บทที่ 3
วิธีดำเนินการศึกษาทดลอง

3.1 วัสดุอุปกรณ์ที่ใช้

3.1.1 วัสดุอุปกรณ์

1. ESP32 คือ ไมโครคอนโทรลเลอร์ที่ใช้ในการรับข้อมูลจากเซ็นเซอร์และส่งข้อมูลไปยัง Flask Web Server ผ่าน Wi-Fi

2. เซ็นเซอร์ DHT22 คือ เซ็นเซอร์สำหรับตรวจวัดอุณหภูมิและความชื้นในอากาศ

3. เซ็นเซอร์ BH1750 คือ เซ็นเซอร์สำหรับตรวจวัดความเข้มแสงในหน่วยลักซ์ (lux)

4. สายไฟและตัวต้านทาน (Jumper Wires & Resistors) คือ ใช้สำหรับเชื่อมต่ออุปกรณ์ต่าง ๆ ภายในระบบ

5. แหล่งจ่ายไฟ (Power Supply / Battery Pack) คือ ใช้สำหรับจ่ายพลังงานให้กับ ESP32 และเซ็นเซอร์

6. บอร์ดทดลอง (Breadboard) หรือ PCB คือ ใช้เป็นฐานสำหรับเชื่อมต่อวงจรและอุปกรณ์ต่าง ๆ

7. คอมพิวเตอร์ (Laptop/PC) คือ ใช้สำหรับพัฒนาและอัปโหลดโค้ดลงใน ESP32 รวมถึงสร้าง Flask Web Server

8. สาย USB Type-C / Micro-USB (ขึ้นอยู่กับบอร์ด ESP32) – ใช้สำหรับเชื่อมต่อ ESP32 กับคอมพิวเตอร์เพื่ออัปโหลดโค้ด

9. ซอฟต์แวร์ที่ใช้พัฒนา

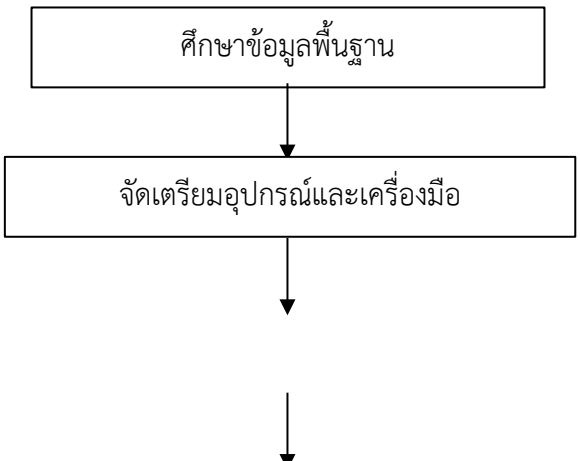
   9.1 Arduino IDE หรือ PlatformIO – สำหรับเขียนและอัปโหลดโค้ดลงใน ESP32

   9.2 Python และ Flask – ใช้สำหรับพัฒนา Web Server และประมวลผลข้อมูล

   9.3 HTML, CSS, JavaScript, Chart.js, Plotly – ใช้สำหรับพัฒนาเว็บไซต์และแสดงข้อมูลในรูปแบบกราฟ

3.2 วิธีการศึกษาทดลอง

3.2.1 ขั้นเตรียม



เตรียมระบบพัฒนาเว็บแอปพลิเคชัน

ทดสอบการเชื่อมต่อเบื้องต้น

3.2.2 ขั้นทดลอง

1. ก่อนเริ่มต้นการพัฒนาระบบตรวจวัดและวิเคราะห์สภาพแวดล้อมของพืช จำเป็นต้องศึกษาเกี่ยวกับเทคโนโลยีที่เกี่ยวข้อง เช่น หลักการทำงานของ ESP32 การอ่านค่าจากเซ็นเซอร์ DHT22 และ BH1750 การเชื่อมต่อ Wi-Fi รวมถึงการพัฒนาเว็บแอปพลิเคชันด้วย Flask และการแสดงผลข้อมูลด้วย JavaScript และไลบรารีสำหรับสร้างกราฟ หลังจากศึกษาข้อมูลที่เกี่ยวข้องแล้ว ต้องทำการดาวน์โหลดและติดตั้งซอฟต์แวร์ที่จำเป็นสำหรับการพัฒนา ได้แก่ Arduino IDE หรือ PlatformIO สำหรับเขียนและอัปโหลดโค้ดไปยัง ESP32, Python เวอร์ชันล่าสุดสำหรับการรัน Flask Web Server, และ ไลบรารีที่จำเป็น เช่น flask, flask-cors, requests และ matplotlib สำหรับจัดการข้อมูลและแสดงผลกราฟ

สำหรับการติดตั้ง Arduino IDE ต้องดาวน์โหลดตัวติดตั้งจากเว็บไซต์อย่างเป็นทางการของ Arduino และทำการติดตั้งลงบนคอมพิวเตอร์ โดยในขั้นตอนการติดตั้งต้องเลือกติดตั้งไดรเวอร์สำหรับบอร์ด ESP32 เพื่อให้สามารถอัปโหลดโค้ดและตรวจสอบการทำงานผ่าน Serial Monitor ได้ ในขณะเดียวกันต้องดาวน์โหลดและติดตั้ง Python จากเว็บไซต์ Python.org และทำการติดตั้ง pip ซึ่งเป็นตัวจัดการแพ็กเกจของ Python เพื่อใช้ติดตั้งไลบรารีเพิ่มเติม หลังจากติดตั้ง Python เรียบร้อยแล้ว ต้องใช้คำสั่ง pip install flask flask-cors requests matplotlib ผ่าน Command Prompt หรือ Terminal เพื่อให้แน่ใจว่าทุกไลบรารีที่จำเป็นถูกติดตั้งครบถ้วน

2. ติดตั้งและตั้งค่าสภาพแวดล้อมการพัฒนาโค้ด

เมื่อซอฟต์แวร์พื้นฐานถูกติดตั้งเรียบร้อยแล้ว ขั้นตอนต่อไปคือการตั้งค่าสภาพแวดล้อมการพัฒนาโค้ด เพื่อให้สามารถพัฒนาและทดสอบระบบได้อย่างมีประสิทธิภาพ หากใช้ Arduino IDE ต้องติดตั้ง ESP32 Board Manager โดยไปที่เมนู Preferences และเพิ่มลิงก์ https://dl.espressif.com/dl/package_esp32_index.json ลงในช่อง Additional Board Manager URLs จากนั้นไปที่ Board Manager และค้นหา "ESP32" เพื่อทำการติดตั้งแพ็กเกจที่จำเป็นสำหรับ ESP32 หากใช้ PlatformIO ต้องติดตั้งส่วนขยาย PlatformIO ใน Visual Studio Code และเพิ่มบอร์ด ESP32 เข้าไปในโปรเจกต์โดยใช้คำสั่ง pio init --board esp32dev เพื่อให้สามารถพัฒนาโค้ดบนบอร์ด ESP32 ได้อย่างสะดวก

สำหรับการตั้งค่า Flask Web Server ต้องสร้างโฟลเดอร์โปรเจกต์ใหม่และภายในโฟลเดอร์นี้ต้องสร้างไฟล์หลักที่ใช้รันเซิร์ฟเวอร์ เช่น app.py และโฟลเดอร์ย่อยสำหรับจัดเก็บไฟล์ที่เกี่ยวข้อง เช่น templates สำหรับไฟล์ HTML และ static สำหรับไฟล์ CSS และ JavaScript จากนั้นต้องเปิด

Terminal หรือ Command Prompt และรันคำสั่ง python app.py เพื่อตรวจสอบว่า Flask Web Server ทำงานได้อย่างถูกต้อง

3. เตรียมอุปกรณ์ฮาร์ดแวร์และประกอบวงจรสำหรับตรวจวัดสภาพแวดล้อมพืช หลังจากตั้งค่าซอฟต์แวร์เรียบร้อยแล้ว ต้องดำเนินการเตรียมอุปกรณ์ฮาร์ดแวร์ที่ใช้ในการตรวจวัดสภาพแวดล้อมของพืช ได้แก่ ESP32, เซ็นเซอร์ DHT22 สำหรับวัดอุณหภูมิและความชื้น, เซ็นเซอร์ BH1750 สำหรับวัดค่าความเข้มแสง, แหล่งจ่ายไฟ 3.3V หรือ 5V, และ สาย Jumper สำหรับเชื่อมต่อวงจร อุปกรณ์เหล่านี้ต้องถูกจัดวางและเชื่อมต่อให้เหมาะสมเพื่อให้การวัดค่ามีความแม่นยำและสัญญาณข้อมูลสามารถส่งผ่านไปยัง ESP32 ได้อย่างถูกต้อง การเชื่อมต่ออุปกรณ์เริ่มจากการต่อสาย **VCC** ของเซ็นเซอร์ทั้งหมดเข้ากับขา 3.3V ของ ESP32, ต่อขา GND ของเซ็นเซอร์ทั้งหมดเข้ากับขา GND ของ ESP32, และต่อขา Data ของเซ็นเซอร์ต่าง ๆ กับขาที่กำหนดไว้บน ESP32 โดย DHT22 ใช้ขา Data เชื่อมกับ GPIO ที่เลือก (เช่น GPIO4) และ BH1750 ใช้ขา SDA และ SCL เชื่อมต่อกับขา I2C ของ ESP32 ที่ GPIO21 และ GPIO22 (หรือ D21 และ D22 สำหรับ ESP32 DEVKITV1) ตามลำดับ จากนั้นต้องตรวจสอบว่าการเชื่อมต่อสายเป็นไปอย่างถูกต้องโดยไม่มีการลัดวงจรหรือขาเชื่อมต่อที่ผิดพลาด

4. ซึ่งเกี่ยวข้องกับการสร้างและปรับแต่งส่วน Frontend และ Backend ของระบบตรวจวัดพืชโดยใช้ Flask และ ESP32 จำเป็นต้องมีโครงสร้างของไฟล์ที่ชัดเจนเพื่อให้ระบบสามารถทำงานได้อย่างเป็นระเบียบและมีประสิทธิภาพ โดยโครงสร้างของไฟล์โครงงานที่จัดเก็บใน GitHub มีองค์ประกอบสำคัญที่ทำงานร่วมกัน ได้แก่ ไฟล์ HTML, CSS, JavaScript สำหรับ Frontend, ไฟล์ Python สำหรับ Backend (Flask Server), ไฟล์กำหนดค่าและการตั้งค่า Deploy เช่น Procfile และ render.yaml, รวมถึง ไฟล์ dependencies ใน requirements.txt รายละเอียดของโครงสร้างและการทำงานของแต่ละไฟล์มีดังนี้

4.1 โครงสร้างของโฟล์เดอร์แต่ละไฟล์ใน GitHub

```
plant-monitoring/
├── templates/            # โฟลเดอร์เก็บไฟล์ HTML
│   └── index.html        # ไฟล์หลักของ Frontend
├── static/               # โฟลเดอร์เก็บไฟล์ CSS และ JavaScript
│   ├── styles.css        # ไฟล์กำหนดรูปแบบ UI
│   └── script.js         # ไฟล์จัดการการทำงานของ Frontend
├── app.py                # ไฟล์หลักของ Flask Backend
├── requirements.txt      # รายการ dependencies ที่ต้องติดตั้ง
├── render.yaml           # ค่าติดตั้งสำหรับ Deploy บน Render
├── Procfile              # กำหนดค่าสั่งเริ่มต้นของเซิร์ฟเวอร์
└── README.md             # เอกสารอธิบายโครงงาน
```

ภาพที่ 3.1 โครงสร้างของโฟล์เดอร์แต่ละไฟล์ใน GitHub

4.2 การทำงานของ Frontend (HTML, CSS, JavaScript)

4.2.1 ไฟล์ index.html (Frontend UI) ไฟล์ index.html เป็นโครงสร้างหลักของ interface ที่ใช้ในการแสดงข้อมูลจากเซ็นเซอร์ และให้ผู้ใช้สามารถโต้ตอบกับระบบเองได้ ประกอบด้วยองค์ประกอบสำคัญดังนี้

4.2.1.1 ส่วนหัวของหน้า (<head>): เชื่อมโยงไฟล์ CSS (styles.css) และไลบรารีที่จำเป็น เช่น Chart.js และ Plotly.js เพื่อใช้สร้างกราฟ

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Plant Sensor Dashboard</title>
    <!-- Link CSS correctly -->
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
</head>
```

ภาพที่ 3.2 ส่วน <head> ของ index.html

4.2.1.2 ส่วนแสดงข้อมูลเซ็นเซอร์: มีช่องแสดงค่าของอุณหภูมิ ความชื้น และความเข้มแสงที่ได้รับจาก ESP32

```
<div class="container">
    <div id="sensor-data">
        <h2>Sensor Data</h2>
        <div class="data">
            <div>Temperature: <span id="temp">--</span> °C</div>
            <div>Humidity: <span id="humidity">--</span> %</div>
            <div>Light Intensity: <span id="light">--</span> lux</div>
        </div>
        <div id="health-status">--</div>
    </div>
```

ภาพที่ 3.3 ภาพของส่วนที่แสดงเซ็นเซอร์

4.2.1.3 ส่วนแสดงกราฟ: ใช้ Chart.js สำหรับแสดงข้อมูลย้อนหลังเป็นกราฟเส้น

```
<div class="chart-container">
    <canvas id="sensor-chart"></canvas>
</div>
```

ภาพที่ 3.4 การประกาศ class “chart-container”

4.2.1.4 ปุ่มโต้ตอบ (Button "Measure Data"): ใช้สำหรับส่งคำขอไปยัง Flask เพื่อเรียกข้อมูลล่าสุดจาก ESP32

```
<!-- Measure Data Button -->
<div class="interaction-container">
    <button id="measure-data-button" class="button">
        Measure Data
        <svg class="icon" viewBox="0 0 24 24" fill="currentColor">
            <path fill-rule="evenodd" d="M12 2.25c-5.385 0-9.75 4.365-9.75 9.75s4.365 9.75 9.75 9.75 9.7
        </svg>
    </button>
</div>
```

ภาพที่ 3.5 การสร้างปุ่มโต้ตอบ

4.2.2 ไฟล์ styles.css (ตกแต่ง UI) ไฟล์นี้ใช้กำหนดลักษณะของหน้าตา UI ให้มีความสวยงาม อ่านง่าย และใช้งานสะดวก โดยมีรายละเอียดที่สำคัญ ประกอบด้วย

4.2.3.1 การออกแบบพื้นหลัง (body): พื้นหลังที่ปรับขนาดอัตโนมัติขึ้นอยู่กับอุปกรณ์ที่ใช้และขนาดการขยายหน้าจอ

```css
/* Global styles */
body {
  font-family: 'Arial', sans-serif;
  margin: 0;
  padding: 0;
  background: url("C:/Users/admin/Pictures/images.jfif") no-repeat center center fixed;
  background-size: cover;
  color: #333;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  justify-content: flex-start;
  align-items: center;
}
```

ภาพที่ 3.6 ส่วน body ของเว็บไซต์

4.2.3.2 การจัดวางโครงสร้าง (container): ใช้ flexbox เพื่อจัดเรียงองค์ประกอบในหน้าเว็บ

```css
.container {
  width: 90%;
  max-width: 1200px;
  background: rgba(255, 255, 255, 0.85);
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);
  display: flex;
  flex-direction: column;
  gap: 20px;
}

/* Sensor data container */
#sensor-data {
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
  background: #ffffff;
  text-align: center;
}

#sensor-data h2 {
  font-size: 1.5rem;
}

#sensor-data .data {
  display: flex;
  justify-content: space-around;
  font-size: 1.2rem;
  gap: 20px;
}
```

ภาพที่ 3.7 ส่วนของ main container ของเว็บไซต์

4.2.3.3 การออกแบบปุ่ม (.button): ใช้สีแบบ gradient และเอฟเฟกต์ hover เพื่อให้มีความสวยงามของ UI

```css
.button {
  display: flex;
  align-items: center;
  justify-content: center;
  gap: 10px;
  padding: 0.75rem 1.5rem;
  background-color: rgb(0 107 179);
  border-radius: 9999px;
  color: #ffffff;
  font-weight: bold;
  font-size: 1rem;
  border: 3px solid #ffffff4d;
  cursor: pointer;
  box-shadow: 0px 10px 20px rgba(0, 0, 0, 0.2);
  transition: all 0.3s ease-in-out;
}

.icon {
  width: 24px;
  height: 24px;
}

.button:hover {
  transform: scale(1.05);
  border-color: #fff9;
}

.button:hover .icon {
  transform: translate(4px);
}
```

ภาพที่ 3.8 การออกแบบปุ่ม measure data

4.2.3.4 การออกแบบกล่องแสดงข้อมูล (#sensor-data, #3d-plot): กำหนดขอบมน (border-radius) และเงา (box-shadow) เพื่อให้ดูมีมิติ

```css
/* Chart container */
.chart-container {
  width: 100%;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
  background: #ffffff;
}

/* 3D plot container */
#3d-plot {
  width: 100%;
  height: 400px;
  border: 1px solid #ccc;
  border-radius: 10px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
}
```

ภาพที่ 3.9 ตกแต่งกราฟ ของ chart.js

4.2.4 ไฟล์ script.js (จัดการฟังก์ชันของหน้าเว็บ)

ไฟล์นี้ทำหน้าที่จัดการเหตุการณ์ที่เกิดขึ้นในหน้าเว็บ เช่น การกดปุ่ม การเรียกข้อมูลจาก Flask และการอัปเดตกราฟ มีฟังก์ชันหลักดังนี้

4.2.4.1 fetchSensorData() – ใช้ fetch() เพื่อดึงข้อมูลจาก Flask (/data)

```
// Fetch sensor data from Flask server
async function fetchSensorData() {
    try {
        const response = await fetch(flaskUrl);
        if (!response.ok) throw new Error(`HTTP error! Status: ${response.status}`);
        return await response.json();
    } catch (error) {
        console.error("❌ Fetch error:", error);
        throw error;
    }
}
```

ภาพที่ 3.10 ใช้ fetch() เพื่อดึงข้อมูล

4.2.4.2 displaySensorData(data) – อัปเดตค่าบน UI เช่น อุณหภูมิ ความชื้น และแสง

```
// Display sensor data on the dashboard
function displaySensorData(data) {
    document.getElementById("temp").innerText = data.temperature;
    document.getElementById("humidity").innerText = data.humidity;
    document.getElementById("light").innerText = data.light;
    updateChart(data);
}
```

ภาพที่ 3.11 อัปเดตค่าบน UI ผ่าน getElementById

4.2.4.3 analyzePlantHealth(data) – ประเมินสภาพแวดล้อมของพืชตามค่าที่ได้รับ โดยการใช้ค่า threshold โดยเฉลี่ยที่ใช้สำหรับพืชในแถบประเทศไทย

```
// Analyze plant health based on sensor data
function analyzePlantHealth(data) {
    if (data.temperature < 18 || data.temperature > 30) return "⚠️ Warning: Temperature out of range! It sh
    if (data.humidity < 50 || data.humidity > 80) return "⚠️ Warning: Humidity out of range! It should stay
    if (data.light < 950 || data.light > 7000) return "⚠️ Warning: Light out of range! It should stays in 9
    return "✅ The plant is healthy.";

}
```

ภาพที่ 3.12 ประเมินสภาพแวดล้อมของพืชผ่าน boolean

4.2.4.4 initChart() และ updateChart(data) – จัดการกราฟโดยใช้ Chart.js

```
// Initialize the chart
function initChart() {
    const ctx = document.getElementById("sensor-chart").getContext("2d");
    window.myChart = new Chart(ctx, {
        type: "line",
        data: {
            labels: [],
            datasets: [
                {
                    label: "Temperature (°C)",
                    data: [],
                    borderColor: "rgb(255, 99, 132)",
                    tension: 0.1,
                },
                {
                    label: "Humidity (%)",
                    data: [],
                    borderColor: "rgb(54, 162, 235)",
                    tension: 0.1,
                },
                {
                    label: "Light Intensity (lux)",
                    data: [],
                    borderColor: "rgb(75, 192, 192)",
                    tension: 0.1,
                },
            ],
        },
```

ภาพที่ 3.13 แสดงผลกราฟ chart.js และ color-code แต่ละข้อมูล

4.2.4.5 การจัดการปุ่ม Measure Data – เมื่อกดปุ่ม จะเรียก API และอัปเดตค่าบน UI

```
// Handle button click for measuring sensor data
measureDataButton.addEventListener("click", async () => {
    try {
        console.log("📡 Fetching sensor data...");
        const data = await fetchSensorData();
        console.log("✅ Data received:", data);

        displaySensorData(data);
        const healthStatus = analyzePlantHealth(data);
        document.getElementById("health-status").innerText = healthStatus;
    } catch (error) {
        console.error("❌ Error fetching data:", error);
        appendToChatbox("Error: Unable to fetch data. Please check the connection.");
    }
});
```

ภาพที่ 3.14 การจัดการปุ่ม measure data

4.3 การทำงานของ Backend (Flask)

4.3.1 ไฟล์ app.py (Flask Server) ไฟล์นี้เป็นแกนหลักของเซิร์ฟเวอร์ที่ทำหน้าที่รับข้อมูลจาก ESP32 และให้บริการ API สำหรับ Frontend การทำงานของเซิร์ฟเวอร์ Flask นี้ช่วยให้ Frontend สามารถดึงข้อมูลจาก ESP32 ได้ผ่าน API โดยที่ Flask ทำหน้าที่เป็นตัวกลางในการจัดการข้อมูล โดยมี endpoint สำคัญดังนี้

4.3.1.1 @app.route("/") – ให้บริการหน้า index.html

```python
@app.route("/")
def index():
    return render_template("index.html")
```

ภาพที่ 3.15 การเรียกใช้ไฟล์ index.html ใน template ของบริการ render

4.3.1.2 @app.route("/data", methods=["POST"]) – รับข้อมูลจาก ESP32 และเก็บไว้ในตัวแปร latest_data ผ่านอินเตอร์เน็ต PlantMonitor ที่สร้างมาเฉพาะกับการส่งข้อมูลข้าม server พร้อมการแจ้งเตือนเมื่อไม่สามารถรับข้อมูลได้

```python
@app.route("/data", methods=["POST"])
def receive_data():
    """Receives sensor data from ESP32 and forwards it if needed."""
    global latest_data
    try:
        sensor_data = request.get_json()
        if sensor_data:
            print(f"✅ Received data: {sensor_data}")
            latest_data = sensor_data  # Save the latest data

            # Forward data to ESP32 if hosted externally (on Render)
            if not request.remote_addr.startswith("192.168."):
                try:
                    response = requests.post(ESP32_LOCAL_URL, json=sensor_data, timeout=3)
                    print(f"📡 Forwarded to ESP32, Response: {response.status_code}")
                except requests.exceptions.RequestException:
                    print("⚠️ Failed to forward data to ESP32")

            return jsonify({"status": "success"}), 200
        else:
            return jsonify({"error": "No data received"}), 400
    except Exception as e:
        print(f"❌ Error processing data: {str(e)}")
        return jsonify({"error": "Failed to process data"}), 500
```

ภาพที่ 3.16 การรับข้อมูลผ่านโมดูล Flask

4.3.1.3 @app.route("/data", methods=["GET"]) – ให้ Frontend ดึงข้อมูลเซ็นเซอร์ล่าสุด

```python
@app.route("/data", methods=["GET"])
def send_data():
    """Sends the latest sensor data to the frontend."""
    if latest_data:
        return jsonify(latest_data)
    else:
        return jsonify({"error": "No sensor data available"}), 404
```

ภาพที่ 3.17 การส่งข้อมูลไปยัง Frontend

4.4 การ Deploy บน Render

4.4.1 ไฟล์ requirements.txt เป็นไฟล์ที่ใช้ระบุแพ็กเกจ Python ที่จำเป็นสำหรับแอปพลิเคชัน Flask เพื่อให้สามารถติดตั้งและทำงานได้อย่างถูกต้อง ในที่นี้ ไฟล์ requirements.txt

ประกอบด้วยแพ็กเกจดังต่อไปนี้ ซึ่งแต่ละตัวมีหน้าที่เฉพาะ

```
Flask==2.2.2
gunicorn==20.1.0
blinker==1.9.0
click==8.1.8
colorama==0.4.6
itsdangerous==2.2.0
Jinja2==3.1.5
MarkupSafe==3.0.2
Werkzeug==2.2.2
Flask-Cors==3.0.10
requests==2.31.0
```

ภาพที่ 3.18 ไฟล์ requirements.txt

4.4.2 ไฟล์ Procfile ใช้กำหนดคำสั่งเริ่มต้นของเซิร์ฟเวอร์ Flask โดยใช้ gunicorn เพื่อให้เซิร์ฟเวอร์รองรับการใช้งานจริง

```
web: gunicorn app:app --bind 0.0.0.0:$PORT
```

ภาพที่ 3.19 ไฟล์ Procfile กำหนดคำสั่ง Deploy เว็บไซต์

4.4.3 ไฟล์ render.yaml กำหนดค่าการตั้งค่า Deploy บน Render โดยใช้แพ็กเกจ gunicorn และกำหนดตัวแปร PORT

```
services:
  - type: web
    name: flask-app
    env: python
    plan: free
    buildCommand: "pip install -r requirements.txt"
    startCommand: "gunicorn app:app"
    envVars:
      - key: FLASK_ENV
        value: production
      - key: PORT
        value: 5000
```

ภาพที่ 3.20 ไฟล์ render.yaml เป็นไฟล์กำหนดและตั้งแค่เว็บไซต์ก่อน Deploy