส้นทางการขนส่งและข้อมูลเซ็นเซอร์ย้อนหลังของรถไฟ

4.2 ทดสอบการสร้างตารางฐานข้อมูลเพื่อให้สามารถใช้งานข้อมูลได้ในส่วนระบบหลังบ้าน

ในการทดสอบนี้จะทำการทดสอบสร้างตารางบนฐานข้อมูล MySQL ด้วย Django โดยใช้เฉพาะเครื่องมือที่มีใน Django ในการสร้างตารางฐานข้อมูลอัตโนมัติ ซึ่งอันดับแรกจะต้องทำการตั้งค่า Django ให้สามารถเชื่อมต่อกับฐานข้อมูลแสดงได้ดังรูปที่ 4.27

รูปที่ 4.27 การตั้งค่าใน Django เพื่อทำการเชื่อมต่อกับฐานข้อมูล MySQL

จากนั้นจะทำการเขียนโปรแกรมใน Django เพื่อเขียนข้อมูลของตารางในฐานข้อมูลที่ต้องการสร้างว่ามีลักษณะอย่างไรแสดงดังรูปที่ 4.28

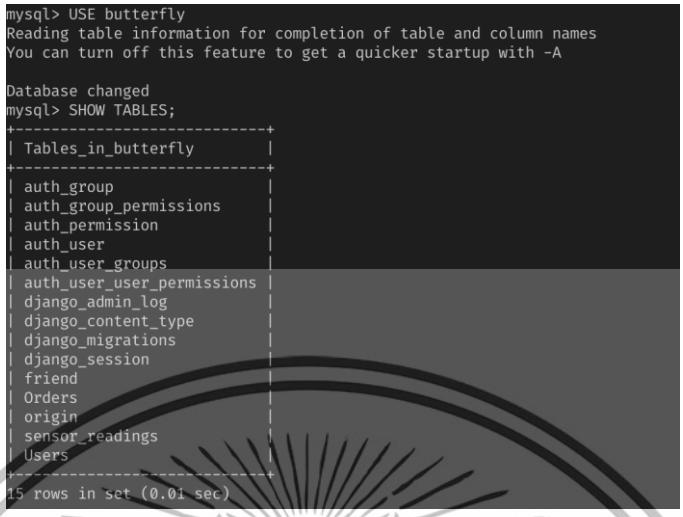
```python
from django.db import models

class Users(models.Model):
    customer_name = models.CharField(max_length=255)
    address = models.TextField()
    latitude = models.DecimalField(max_digits=50, decimal_places=30, null=True, blank=True)  # Latitude
    longitude = models.DecimalField(max_digits=50, decimal_places=30, null=True, blank=True)  # Longitude

    class Meta:
        db_table = 'Users'

class Orders(models.Model):
    customer_name = models.ForeignKey(Users, on_delete=models.CASCADE)
    product = models.TextField()
    delivery_date = models.DateField()

    class Meta:
        db_table = 'Orders'


class Friend(models.Model):
    customer_name = models.CharField(max_length=255)
    address = models.TextField()
    product = models.TextField()

    class Meta:
        db_table = 'friend'
        managed = False  # uen Django
class Origin(models.Model):
    origin = models.CharField(max_length=255)
    latitude = models.DecimalField(max_digits=50, decimal_places=30, null=True, blank=True)  # Latitude
    longitude = models.DecimalField(max_digits=50, decimal_places=30, null=True, blank=True)  # Longitude
    class Meta:
        db_table = 'origin'
class SensorData(models.Model):
    device = models.CharField(max_length=10, blank=True, null=True)  # varchar(10)
    temperature = models.FloatField(blank=True, null=True)  # float
    humidity = models.FloatField(blank=True, null=True)  # float
    latitude = models.FloatField(blank=True, null=True)  # float
    longitude = models.FloatField(blank=True, null=True)  # float
    timestamp = models.DateTimeField(auto_now_add=True, blank=True, null=True)  # timestamp

    class Meta:
        db_table = 'sensor_readings'
```

รูปที่ 4.28 การเขียนโปรแกรมใน Django เพื่อเขียนข้อมูลลักษณะของตารางในฐานข้อมูล MySQL

ในขั้นตอนสุดท้ายจะทำการ make migrations และ migrate ผ่าน manage.py เพื่อปรับให้ข้อมูลในฐานข้อมูลกับข้อมูลใน Django ตรงกันจากนั้นตารางจะถูกสร้างลงบน Database อัตโนมัติ แสดงได้ดังรูปที่ 4.29 และ 4.30

```
(.venv) EARTH ~/Desktop/ButterFlyproject/backend_django/API_Service → python manage.py makemigrations
No changes detected
(.venv) EARTH ~/Desktop/ButterFlyproject/backend_django/API_Service → python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, database, sessions
Running migrations:
  No migrations to apply.
```

รูปที่ 4.29 การทำ makemigration และ migrate ผ่าน manage.py



รูปที่ 4.30 ผลลัพธ์ที่ได้หลังจากการทำ makemigrations และ migrate ผ่าน manage.py

ต่อไปทำการตรวจสอบชนิดของข้อมูลคอลัมน์ในแต่ละตาราง MySQL ซึ่งแสดงตาราง Orders ดังรูปที่ 4.31

รูปที่ 4.31 ชนิดของข้อมูลคอลัมน์ในแต่ละตาราง Orders

4.3 การทดสอบการทำงานของระบบค้นหาเส้นทาง

4.3.1 ทดสอบประสิทธิภาพของการแบ่งกลุ่มพิกัดด้วยวิธีแบบ K-means Clustering

ในส่วนนี้จะนำเสนอผลการทดสอบประสิทธิภาพของการแบ่งกลุ่มข้อมูลโดยใช้วิธี K-means Clustering ซึ่งเป็นวิธีการที่นิยมใช้ในงานวิเคราะห์ข้อมูลเพื่อการจัดกลุ่ม (clustering) ข้อมูลที่มีลักษณะคล้ายคลึงกัน โดยการทดสอบนี้จะดำเนินการในสามสถานการณ์ ได้แก่ การทดสอบกับข้อมูล 10 รายการ และ 20 รายการ โดยมีขั้นตอนการทดสอบดังนี้

4.3.1.1 เตรียมข้อมูลที่ใช้ในการทดสอบการแบ่งกลุ่มด้วยวิธีแบบ K-means Clustering โดยจะทำการสมมติพิกัดขึ้นมา 10 จุด , 20 จุด แสดงดังรูปที่ 4.32 และ รูปที่ 4.33

```
str_latlng = [
    ("13.7563", "100.5018"),  # Point 1
    ("13.7565", "100.5020"),  # Point 2
    ("13.8120", "100.5968"),  # Point 3
    ("13.8130", "100.5978"),  # Point 4
    ("13.7501", "100.5002"),  # Point 5
    ("13.7800", "100.5600"),  # Point 6
    ("13.7700", "100.5700"),  # Point 7
    ("13.7600", "100.5200"),  # Point 8
    ("13.7550", "100.5100"),  # Point 9
    ("13.7510", "100.5010")   # Point 10
]
```

รูปที่ 4.32 พิกัดที่สมมุติขึ้นมา 10 จุด

```
str_latlng = [
    ("13.7563", "100.5018"),  # Point 1
    ("13.7565", "100.5020"),  # Point 2
    ("13.8120", "100.5968"),  # Point 3
    ("13.8130", "100.5978"),  # Point 4
    ("13.7501", "100.5002"),  # Point 5
    ("13.7800", "100.5600"),  # Point 6
    ("13.7700", "100.5700"),  # Point 7
    ("13.7600", "100.5200"),  # Point 8
    ("13.7550", "100.5100"),  # Point 9
    ("13.7510", "100.5010"),  # Point 10
    ("13.7540", "100.5400"),  # Point 11
    ("13.7555", "100.5800"),  # Point 12
    ("13.7610", "100.5900"),  # Point 13
    ("13.7620", "100.5920"),  # Point 14
    ("13.7705", "100.5805"),  # Point 15
    ("13.7715", "100.5750"),  # Point 16
    ("13.7780", "100.5590"),  # Point 17
    ("13.7590", "100.5220"),  # Point 18
    ("13.7640", "100.5040"),  # Point 19
    ("13.7520", "100.5500")   # Point 20
]
```

รูปที่ 4.33 พิกัดที่สมมุติขึ้นมา 20 จุด

พิกัดจุดเริ่มต้น 1 แสดงดังรูปที่ 4.34

```
str_origin = ("13.7560", "100.5500")  # Origin point, central location
```

รูปที่ 4.34 สมมติพิกัดจุดเริ่มต้นขึ้นมา 1 จุด

4.3.1.2 แบ่งกลุ่มข้อมูลด้วยเทคนิคแบบ K-means Clustering ซึ่งใช้ภาษา Python ในการเขียนโปรแกรมและใช้ Library Scikit-learn ในการ import KMeans จาก sklearn แสดงได้ดังรูปที่ 4.35

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

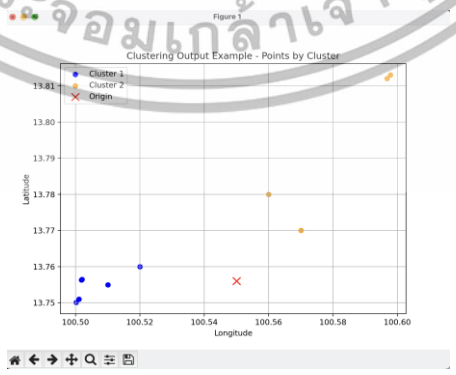
รูปที่ 4.35 การ Import Library Kmeans และ silhouette_score

1) ผลลัพธ์การแบ่งกลุ่ม

โดยในขั้นตอนนี้จะแบ่งกลุ่มออกเป็นสองกลุ่มและแสดงพิกัดของจุดว่าถูกแบ่งไปให้กลุ่มไหนสำหรับชุดตัวอย่างพิกัดสมมติ 10 จุด แสดงได้ดังรูปที่ 4.36 และรูปที่ 4.37

```
Route 1
Point 1: [13.756, 100.55]
Point 2: [13.7563, 100.5018]
Point 3: [13.7565, 100.502]
Point 4: [13.7501, 100.5002]
Point 5: [13.76, 100.52]
Point 6: [13.755, 100.51]
Point 7: [13.751, 100.501]
Point 8: [13.756, 100.55]
Route 2
Point 1: [13.756, 100.55]
Point 2: [13.812, 100.5968]
Point 3: [13.813, 100.5978]
Point 4: [13.78, 100.56]
Point 5: [13.77, 100.57]
Point 6: [13.756, 100.55]
```

รูปที่ 4.36 ผลลัพธ์การแบ่งกลุ่มด้วยวิธีแบบ K-means Clustering ของตัวอย่างพิกัด 10 จุด

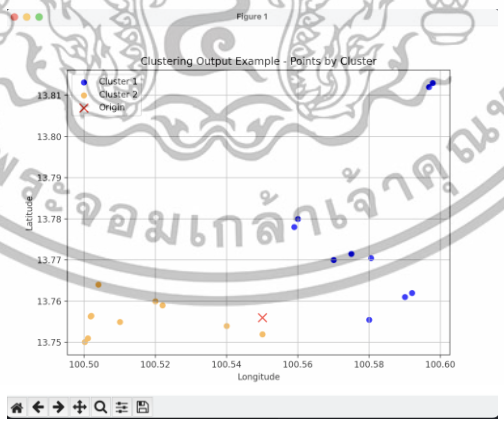


รูปที่ 4.37 กราฟผลลัพธ์การแบ่งกลุ่มด้วยวิธีแบบ K-means Clustering ของตัวอย่างพิกัด 10 จุด

ต่อไปเป็นชุดตัวอย่างพิกัดสมมติ 20 จุด แสดงได้ดังรูปที่ 4.38 และรูปที่ 4.39

```
Route 1
Point 1: [13.756, 100.55]
Point 2: [13.812, 100.5968]
Point 3: [13.813, 100.5978]
Point 4: [13.78, 100.56]
Point 5: [13.77, 100.57]
Point 6: [13.7555, 100.58]
Point 7: [13.761, 100.59]
Point 8: [13.762, 100.592]
Point 9: [13.7705, 100.5805]
Point 10: [13.7715, 100.575]
Point 11: [13.778, 100.559]
Point 12: [13.756, 100.55]
Route 2
Point 1: [13.756, 100.55]
Point 2: [13.7563, 100.5018]
Point 3: [13.7565, 100.502]
Point 4: [13.7501, 100.5002]
Point 5: [13.76, 100.52]
Point 6: [13.755, 100.51]
Point 7: [13.751, 100.501]
Point 8: [13.754, 100.54]
Point 9: [13.759, 100.522]
Point 10: [13.764, 100.504]
Point 11: [13.752, 100.55]
Point 12: [13.756, 100.55]
```

รูปที่ 4.38 ผลลัพธ์การแบ่งกลุ่มด้วยวิธีแบบ K-means Clustering ของตัวอย่างพิกัด 20 จุด



รูปที่ 4.39 กราฟผลลัพธ์การแบ่งกลุ่มด้วยวิธีแบบ K-means Clustering ของตัวอย่างพิกัด 20 จุด

2) แสดงผลลัพธ์ Silhouette Score

เพื่อทดสอบความน่าเชื่อถือของการแบ่งกลุ่ม โดยค่ายิ่งใกล้ 1 มากเท่าไร ยิ่งมีความน่าเชื่อถือมากขึ้นเท่านั้น ซึ่งแสดงผลลัพธ์ได้ดังรูปที่ 4.40 และรูปที่ 4.41

```
Silhouette Score: 0.741
```

รูปที่ 4.40 ผลลัพธ์ของค่า Silhouette Score สำหรับชุดพิกัดตัวอย่าง 10 จุด

```
Silhouette Score: 0.615
```

รูปที่ 4.41 ผลลัพธ์ของค่า Silhouette Score สำหรับชุดพิกัดตัวอย่าง 20 จุด

4.3.1 การทดสอบประสิทธิภาพของการจัดลำดับเส้นทางที่สั้นที่สุดด้วยวิธีแบบ Brute Force

ในการทดสอบนี้เราจะทำการทดสอบวิธีการแก้ปัญหา Traveling Salesman Problem โดยหนึ่งใน Solution ที่สามารถนำมาประยุกต์ใช้ในการแก้ปัญหาคือวิธีการทำแบบ Brute Force โดยจะทำการสร้างลำดับเส้นทางทั้งหมดที่เป็นไปได้จากนั้นจะทำการคำนวนระยะทางทั้งหมดโดยใช้ Distance Matrix มาเป็นตัวช่วยและจะเลือกลำดับเส้นทางที่มีระยะเวลาการเดินทางสั้นที่สุดโดยมีขั้นตอนการทดสอบดังนี้การสร้าง Distance Matrix

4.3.1.1 การสร้าง Distance Matrix

โดยในขั้นตอนนี้จะสร้างเพื่อใช้ในการสมมติค่าระยะทางระหว่างจุด 12 จุด (พิกัดจุดหมายที่ต้องเดินทางไปส่ง 11 จุดและจุดเริ่มต้น 1 จุด) ดังแสดงในรูปที่ 4.42

```
distance_matrix = [
    [0, 19, 76, 33, 61, 62, 61, 26, 5, 18, 84, 72],
[19, 0, 67, 41, 63, 58, 76, 53, 59, 11, 8, 50],
[76, 67, 0, 7, 10, 89, 52, 17, 83, 97, 38, 72],
[33, 41, 7, 0, 51, 42, 73, 41, 89, 34, 62, 88],
[61, 63, 10, 51, 0, 30, 13, 74, 38, 71, 25, 44],
[62, 58, 89, 42, 30, 0, 92, 60, 31, 78, 8, 63],
[61, 76, 52, 73, 13, 92, 0, 12, 41, 86, 11, 67],
[26, 53, 17, 41, 74, 60, 12, 0, 44, 80, 80, 14],
[5, 59, 83, 89, 38, 31, 41, 44, 0, 42, 86, 95],
[18, 11, 97, 34, 71, 78, 86, 80, 42, 0, 29, 100],
[84, 8, 38, 62, 25, 8, 11, 80, 86, 29, 0, 50],
[72, 50, 72, 88, 44, 63, 67, 14, 95, 100, 50, 0],
```

รูปที่ 4.42 Distance Matrix ของ ระยะทางระหว่างจุด 12 จุด

4.3.1.2 การเขียนโปรแกรมคำนวนลำดับเส้นทางที่ดีที่สุดด้วยวิธีแบบ Brute Force

โดยจะสร้างเส้นทางทุกเส้นทางที่เป็นไปได้จากนั้นจะคำนวณระยะทางรวมและจะเลือกเส้นทางที่มีระยะรวมสั้นที่สุดเป็นผลลัพธ์ซึ่งแสดงรายละเอียดการเขียนโปรแกรมและฟังก์ชันที่ใช้แสดงดังรูปที่ 4.43

รูปที่ 4.43 โปรแกรมการแก้ปัญหา Traveling Salesman Problem ด้วยวิธีแบบ Brute Force

4.3.1.3 การวิเคราะห์ผลลัพธ์การคำนวณเส้นทางที่ดีที่สุดด้วยวิธีแบบ Brute Force

จากการคำนวณพบว่าเวลาที่ใช้ในการคำนวณข้อมูลของพิกัดที่ต้องไปส่ง 11 จุดและกลับมายังจุดเริ่มต้นใช้เวลาไปทั้งสิ้น 45.5608 วินาทีและมีลำดับเส้นทางและระยะทางรวมซึ่งแสดงได้ดังรูปที่ 4.44

รูปที่ 4.44 ผลลัพธ์จากการคำนวณการจัดลำดับเส้นทางด้วยวิธีแบบ Brute Force

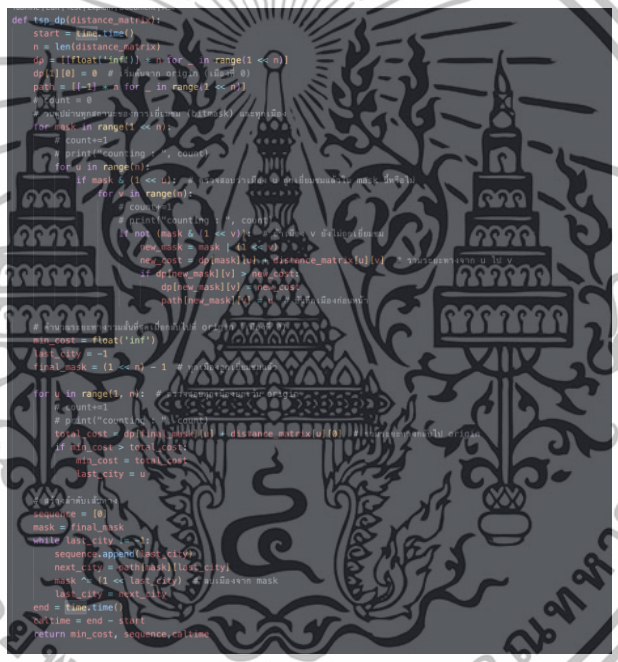
4.3.2 การทดสอบประสิทธิภาพของการจัดลำดับเส้นทางที่สั้นที่สุดด้วยวิธีแบบ Dynamic Programming

ในการทดสอบนี้เราจะทำการทดสอบวิธีการแก้ปัญหา Traveling Salesman Problem โดยอีกหนึ่งใน Solution ที่สามารถนำมาประยุกต์ใช้ในการแก้ปัญหาคือวิธีการทำแบบ

Dynamic Programming โดยจะพิจารณาเส้นทางที่เป็นไปได้ทั้งหมด และแบ่งการคำนวณออกเป็นขั้นตอนย่อย ๆ เพื่อหาคำตอบที่สั้นที่สุด และเก็บผลลัพธ์ของขั้นตอนย่อยไว้เพื่อลดการคำนวณซ้ำในอนาคต

4.3.2.1 การเขียนโปรแกรมคำนวนลำดับเส้นทางที่ดีที่สุดด้วยวิธีแบบ Dynamic Programming

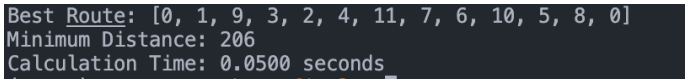
โดยในขั้นตอนนี้จะใช้วิธีที่เรียกว่า Memorization หรือการบันทึกผลลัพธ์ของปัญหาย่อย ๆ ที่เคยคำนวณไปแล้ว เพื่อนำมาใช้ซ้ำในการคำนวณโดยแสดงโปรแกรมการคำนวณดังรูปที่ 4.45



รูปที่ 4.45 โปรแกรมการแก้ปัญหา Traveling Salesman Problem ด้วยวิธีแบบ Dynamic Programming

4.3.2.2 การวิเคราะห์ผลลัพธ์การคำนวณลำดับเส้นทางที่ดีที่สุดด้วยวิธีแบบ Dynamic Programming

จากการคำนวณพบว่าเวลาที่ใช้ในการคำนวณข้อมูลของพิกัดที่ต้องไปส่ง 11 จุดและกลับมายังจุดเริ่มต้นใช้เวลาไปทั้งสิ้น 0.500 วินาทีและมีลำดับเส้นทางและระยะทางรวมซึ่งแสดงได้ดังรูปที่ 4.46



รูปที่ 4.46 ผลลัพธ์จากการคำนวณการจัดลำดับเส้นทางด้วยวิธีแบบ Dynamic Programming

4.3.3 การทดสอบเปรียบเทียบประสิทธิภาพของวิธีการจัดลำดับเส้นทางที่สั้นที่สุดระหว่าง Brute Force และ Dynamic Programming

ในการทดสอบนี้เราต้องการเปรียบเทียบประสิทธิภาพระหว่างวิธี Brute Force และ Dynamic Programming ในการหาลำดับเส้นทางที่สั้นที่สุดสำหรับปัญหา Traveling Salesman Problem (TSP) โดยเฉพาะเมื่อจำนวนจุดที่ต้องเดินทางมีจำนวนเพิ่มขึ้นเรื่อย ๆ จะยิ่งเพิ่มเวลาในการคำนวนซึ่งเราได้ทำการกำหนดสภาพแวดล้อมของวิธีทั้งสองให้เหมือนกันและทำการเปรียบเทียบเพื่อดูความแตกต่างโดยจะมีขั้นตอนการทดสอบดังนี้

4.3.3.1 การตรวจสอบว่าเส้นทางการเดินทางของวิธีทั้งสองเหมือนหรือแตกต่างกันหรือไม่ ซึ่งจากผลลัพธ์การคำนวณเส้นทางแบบวิธี Brute Force แสดงได้ดังรูปที่ 4.47

รูปที่ 4.47 ผลลัพธ์การคำนวณเส้นทางด้วยวิธีแบบ Brute Force

การคำนวณเส้นทางแบบวิธี Dynamic Programming แสดงได้ดังรูป 4.48

รูปที่ 4.48 ผลลัพธ์การคำนวณเส้นทางด้วยวิธีแบบ Dynamic Programming

4.3.3.2 การตรวจสอบว่าระยะทางรวมนั้นเท่ากันหรือไม่ จากการเปรียบเทียบผลลัพธ์ที่ได้พบว่าค่าของลำดับเส้นทางและระยะทางที่ได้จากวิธีการทั้งสองนั้นได้เหมือนกันจึงสรุปได้ว่าทั้งสองวิธีให้ผลลัพธ์ที่เท่ากัน แสดงได้ดังรูปที่ 4.49 และ 4.50

รูปที่ 4.49 ผลลัพธ์ระยะทางด้วยวิธีแบบ Brute Force

รูปที่ 4.50 ผลลัพธ์ระยะทางด้วยวิธีแบบ Dynamic Programming

4.3.3.3 ตรวจสอบเวลาที่ใช้ในการคำนวนของทั้งสองวิธีและเปรียบเทียบถึงความแตกต่าง

จากการตรวจสอบเวลาที่ใช้ในการคำนวณด้วยวิธีแบบ Brute Force และเวลาที่ใช้ในการคำนวณด้วยวิธีแบบ Dynamic Programming พบว่าเวลาที่ใช้ในการคำนวณของการขนส่งที่มีพิกัด 11 จุดวิธีแบบ Brute Force ใช้เวลามากกว่าวิธีแบบ Dynamic Programming ถึง 90 เท่า แสดงได้ดังรูปที่ 4.51 และรูปที่ 4.52

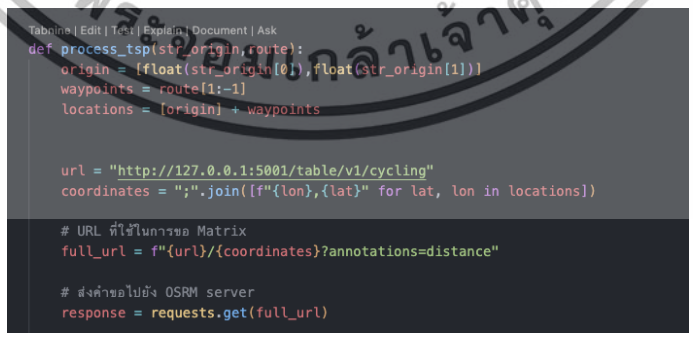
รูปที่ 4.51 ผลลัพธ์เวลาที่ใช้ในการคำนวนด้วยวิธีแบบ Brute Force

รูปที่ 4.52 ผลลัพธ์เวลาที่ใช้ในการคำนวนด้วยวิธีแบบ Dynamic Programming

4.3.4 การทดสอบการรวมการทำงานของระบบย่อยต่างๆ ให้ใช้เส้นทางจริงในการคำนวนเส้นทางอัจฉริยะ

4.3.5.1 การเขียนโปรแกรมเพื่อรวมทุกฟังก์ชันเข้าด้วยกัน

หลังจากที่เราทำการทดสอบการทำงานของระบบต่างๆ แล้วพบว่าระบบแบบแยกยังสามารถทำงานได้ดีซึ่งขั้นต่อไปเราจะนำส่วนต่างๆมาประกอบเข้าด้วยกันและจะทำการเรียกใช้ OSRM ในฟังก์ชันของ Dynamic programming เพื่อใช้งานข้อมูลเส้นทางจริงเป็น Distance Matrix ในการคำนวณ แสดงได้ดังรูปที่ 4.53



รูปที่ 4.53 การดึงข้อมูล Distance Matrix จาก OSRM ในฟังก์ชัน Dynamic programming

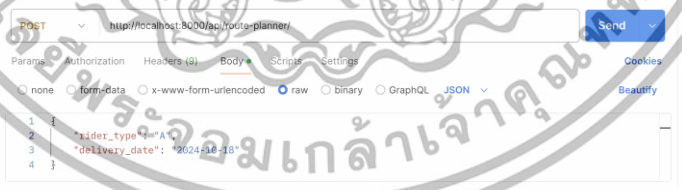
4.3.5.2 การเตรียมข้อมูลใน Database เพื่อให้พร้อมใช้ในการทดสอบ

หลังจากที่เราทำการเขียนโปรแกรมเพื่อใช้ในส่วนต่างๆ เรียบร้อยแล้วต่อมาเราจะทำการเพิ่มข้อมูลลงไปในฐานข้อมูลเพื่อทดสอบการใช้งานจริงทั้งระบบดัง แสดงในรูปที่ 4.54

รูปที่ 4.54 ตัวอย่างข้อมูลในตาราง Orders

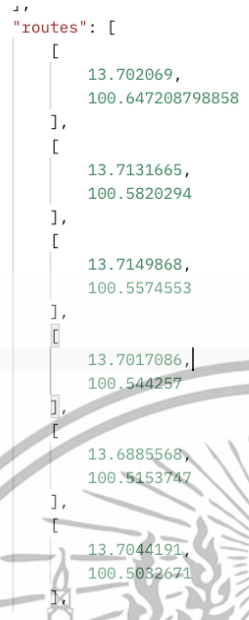
4.3.5.3 การทดสอบการ POST Request เพื่อดูการตอบสนองต่อ Client จาก Server เมื่อทำการขอข้อมูลลำดับเส้นทาง

หลังจากที่เราทำการเตรียมข้อมูลในตารางฐานข้อมูลและทำการเขียนโปรแกรมเรียบร้อยแล้วเราจะทำการทดสอบการเรียกใช้ข้อมูลจากฝั่ง Client โดยใช้ Postman ในการ POST Request ซึ่งแสดงดังรูปที่ 4.55



รูปที่ 4.55 ตัวอย่างการ POST Request เพื่อขอลำดับเส้นทางจาก Django

เมื่อทำการ Send Request แล้วจะแสดงผลการตอบสนองจาก Django ดังรูปที่ 4.56



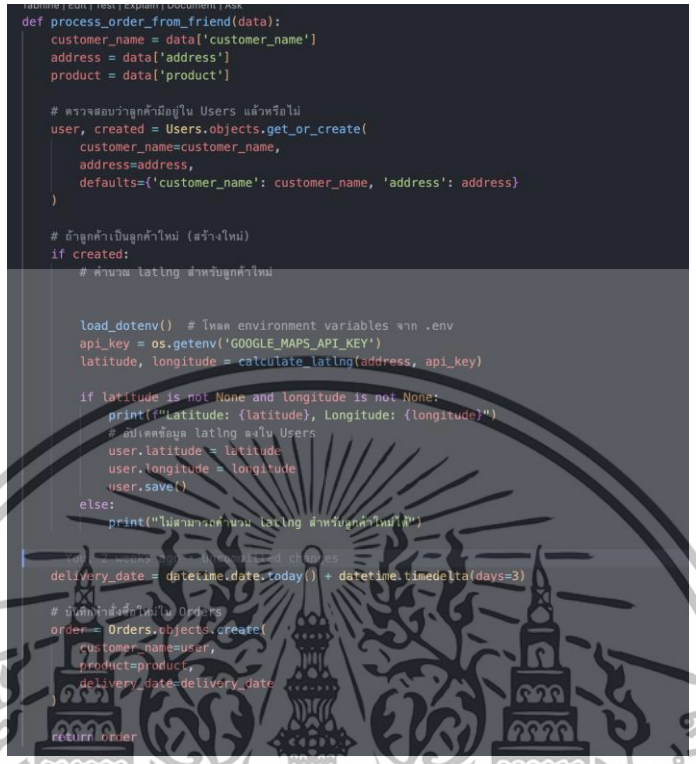
รูปที่ 4.56 ผลการตอบสนองจาก Django เข้ามาที่ Postman

4.4 การทดสอบการทำงานของระบบการแปลงข้อมูลที่อยู่เป็นพิกัดละติจูด ลองจิจูด

จากการทดสอบเพิ่มข้อมูลลงใน Database ผ่าน API สำเร็จแล้วต่อมาจะทำการทดสอบการแปลงข้อมูลที่อยู่ในรูปแบบข้อความให้อยู่ในรูปแบบของพิกัดโดยใช้บริการของ Google API ในการแปลงซึ่งมีขั้นตอนการทดสอบดังนี้

4.4.1 การเขียนโปรแกรมเพื่อทำการแปลงข้อมูลที่อยู่ในรูปแบบข้อความให้อยู่ในรูปแบบพิกัดละติจูด ลองจิจูด

หลังจากที่มีการ POST Request เพื่อทำการเพิ่มข้อมูลคำสั่งซื้อลงในตารางฐานข้อมูล ระบบจะทำการแปลงข้อมูลอยู่ที่อยู่ ที่ยังไม่มีพิกัดอยู่ในฐานข้อมูล ระบบจะทำการเรียกใช้บริการ Google API เพื่อทำการแปลงข้อมูลที่อยู่ในรูปแบบข้อความให้อยู่ในรูปพิกัด โดยแสดงได้ดังรูปที่ 4.57



```python
def process_order_from_friend(data):
    customer_name = data['customer_name']
    address = data['address']
    product = data['product']

    # ตรวจสอบว่าลูกค้ามีอยู่ใน Users แล้วหรือไม่
    user, created = Users.objects.get_or_create(
        customer_name=customer_name,
        address=address,
        defaults={'customer_name': customer_name, 'address': address}
    )

    # ถ้าลูกค้าเป็นลูกค้าใหม่ (สร้างใหม่)
    if created:
        # คำนวณ latlng สำหรับลูกค้าใหม่


        load_dotenv()  # โหลด environment variables จาก .env
        api_key = os.getenv('GOOGLE_MAPS_API_KEY')
        latitude, longitude = calculate_latlng(address, api_key)

        if latitude is not None and longitude is not None:
            print(f"Latitude: {latitude}, Longitude: {longitude}")
            # อัปเดตข้อมูล latlng ลงใน Users
            user.latitude = latitude
            user.longitude = longitude
            user.save()
        else:
            print("ไม่สามารถคำนวณ latlng สำหรับลูกค้าใหม่ได้")

    delivery_date = datetime.date.today() + datetime.timedelta(days=3)

    # บันทึกคำสั่งซื้อใหม่ใน Orders
    order = Orders.objects.create(
        customer_name=user,
        product=product,
        delivery_date=delivery_date
    )
    return order
```

รูปที่ 4.57 โปรแกรมการแปลงที่อยู่ในรูปแบบข้อความให้เป็นพิกัด

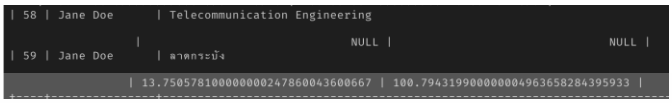
4.4.2 ทำการเพิ่มรายการคำสั่งซื้อลงไปใน Database ผ่าน API

ทำการทดสอบโดยทำการเพิ่มรายการคำสั่งซื้อลงไปใน Database ซึ่งแสดงได้ดังรูปที่ 4.58

รูปที่ 4.58 การเพิ่มรายการคำสั่งซื้อลงในฐานข้อมูลโดยการใช้ Postman ตรวจสอบผลลัพธ์

4.4.3 ตรวจสอบผลลัพธ์

ตรวจสอบข้อมูลในตาราง Users ว่ามีชื่อผู้ใช้และพิกัดที่อยู่ถูกเพิ่มเข้ามาในตารางแล้ว แสดงได้ดังรูปที่ 4.59



รูปที่ 4.59 ตรวจสอบข้อมูลในตาราง Users

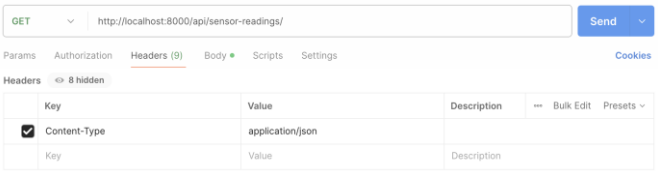
ตรวจสอบ Logs จาก Django ว่ามีการตอบสนองอย่างไรแสดงได้ดังรูปที่ 4.60

รูปที่ 4.60 ตรวจสอบ Logs จาก Django

4.5 การทดสอบการทำงานของระบบหลังบ้าน

4.5.1 การทดสอบการส่งข้อมูลของเซนเซอร์อุณหภูมิ ความชื้นและข้อมูลพิกัดของอุปกรณ์จากฐานข้อมูลผ่าน API

ในการทดการทดสอบการส่งข้อมูลของเซนเซอร์อุณหภูมิ ความชื้นและข้อมูลพิกัดของอุปกรณ์จาก Database ผ่าน API เพื่อทำการตรวจสอบความถูกต้องของการส่งข้อมูลระหว่าง Client และ Server ผ่าน API โดยจะใช้ Postman แทนฝั่ง Client ในการส่งคำขอไปยัง Server ที่ใช้ Django ซึ่งแสดงได้ดังรูปที่ 4.61



รูปที่ 4.61 การส่งคำขอข้อมูลของเซนเซอร์อุณหภูมิ และข้อมูลพิกัดของอุปกรณ์ผ่าน Postman

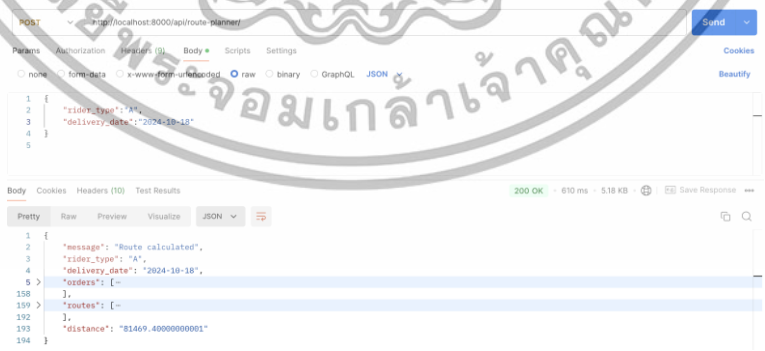
ผลการตอบสนองจากฝั่งของ Server แสดงได้ดังรูปที่ 4.62


```
{
    "id": 328,
    "device": "A",
    "temperature": 22.75,
    "humidity": 61.0,
    "latitude": 13.7129,
    "longitude": 100.771,
    "timestamp": "2024-10-10T00:53:09Z"
},
{
    "id": 329,
    "device": "B",
    "temperature": 22.76,
    "humidity": 64.0,
    "latitude": 13.7129,
    "longitude": 100.771,
    "timestamp": "2024-10-10T00:53:24Z"
},
```


รูปที่ 4.62 ผลการตอบสนองจาก Server ที่ส่งเข้ามายัง Postman

4.5.2 การออกแบบระบบทดสอบการส่งข้อมูลลำดับเส้นทางที่สั้นที่สุดของแต่ละพนักงานขนส่งในวันที่ระบุผ่าน API

ในการทดสอบการส่งข้อมูลของลำดับเส้นทางที่สั้นที่สุดและรายละเอียดที่เกี่ยวข้องผ่าน API เพื่อทำการตรวจสอบความถูกต้องของการส่งข้อมูลระหว่าง Client และ Server ผ่าน API โดยจะใช้ Postman แทนฝั่ง Client ในการส่งคำขอไปยัง Server ที่ใช้ Django และแสดงผลการตอบสนองจากฝั่งของ Server ซึ่งแสดงดังรูปที่ 4.63

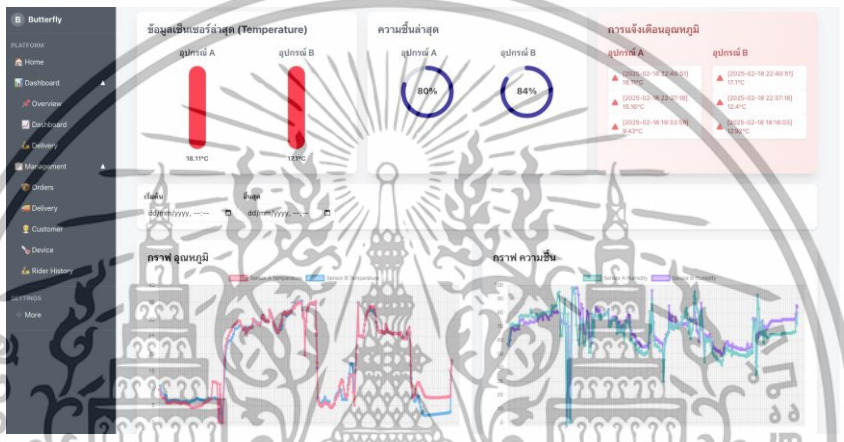


รูปที่ 4.63 ตัวอย่างแสดงคำขอข้อมูลสำหรับพนักงานขนส่งซึ่งประกอบไปด้วยลำดับเส้นทาง, ระยะทาง และและรายละเอียดคำสั่งซื้อ

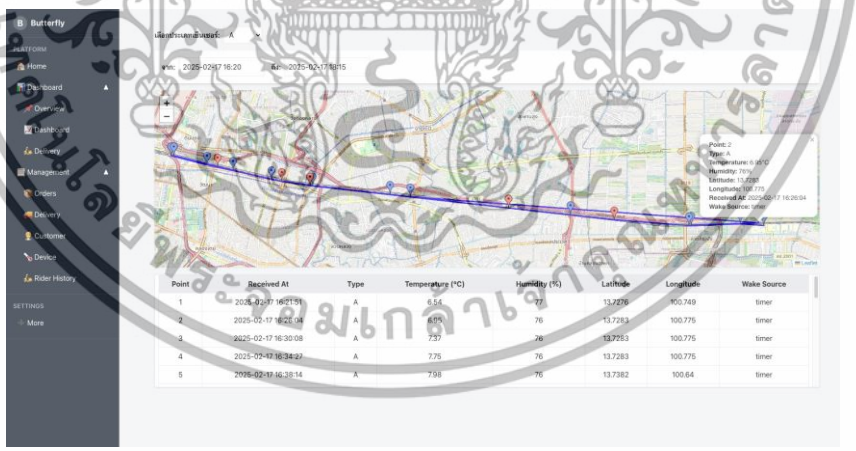
4.6 การทดสอบการทำงานของระบบหน้าบ้าน

4.6.1 การทดสอบการแสดงผลข้อมูลเซนเซอร์อุณหภูมิ ความชื้นและข้อมูลพิกัดของอุปกรณ์บนหน้า Dashboard

การทดสอบนี้เป็นการทดสอบเพื่อตรวจสอบการแสดงผลข้อมูลในส่วนของ User Interface การดึงข้อมูลจาก Database และ Server ผ่าน API โดยใช้ ReactJS ซึ่งแสดงได้ดังรูปที่ 4.64 รูปที่ 4.65



รูปที่ 4.64 ตัวอย่างการแสดงข้อมูลบน Dashboard

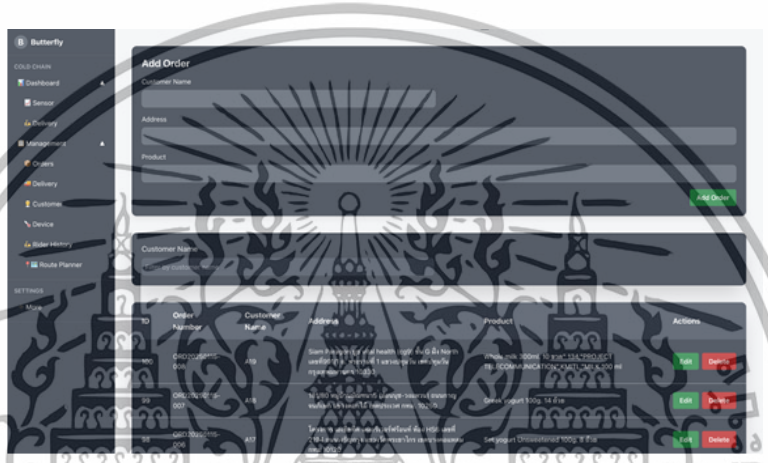


รูปที่ 4.65 ตัวอย่างการแสดงข้อมูลบน Dashboard

4.6.2 การทดสอบการแสดงผลข้อมูลในส่วนของหน้าการจัดการของผู้ดูแลระบบ

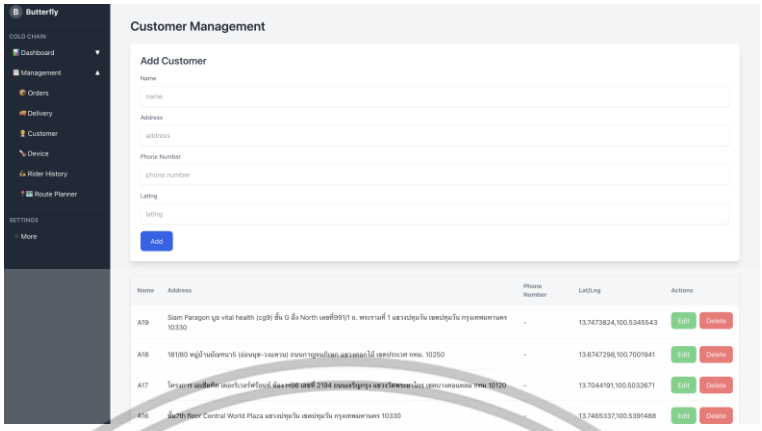
การทดสอบนี้เป็นการทดสอบเพื่อตรวจสอบการแสดงผลข้อมูลในส่วนของ User Interface และการดึงข้อมูลจาก Server ผ่าน API โดยใช้ ReactJS เป็น frontend ซึ่งจะแสดงการเขียนโปรแกรมการดึงข้อมูลจาก Server ในส่วนของหน้าจัดการรายการสินค้าแสดงได้ดังรูปที่ 4.66

หน้าจัดการรายการสินค้าที่จะต้องจัดส่งแสดง ได้ดังรูปที่ 4.67 หน้าจัดการรายชื่อลูกค้าแสดงได้ดังรูป 4.64 หน้าจัดการรายการข้อมูลอุปกรณ์แสดงได้ดังรูปที่ 4.68



รูปที่ 4.66 ตัวอย่างการแสดงผลข้อมูลรายการคำสั่งซื้อบนหน้า Orders

รูปที่ 4.67 ตัวอย่างการแสดงผลข้อมูลรายการคำสั่งซื้อบนหน้า Orders

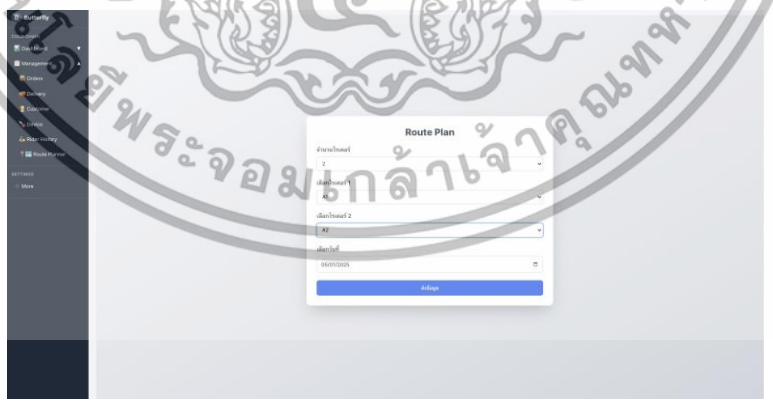


รูปที่ 4.68 ตัวอย่างการแสดงผลข้อมูลรายการคำสั่งซื้อบนหน้า Orders

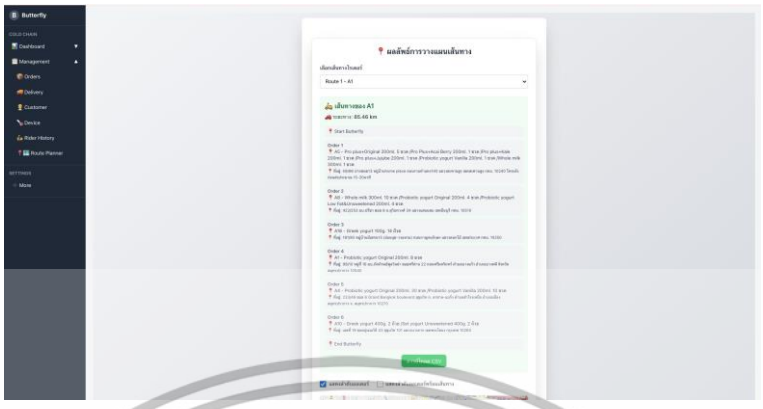
4.6.3 การทดสอบการแสดงผลข้อมูลของการจัดลำดับเส้นทางของพนักงานแต่ละคน พร้อมทั้งการ Export CSV.file

ในการทดสอบนี้จะเป็นการทดสอบการแสดงผลข้อมูลของการจัดลำดับเส้นทางบนหน้าเว็บไซต์พร้อมทั้งการทดสอบการโหลดข้อมูล CSV.file เพื่อให้มีความสามารถในการวางแผนการเดินทางล่วงหน้าได้อย่างสะดวกและแม่นยำ

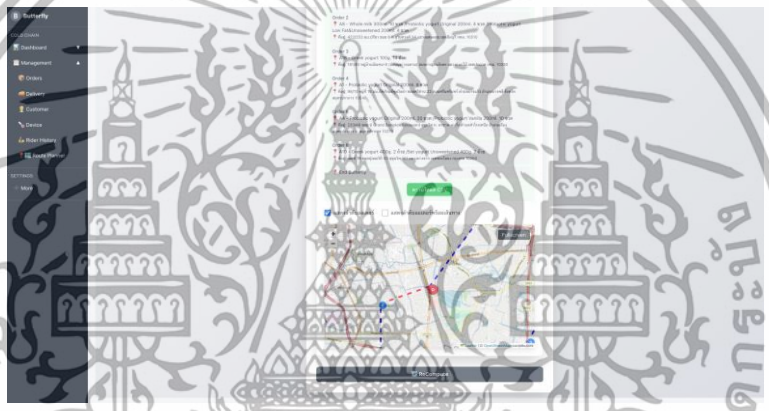
ทำการเลือกจำนวนของพนักงานที่จะต้องใช้จัดส่งในวันนั้นๆที่จัดส่ง พร้อมทั้งระบุรายชื่อของพนักงานที่จะทำการเดินทางจัดส่งและวันที่จะจัดส่ง แสดงได้ดังรูปที่ 4.69 รูปที่ 4.70 รูปที่ 4.71 รูปที่ 4.72 และรูปที่ 4.73



รูปที่ 4.69 การเลือกข้อมูลของหน้า Route Plan



รูปที่ 4.70 ผลลัพธ์ของข้อมูลเส้นทางลำดับการเดินทางและรายการสินค้า



รูปที่ 4.71 เส้นทางและลำดับเส้นทาง

รูปที่ 4.72 เส้นทางในระยะการกระจัด

รูปที่ 4.73 เส้นทางและลำดับเส้นทางโดยแสดงการวิ่งตามถนนจริง

บทที่ 5

สรุปผลและข้อเสนอแนะ

5.1 สรุปผล

ปริญญานิพนธ์นี้ประสบความสำเร็จในการพัฒนาระบบติดตามการขนส่งที่มีประสิทธิภาพสูง โดยใช้เซ็นเซอร์วัดอุณหภูมิ ความชื้น และโมดูล GPS ในการเก็บข้อมูลสิ่งแวดล้อมระหว่างการขนส่ง ข้อมูลที่เก็บได้ถูกส่งไปยังฐานข้อมูลทุกๆ 3 นาทีผ่านโครงข่าย LTE และใช้พลังงานจากแบตเตอรี่ 18650 จากการทดสอบพบว่าระบบสามารถทำงานได้อย่างต่อเนื่องเป็นเวลา 16 ชั่วโมง 28 นาที นอกจากนี้ยังได้พัฒนาระบบค้นหาเส้นทางอัจฉริยะโดยประสบความสำเร็จในการใช้เทคนิคการจัดกลุ่ม K-means Clustering เพื่อแบ่งกลุ่มของรายการสินค้า และประยุกต์ใช้เทคโนโลยีโอเพนซอร์ส OSRM (Open Source Routing Machine) ในการสร้างตารางระยะทาง (Distance Matrix) จากพิกัดที่ใช้ในการคำนวณลำดับเส้นทางโดยใช้วิธี Dynamic Programming จากการทดสอบประสิทธิภาพของระบบเทียบกับบริการนำทางอื่น พบว่าผลการคำนวณเส้นทางมีความใกล้เคียงกับผลที่ได้จากบริการเชิงพาณิชย์ และสามารถให้ระยะทางที่สั้นที่สุดได้จริง ซึ่งเป็นข้อบ่งชี้ถึงความแม่นยำและประสิทธิภาพของระบบในการประยุกต์ใช้ในบริบทจริง

5.2 ข้อเสนอแนะ

เพื่อเพิ่มประสิทธิภาพระบบติดตามการขนส่ง ควรพิจารณาเพิ่มอายุการใช้งานแบตเตอรี่ เช่น ใช้เทคโนโลยีจัดการพลังงานที่ดีขึ้นหรือแผงโซล่าเซลล์ พร้อมทั้งเพิ่มความยืดหยุ่นในการเชื่อมต่อโดยรองรับ NB-IoT สำหรับพื้นที่ที่สัญญาณ LTE ไม่เสถียร ในส่วนของระบบค้นหาเส้นทางอัจฉริยะ ควรปรับแต่งคอนฟิก Car.lua ให้เหมาะกับรถจักรยานยนต์เพื่อเพิ่มความคล่องตัว และเพิ่มฟังก์ชันสร้างเส้นทางระหว่างพิกัดเพื่อให้แสดงเส้นทางที่ชัดเจนขึ้น

บรรณานุกรม

[1] Lilygo. “*LILYGO® T-SIM7670E Documentation*.” https://github.com/Xinyuan-LilyGO, n.d.

[2] Xiaomi. “*Mi Temperature and Humidity Monitor 2 Specifications*.” https://www.mi.com/global/, n.d.

[3] U-Blox. NEO-6M GPS Modules Data Sheet. https://www.u-blox.com/, 2011.

[4] Blum, Jeremy. Exploring Arduino: Tools and Techniques for Engineering Wizardry. Wiley, 2013.

[5] Margolis, Michael. Arduino Cookbook. 3rd ed., O'Reilly Media, 2020.

[6] Tektronix. “*I2C Protocol Overview*.” https://www.tek.com/, n.d.

[7] Texas Instruments. “*Understanding the I2C Bus*.” https://www.ti.com/, n.d.

[8] Hastie, Trevor, Tibshirani, Robert, and Friedman, Jerome. The Elements of Statistical Learning: “*Data Mining, Inference, and Prediction.*” 2nd ed., Springer, 2009.

[9] Lawler, Eugene L., Lenstra, Jan K., Rinnooy Kan, A. H., and Shmoys, David B., editors. “*The Traveling Salesman Problem*” A Guided Tour of Combinatorial Optimization. Wiley-Interscience, 2010.

[10] Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., and Stein, Clifford. “*Introduction to Algorithms*”. 3rd ed., MIT Press, 2009.

[11] Bellman, Richard. “*Dynamic Programming*” Princeton University Press, 2017.

[12] Arduino.cc. “*Arduino IDE Documentation*.” https://docs.arduino.cc/software/ide-v1, n.d.

บรรณานุกรม (ต่อ)

[13] OASIS Open. MQTT Version 5.0 (OASIS Standard). https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html, 2020.

[14] Python Software Foundation. "*Python Documentation*." https://docs.python.org/3/, n.d.

[15] Postman. "*Postman Learning Center*." https://learning.postman.com/, n.d.

[16] Meta Platforms, Inc. "*React Documentation*." https://reactjs.org/docs/getting-started.html, n.d.

[17] Django Software Foundation. "*Django Documentation*." https://docs.djangoproject.com/en/stable/, n.d.

[18] OpenStreetMap Foundation. "*Welcome to OpenStreetMap*." https://www.openstreetmap.org/, n.d.

[19] Project-OSRM. "*OSRM Documentation*." http://project-osrm.org/docs/, n.d.

[20] Docker, Inc. "*Docker Documentation*." https://docs.docker.com/, n.d.

[21] Oracle Corporation. "*MySQL Documentation*." https://dev.mysql.com/doc/, n.d.

[22] Google Developers. "*Google Maps Platform Documentation.*" https://developers.google.com/maps/documentation, n.d.

ภาคผนวก ก
โค้ดสำหรับไมโครคอนโทลเลอร์

```
#define TINY_GSM_MODEM_SIM7600

#include <Arduino.h>
#include <HardwareSerial.h>
#include <TinyGsmClient.h>
#include <PubSubClient.h>
#include <BLEDevice.h>
#include <BLEClient.h>
#include <BLEUtils.h>
#include <BLEScan.h>
#include <TinyGPS++.h>
#include <esp_sleep.h>

// ---------------------- SIM7600 and MQTT Configuration ----------------------

// Define pins for TTGO T-SIM A7670E with SIM7600
#define PIN_DTR 25
#define PIN_RI 33
#define PIN_TX 26
#define PIN_RX 27
#define PWR_PIN 4

// APN for your mobile network
const char apn[]  = "internet";  // Replace with your actual APN

// Baud Rate for communication with SIM7600
#define GSM_BAUD 115200

// MQTT Broker settings
const char* mqtt_server = "34.44.210.178";
const int mqtt_port = 1883;
const char* mqtt_user = "pokpong";
const char* mqtt_pass = "pokpong";

// MQTT Topic
const char* mqtt_topic = "device/data";

// Initialize HardwareSerial for SIM7600 (Serial1)
HardwareSerial SerialAT(1); // Use UART1 (Serial1) on ESP32

// Initialize TinyGSM modem
TinyGsm modem(SerialAT);

// Initialize the MQTT client
TinyGsmClient gsmClient(modem);
PubSubClient mqttClient(gsmClient);

// ---------------------- GPS Configuration ----------------------

// Define GPS pins
```

```
#define RXPin_GPS 22
#define TXPin_GPS 21

static const uint32_t GPSBaud = 9600;

// The TinyGPS++ object
TinyGPSPlus gps;

// The serial connection to the GPS device
HardwareSerial ssGPS(2);

// ---------------------- BLE Configuration ----------------------

// Define BLE Devices
BLEAddress deviceA_addr("A4:C1:38:E1:5E:96");
BLEAddress deviceB_addr("A4:C1:38:24:27:29");

// Define Service and Characteristic UUIDs
BLEUUID serviceUUID("ebe0ccb0-7a0a-4b0c-8a1a-6ff2997da3a6");
BLEUUID charUUID("ebe0ccc1-7a0a-4b0c-8a1a-6ff2997da3a6");
BLEUUID batteryServiceUUID("0000180F-0000-1000-8000-00805F9B34FB");
BLEUUID batteryCharUUID("00002A19-0000-1000-8000-00805F9B34FB");

// BLE Clients
BLEClient* pClientA;
BLEClient* pClientB;

// Sensor Data Variables
float tempA = 0, humiA = 0, tempB = 0, humiB = 0;
int batteryA = 0, batteryB = 0;

float currentLatitude = 0.0;
float currentLongitude = 0.0;

// ---------------------- Data Point Structure ----------------------

// Structure to hold data points
struct DataPoint {
    String device;
    float temperature;
    float humidity;
    float latitude;
    float longitude;
};

// ---------------------- Unsent Data Buffer Configuration ----------------------

// Buffer to store unsent data points
#define UNSENT_DATA_POINTS 20 // Increased to store more data points
DataPoint unsentDataBuffer[UNSENT_DATA_POINTS];
```

```
int unsentDataIndex = 0;

// ---------------------- Timing Configuration ----------------------

// For Deep Sleep
#define uS_TO_S_FACTOR 1000000ULL  /* Conversion factor for micro seconds to seconds */
#define TIME_TO_SLEEP  180         /* Time ESP32 will go to sleep (in seconds) */

// ---------------------- Function Prototypes ----------------------

String sendAT(String command, int timeout, bool debug);
void printNetworkInfo();
void reconnectMQTT();
bool publishWithRetry(const char* topic, const char* payload);
bool publishDataPoint(const DataPoint& dp);
bool addToUnsentBuffer(const DataPoint& dp);
void getSIMLocation(float& latitude, float& longitude);
void getGPSLocation(float& latitude, float& longitude);
void connectAndReadBLE(BLEClient* pClient, BLEAddress address, const char* deviceID, float& temp, float& humi, int& battery);
void modemPowerOn();
void modemPowerOff();
void readGPS();
DataPoint collectData(String deviceID, float temp, float humi);
void enterDeepSleep();

// ---------------------- Function Implementations ----------------------

// Function to send AT commands and get response
String sendAT(String command, int timeout, bool debug) {
    String response = "";
    SerialAT.println(command);
    long int startTime = millis();
    while ((millis() - startTime) < timeout) {
        while (SerialAT.available() > 0) {
            char c = SerialAT.read();
            response += c;
        }
    }
    SerialAT.flush();
    if (debug) {
        Serial.print("Command: ");
        Serial.println(command);
        Serial.print("Response: ");
        Serial.println(response);
    }
    return response;
}

// Function to print network information
void printNetworkInfo() {
```

```
    Serial.println("\n------- Network Information --------");

    Serial.print("Signal Quality: ");
    Serial.println(modem.getSignalQuality());

    Serial.print("Operator: ");
    Serial.println(modem.getOperator());

    Serial.print("IMEI: ");
    Serial.println(modem.getIMEI());

    Serial.print("IMSI: ");
    Serial.println(modem.getIMSI());

    Serial.print("CCID: ");
    Serial.println(modem.getSimCCID());

    Serial.println("-----------------------------------\n");
}

// Function to reconnect to MQTT broker
void reconnectMQTT() {
    // Loop until reconnected
    while (!mqttClient.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (mqttClient.connect("ESP32Client", mqtt_user, mqtt_pass)) {
            Serial.println("connected");
            // Subscribe to topics if needed
            // mqttClient.subscribe("your/subscription/topic");
        } else {
            Serial.print("failed, rc=");
            Serial.print(mqttClient.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

// Function to publish with retries
bool publishWithRetry(const char* topic, const char* payload) {
    // Ensure MQTT is connected
    if (!mqttClient.connected()) {
        reconnectMQTT();
    }

    Serial.print("Payload: ");
    Serial.println(payload);
```

```
    // Attempt to publish
    if (mqttClient.publish(topic, payload)) {
        //Serial.println("Data published to MQTT: " + String(payload));
        Serial.println("Published to MQTT");
        return true;
    } else {
        Serial.println("Failed to publish data to MQTT. Attempt 1 of 2");
        Serial.print("MQTT Client State: ");
        Serial.println(mqttClient.state());
        delay(1000); // Wait 1 second before retrying

        // Ensure MQTT is connected before retrying
        if (!mqttClient.connected()) {
            reconnectMQTT();
        }

        // Retry publishing
        if (mqttClient.publish(topic, payload)) {
            Serial.println("Data published to MQTT on retry: " + String(payload));
            return true;
        } else {
            Serial.println("Failed to publish data to MQTT. Attempt 2 of 2");
            Serial.print("MQTT Client State: ");
            Serial.println(mqttClient.state());
            return false;
        }
    }
}

// Function to publish a single DataPoint with retries
bool publishDataPoint(const DataPoint& dp) {
    // Create JSON payload for single DataPoint
    String jsonPayload = "{";
    jsonPayload += "\"device\": \"" + dp.device + "\", ";
    jsonPayload += "\"temperature\": " + String(dp.temperature, 2) + ", ";
    jsonPayload += "\"humidity\": " + String(dp.humidity, 2) + ", ";
    jsonPayload += "\"latitude\": " + String(dp.latitude, 6) + ", ";
    jsonPayload += "\"longitude\": " + String(dp.longitude, 6);
    jsonPayload += "}";

    // Attempt to publish with retries
    if (publishWithRetry(mqtt_topic, jsonPayload.c_str())) {
        return true;
    } else {
        return false;
    }
}

// Function to add a DataPoint to the unsent buffer
bool addToUnsentBuffer(const DataPoint& dp) {
```

```
    if (unsentDataIndex < UNSENT_DATA_POINTS) {
        unsentDataBuffer[unsentDataIndex++] = dp;
        Serial.println("Added DataPoint to unsent buffer.");
        return true;
    } else {
        Serial.println("Unsent buffer is full. Dropping DataPoint.");
        return false;
    }
}

// Function to get location from SIM
void getSIMLocation(float& latitude, float& longitude) {
    Serial.println("Obtaining location from SIM...");
    String clbsResponse = sendAT("AT+CLBS=4", 10000, true); // Use "AT+CLBS=4,1" for extended data
    int index = clbsResponse.indexOf("+CLBS:");

    if (index != -1) {
        String data = clbsResponse.substring(index + 6); // Adjusted to +CLBS:
        data.trim(); // Remove leading and trailing whitespace
        int firstComma = data.indexOf(',');
        int secondComma = data.indexOf(',', firstComma + 1);
        int thirdComma = data.indexOf(',', secondComma + 1);

        if (firstComma != -1 && secondComma != -1 && thirdComma != -1) {
            // String status = data.substring(0, firstComma); // You can use status if needed
            String latStr = data.substring(firstComma + 1, secondComma);
            String lonStr = data.substring(secondComma + 1, thirdComma);

            latitude = latStr.toFloat();
            longitude = lonStr.toFloat();

            Serial.println("SIM Latitude: " + String(latitude, 6));
            Serial.println("SIM Longitude: " + String(longitude, 6));
        } else {
            Serial.println("Failed to parse SIM location data.");
            latitude = 0.0;
            longitude = 0.0;
        }
    } else {
        Serial.println("Failed to get location from SIM.");
        latitude = 0.0;
        longitude = 0.0;
    }
}

// Function to get location from GPS
void getGPSLocation(float& latitude, float& longitude) {
    if (gps.location.isValid()) {
        latitude = gps.location.lat();
        longitude = gps.location.lng();
```

```
        Serial.println("GPS Latitude: " + String(latitude, 6));
        Serial.println("GPS Longitude: " + String(longitude, 6));
    } else {
        Serial.println("GPS location invalid.");
        latitude = 0.0;
        longitude = 0.0;
    }
}

// Function to connect to BLE device and read data
void connectAndReadBLE(BLEClient* pClient, BLEAddress address, const char* deviceID, float& temp, float& humi, int& battery) {
    const int maxRetries = 3;  // Number of retries
    bool connected = false;

    for (int attempt = 1; attempt <= maxRetries; attempt++) {
        Serial.printf("Attempt %d to connect to device %s\n", attempt, deviceID);
        if (pClient->connect(address)) {
            connected = true;
            Serial.printf("Successfully connected to device %s on attempt %d\n", deviceID, attempt);
            break;
        } else {
            Serial.printf("Failed to connect to device %s on attempt %d\n", deviceID, attempt);
            delay(1000);  // Wait before retrying
        }
    }

    if (connected) {
        // Read temperature and humidity
        BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
        BLERemoteCharacteristic* pChar = nullptr;
        if (pRemoteService != nullptr) {
            pChar = pRemoteService->getCharacteristic(charUUID);
            if (pChar != nullptr) {
                String value = pChar->readValue();
                Serial.print("Raw data: ");
                for (size_t i = 0; i < value.length(); i++) {
                    Serial.print((uint8_t)value[i], HEX);
                    Serial.print(" ");
                }
                Serial.println();

                if (value.length() >= 3) { // Check if enough data
                    // Convert to signed 16-bit integer
                    int16_t tempRaw = (int16_t)((uint8_t)value[1] << 8 | (uint8_t)value[0]);
                    temp = tempRaw * 0.01;
                    humi = (float)(uint8_t)value[2];
                } else {
                    Serial.println("Insufficient data length for temperature and humidity.");
                    temp = 0.0;
                    humi = 0.0;
```

```
            }
        } else {
            Serial.println("Failed to find characteristic for temperature and humidity.");
            temp = 0.0;
            humi = 0.0;
        }
    } else {
        Serial.println("Failed to find the remote service for temperature and humidity.");
        temp = 0.0;
        humi = 0.0;
    }

    // Read battery level
    BLERemoteService* pBatteryService = pClient->getService(batteryServiceUUID);
    BLERemoteCharacteristic* pBatteryCharacteristic = nullptr;
    if (pBatteryService != nullptr) {
        pBatteryCharacteristic = pBatteryService->getCharacteristic(batteryCharUUID);
        if (pBatteryCharacteristic != nullptr) {
            battery = pBatteryCharacteristic->readUint8();
        } else {
            Serial.println("Failed to find characteristic for battery.");
            battery = 0;
        }
    } else {
        Serial.println("Failed to find the remote service for battery.");
        battery = 0;
    }
} else {
    Serial.printf("Could not connect to device %s after %d attempts.\n", deviceID, maxRetries);
    // Handle the failure as needed (e.g., log the error, set default values)
    temp = 0.0;
    humi = 0.0;
    battery = 0;
}

if (pClient->isConnected()) {
    pClient->disconnect();
}
}

// Function to power on the modem
void modemPowerOn() {
    const int SUPPLY_PIN = 12;
    const int RESET_PIN = 5;
    const int POWER_PIN = PWR_PIN;

    pinMode(SUPPLY_PIN, OUTPUT);
    digitalWrite(SUPPLY_PIN, HIGH); // Turn on SUPPLY
    pinMode(RESET_PIN, OUTPUT);
    digitalWrite(RESET_PIN, LOW);
```

```
    delay(100);
    digitalWrite(RESET_PIN, HIGH);
    delay(3000);
    digitalWrite(RESET_PIN, LOW);
    pinMode(POWER_PIN, OUTPUT);
    digitalWrite(POWER_PIN, LOW);
    delay(100);
    digitalWrite(POWER_PIN, HIGH);
    delay(1000);
    digitalWrite(POWER_PIN, LOW);
}

// Function to power off the modem
void modemPowerOff() {
    Serial.println("Powering off modem...");
    modem.poweroff();
    // Turn off power to the modem if necessary
    digitalWrite(PWR_PIN, LOW);
}

// Function to read GPS data
void readGPS() {
    unsigned long startTime = millis();
    unsigned long timeout = 5000; // 5 seconds
    while ((millis() - startTime) < timeout) {
        while (ssGPS.available() > 0) {
            char c = ssGPS.read();
            gps.encode(c);
        }
    }
}

// Function to collect data for a device
DataPoint collectData(String deviceID, float temp, float humi) {
    Serial.println("Collecting data for " + deviceID + "...");

    DataPoint dp;
    dp.device = deviceID;
    dp.temperature = temp;
    dp.humidity = humi;
    dp.latitude = currentLatitude;
    dp.longitude = currentLongitude;

    Serial.println("Data collected for " + deviceID + ".");

    return dp;
}

void enterDeepSleep() {
    Serial.println("Preparing to enter Deep Sleep...");
```

```
    // Disconnect MQTT
    if (mqttClient.connected()) {
        mqttClient.disconnect();
    }

    // Power off the modem
    modemPowerOff();

    // Deinitialize BLE
    BLEDevice::deinit();

    // Stop GPS
    ssGPS.end();

    // Set Wake-up Timer
    esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);

    Serial.println("Deep Sleep...");
    Serial.flush();
    esp_deep_sleep_start();
}

// ---------------------- Setup Function ----------------------

void setup() {
    // Initialize Serial Monitor
    Serial.begin(115200);
    delay(1000);
    Serial.println("Starting...");

    // Initialize SIM7600 Communication
    SerialAT.begin(GSM_BAUD, SERIAL_8N1, PIN_RX, PIN_TX);
    delay(3000);

    // Set GPIO pins
    pinMode(PWR_PIN, OUTPUT);
    pinMode(PIN_DTR, OUTPUT);
    pinMode(PIN_RI, INPUT);

    // Power on the modem
    modemPowerOn();
    delay(500);

    // Set DTR
    digitalWrite(PIN_DTR, HIGH);
    delay(1000);

    Serial.println("Initializing modem...");
    if (!modem.init()) {
```

```
    Serial.println("Failed to initialize modem");
    enterDeepSleep();
}
delay(1000);

// Check SIM status
if (modem.getSimStatus() != SIM_READY) {
    Serial.println("SIM not ready. Check SIM card.");
    enterDeepSleep();
}

Serial.println("Waiting for network...");
if (!modem.waitForNetwork()) {
    Serial.println("Network registration failed");
    enterDeepSleep();
}
Serial.println("Network registered");

Serial.print("Setting APN: ");
Serial.println(apn);
modem.sendAT("+CGDCONT=1,\"IP\",\"" + String(apn) + "\"");
if (modem.waitResponse(10000L, "OK") != 1) {
    Serial.println("Failed to set APN");
}

Serial.print("Connecting to LTE using APN: ");
Serial.println(apn);
if (!modem.gprsConnect(apn, "", "")) {
    Serial.println("LTE connection failed");
    enterDeepSleep();
}
Serial.println("LTE connected successfully");

IPAddress ip = modem.localIP();
Serial.print("Modem IP Address: ");
Serial.println(ip.toString());

// Print network information
printNetworkInfo();

// Set up MQTT
mqttClient.setServer(mqtt_server, mqtt_port);

// Initialize BLE
BLEDevice::init("");
pClientA = BLEDevice::createClient();
pClientB = BLEDevice::createClient();

// Attempt MQTT connection
reconnectMQTT();
```

```
// Initialize GPS
ssGPS.begin(GPSBaud, SERIAL_8N1, RXPin_GPS, TXPin_GPS, false);
Serial.println(TinyGPSPlus::libraryVersion());

// Read GPS data once
Serial.println("Reading GPS data...");
readGPS();

// Obtain location
if (gps.location.isValid()) {
    getGPSLocation(currentLatitude, currentLongitude);
} else {
    getSIMLocation(currentLatitude, currentLongitude);
}

// Attempt to send any unsent data immediately
if (unsentDataIndex > 0) {
    Serial.println("Attempting to send unsent data...");
    for (int i = 0; i < unsentDataIndex; ) {
        if (publishDataPoint(unsentDataBuffer[i])) {
            // Successfully sent, remove from unsentDataBuffer
            for (int j = i; j < unsentDataIndex - 1; j++) {
                unsentDataBuffer[j] = unsentDataBuffer[j + 1];
            }
            unsentDataIndex--;
            Serial.println("Unsent DataPoint sent successfully and removed from unsent buffer.");
        } else {
            Serial.println("Failed to resend unsent DataPoint.");
            i++; // Move to next DataPoint
        }
    }
}

// Read BLE sensors
connectAndReadBLE(pClientA, deviceA_addr, "A", tempA, humiA, batteryA);
connectAndReadBLE(pClientB, deviceB_addr, "B", tempB, humiB, batteryB);

// Collect data for Device A
DataPoint dpA = collectData("A", tempA, humiA);

// Publish data for Device A
Serial.println("Publishing data for Device A to MQTT...");
if (publishDataPoint(dpA)) {
    Serial.println("Data for Device A published successfully.");
} else {
    Serial.println("Failed to publish data for Device A. Adding to unsent buffer.");
    if (!addToUnsentBuffer(dpA)) {
        Serial.println("Unsent buffer is full. DataPoint cannot be added.");
    }
```

```
    }

    // Collect data for Device B
    DataPoint dpB = collectData("B", tempB, humiB);

    // Publish data for Device B
    Serial.println("Publishing data for Device B to MQTT...");
    if (publishDataPoint(dpB)) {
        Serial.println("Data for Device B published successfully.");
    } else {
        Serial.println("Failed to publish data for Device B. Adding to unsent buffer.");
        if (!addToUnsentBuffer(dpB)) {
            Serial.println("Unsent buffer is full. DataPoint cannot be added.");
        }
    }

    // Enter Deep Sleep
    enterDeepSleep();
}

// ---------------------- Loop Function ----------------------

void loop() {
    // No code needed here since all operations are done in setup()
}
```

ภาคผนวก ข

โค้ดส่วนการทำ K-means Clustering

```
from sklearn.cluster import KMeans

def cluster(str_latlng,str_origin):
    origin = [float(str_origin[0]),float(str_origin[1])]
    customer_latlng_results = [[float(lat), float(lng)] for lat, lng in str_latlng]

    n_clusters = 2
    kmeans = KMeans(n_clusters=n_clusters, init='k-means++', n_init=100, max_iter=1000)

    kmeans.fit(customer_latlng_results)

    # ผลลัพธ์ของการแบ่งกลุ่ม
    clusters = kmeans.labels_
    routes = []
    routes_json = {}
    for i in range(n_clusters):
        cluster_points = [customer_latlng_results[j] for j in range(len(customer_latlng_results)) if clusters[j] == i]
        routes.append(create_route(cluster_points, origin))
        routes_json[f"Rider {i+1}"] = create_route(cluster_points, origin)
    return routes[0],routes[1]

def create_route(cluster, origin):
    waypoints = [origin] + [list(latlng) for latlng in cluster] + [origin]  # เพิ่ม origin เป็นจุดเริ่มต้นและจุดสิ้นสุด
    return waypoints
```

ภาคผนวก ค

โค้ดส่วนการทำ Dynamic programming

```
import requests
import folium
from folium.plugins import AntPath

def process_tsp(str_origin,route):
    origin = [float(str_origin[0]),float(str_origin[1])]
    waypoints = route[1:-1]
    locations = [origin] + waypoints

    url = "http://127.0.0.1:5001/table/v1/cycling"
    coordinates = ";".join([f"{lon},{lat}" for lat, lon in locations])

    # URL ที่ใช้ในการขอ Matrix
    full_url = f"{url}/{coordinates}?annotations=distance"

    # ส่งคำขอไปยัง OSRM server
    response = requests.get(full_url)

    # ตรวจสอบสถานะคำขอ
    if response.status_code == 200:
        # แสดงผลลัพธ์ในรูปแบบ JSON
        data = response.json()
        print("connected ")
        # print(json.dumps(data, indent=2))
    else:
        print(f"Error: {response.status_code}")
    # print(response.json()["distances"])

    distance_matrix = response.json()["distances"]
    # คำนวณระยะทางที่สั้นที่สุดและลำดับ
    shortest_distance, route = tsp_dp(distance_matrix=distance_matrix)
    route_locations = [locations[i] for i in route]

    return route_locations+[origin],shortest_distance

# # ฟังก์ชัน dynamic programming สำหรับ TSP
def tsp_dp(distance_matrix):
    n = len(distance_matrix)
    dp = [[float('inf')] * n for _ in range(1 << n)]
    dp[1][0] = 0  # เริ่มต้นจาก origin
    path = [[-1] * n for _ in range(1 << n)]  # เก็บเส้นทาง

    for mask in range(1 << n):
        for u in range(n):
            if mask & (1 << u):  # ถ้าเมือง u ถูกเยี่ยมชมใน mask
                for v in range(n):
                    if not (mask & (1 << v)):  # ถ้าเมือง v ยังไม่ถูกเยี่ยมชม
                        new_mask = mask | (1 << v)
                        new_cost = dp[mask][u] + distance_matrix[u][v]  # รวมระยะทางจาก u ไป v
                        if dp[new_mask][v] > new_cost:
```

```
                    dp[new_mask][v] = new_cost
                    path[new_mask][v] = u  # บันทึกเมืองก่อนหน้า

    # คำนวณระยะทางรวมสั้นที่สุดเมื่อกลับไปที่ origin (เมือง 0)
    min_cost = float('inf')
    last_city = -1
    final_mask = (1 << n) - 1  # ทุกเมืองถูกเยี่ยมชมแล้ว

    for u in range(1, n):  # ตรวจสอบทุกเมืองยกเว้น origin
        total_cost = dp[final_mask][u] + distance_matrix[u][0]  # รวมระยะทางกลับไป origin
        if min_cost > total_cost:
            min_cost = total_cost
            last_city = u

    # สร้างลำดับเส้นทาง
    sequence = []
    mask = final_mask
    while last_city != -1:
        sequence.append(last_city)
        next_city = path[mask][last_city]
        mask ^= (1 << last_city)  # ลบเมืองจาก mask
        last_city = next_city
    sequence.reverse()  # กลับลำดับ
    return min_cost, sequence
def map(origin,route_locations):
    # สร้างแผนที่ที่จุดเริ่มต้น
    m = folium.Map(location=origin, zoom_start=12)
    # เพิ่ม Google Maps Tiles (แบบไม่ใช้ API Key)
    google_maps_tile = folium.TileLayer(
        tiles='http://{s}.google.com/vt?lyrs=m&x={x}&y={y}&z={z}',
        attr='Google',
        name='Google Maps',
        subdomains=['mt0', 'mt1', 'mt2', 'mt3'],  # subdomains ของ Google Maps
        overlay=True,
        control=True  )
    google_maps_tile.add_to(m)

    # วาดเส้นโค้งแบบเส้นประระหว่างจุดบนแผนที่
    AntPath(route_locations, color="black", weight=5, dash_array=[10, 20], delay=500).add_to(m)

    # เพิ่ม Marker สำหรับ origin และ waypoints พร้อมกับหมุดและตัวเลข
    folium.Marker(origin, tooltip="Origin", icon=folium.Icon(color='red', icon='info-sign')).add_to(m)

    for idx, point in enumerate(route_locations[1:], start=1):
        # Custom Icon with number inside
        icon_html = f"""
        <div style="background-color: #007bff; color: white; border-radius: 50%; text-align: center; width: 32px; height: 32px; line-height:
32px; font-weight: bold;">
            {idx}
        </div>
```

```
"""
icon = folium.DivIcon(html=icon_html)
folium.Marker(point, icon=icon).add_to(m)
```