วารสารบัณฑิตศึกษา มหาวิทยาลัยธุรกิจบัณฑิตย์
ปีที่ 1 ฉบับที่ 13 (สิงหาคม 2567-พฤศจิกายน 2567)

การสร้างส่วนต่อประสานโปรแกรมประยุกต์แบบ Minimal API ด้วยโครงสร้างสถาปัตยกรรม N-Tier

CREATING MINIMAL APIS WITH N-TIER ARCHITECTURE

ณัชพล เมฆลาย[1]
ผศ.ดร.วิลาวัลย์ อินทร์ชำนาญ[2]

บทคัดย่อ

การสร้างส่วนต่อโปรแกรมประยุกต์ (APIs) ในปัจจุบันนี้นอกจากจะต้องคำนึงถึงการใช้งานจากผู้ใช้งานให้มีความครบถ้วนสมบูรณ์แล้ว การทำให้ส่วนต่อประสานโปรแกรมประยุกต์ที่ ถูกสร้างขึ้นนั้นจะต้องสามารถทำความเข้าใจได้ง่าย มีความเป็นระเบียบ ลดความยุ่งยากในการปรับปรุงหรือเปลี่ยนแปลง รวมถึงความเรียบง่ายในการสร้างส่วนต่อประสานโปรแกรมประยุกต์ ขึ้นมาใหม่ ก็เป็นอีกส่วนหนึ่งที่จะต้องคำนึงถึงในเวลานี้เช่นกัน

ซึ่งความต้องการในส่วนต่าง ๆ ที่ได้กล่าวมาคือ ความสามารถของการสร้างส่วนต่อประสานโปรแกรมประยุกต์ แบบมินิมอล)Minimal API) โดยใช้ดอตเน็ตเฟรมเวิร์ค (.Net Framework) และภาษาซีชาร์ป (C#) ซึ่งจะเข้ามาตอบโจทย์สำหรับการสร้างส่วนต่อประสานโปรแกรมประยุกต์ที่มีความเรียบง่ายและสามารถปรับปรุงเปลี่ยนแปลงโครงสร้างข้อมูลต่าง ๆ ได้อย่างเป็นอิสระมากยิ่งขึ้น จึงทำให้การเลือกใช้สถาปัตยกรรมแบบลำดับชั้น) N-Tier Architecture) ที่เป็นสถาปัตยกรรมที่จะทำให้การสร้างส่วนต่อประสานโปรแกรมประยุกต์มีความยืดหยุ่น ความเป็นระเบียบ และสามารถทำได้อย่างสะดวก รวดเร็ว อีกทั้งผลลัพธ์ที่ออกมาคือโค้ดที่ทำความเข้าใจได้ง่ายมากขึ้นอีกด้วย

โดยมีผลลัพธ์เมื่อนำมาใช้สำหรับการพัฒนาส่วนต่อโปรแกรมประยุกต์)APIs) ภายในระบบเว็บแอปพลิเคชันจัดการเอกสารสำหรับกิจกรรมส่งเสริมการขายสินค้าขององค์กรส่วนงานสุรา (E-Activity Spirits) แล้วนั้นพบว่าผู้ใช้งานส่วนใหญ่มีความพึ่งพอใจในการใช้งานอยู่ในระดับมาก (คะแนนเฉลี่ยที่ 4.15 คะแนนจากคะแนนทั้งหมด 5 คะแนน) จากผู้ใช้งานที่ตอบแบบสอบถามความพึงพอใจในการใช้งานระบบจำนวนทั้งหมด 304 คน ทั้งนี้ระบบยังคงมีช่องทางที่จะสามารถขยับขยายไปใช้ในหลาย ๆ ภาคส่วนขององค์กร รวมถึงโอกาสในการพัฒนาความสามารถภายในระบบก็สามารถที่จะทำได้อยู่เช่นกันโดย

---

[1] นักศึกษาหลักสูตรวิทยาศาสตรมหาบัณฑิตสาขาวิชาวิสวกรรมเว็บและการพัฒนาแอปพลิเคชันบนอุปกรณ์พกพา

[2] อาจารย์ที่ปรึกษา วิทยาลัยครีเอทีฟดีไซน์ แอนด์ เอ็นเตอร์เทนเมนต์เทคโนโลยี

คำสำคัญ: ส่วนต่อประสานโปรแกรมประยุกต์แบบมินิมอล สถาปัตยกรรมแบบหลายชั้น ดอตเน็ตเฟรมเวิร์ค ซีชาร์ป

ABSTRACT

In creation of Application Programming Interfaces (APIs) extends beyond ensuring comprehensive user functionality. A critical consideration in modern API development encompasses the implementation of interfaces that are readily comprehensible, systematically organized, and amenable to modifications. Furthermore, the ease of reconstruction and maintenance of these application interfaces represents an equally significant aspect that demands careful consideration in the current development paradigm.

These afore-mentioned requirements are effectively addressed through the implementation of Minimal API utilizing by .Net Framework and C# language. This approach provides an optimal solution for developing streamlined application interfaces while facilitating enhanced autonomy in data structure modifications. Consequently, the adoption of N-Tier Architecture becomes particularly advantageous, as this architectural pattern promotes flexibility and systematic organization in API development. This methodology not only expedites the development process but also yields more comprehensible code structures, thereby enhancing overall maintainability and efficiency of the system.

In a practical application, this methodology was using in developing a web application API for managing documentation related to sales promotion activities within a spirits organization's electronic activity system (E-Activity Spirits). The implementation demonstrated significant success, with user satisfaction surveys revealing high levels (score 4.15 from 5) of acceptance among 304 respondents. Moreover, the system exhibits considerable potential for scalability across multiple organizational sectors and future capability enhancements and on strategic approach emphasizes not just technical implementation, but a holistic perspective on API design that balances functionality, user experience, and systemic adaptability.

Keywords: Minimal API, N-tier Architecture, .Net Framework, C#

1. บทนำ

ในปัจจุบัน สภาวะการแข่งขันทางธุรกิจที่ทวีความรุนแรงส่งผลให้องค์กรต้องปรับเปลี่ยนกลยุทธ์อย่างต่อเนื่อง เพื่อให้สอดรับกับบริบทและความท้าทายที่เกิดขึ้น ด้วยเหตุนี้การพัฒนาระบบงานต่าง ๆ ไม่ว่าจะ

เป็นซอฟต์แวร์หรือเว็บแอปพลิเคชันด้วยวิธีดั้งเดิมจึงประสบปัญหาในการตอบสนองต่อการเปลี่ยนแปลงอย่างทันท่วงที การปรับปรุงซอฟต์แวร์หรือเว็บแอปพลิเคชันภายใต้ข้อจำกัดของระบบเดิมต้องอาศัยความเข้าใจอย่างลึกซึ้งในโครงสร้างการทำงานที่ซับซ้อน โดยเฉพาะในองค์กรขนาดใหญ่ที่มีการแบ่งระบบงานออกเป็นหน่วยงานย่อยตามภาระหน้าที่ ความเชื่อมโยงระหว่างระบบงานที่แนบแน่นนี้ทำให้การเปลี่ยนแปลงเงื่อนไขในส่วนหนึ่งส่งผลกระทบต่อระบบอื่นๆ อย่างหลีกเลี่ยงมิได้ ผลที่ตามมาคือนักพัฒนาจำเป็นต้องทำการปรับแก้ระบบงานที่เกี่ยวข้องทั้งหมด เพื่อให้ตรงตามความต้องการทางธุรกิจอย่างต่อเนื่อง ซึ่งกระบวนการดังกล่าวต้องใช้ระยะเวลาและทรัพยากรอย่างมาก

ดังนั้นในบทความนี้จะกล่าวถึงการสร้างส่วนต่อโปรแกรมประยุกต์ (APIs) ซึ่งเป็นส่วนประกอบสำคัญสำหรับเว็บแอปพลิเคชันในการใช้รับส่งข้อมูลระหว่างส่วนหน้าของระบบ (Front-end) และส่วนของฐานข้อมูล โดยจะสร้างส่วนประสานโปรแกรมประยุกต์ แบบมินิมอล (Minimal API) โดยใช้ดอตเน็ตคอร์เฟรมเวิร์ค (.Net Core Framework) และภาษาซีชาร์ป (C#) รวมถึงการใช้สถาปัตยกรรมแบบลำดับชั้น) N-Tier Architecture) มาเป็นแม่แบบในการจัดวางโครงสร้างสำหรับการสร้างส่วนต่อโปรแกรมประยุกต์ภายใต้สารนิพนธ์หัวข้อการพัฒนาเว็บแอปพลิเคชันจัดการเอกสารสำหรับกิจกรรมส่งเสริมการขายสินค้าขององค์กรส่วนงานสุรา

2. ทฤษฎี และงานวิจัยที่เกี่ยวข้อง

2.1 ส่วนประสานโปรแกรมประยุกต์ แบบมินิมอล (Minimal API)

ส่วนประสานโปรแกรมประยุกต์แบบมินิมอล ต่อไปนี้จะขอเรียกว่า Minimal API เป็นนวัตกรรมที่ถูกนำมาใช้ใน .NET 6 ซึ่งเป็นแพลตฟอร์มโอเพ่นซอร์สข้ามแพลตฟอร์มสำหรับการพัฒนาซอฟต์แวร์ โดยมีวัตถุประสงค์เพื่อปรับปรุงประสิทธิภาพและลดความซับซ้อนในการสร้าง HTTP API

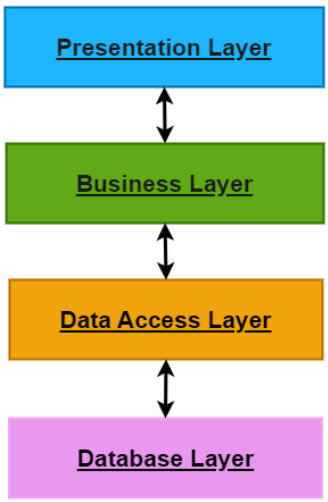
ซึ่งในส่วนของ .NET Framework นั้นมีพัฒนาการมาตั้งแต่ต้นทศวรรษ 2000 โดย Microsoft ซึ่งในปี 2016 ได้เริ่มพัฒนา .NET Core ซึ่งเป็นจุดเปลี่ยนสำคัญในระบบนิเวศของแพลตฟอร์ม ต่อมาในปี 2020 Microsoft ได้รวม .NET Framework และ .NET Core เป็น .NET 5 ซึ่งนำไปสู่การพัฒนาอย่างต่อเนื่องจนถึง .NET 8 และภายในตัวของ .NET 8 นั้นยังมีฟีเจอร์ที่เรียกว่า Minimal API ซึ่งถูกพัฒนามาตั้งแต่ .NET 6 โดย Minimal API นั้นจะใช้สำหรับการเขียน HTTP API เพื่อติดต่อหรือร้องขอข้อมูลจากฐานข้อมูลอย่างเรียบง่าย ซึ่งแนวคิดนี้ได้รับแรงบันดาลใจจากความเรียบง่ายและความยืดหยุ่นของเว็บเฟรมเวิร์กที่มีน้ำหนักเบาและสามารถอ่านทำความเข้าใจได้อย่างชัดเจนตัวอื่นๆ เช่น Express.js ใน Node.js และ Flask ใน Python โดย Minimal API มีจุดมุ่งหมายเพื่อนำความเรียบง่ายนี้มาสู่ระบบนิเวศของ .NET สำหรับการสร้าง HTTP API [1] โดยการทำ Minimal API ผ่าน .Net 8 นั้นจะได้รับข้อดีต่าง ๆ ไม่ว่าจะเป็นการกำหนดปลายทาง HTTP ได้ง่ายขึ้น ลดจำนวนโค้ดต้นแบบ (Boilerplate) ที่จำเป็นในการตั้งค่า API ทำให้ง่ายต่อการอ่านและทำความเข้าใจโค้ดเบส และยังปรับปรุงประสิทธิภาพของ API ทำให้สร้างโค้ดที่สามารถบำรุงรักษาได้ดีและเป็นรูปแบบยิ่งขึ้น โดยใช้ประโยชน์จากความสามารถของ .Net 8 ที่มีความซับซ้อนน้อยลง

โดยมีการเปรียบเทียบการทำงานระหว่างส่วนประสานโปรแกรมประยุกต์แบบมินิมอลกับส่วนประสานโปรแกรมประยุกต์แบบตัวควบคุม (Controller base) แล้วพบว่าส่วนประสานโปรแกรมประยุกต์แบบมินิมอลนั้นใช้งานหน่วยความจำน้อยกว่าแบบที่เป็นตัวควบคุมถึงประมาณ 50% รวมถึงยังมีใช้เวลาในการตอบสนองที่รวดเร็วกว่าอีกด้วย [2] รวมถึงเมื่อมีการนำไปเปรียบเทียบกับการสร้างส่วนประสานโปรแกรมประยุกต์โดยใช้ Express ที่เป็นเครื่องมือสำหรับการสร้างส่วนประสานโปรแกรมประยุกต์โดยใช้ภาษาจาวาสคริปต์ (JavaScript) ซึ่งผลลัพธ์คือส่วนประสานโปรแกรมประยุกต์แบบมินิมอลนั้นมีการใช้งานหน่วยความจำน้อยกว่าประมาณ 50% เช่นกันและมีความเร็วในการทำงานมากกว่าถึงประมาณ 85% [3]

2.2 สถาปัตยกรรมแบบลำดับชั้น) N-Tier Architecture)

สถาปัตยกรรมแบบลำดับชั้น (N-Tier Architecture) เป็นวิธีการออกแบบซอฟต์แวร์ที่แบ่งระบบออกเป็นหลายชั้น)Layer) โดยแต่ละชั้นมีหน้าที่และความรับผิดชอบที่แตกต่างกัน เพื่อให้ระบบมีความยืดหยุ่น ง่ายต่อการบำรุงรักษา และขยายระบบ

โดยในขั้นตอนสำหรับการสร้างส่วนประสานโปรแกรมประยุกต์ (APIs) นั้นจะมีลำดับชั้นที่สำคัญอยู่ด้วยกันทั้งหมด 4 ลำดับชั้น แสดงดังภาพที่ 2.1 โดยประกอบไปด้วย



ภาพที่ 2.1 ลำดับชั้นที่สำคัญในสถาปัตยกรรมแบบลำดับชั้น

2.2.1 พรีเซนท์เทชันเลเยอร์ (Presentation layer)

เป็นชั้นที่ใช้ติดต่อระหว่างผู้ใช้งาน โดยภายในการสร้างส่วนประสานโปรแกรมประยุกต์จะหมายถึงส่วนของตัวควบคุม (Controllers) หรือส่วนของจุดสิ้นสุด (Endpoints) ร่วมถึงยังมีตัวตรวจสอบความถูกต้อง (Validator) ของข้อมูลที่รับเข้ามาเพื่อตรวจสอบความถูกต้องและส่งข้อผิดพลาดกลับในทันทีกรณีที่ข้อมูลที่รับเข้ามานั้นไม่ถูกต้อง และในระดับชั้นนี้จะไม่มีโค้ดที่เกี่ยวข้องกับฐานข้อมูล มีเพียงแต่การเรียกใช้ออบเจ็กต์หรือคลาสจากบิสซิเนสเลเยอร์[4] เพื่อนำข้อมูลมาเรียบเรียงเข้าสู่ออบเจ็กต์สำหรับส่งข้อมูล (Data Tranasfer Objects (DTOs)) กลับไปยังผู้ใช้งานพร้อมกับระบุ HTTP Code ที่เหมาะสม

2.2.2 บิสซิเนสเลเยอร์ (Business layer)

เป็นชั้นที่ทำหน้าที่เกี่ยวกับความต้องการของระบบกฎเกณฑ์ในดำเนินแผนงานทางธุรกิจอีกทั้งยังเป็นตัวกำหนดแนวทางหรือทิศทางในการทำงานของแอปพลิเคชันโดยอ้างอิงจากความต้องการทางธุรกิจเป็นหลักและผลลัพธ์ที่ได้ต้องสอดคล้องถูกต้องตามแผนธุรกิจที่วางไว้[4][5] ลำดับชั้นนี้ผู้ออกแบบและพัฒนาระบบต้องมีความเข้าใจในระบบมากในการออกแบบและพัฒนาระบบให้เกิดความยืดหยุ่น เพื่อรองรับการเปลี่ยนแปลงความต้องการทางธุรกิจที่เกิดขึ้นได้ตลอดเวลา

2.2.3 ดาต้าแอคเซสเลเยอร์ (Data Access Layer)

เป็นชั้นที่ติดต่อ และการเข้าถึงในส่วนของข้อมูลจะประกอบไปด้วยคลาสที่ทำหน้าที่เป็นตัวแทนของ DBMS ที่ใช้จริง[4][6] เช่นชั้นบิสซิเนสเลเยอร์จะมาขอให้คลาสในชั้นดาต้าแอคเซสเลเยอร์ส่งข้อมูลออบเจ็กต์ของลูกค้าทั้งหมดมาให้หรือนำออบเจ็กต์ รายการสั่งซื้อไปบันทึกเก็บไว้ เป็นต้น โดยการเข้าถึงข้อมูลสามารถผ่านเข้าถึงข้อมูลได้โดยผ่านเซอร์วิสต่าง ๆ ในชั้นดาต้าแอคเซสเลเยอร์ และจะใช้อินเทอร์เฟสที่เป็นมาตรฐาน ซึ่งทำให้ชั้นบิสซิเนสเลเยอร์สามารถเรียกใช้งานได้ผ่านทางอินเทอร์เฟสดังกล่าว โดยในการสร้างส่วนประสานโปรแกรมประยุกต์นั้นจะมีการสร้างคลาสของออบเจ็กต์และเอนติตี้ในการติดต่อประสานงานไปยังดาต้าเบสเลเยอร์ซึ่งจะถูกเรียกว่า Object-Relational Mapping (ORM) ซึ่งจะทำให้ดาต้าแอคเซสเลเยอร์เข้าถึงข้อมูลภายใต้ดาต้าเบสเลเยอร์ที่กำหนดไว้ได้ง่ายและเป็นระเบียบมากยิ่งขึ้น

2.2.4 ดาต้าเบสเลเยอร์ (Database Layer)

เป็นชั้นที่ทำหน้าที่เกี่ยวกับการจัดเก็บข้อมูลจริง รวมถึงการดูแลรักษาข้อมูลอย่างเป็นระเบียบ หรือพูดอีกนัยหนึ่งในลำดับชั้นนี้คือตัวของฐานข้อมูล (Database) นั้นเอง โดยในบทความนี้จะเลือกใช้เป็นฐานข้อมูลที่มีประเภทของภาษาแบบมีความสัมพันธ์ของข้อมูล (Structured query language (SQL)) โดยจะต้องกำหนดค่าความสัมพันธ์ต่าง ๆ ให้ถูกต้อง อาทิเช่น การกำหนดไพรมาร์รี่คีย์ (Primary Key (PK)) การกำหนดฟอเรนคีย์ (Foreign Key (FK)) ให้กับฟิลด์ข้อมูลที่ถูกต้อง ในทุกๆตารางข้อมูลรวมถึงการกำหนดประเภทของค่าข้อมูล (Data type) ที่ใช้ในการเก็บข้อมูลในแต่ฟิลด์ข้อมูลให้ถูกต้อง ซึ่งนี้จะช่วยเพิ่มประสิทธิภาพในการค้นหาและการทำงานของฐานข้อมูลได้อย่างรวดเร็ว ถูกต้อง และแม่นยำมากยิ่งขึ้น

2.3 เอนติตี้ เฟรมเวิร์ค คอร์ (Entity Framework Core (EF Core))

เอนติตี้ เฟรมเวิร์ค คอร์ (Entity Framework Core (EF Core)) เป็นเครื่องมือสำหรับสร้าง Object-Relational Mapper (ORM) ที่พัฒนาโดยไมโครซอร์ฟ (Microsoft) ซึ่งเป็นโครงการโอเพ่นซอร์สที่สร้างขึ้นเพื่อเลียนแบบโครงการเอนติตี้ เฟรมเวิร์ค (Entity Framework) ซึ่งเป็นของไมโครซอร์ฟเองเช่นกัน เพียงแต่การทำงานเอนติตี้ เฟรมเวิร์ค นั้นจะถูกสร้างขึ้นสำหรับวินโดวส์ (Windows) และดอตเน็ตเฟรมเวิร์ค (.NET Framework) แต่สำหรับ เอนติตี้ เฟรมเวิร์ค คอร์ นั้นถูกสร้างขึ้นสำหรับการใช้งานแบบข้ามแพลตฟอร์ม (Cross-platform) สำหรับการทำงานกับดอตเน็ตคอร์เฟรมเวิร์ค (.Net Core Framework) [7]

โดยตัวของเอนติตี้ เฟรมเวิร์ค คอร์ นั้นจะช่วยให้นักพัฒนาสามารถทำงานกับฐานข้อมูลได้ง่ายขึ้นโดยการใช้ออบเจ็กต์ในภาษาซีชาร์ป (C#) แทนการเขียนคำสั่ง SQL (SQL Query) โดยตรง ซึ่ง เอนติตี้ เฟรม

เวิร์ค คอร์ ถูกออกแบบมาให้มีความยืดหยุ่นและรองรับการทำงานกับฐานข้อมูลหลายประเภท เช่น SQL Server, SQLite, PostgreSQL และ MySQL รวมถึงยังสามารถรองรับการทำไมเกรชั่นดาต้า (Migration Data) จากการเขียนแบบโค้ดไปยังฐานข้อมูล (Code-first migration) อีกด้วย

โดยหลักการทำงานของ เอนติตี้ เฟรมเวิร์ค คอร์ นั้นจะทำการสร้างคอนเท็กคลาส (Context Class) มาหนึ่งตัวเพื่อใช้สำหรับเรียกใช้งานและเข้าสู่ฐานข้อมูล รวมถึงมีการประกาศคลาสโมเดล (Model Class) ต่าง ๆ ตามตารางข้อมูลที่อยู่ในฐานข้อมูลที่คอนเท็กคลาสเชื่อมต่ออยู่เพื่อสามารถเรียกใช้งานได้ผ่านการเขียนคำสั่งภาษาคิวรี่แบบบูรณาการ (Language Integrated Query (LINQ)) [7]

3. วิธีการดำเนินงาน

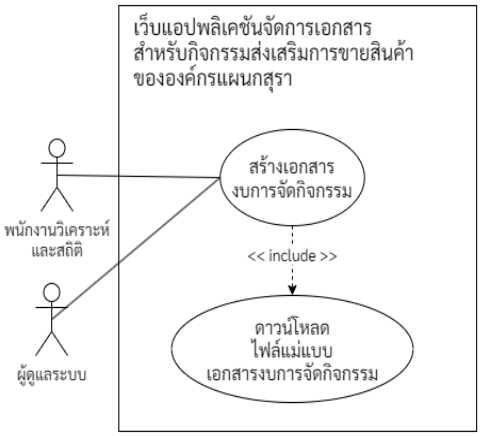
การดำเนินการสร้างส่วนต่อประสานโปรแกรมประยุกต์แบบ Minimal API ด้วยโครงสร้างสถาปัตยกรรม แบบลำดับชั้นภายใต้สารนิพนธ์หัวข้อการพัฒนาเว็บแอปพลิเคชันจัดการเอกสารสำหรับกิจกรรมส่งเสริมการขายสินค้าขององค์กรส่วนงานสุรา จะมีขั้นตอนวิธีการดำเนินงานและเครื่องมือที่ใช้ดังต่อไปนี้

3.1 การวิเคราะห์ปัญหาและศึกษาค้นคว้าข้อมูล

ปัญหาที่พบได้บ่อยในการทำงานขององค์กรในส่วนของการจัดทำเอกสารการจัดทำและจ่ายค่าใช้จ่ายของกิจกรรมส่งเสริมการขายคือการที่เอกสารนั้น มีการกำกับเลขที่กำกับเอกสารไม่ถูกต้อง หรือซ้ำกันกับตัวเลขเก่า จึงทำให้เกิดการเสียเวลาในการคำนวณหาเลขที่กำกับเอกสารที่ถูกต้องและตรงตามมาตรฐานที่องค์กรกำหนด อีกทั้งในช่วงของการอนุมัติเอกสาร ทั้งส่วนของการที่จำเป็นต้องพิมพ์เอกสารที่ต้องการขออนุมัติ และทำการตามหาผู้อนุมัติ ซึ่งบางครั้งไม่ได้เข้ามา ณ สถานที่ทำงาน หรือทั้งการที่เอกสารที่ขออนุมัตินั้นมีรายละเอียดที่ผิด หรือขาดตกบกพร่องไป ผู้อนุมัติจำเป็นที่จะต้องปฏิเสธการอนุมัติเอกสารใบนั้นไป และจำเป็นต้องทิ้งและทำลายเอกสารใบนั้น เพื่อไปทำเอกสารใหม่ให้มีข้อมูลที่ครบถ้วนสมบูรณ์ รวมถึงยังไม่สามารถติดตามเอกสารแต่ละใบว่าอยู่ในขั้นตอนใดแล้วบ้าง ทำให้เกิดปัญหาเอกสารตกหล่นและสูญหาย จนทำให้เกิดปัญหาในช่วงที่ทำการเบิกสินค้าไปจัดกิจกรรมส่งเสริมการขาย

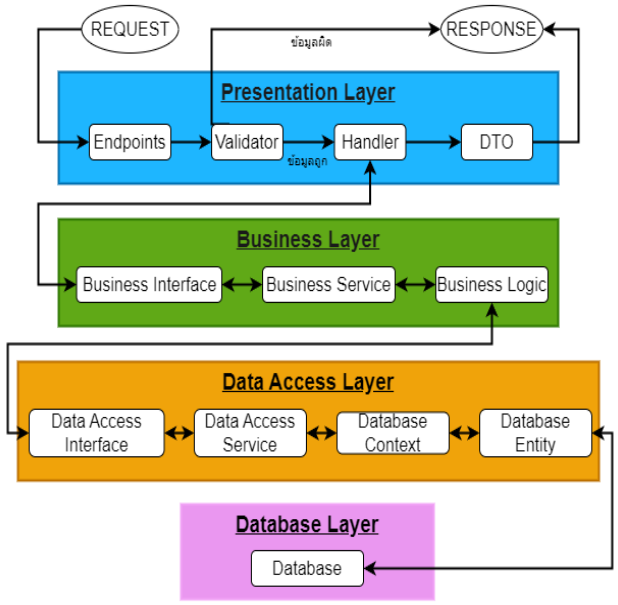
3.2 การวิเคราะห์ และออกแบบระบบ

ผู้วิจัยเริ่มจากการออกแบบ Use Case Diagramโดยอ้างอิงจากการศึกษาปัญหา การออกแบบระบบเพื่อให้ตรงตามวัตถุประสงค์ของผู้ใช้งาน เช่นตัวอย่างของ Use Case การสร้างเอกสารงบการจัดกิจกรรมที่แสดงในภาพที่ 3.1



ภาพที่ 3.1 ตัวอย่าง Use Case Diagram การสร้างเอกสารงบการจัดกิจกรรม

และในขั้นตอนต่อมาผู้วิจัยได้ทำการออกแบบส่วนต่อประสานโปรแกรมประยุกต์ภายใต้สถาปัตยกรรมแบบลำดับชั้นดังที่แสดงในภาพที่ 3.2



ภาพที่ 3.2 ภาพรวมการทำงานของส่วนต่อประสานโปรแกรมประยุกต์ภายในระบบ

โดยการวางแผนในการสร้างส่วนต่อประสานโปรแกรมประยุกต์ ต่อไปนี้จะขอเรียกว่า API ภายในระบบเว็บแอปพลิเคชันจัดการเอกสารสำหรับกิจกรรมส่งเสริมการขายสินค้าขององค์กรส่วนงานสุรา จะเริ่มจากการที่ฝั่งของไคลเอนต์ (Client) ได้ส่งคำร้องขอข้อมูล (Request) พร้อมกับข้อมูลที่ระบบต้องการใช้งานเข้ามายังจุดสิ้นสุด (Endpoint) ของ API ซึ่งจะถูกแบ่งเป็นหลาย ๆ Endpoint ตามชุดข้อมูลที่เราต้องการให้ API ส่งข้อมูลกลับ (Response) ไปหาทางฝั่งไคลเอนต์ หลังจากที่เข้ามายัง Endpoint แล้วจะมีส่วนของตัวตรวจสอบข้อมูล (Validator) ที่จะค่อยทำการตรวจสอบข้อมูลที่ส่งมากับ Request นั้น ๆ ว่าข้อมูลนั้นถูกต้องตามที่ API ต้องการหรือไม่ ถ้าข้อมูลไม่ถูกต้องตัวตรวจสอบจะทำการส่ง HTTP Error Code กลับไปหาทาง

ไคลเอนต์โดยทันที แต่ถ้าข้อมูลที่ส่งมานั้นถูกต้องจะถูกนำไปยังส่วนของตัวจัดการ (Handler) ซึ่งจะมีหน้าที่เรียกหาไปยังส่วนของบิสซิเนสอินเตอร์เฟส (Business Interface) ที่เป็นส่วนต่อประสานไปยังบิสซิเนสเซอร์วิส (Business Service) ซึ่งภายในนี้จะประกอบไปด้วยตรรกะทางธุรกิจต่าง ๆ (Business Logic) เพื่อที่จะได้นำข้อมูลมาปรับปรุงเปลี่ยนแปลง หรือร้องขอข้อมูลเพิ่มเติมจากฐานข้อมูลได้จากดาต้าแอคเซสอินเตอร์เฟส (Data Access Interface) ซึ่งจะเป็นส่วนติดต่อไปยังดาต้าแอคเซสเซอร์วิส (Data Access Service) ซึ่งภายในนี้จะประกอบไปด้วยดาต้าเบสคอนเท็ก (Database Context) และดาต้าเบสเอนติตี้ (Database Entity) ที่ถูกสร้างมาจากเอนติตี้ เฟรมเวิร์ค คอร์(Entity Framework Core) เพื่อทำการสร้าง อ่าน แก้ไข หรือลบ (Create, Read, Update and Delete (CRUD)) ไปยังฐานข้อมูลตามที่ดาต้าเบสคอนเท็ก ได้ทำการสร้างการเชื่อมต่อไว้ และในกรณีที่มีความจำเป็นต้องส่งกลับข้อมูลไปยังไคลเอนต์ ข้อมูลจากฐานข้อมูลก็จะถูกส่งกลับไปตามเส้นทางข้อมูลเดิมจนถึงส่วนของตรรกะทางธุรกิจเพื่อจัดการข้อมูลให้ถูกต้องตามต้องการ และจะถูกส่งกลับไปยังตัวจัดการ (Handler) เพื่อนำข้อมูลที่ได้มาปรับให้เข้ากับออบเจ็กต์สำหรับส่งข้อมูล (Data Transfer Objects (DTO)) เพื่อที่จะปรับแต่งข้อมูลบางส่วนที่ไม่ต้องการให้ไคลเอนต์ได้รับและส่งข้อมูลกลับ (Response) ไปพร้อมกับ HTTP Code ที่ถูกต้อง

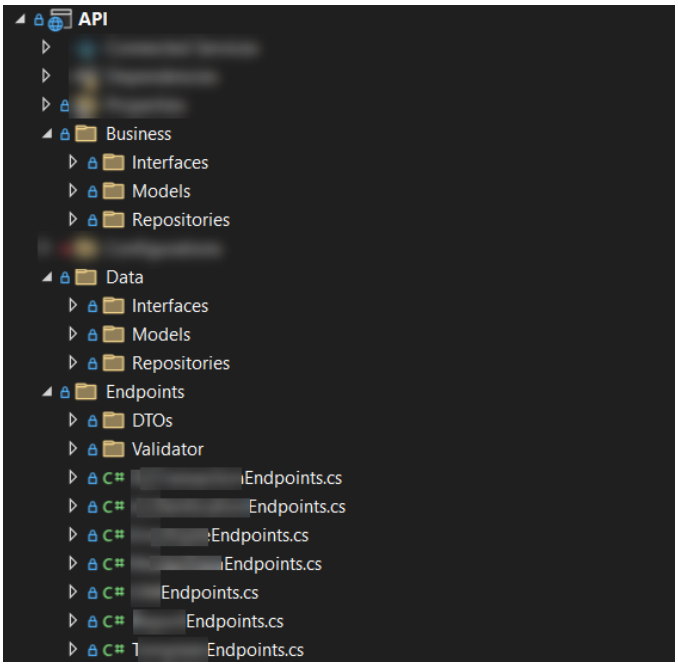
4. ผลการดำเนินงาน

ผู้วิจัยได้ทำการสร้าง API โดยอ้างอิงจาก Use Case Diagram เป็นจำนวนรวมทั้งสิ้น 48 เส้นทางด้วยกัน พร้อมทั้งได้สร้างหน้าสำหรับเอกสาร API (API Document) ด้วย Swagger UI ดังภาพที่ 4.1

ภาพที่ 4.1 ตัวอย่าง API Endpoint ผ่าน Swagger UI

โดย API แต่ละเส้นทางจะถูกสร้างอยู่ภายใต้สถาปัตยกรรมแบบลำดับชั้น โดยจะมีโฟลเดอร์แยกแต่ละลำดับชั้นออกจากกัน ซึ่งจะประกอบไปด้วยโฟลเดอร์ Endpoints ทำหน้าที่เป็น Presentation Layer ประกอบไปด้วยไฟล์ของกลุ่ม Endpoints แยกตามประเภท และโฟลเดอร์ DTOs ที่ภายในจะใช้สำหรับเก็บไฟล์ออบเจ็กต์สำหรับส่งข้อมูล ต่อมาโฟลเดอร์ Business สำหรับ Business Layer และ โฟลเดอร์ Data สำหรับ Data Access Layer ซึ่งใน โฟลเดอร์ Business และ โฟลเดอร์ Data จะยังประกอบไปด้วยโฟลเดอร์ที่

เหมือนกันคือโฟลเดอร์ Interface โฟลเดอร์ Repository และโฟลเดอร์ Models สำหรับเก็บอินเตอร์เฟสเซอร์วิสและคลาสโมเดลสำหรับใช้งานตามลำดับ แต่สำหรับในโฟลเดอร์ Data ในโฟลเดอร์ Model จะเป็นที่เก็บ Database Context และ Database Entity ที่ถูกสร้างขึ้นจากการใช้งาน Entity Framework Core ดังภาพที่ 4.2



ภาพที่ 4.2 โครงสร้าง API ตามสถาปัตยกรรมแบบลำดับชั้น

ในส่วนของประสิทธิภาพจะทำการยกตัวอย่างจาก API เส้นทางหนึ่งในระบบชื่อ GetApTransactions ซึ่งเป็น API ที่ใช้ในการดึงข้อมูลที่เกี่ยวข้องกับผู้ใช้งานที่เข้าระบบมาว่าผู้ใช้งานคนนั้นสามารถมองเห็นเอกสารใดได้บ้างและนำมาแสดงที่หน้าของตารางข้อมูลเพื่อให้ผู้ใช้ได้ทำการเลือกเอกสารที่ต้องการได้อย่างถูกต้อง โดยจะใช้โปรแกรม Postman ในการทดสอบประสิทธิภาพการใช้งาน (Performance Test) โดยมีรูปแบบการทดสอบแบบพีค (Peak Pattern) โดยมีจำนวนฐานผู้ใช้จำนวน 5 ผู้ใช้ และในช่วงเวลาสูงสุดจะใช้งานอยู่ที่จำนวน 30 ผู้ใช้ และการทดสอบนี้ใช้เวลา 20 นาที

ภาพที่ 4.3 กราฟแสดงผลการทดสอบประสิทธิภาพของ API ชื่อ GetApTransactions

User name	CPU	Memory (private working ...	Description
eas2_api_prd	01	447,828 K	IIS Worker Process

ภาพที่ 4.4 แสดงจำนวนหน่วยความจำที่ใช้ขณะรับ request จากผู้ใช้จำนวน 5 คน

User name	CPU	Memory (private working ...	Description
eas2_api_prd	03	2,607,220 K	IIS Worker Process

ภาพที่ 4.5 แสดงจำนวนหน่วยความจำที่ใช้ขณะรับ request จากผู้ใช้จำนวน 30 คน

จากภาพที่ 4.3 ถึงภาพที่ 4.5 จะแสดงผลลัพธ์การทดสอบประสิทธิภาพโดยแสดงให้เห็นว่าสามารถลองรับจำนวนผู้ใช้งานที่มีการ request เข้ามาทุกๆ 1 วินาทีเป็นจำนวน 2 ครั้งต่อผู้ใช้งานได้เป็นอย่างดีโดยไม่มี request ใดที่ได้รับข้อผิดพลาด (Error request) แต่ยังมีข้อกังวลในบ้างส่วนเรื่องของหน่วยความจำที่มีการใช้งานสูงขณะที่มีการได้รับ request เข้ามาเพื่อทำการสร้างข้อมูลเพื่อส่งกลับไป (Response) หายังทางผู้ใช้งาน

5. สรุป

ด้วยการสร้าง API แบบ Minimal API นั้นมีการออกแบบที่เรียบง่ายและมีโอเวอร์เฮดต่ำ กล่าวคือเป็นการเขียน API โดยที่ตัว API เองถูกโดนกำจัดส่วนที่เป็นตัวควบคุม (Controller) ที่ไม่จำเป็นออกไป ทำให้สามารถเขียนและทำความเข้าใจระบบ API ได้ง่ายขึ้น[8] รวมถึงทำให้สามารถรองรับการเพิ่มทรัพยากรระบบได้อย่างมีประสิทธิภาพ การใช้หน่วยความจำและการประมวลผลที่น้อยกว่าเมื่อเทียบกับ Controller API หรือเมื่อเทียบกับภาษาโปรแกรมอื่น ๆ [9] ส่งผลให้การเพิ่มประสิทธิภาพของฮาร์ดแวร์สามารถรองรับผู้ใช้งานได้มากขึ้นอย่างเป็นสัดส่วนจึงทำให้มีความสามารถในการปรับขยายในแนวตั้ง)Vertical Scaling) และในส่วนของการนำสถาปัตยกรรมแบบลำดับชั้นมาใช้งานในการแยกส่วนของ API ออกเป็นแต่ลำลำดับชั้น ทำให้สามารถทำความเข้าใจระบบรวมถึงการนำไปใช้บนเซิร์ฟเวอร์หลาย ๆ ตัวได้อย่างอิสระ โดยที่ตัว Minimal API เองนั้นมีขนาดที่เล็กและการพึ่งพาที่น้อย ทำให้การ Deploy หลายอินสแตนซ์ทำได้ง่ายและใช้ทรัพยากรน้อย จึงทำให้มีความสามารถในการปรับขยายในแนวนอน)Horizontal Scaling) อีกด้วย

และเมื่อพูดถึงความสามารถในด้านความยืดหยุ่น (Flexibility) ของระบบแล้วนั้นแม้ Minimal API จะเน้นความเรียบง่าย แต่ภายใต้สถาปัตยกรรมแบบลำดับชั้นสามารถรองรับการขยายฟังก์ชันการทำงานได้หลายรูปแบบ อาทิเช่น การรองรับการทำงานแบบ Async/Await สำหรับการประมวลผลที่ซับซ้อน หรือความสามารถในการเชื่อมต่อกับ Business Logic Layer ได้หลากหลายรูปแบบ เป็นต้น และในด้านของการบำรุงรักษาและพัฒนาต่อด้วยโครงสร้างที่เรียบง่ายของ Minimal API ส่งผลดีต่อการพัฒนาในระยะยาว

กล่าวคือการแก้ไขและปรับปรุงโค้ดทำได้ง่ายเนื่องจากมีความซับซ้อนน้อย การทดสอบทำได้ตรงจุดเพราะแต่ละ Endpoint มีหน้าที่ชัดเจน และรวมถึงการถ่ายทอดความรู้ให้ทีมพัฒนาใหม่ทำได้รวดเร็ว

แต่ทั้งนี้ Minimal API ก็ยังมีบางฟังก์ชันที่ยังไม่สามารถใช้งานได้ [10] เช่น การใช้ทำ API Endpoint ประเภท PATCH ที่จะแก้ไขข้อมูลบางส่วนจะยังไม่สามารถทำได้ ส่วนของ built-in validation หรือ built-in model binding จะไม่มีอยู่ในการสร้าง API แบบ Minimal API จึงจำเป็นต้องทำการสร้างฟังก์ชันการ validation หรือ model binding ด้วยตนเอง เป็นต้น รวมถึงในระยะยาวด้วยจำนวนข้อมูลที่เพิ่มขึ้นอาจจะต้องมีการคำนึงถึงการใช้ระบบ Cache Database อาทิเช่น Redis เป็นต้น มาช่วยในการเก็บค่าข้อมูลคงที่ที่ถูกเรียกใช้บ่อยครับ จะช่วยเพิ่มประสิทธิภาพในการทำงานของระบบได้ดีมากยิ่งขึ้น

บรรณานุกรม

Baptista, Ruth and Shah, Deven. (2018). 3-Tier Website As a SAAS Model. *Proceedings of International Conference on Communication, Security and Optimization of Decision Support Systems*, ISBN-978-0-9994483-1-1, pp. 33-38.

Dhali, Salle. (2012). Web application development with .NET. *Bachelor's thesis,* Mid Sweden University Department of Information Technology and Media (ITM).

Ferrandez, Tess. (2020). Organizing ASP.NET Core Minimal APIs. Retrieved from https://www.tessferrandez.com/blog/2023/10/31/organizing-minimal-apis.html

Giesel, S. (2024). *Comparing the performance between the Minimal API and classic Controllers*. Retrieved from https://steven-giesel.com/blogPost/698c45c3-58c5-4157-b4da-2cde4e27862e

Mayak, C. (2024). Node.js (Express) vs .Net: Hello World Performance. Retrieved from https://medium.com/deno-the-complete-reference/node-js-express-vs-net-hello-world-performance-58da841bd82f

Microsoft Learn. (2023). Choose between controller-based APIs and minimal APIs. Retrieved from https://learn.microsoft.com/en-us/aspnet/core/fundamentals/apis?view=aspnetcore-9.0

Patel, Ravi. (2024). A Beginner's Guide to Entity Framework Core (EF Core). Retrieved from https://medium.com/@ravipatel.it/a-beginners-guide-to-entity-framework-core-ef-core-5cde48fc7f7a

Songpon Ninwong. (2023). N-tier Architecture สำหรับ Golang REST API Application. สืบค้น 19 ธันวาคม 2567. https://blog.finnomena.com/n-tier-architecture-สำหรับ- golang-rest-api-application-9826abe1c5d6, 2566

TechEmpower. (2023).Web Framework Benchmarks Round 22 (Composite scores). Retrieved from https://www.techempower.com/benchmarks/#section=data-r22&hw=ph&test=composite

Valerio De Sanctis. (2023). *Building Web APIs with ASP.NET Core,* Manning Publications Co. New York.