การประยุกต์สถาปัตยกรรมองค์กรเต็มรูปแบบเพื่อพัฒนาเว็บแอปพลิเคชัน

FULL STACK PATTERNS OF ENTERPRISE ARCHITECTURE FOR WEB APPLICATION DEVELOPMENT

ธีระพล ลิ้มศรัทธา[1]*, รัตนาภรณ์ นิลพงษ์[2], พุทธิพงษ์ พิพัฒน์วุฒิกร[3] และ สุชาติ รมณียารักษ์[4]
Theerapol Limsatta[1], Rattanaporn Nilpong [2], Pudthiphong Piphatwuttikorn [3] and Suchart Rommaneeyarak [4]

สาขาวิชาเทคโนโลยีสารสนเทศ คณะเทคโนโลยีดิจิทัลและนวัตกรรม, มหาวิทยาลัยเซาธ์อีสท์บางกอก [1,2,4]
สาขาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์, มหาวิทยาลัยปทุมธานี [3]
Department of Information Technology, Faculty of Digital Technology and Innovation,
Southeast Bangkok University [1,2,4]
Electrical of engineering, Faculty of Engineering, Pathumthani University [3]
Corresponding author email: theerapol.lim@gmail.com [1]*, rattanaporn_n@southeast.ac.th[2],
ipareng@gmail.com[3], suchart@southeast.ac.th[4]

Received: September 25, 2024
Revised: December 1, 2024
Accepted: December 9, 2024

บทคัดย่อ

การพัฒนาระบบซอฟต์แวร์ในปัจจุบัน ด้วยขนาดของระบบที่มีความซับซ้อนและมีการขยายตัวเพิ่มขึ้นจากความต้องการการใช้งานที่เปลี่ยนแปลง ส่งผลให้เกิดแนวทางการพัฒนาการดำเนินงานบนระบบซอฟต์แวร์ AngularJS โดยใช้แนวคิดสถาปัตยกรรม Model-View-Controller มาศึกษาเพื่อแยกส่วนการทำงานอย่างอิสระต่อกันในงานวิจัยนี้ วัตถุประสงค์ของงานวิจัย 1) เพื่อพัฒนาและจัดการทุกชั้นของระบบเว็บแอปพลิเคชัน ตั้งแต่ frontend และ backend ไปจนถึง infrastructure และ 2) เพื่อพัฒนาระบบสารสนเทศเพื่อการจัดการเอกสาร มคอ.3 และ มคอ.5 จากการสุ่มกลุ่มตัวอย่าง จำนวน 40 คน เป็นอาจารย์ผู้สอนของมหาวิทยาลัยเซาธ์อีสท์บางกอกที่มีแผนการจัดการเรียนการสอนและมีความเข้าใจในเอกสาร มคอ. จึงเหมาะต่อการเก็บกลุ่มตัวอย่างผู้ใช้งาน เครื่องมือที่ใช้วัดและประเมินผลเป็นแบบประเมินประสิทธิภาพ โดยการวัดแบบมาตราส่วน 5 ระดับ ตามแนวคิดของ Good ที่ทำให้เห็นภาพรวมผลการดำเนินงานและความสามารถในการทำงานตามมาตรฐาน และใช้ซอฟต์แวร์ AngularJS แบบ Model-View-Controller Architecture เพื่อพัฒนาและออกแบบเว็บแอปพลิเคชัน ตามแนวคิดของ Martin Fowler จากผลการวิจัย พบว่าซอฟต์แวร์ที่ใช้สามารถแยกความรับผิดชอบแต่ละส่วนอย่างอิสระต่อกัน และเกิดประสิทธิภาพต่อผู้ใช้งานระบบ ในด้านของหมวดการเรียนรู้ที่ต้องการใช้งาน หมวดการใช้งานง่าย เมื่อเทียบกับระบบเดิม (เอกสาร) และหมวดการเข้าถึงระบบและการเรียกใช้ออนไลน์ ซึ่งภาพรวมอยู่ในระดับ (ดี) มากที่สุด ($\bar{X}$ = 4.56, S.D. = 0.48) ดังนั้น การออกแบบและพัฒนาเว็บแอปพลิเคชันด้วยแนวทางนี้จึงเป็นทางเลือกที่เหมาะสมกับยุคที่ต้องการการพัฒนาซอฟต์แวร์ที่รวดเร็ว ยืดหยุ่น และสามารถบำรุงรักษาได้ในระยะยาว

คำสำคัญ: รูปแบบสถาปัตยกรรมองค์กร, รูปแบบการออกแบบ, เว็บแอปพลิเคชัน, ระบบการจัดการเอกสาร มคอ.

Abstract

The Development of Software Systems in the Present Era. The increasing complexity and scalability of software systems driven by changing user demands have led to the adoption of development approaches leveraging AngularJS with the Model-View-Controller (MVC) architecture. This study aims to explore how system components can be decoupled to operate independently. The research objectives are: (1) to develop and manage all layers of a web application system, including the frontend, backend, and infrastructure, and (2) to create an information system for managing TQF3 and TQF5 academic documents. The sample consisted of 40 lecturers from Southeast Bangkok University who were involved in academic planning and familiar with TQF documentation, making them a suitable group for system evaluation. Performance assessment tools using a 5-level Likert scale, based on Good's conceptual framework, were employed to provide an overview of system functionality and operational efficiency in meeting established standards. The system was developed using AngularJS and the MVC architecture, applying Martin Fowler's principles, which emphasize separation of concerns. Results demonstrated that the software effectively separated functional responsibilities into independent modules, enhancing system performance in key areas. These included usability for academic learning, ease of use compared to traditional paper-based systems, and improved accessibility through online operations. The overall performance rating was exceptionally high (Mean = 4.56, S.D. = 0.48). In conclusion, the adoption of this approach to designing and developing web applications is well-suited to modern software development needs, offering speed, flexibility, and maintainability for long-term use.

Keywords: pattern of enterprise application architecture, design pattern, web application, TQF document management system

บทนำ

ปัญหาสำคัญของนักพัฒนาซอฟต์แวร์อย่างหนึ่งคือ ความซับซ้อนที่เพิ่มมากขึ้นเรื่อยๆ เมื่อมีการใช้งานในระยะยาว ด้วยความต้องการที่เพิ่มขึ้น หรือขอบเขตงานที่มากขึ้น ทำให้โครงสร้างระบบซอฟต์แวร์ หรือสถาปัตยกรรมซอฟต์แวร์ จัดการกับปัญหานี้ได้ยากขึ้น และลดประสิทธิภาพการใช้งานระบบ รูปแบบสถาปัตยกรรมซอฟต์แวร์ต่างๆ จึงได้ถูกสร้างขึ้นมากมาย เพื่อแก้ปัญหาระดับการยึดโยงหรือการผูกติดกันของส่วนประกอบในซอฟต์แวร์ ในระดับใดที่เหมาะสม หนึ่งในการแก้ปัญหานั้นคือ รูปแบบสถาปัตยกรรมองค์กร (Pattern of enterprise application architecture) ตามแนวคิดของ Fowler [1] ซึ่งต่อไปจะเรียกเพียงรูปแบบสถาปัตยกรรมองค์กรที่ได้ออกแบบไว้มากมาย รูปแบบเหล่านี้เกิดขึ้นจากการรวบรวมและพัฒนาจากแนวการปฏิบัติที่เกิดขึ้นจริงในการใช้งานกับธุรกิจ เช่น การอ่านฐานข้อมูล พบว่าการออกแบบคลาสให้สามารถดำเนินการกับฐานข้อมูลได้ครบทุกหน้าที่มาตราฐานควรเก็บไว้ที่เดียวกัน ซึ่งพบในรูปแบบการเข้าถึงข้อมูลในตาราง (Table data gateway) โดยรูปแบบนี้เหมาะสมกับลักษณะงานที่เกี่ยวเนื่องกับส่วนงานอื่นๆ การเกี่ยวเนื่องกับส่วนงานอื่นๆ จะเหมาะสมเพียงใดก็ขึ้นอยู่กับส่วนเกี่ยวเนื่องภายในโครงสร้างหลัก หรือสถาปัตยกรรมซอฟต์แวร์ย่อยๆ ของระบบ รูปแบบสถาปัตยกรรมหนึ่งๆ จึงไม่ได้ใช้งานได้ดีเสมอกับทุกงานอื่นๆ เสมอไป จึงเป็นหน้าที่นักพัฒนาซอฟต์แวร์จะเลือกออกแบบรูปสถาปัตยกรรมที่เหมาะสมประกอบกันขึ้นภายในโครงสร้างหลัก ซึ่งขึ้นอยู่กับลักษณะเฉพาะกับ

การนำไปใช้งานในลักษณะหนึ่งๆ เท่านั้น ภายใต้ความซับซ้อนที่เกิดขึ้นภายหลัง เมื่อความเหมาะสมเป็นเรื่องหลักในการพิจารณาภายใต้ขนาดระบบที่ขยายโตขึ้นเรื่อยๆ ความท้าทายของนักพัฒนาระบบซอฟต์แวร์ คือ การเลือกรูปแบบสถาปัตยกรรมของระบบ ซึ่งเปลี่ยนแปลงได้ตลอดเวลาที่พัฒนาระบบเมื่อพบทางที่ดีกว่าระหว่างพัฒนาระบบ ดังกฎข้อ 11 [2] ที่กล่าวว่าสถาปัตยกรรมที่ดีที่สุด เกิดจากความต้องการและออกแบบระบบ ซึ่งจะเกิดขึ้นเอง จากการจัดการตัวเองภายในคณะทำงาน ทำให้การเลือกรูปแบบสถาปัตยกรรมหนึ่งๆ ไม่มีรูปแบบตายตัว นั้นหมายความว่า การออกแบบระบบซอฟต์แวร์ที่ดีจะต้องมีความยืดหยุ่นสูงที่พร้อมจะรองรับความความเปลี่ยนแปลงที่เกิดขึ้นภายหลังได้ ไม่ว่าจะเป็นด้วยความต้องการของผู้ใช้ซอฟต์แวร์ที่เปลี่ยนแปลงหรือเทคโนโลยีที่เปลี่ยนแปลง งานพัฒนาซอฟต์แวร์นี้จะเป็นกรณีศึกษา เรื่องพัฒนาเว็บแอปพลิเคชันเพื่อการจัดการเอกสาร มคอ. (กรอบมาตรฐานคุณวุฒิระดับอุดมศึกษาแห่งชาติ) โดยเลือกใช้รูปแบบสถาปัตยกรรมวิสาหกิจ มีโครงสร้างหลักเป็นชั้นงาน (Layer) และเลือกรูปแบบสถาปัตยกรรมองค์กรบรรจุลงในทุกชั้นงาน ซึ่งแต่ละส่วนมีความอิสระรอรับการเปลี่ยนแปลงที่เกิดขึ้นได้ภายหลัง การเลือกใช้รูปแบบเพื่อพัฒนาระบบกับกรณีศึกษาระบบเว็บแอปพลิเคชัน ซึ่งเน้นไปที่โครงสร้างสถาปัตยกรรมองค์กร โดยแยกการประยุกต์รูปแบบออกเป็นการทำงานฝั่งไคลเอนต์ (Client) และเซิร์ฟเวอร์ (Server)

1. วัตถุประสงค์การวิจัย

1.1 เพื่อพัฒนาและการจัดการทุกชั้นของระบบเว็บแอปพลิเคชัน ตั้งแต่ frontend และ backend ไปจนถึง infrastructure

1.2 เพื่อพัฒนาระบบสารสนเทศเพื่อการจัดการเอกสาร มคอ.3 และ 5

2. เอกสารและงานวิจัยที่เกี่ยวข้อง

นิยามที่แท้จริงสถาปัตยกรรมองค์กร ยังไม่มีความหมายที่ชัดเจน และยังเป็นที่ปัญหาค้นคว้ากันต่อไป [4], [5] อย่างไรก็ตาม ในความเห็นของผู้เชี่ยวชาญบางท่าน เน้นไปที่ข้อตกลงพื้นฐานขึ้นโครงสร้าง เมื่อตกลงกันแล้วก็ยากที่จะเปลี่ยนแปลง แต่การเปลี่ยนแปลงก็เกิดขึ้นได้ในระดับย่อยๆ [6] ตามแนวทางนี้จำเป็นต้องสร้างความยืดหยุ่นที่เปลี่ยนแปลงได้เพื่อทีมพัฒนาซอฟต์แวร์พบวิธีทางที่ดีกว่าหรือความต้องการระบบที่เปลี่ยนแปลง เมื่อเกิดการเปลี่ยนแปลง Fowler แนะนำว่าโครงสร้างหลักที่ลดต้นทุนการเปลี่ยนแปลงได้ดีคือ การใช้ระบบโครงสร้าง แบบชั้นงาน ในขณะที่ Philippe Drchten, Grady Booch, Kurt Bittner และ Rich Reithman [3] เน้นความสำคัญองค์การทำงานร่วมกันของประกอบต่าง ๆ ในโครงสร้างที่ให้ความยืดหยุ่น วิธีการทำให้ระบบมีความยืดหยุ่น พิจารณาในประเด็นในหลักการ SOLID 5 ข้อ [2] กล่าวถึงการออกแบบคลาส (Class) ให้มีความพร้อมรับความเปลี่ยนแปลง เช่น เปิดช่องให้เขียนต่อเติมได้ โดยไม่มีผลกระทบกับของเดิม การลดการขึ้นต่อกันกับส่วนงานที่ไม่เกี่ยวข้อง โดยความหมายรวม มีการกล่าวถึงโครงสร้างหลัก โครงสร้างย่อย และการต่อเชื่อมระหว่างกัน เท่านั้นที่พอจะนิยามสถาปัตยกรรมองค์กรได้ [7] ยังมีอีกความหมายที่ให้มุมมองด้านองค์กรมากกว่ากระบวนการพัฒนาซอฟต์แวร์ กล่าวคือ เป็นแบบจำลองทางธุรกิจ ที่มีผู้มีส่วนได้ส่วนเสีย มีลำดับการดำเนินงานกิจกรรมทางธุรกิจ เหตุผลทางธุรกิจ และข้อบังคับในทางธุรกิจ [8] แต่แนวคิดนี้ตีความให้เห็นเป็นรูปธรรมในเชิงโครงสร้างซอฟต์แวร์ได้ยาก และใช้เวลามากเกินข้อจำกัดแค่การพัฒนาซอฟต์แวร์ กลายเป็นกรอบงานที่ใช้สำหรับจัดการโครงสร้างและกลยุทธ์ในองค์กรทางธุรกิจ [9] อย่างไรก็ตามข้อจำกัดทางด้านการพัฒนาซอฟต์แวร์ จะเห็นว่าจากแนวคิดเชิงโครงสร้างหลักและโครงสร้างพื้นฐานที่ใช้งานทางปฏิบัติมีความเป็นไปได้ คือ รูปแบบโครงสร้างแบบชั้นงาน (Layer) [10] รูปแบบนี้นักพัฒนาซอฟต์แวร์มักนิยมใช้และมีการประยุกต์กับงานด้านอื่นๆ เช่น ใช้ในการออกแบบโครงสร้างการระบบสื่อสาร ที่รู้จักกันในชื่อแบบจำลอง OSI (Open Systems Interconnection) โดยโครงสร้างหลักจะมีรูปแบบสถาปัตยกรรมต่างๆ อยู่ภายใน เช่น Model - View - Controller ซึ่งเป็นอีกหนึ่งรูปแบบที่ได้รับความนิยมในการพัฒนาเว็บแอปพลิเคชัน เห็นได้จาก Open source อย่างเช่น

AngularJS โดยมีคอนโทรลเลอร์ ทำหน้าที่รับการตอบสนองที่จะเลือกโมเดล หรือข้อมูลงานและ View หรือส่วนแสดงผล ดังนั้นการพัฒนาเว็บไซต์ จึงมีสองมุมมอง คือ มุมมองแบบชั้นงาน (Layer) กับมุมมอง MVC (Model – View - Controller) โดยมุมมองแบบชั้นงานเน้นการทำงานฝั่งเซิร์ฟเวอร์ (Server) ส่วนมุมมอง MVC เน้นการทำงานฝั่งไคลเอนต์ (Client) ในด้านการออกแบบรูปแบบสถาปัตยกรรมฝั่งเซิร์ฟเวอร์ (Server) มีความหลากหลายขึ้นอยู่กับการเลือกรูปแบบ เช่นเดียวกันกับรูปแบบฝั่งไคลเอนต์ (Client) ที่มีรูปแบบให้เลือกมากมายเช่นกัน

การวางรูปแบบสถาปัตยกรรมองค์กรแบบชั้นงานหลัก 3 ชั้น คือ ชั้นงานโดเมนลอจิก ชั้นงานข้อมูล ชั้นงานแสดงผล นอกจากนี้ยังมีกลุ่มงานสนับสนุนชั้นงานหลัก คือ กลุ่มงานบริการ

2.1 ชั้นงานโดเมนลอจิก (Domain Logic Layer)

จุดเริ่มต้นแรกคือ การวิเคราะห์ ระบบการใช้งาน ใช้ในกิจกรรมประเภทใด เช่น การใช้งานเป็นระบบเว็บทางเลือกที่แคบลงมักจะลงท้ายด้วยการเน้นรูปแบบแสดงผลในรูปแบบ Model - View - Controller แต่ถ้าเน้นให้บริการข้อมูล มักเลือกเว็บเซอร์วิส (Web service) แต่สิ่งที่ยากที่สุดคือ การวิเคราะห์โดเมนลอจิก ซึ่งมีทางเลือกที่สัมพันธ์ต่อไปกับการเลือกรูปแบบในชั้นงานข้อมูล

สิ่งที่ทำให้ระบบงานในชั้นโดเมนลอจิกดูง่ายที่สุดคือ ยังไม่ต้องเริ่มต้นพัฒนาหรือไม่ออกแบบอะไรในชั้นนี้ ปล่อยให้ขั้นตอนการออกแบบมาจากชั้นงานแสดงผล ซึ่งหมายความว่าข้ามขั้นตอนนี้ไปก่อน หากเลือกแนวทางนี้ ชั้นงานข้อมูล จะแสดงบทบาทสำคัญ รูปแบบการใช้ข้อมูล จึงควรเลือกรูปแบบเป็นกลางๆ และอย่างง่าย รูปแบบนี้คือ รูปแบบการเข้าถึงข้อมูลต่างๆ เช่น การเข้าถึงข้อมูลดิบ (Raw Data Gateway) หรือ การเข้าถึงข้อมูลในตาราง (Data Table Gateway) ในขั้นตอนการแสดงผล จะเลือกได้เองว่าจะนำข้อมูลใด รูปแบบที่คิดได้ในตอนนั้นก็คือ รูปแบบไฟล์ธุรกรรม (Transaction Script) หรือตารางโมดูล (Table Module) เป็นการคิดอย่างง่ายที่เหมาะสำหรับการพัฒนาในตอนเริ่มต้นของโครงการ เมื่อพัฒนาได้ระยะหนึ่งจะถึงเห็นความซับซ้อนที่ค่อยคลี่คลายให้เห็นในภาพรวม และหากพบว่ามีวิธีการดีกว่าที่สามารถทำได้ จึงค่อยมีการปรับเปลี่ยนรูปแบบไปสู่แบบจำลองโดเมน (Domain Model)

กรณีที่เป็นโครงการขนาดใหญ่ จำเป็นต้องใช้ระยะเวลาวิเคราะห์การเริ่มต้น ด้วยรูปแบบจำลองโดเมน แต่ใช้รูปแบบการเข้าถึงข้อมูล (Data Gateway) การทำงานกับฐานข้อมูลเป็นรากฐานก่อน เพราะต้องการใช้ความสำคัญกับชั้นงานโดเมนลอจิก ควรเลือกรูปแบบจำลองโดเมน เป็นคิดบนความจำเป็นต้องใช้ข้อมูลที่ได้จากข้อมูลที่ต้องการใช้จริงๆ โดยไม่ได้สนใจในชั้นงานข้อมูล (Data Layer) และต้องการให้ข้อมูลเป็นอิสระต่อการคิดวิเคราะห์ ความต้องการใช้ข้อมูลที่มาก จากหลากหลายลักษณะการวิเคราะห์ระบบในลักษณะงานเว็บ บางครั้งคิดจากมุมมองของผู้ใช้ข้อมูลที่เป็นไฟล์เอกสาร (กระดาษ) และลักษณะการใช้ประโยชน์ของข้อมูลที่จะนำมาตอบคำถาม รวมทั้งประโยชน์ที่ควรได้ของผู้ต้องการใช้ข้อมูล

2.2 ชั้นงานข้อมูล (Data Layer)

ในชั้นงานนี้ อาจไม่ใช่ ขั้นตอนต่อจากการวิเคราะห์โดเมนลอจิก แต่เป็นการจำลองข้อมูลในขั้นตอนการวิเคราะห์และออกแบบระบบ ผู้พัฒนาระบบมักจะเริ่มจากการวิเคราะห์กระบวนการทำงานผ่านแผนภาพ คือ แผนภาพแสดงทิศทางการไหลของข้อมูล (Data Flow Diagram - DFD) ต่อด้วยการทำแผนภาพความสัมพันธ์ของเอ็นทิตี (Entity Relationship Diagram - ERD) ไปพร้อมกัน ซึ่งถือเป็นการทำงานขนานกันและหากจะเลือกทำ ERD ภายหลัง DFD ก็ไม่เป็นข้อจำกัดในการทำงาน

การพัฒนาระบบซอฟต์แวร์ในขั้นตอนนี้ ไม่จำเป็นต้องเลือกทำ DFD ก่อนหรือหลัง หรือขนานกัน การเลือกรูปแบบการเข้าถึงข้อมูลในอีกรูปแบบหนึ่งที่เป็นกลางกับทุกชนิดฐานข้อมูลและตารางข้อมูล คือ รูปแบบ Metadata Mapper หากเลือกขั้นตอนการวิเคราะห์เป็นขั้นตอนแรกอย่างละเอียดก่อน เช่น การวิเคราะห์ในระดับแบบจำลองโดเมน จะมีผลถัดมาที่จำเป็นต้องเลือกใช้ส่วนขยายของรูปแบบ Metadata Mapper มาสู่ Query Object หรือ Data Mapper

เมื่อระบบพัฒนาไปถึงระยะหนึ่งของการปรับปรุงประสิทธิภาพการทำงาน ก็เริ่มที่จะนำรูปแบบการดำเนินงานกับฐานข้อมูลเชิงพฤติกรรม เช่น การใช้รูปแบบหน่วยการทำงาน (Unit of Work - UoW) เพื่อจัดการกับธุรกรรมให้เหมาะสม

2.3 ชั้นงานแสดงผล (Presentation Layer)

สำหรับงานระบบเว็บทางเลือกหลัก คือ Model - View - Controller (MVC) เป็นทางเลือกไม่ว่าระบบงานเว็บจะเน้นการทำงานบนเซิร์ฟเวอร์ (Server) หรือไคลเอนต์ (Client) หากเลือกเน้นบนเซิร์ฟเวอร์ (Server) จะมีทางเลือกเพิ่มขึ้นอีก คือ การใช้รูปแบบ Template view ที่พึ่งพาความสามารถของเซิร์ฟเวอร์ (Server) ซึ่งให้ความแน่นอนและปลอดภัยสูงกว่าการทำงานฝั่งไคลเอนต์ (Client) แต่การใช้ความสามารถของไคลเอนต์นั้น ก็ส่งผลให้การตอบสนองเชิงประสบการณ์ของผู้ใช้งานได้ดีกว่า รวดเร็วกว่า การเลือกใช้การทำงานฝั่งไคลเอนต์ (Client) มีทางเลือกเพิ่มคือ การใช้รูปแบบ Transform View โดยใช้ภาษา JavaScript เป็นหลักในการประมวลผลหน้าแสดงผล รวมทั้งการสร้างการสืบค้นไปยังเซิร์ฟเวอร์ (Server) โดยเทคโนโลยี AJAX

หากระบบงานเว็บประกอบด้วยหน้าเว็บจำนวนมากและมีความซับซ้อนมาก การเลือกใช้รูปแบบ Front Controller ที่มีตัวช่วยการจัดการคำขอในการเลือกคอนโทรลเลอร์ ร่วมกับรูปแบบ Transform View หรือ รูปแบบ Template view ได้ ขึ้นอยู่กับว่าจะเน้นการทำงานบนฝั่งใดของระบบเว็บ

2.4 กลุ่มงานบริการ (Service Layer)

กลุ่มงานบริการเป็นส่วนประกอบเสริมที่จำเป็นต้องมีและสำคัญในการเสริมประสิทธิภาพของระบบงานในด้านต่างๆ เช่น ด้านความปลอดภัย ด้านการจัดการ Session หรือด้านการคงสถานะอยู่ในระบบ นอกจากนี้ยังสามารถทำงานข้ามเขตชั้นงานได้ เช่น การตรวจสอบการคงอยู่ของระบบผ่านหน้าเว็บต่างๆ ซึ่งเป็นส่วนแสดงผลกับสถานะข้อมูลผ่านการประมวลผลโดเมน

งานบริการสื่อสารจดหมายอิเล็กทรอนิกส์ (Email service) ที่เสริมความสำเร็จในการประมวลผลของโดเมนลอจิก เช่น การยืนยันตัวตนของสมาชิกหลังจากการลงทะเบียนผ่านจดหมายอิเล็กทรอนิกส์

งานบริการของ RESTful มีส่วนประกอบที่ส่งผ่านงานคำขอบริการไปยังคอนโทรลเลอร์ได้ ซึ่งมองว่า RESTful จะคล้ายกับ Front Controller ที่เป็นตัวกลางในการผ่านงานของระบบ

รูปแบบโปรแกรมอรรถประโยชน์ (Utility Programs) สามารถจัดอยู่ในส่วนชั้นงานบริการได้ รูปแบบที่มักใช้ในชั้นงานบริการ คือ รูปแบบที่เป็นรากฐานใช้งานทั่วไป เช่น รูปแบบลงทะเบียน (Registry pattern) รูปแบบกรณีพิเศษ (Special case pattern) และรูปแบบซุปเปอร์ไทป์ (Super type pattern)

กลุ่มงานบริการ (Service Layer) สามารถสร้างเป็นชั้นงานบริการในแนวนอนที่อยู่ระหว่างชั้นอื่นๆ ได้ เช่น อยู่ระหว่างชั้นงานโดเมนลอจิก ที่ช่วยส่งเสริมกระบวนการธุรกรรม เช่น การจัดการกับ Unit of Work (UoW) เพื่อให้การทำงานของธุรกรรมเสร็จสมบูรณ์ ซึ่งเป็นการใช้รูปแบบที่ช่วยให้สามารถจัดการกับสถานะของข้อมูลในการดำเนินการหลายขั้นตอนภายในหนึ่งกระบวนการธุรกรรม และอีกกรณีหนึ่งของชั้นงานบริการในแนวตั้ง เป็นลักษณะการจัดวางที่สามารถเข้าถึงทุกชั้นงานภายในระบบได้ เช่น การใช้รูปแบบขนถ่ายข้อมูลวัตถุ (Data transfer object - DTO) ที่ให้บริการขนถ่ายข้อมูลในรูปแบบออบเจ็กต์ระหว่างโดเมนลอจิกกับชั้นงานแสดงผล รวมทั้งการขนถ่ายข้อมูลระหว่างชั้นงานข้อมูลกับชั้นงานโดเมนลอจิก โดย DTO จะเป็นตัวกลางในการจัดส่งข้อมูลจากชั้นหนึ่งไปยังอีกชั้นหนึ่งในรูปแบบที่สามารถจัดการได้ง่าย

3. วิธีดำเนินการวิจัย

3.1 เครื่องมือการวิจัย

3.1.1 แบบประเมินประสิทธิภาพ โดยการวัดแบบมาตราส่วน 5 ระดับ [Good] ที่ทำให้เห็นภาพรวมผลการดำเนินงานและความสามารถในการทำงานตามมาตรฐาน

3.1.2 ชุดซอฟต์แวร์ AngularJS แบบ Model-View-Controller Architecture เพื่อพัฒนาและออกแบบเว็บแอปพลิเคชัน [Martin Fowler]

3.2 กลุ่มเป้าหมาย

3.2.1 ประชากรเป็นอาจารย์ผู้ใช้ระบบเอกสาร มคอ. จากประชากรอาจารย์ผู้ใช้งานจริง จำนวน 40 คน ของมหาวิทยาลัยเซาธ์อีสท์บางกอก สาขาวิชาต่าง ๆ โดยใช้เกณฑ์การเลือกแบบสุ่มเลือก (random sampling) ใช้เกณฑ์เลือกจำนวนจากสูตรของ Yamane ได้จำนวนตัวอย่างประมาณ 29 คนที่ระดับนัยสำคัญ 0.1

3.3 ขั้นตอนการดำเนินการวิจัย

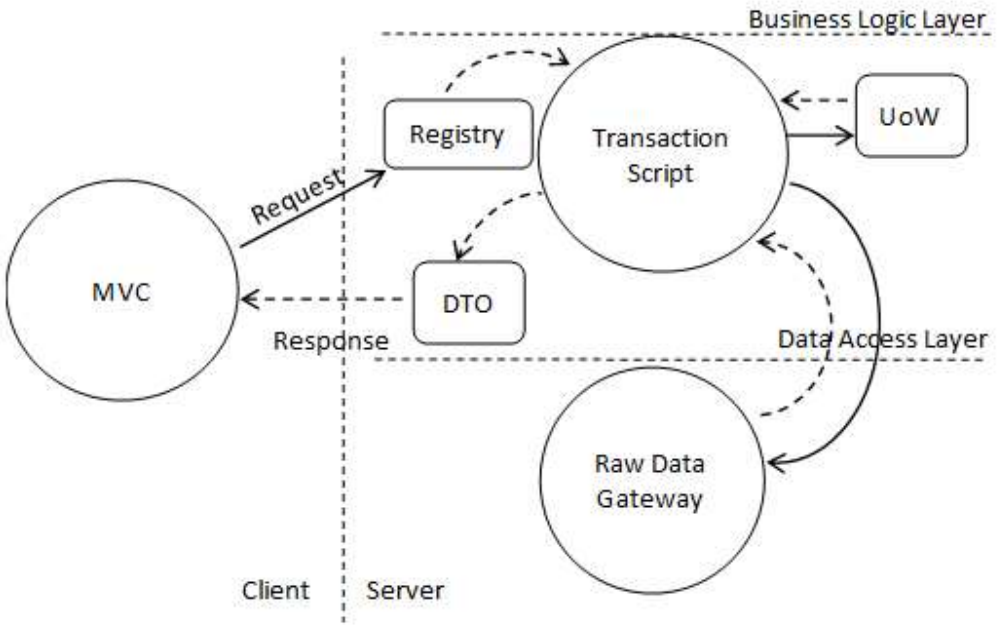
3.3.1 ศึกษาระบบงานเดิม

กิจกรรมของระบบงานเดิม เพื่อให้สามารถรับรู้ถึงขอบเขตงาน ผู้ที่เกี่ยวข้อง และข้อมูลจากสัมภาษณ์ที่เกี่ยวข้องกับความต้องการของระบบ รวมถึงข้อมูลที่ไม่ได้ระบุไว้ในเอกสาร การดำเนินกิจกรรมดังกล่าวทำให้เข้าใจระบบงานและทำให้ทราบความต้องการของระบบทั่วไป ซึ่งจะนำไปสู่การพัฒนาระบบให้ตอบสนองความต้องการได้

3.3.2 เลือกรูปแบบสถาปัตยกรรมที่เหมาะสม

ขั้นตอนการพัฒนาระบบ คือ การเลือกรูปแบบบสถาปัตยกรรมองค์กรที่เหมาะสม ซึ่งเป็นเป้าหมายหลักของงานวิจัยนี้ การออกแบบโครงสร้างหลักเลือกเป็นแบบชั้นงาน โดยการเริ่มต้นเลือกออกแบบโดเมนลอจิกและบรรจุชั้นงานนี้ในลักษณะระบบชั้นงาน ซึ่งเป็นวิธีที่ง่ายที่สุด [1] การเลือกแบบจำลองโดเมน ตั้งแต่แรกพัฒนาทางเลือกนี้อาจซับซ้อนและใช้เวลานาน เพราะระบบงานจริงไม่ซับซ้อนมาก ไม่มีส่วนเกี่ยวข้องกับหลายหน่วยงาน จึงเลือกรูปแบบที่เร็วกว่า คือการเลือกรูปไฟล์ธุรกรรม เพราะการใช้เพียงไฟล์หนึ่งไฟล์ สำหรับการดำเนินการธุรกรรมหนึ่งอย่างกับระบบเท่านั้น และพร้อมจะเพิ่มไฟล์หรือลบไฟล์ออกได้อิสระ โดยไม่มีผลกระทบกับส่วนงานอื่น ส่งผลทำให้เลื่อนข้ามชั้นตอนการออกแบบ ไฟล์ธุรกรรมไปก่อนได้ ส่วนประกอบอื่นของชั้นงานนี้ เช่น การสร้างรูปแบบหน่วยการทำงาน (Unit of Work - UoW) แม้งานนี้จะทำงานใกล้ชิดกันมากกับระบบฐานข้อมูล แต่จัดอยู่ในชั้นงานนี้ เพราะโดเมนลอจิกไม่สัมพันธ์กับระบบฐานข้อมูลโดยตรง อย่างไรก็ตาม หากการเลือกใช้รูปแบบการเข้าถึงข้อมูลดิบ (Raw Data Gateway) ก็จะทำให้รูปแบบ UoW ขยับความใกล้ชิดชั้นงานข้อมูล (Data Layer) การเลือกใช้รูปแบบ UoW จึงขึ้นกับการตัดสินใจการใช้รูปแบบการทำงานกับฐานข้อมูล ซึ่งแน่นอนว่าการเลือกให้มีความใกล้ชิดกับฐานข้อมูล จะทำให้สร้าง UoW ได้ง่ายกว่า โดยเฉพาะเมื่อเลือกใช้กับรูปแบบการเข้าถึงข้อมูลดิบ เพราะมีการทำงานกับฐานข้อมูลในรูปแบบมาตรฐาน เช่น มีการเลือกทั้งหมดของตาราง การลบแถว การแทรกแถว และปรับปรุงข้อมูลเฉพาะแถว

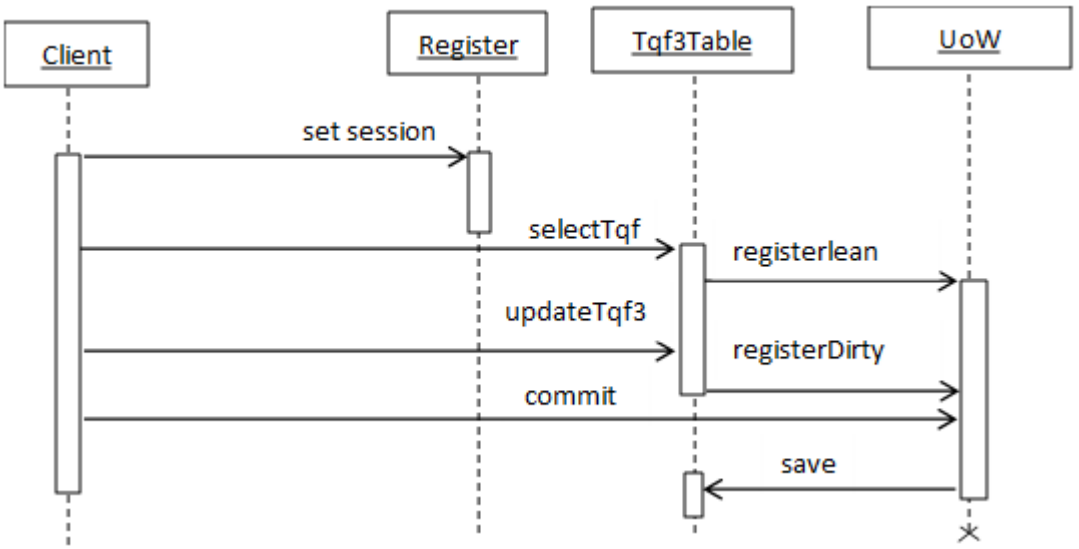
ต่อไปก็จะเป็นการออกแบบกลุ่มงานบริการ (Service Layer) การเลือกรูปแบบมาตรฐานที่สื่อสารกับไคลเอนต์ (Client) ได้ง่าย และเป็นอิสระกับการเลือกใช้เฟรมเวิร์ค (Framework) ของไคลเอนต์ (Client) สนับสนุนรับส่งข้อมูลรูปแบบ JSON จึงถูกเลือกในการขนถ่ายข้อมูล รวมทั้งการเข้ารหัสอักขระก็ทำได้หลายมาตรฐาน เช่น ภาษา PHP มีฟังก์ชันสนับสนุนให้ใช้งานได้ง่าย อีกงานบริการหนึ่งซึ่งทำตัวเป็นตัวกลางส่งต่อคำสั่งในการเลือกโดเมนลอจิกจากคำขอบริการของไคลเอนต์ คือ งานบริการลงทะเบียน (Registry) ซึ่งคล้ายกับรูปแบบการออกแบบ Mediator



รูปที่ 1 การใช้รูปแบบสถาปัตยกรรมองค์กรบนชั้นงานฝั่งเซิร์ฟเวอร์ (Server) และการติดต่อกับไคลเอนต์ (Client)

จากรูปที่ 1 แสดงการเลือกใช้รูปแบบสถาปัตยกรรมองค์กรบนฝั่งเซิร์ฟเวอร์ โดยเริ่มจากมีคำขอบริการ (Request) ไปยังเซิร์ฟเวอร์ โดยมีงานบริการรับคำขอเป็นตัวกลาง (Registry) ทำหน้าที่กำหนดความปลอดภัย ก่อนจะส่งต่อไปยังไฟล์ธุรกรรม เมื่อความจำเป็นต้องใช้ฐานข้อมูล จะส่งคำขอไปรูปแบบการเข้าถึงข้อมูลดิบ ก่อนการส่งข้อมูลไปยังไคลเอนต์จะผ่านบันทึกอยู่ในรูปแบบ (UoW) โดยเฉพาะเมื่อต้องทำงานกับฐานข้อมูลเพื่อรองรับความสำเร็จของงาน ก่อนจะดำเนินงานจริงกับฐานข้อมูล เมื่องานโดเมนลอจิกพร้อมตอบสนองจะบรรจุข้อมูลพร้อมส่งในรูปแบบมาตรฐานที่ไคลเอนต์รองรับได้ (DTO)

สำหรับลำดับการทำงานของออบเจ็กต์ เมื่อมีคำขอจากผู้ใช้รายหนึ่ง เพื่อการปรับข้อมูล TQF3 เริ่มจากทุกครั้งที่เข้าสู่ระบบได้ จะต้องลงทะเบียนความปลอดภัย ผู้ใช้งานเลือกออบเจ็กต์ Tqf3Table เป็นตัวเดียวกับออบเจ็กต์ที่ทำงานกับฐานข้อมูล เมื่อได้ออบเจ็กต์ตามต้องการแล้ว จะลงทะเบียนกับ UoW เพื่อแจ้งว่ามีออบเจ็กต์เกิดใหม่ที่จะทำงานแล้ว (registerlean) ต่อมาผู้ใช้ปรับปรุงข้อมูลออบเจ็กต์ Tqf3Table และต้องนำไปลงทะเบียนอีกครั้ง ว่ามีการปรับเปลี่ยนเกิดขึ้น (registerDirty) จนกระทั่งผู้ใช้ยืนยันการเปลี่ยนแปลง (commit) UoW จึงสร้างการเปลี่ยนแปลงที่ฐานข้อมูลบน Tqf3Table (save) ดังรูปที่ 2

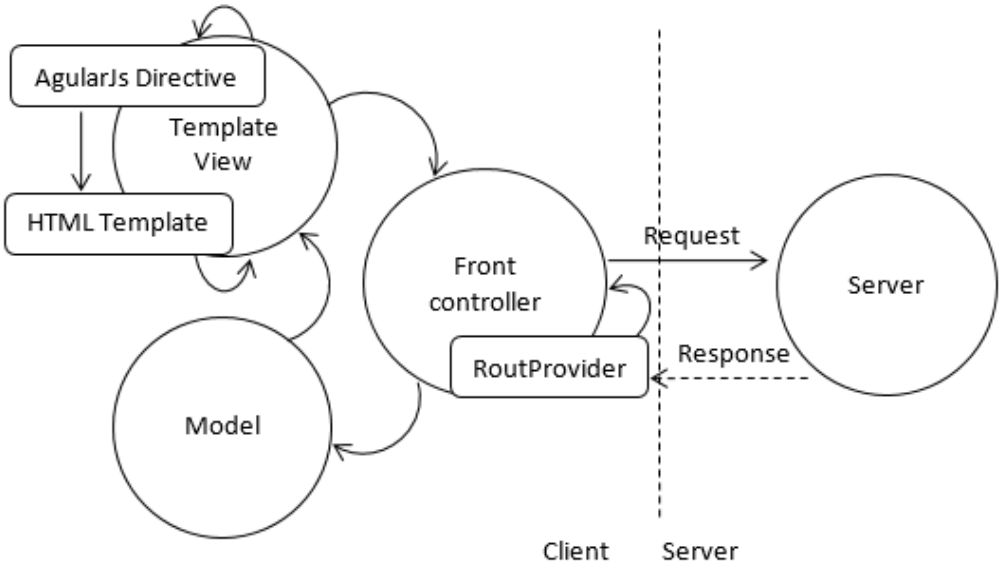


รูปที่ 2 การใช้รูปแบบสถาปัตยกรรมองค์กร รูปแบบ Transaction Script เพื่อการปรับปรุงข้อมูล Tqf3

สำหรับการออกแบบในฝั่งไคลเอนต์อยู่ในสภาพค่อนข้างจำกัด เนื่องด้วยเลือกใช้เฟรมเวิร์ค (Framework) ซึ่งเลือกใช้ AngularJS ซึ่งเป็นรูปแบบแอพพลิเคชันเว็บหนึ่งหน้า (Single page web application) ในความหมายคือ มีเพียงหนึ่งหน้าหลัก แต่ภายในหนึ่งหน้าพร้อมสลับสับเปลี่ยนเนื้อหาได้อย่างไดนามิก และจัดเป็นรูปแบบ MVC ชนิดหนึ่ง มีรูปแบบสถาปัตยกรรมองค์กรถูกผนวกใน AngularJS อยู่แล้ว การประกอบโครงร่างของ AngularJS สามารถแยกส่วนเป็นรูปแบบ Front controller และรูปแบบ Template View

Front Controller ทำหน้าที่เป็นศูนย์กลางการจัดการกับขอบริการ ก่อนที่ส่งต่อให้คอนโทรลเลอร์ทำงาน สำหรับ AngularJS ใช้งานบริการจัดการเส้นทางชื่อ Route Provider เป็นชื่อของงานบริการของ AngularJS รองรับทุกหน้าที่ต้องการแสดงผล ด้วยไฟล์เดียวเปรียบเสมือนทำหน้าที่ควบคุมการเลือกคอนโทรลเลอร์ให้ทำงาน แทนที่จะมีแต่ละไฟล์ที่มาจากคำขอของไคลเอนต์ สำหรับเลือกคอนโทรลเลอร์ การจัดการแบบศูนย์กลาง การจัดการกับคำขอบริการ สามารถกำหนดข้อกำหนดอื่นๆ เพิ่มเติมได้ นอกจากให้คอนโทรลเลอร์ทำหน้าที่แทนทั้งหมด ผลคือทำให้คอนโทรลเลอร์รับผิดชอบงานหลักของตนเองได้เต็มที่ ส่งผลให้การจัดการระบบทำให้ง่ายขึ้น

การจัดการแสดงผลให้อยู่ที่ฝั่งไคลเอนต์ (Client) กลไกสำคัญคือการใช้ภาษาที่ฝั่งไคลเอนต์ (Client) จัดการได้ตนเอง ต้องมีตัวช่วยนอกจากภาษาของ HTML และ JavaScript เปรียบเสมือนเป็น Template คือ คำสั่งของ AngularJS สำหรับผูกข้อมูลให้กับ HTML (HTML-AngularJS Directive) ซึ่งการแสดงผลในลักษณะนี้เป็นรูปแบบที่เรียกว่า Template View



รูปที่ 3 การใช้รูปแบบสถาปัตยกรรมองค์กร บนฝั่งไคลเอนต์ (Client) และการติดต่อกับเซิร์ฟเวอร์ (Server)

จากรูปที่ 3 เซิร์ฟเวอร์ ส่งข้อมูลในรูปแบบมาตรฐานที่ฝั่งไคลเอนต์รับได้ จะผ่านงานบริการจัดการเส้นทาง (RoutProvider) ทำหน้าที่เป็นหน้าด่านเหมือนรูปแบบลงทะเบียนจับคู่คอนโทรลเลอร์กับคำตอบสนอง (Response) เพื่อให้ได้คอนโทรลเลอร์เป้าหมายต่อไป คอนโทรลเลอร์จะเลือกแบบจำลองข้อมูล (Data model) และนำแบบจำลองข้อมูลไปใส่ใน Template View ในขั้นตอนทั้งหมดเป็นขั้นตอนแสดงผลที่จัดการฝั่งไคลเอนต์ได้เองทั้งหมด

3.3.3 พัฒนาระบบและทดสอบการใช้งาน

ในขั้นตอนการพัฒนาใช้การพัฒนาในฝั่งเซิร์ฟเวอร์ก่อน โดยใช้ ภาษา PHP และฐานข้อมูล MySQL และทดสอบการส่งค่าข้อมูล หลังจากนั้นพัฒนาในฝั่งไคลเอนต์ โดยใช้ Angular Library ในระหว่างดำเนินการพัฒนาเมื่อเสร็จเป็นระยะที่พอใช้งานได้ นำไปให้ทดลองใช้งาน เพื่อขอทราบผลตอบรับ ซึ่งได้รับคำแนะนำที่ดีกลับมาแก้ไขต่อไป

4. เกณฑ์การวัดในงานวิจัย

4.1 เกณฑ์วัดประเมินผล

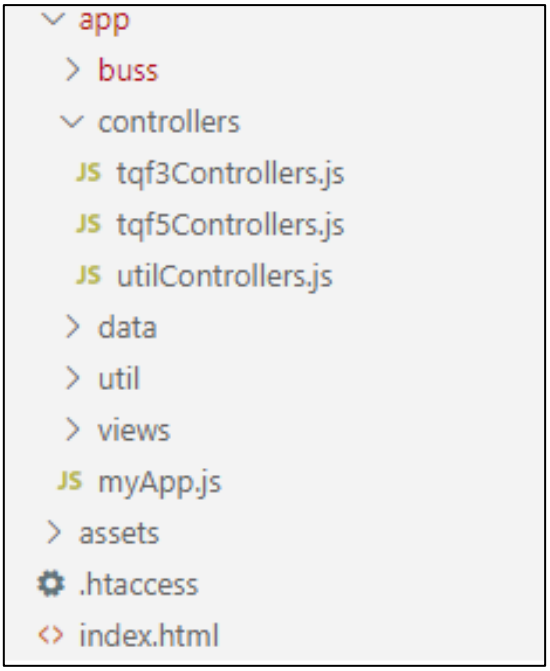
ใช้ค่าสถิติพรรณนา เช่น ค่ากลาง ค่าความแปรปรวน ดังใช้ ค่าเฉลี่ย ด้วยการวัดขนาด 5 ระดับคะแนน Likert โดยนำผลที่ได้เทียบกับเกณฑ์การประเมิน มีค่าทางสถิติดังนี้

ค่าเฉลี่ยเท่ากับ 4.51 – 5.00 หมายความว่า ระดับมากที่สุด
ค่าเฉลี่ยเท่ากับ 3.51 – 4.50 หมายความว่า ระดับมาก
ค่าเฉลี่ยเท่ากับ 2.51 – 3.50 หมายความว่า ระดับปานกลาง
ค่าเฉลี่ยเท่ากับ 1.51 – 2.50 หมายความว่า ระดับน้อย
ค่าเฉลี่ยเท่ากับ 1.01 – 1.50 หมายความว่า ระดับน้อยที่สุด

5. ผลการวิจัย

5.1 ผลการพัฒนาระบบสารสนเทศเพื่อการจัดการเอกสาร มคอ.3 และ มคอ.5

เมื่อออกแบบรูปแบบสถาปัตยกรรมต่างๆ แล้ว โครงสร้างระบบไฟล์ก็สามารถจัดให้เป็นระบบระเบียบตามโครงสร้างรูปแบบสถาปัตยกรรม เพื่อให้ง่ายต่อการดูแลรักษา โดยมีระบบโครงสร้างไฟล์ แยกเป็นโครงสร้างหลักคือ มีโฟลเดอร์ (Folder) buss ใช้เก็บไฟล์ธุรกรรม โฟลเดอร์ controllers เก็บไฟล์คอนโทรลเลอร์ ซึ่งมีเพียง 3 คอนโทรลเลอร์หลัก โฟลเดอร์ data เก็บไฟล์ที่ทำงานกับฐานข้อมูล โฟลเดอร์ util เก็บไฟล์งานบริการต่างๆ เช่น งานบริการจดหมายอิเล็กทรอนิกส์ งานด้านความปลอดภัย โฟลเดอร์ views เก็บไฟล์เพื่อใช้แสดงผล สำหรับไฟล์ myApp.js เป็นเหมือนไฟล์ลงทะเบียนของเส้นทางที่จับคู่คอนโทรลเลอร์กับ Template View และเมื่อแยกฝั่งการทำงาน มีเพียงไฟล์จาก data ไฟล์จาก buss และไฟล์จาก util ทำงานฝั่งเซิร์ฟเวอร์ ส่วนที่เหลือทำงานฝั่งไคลเอนต์ ดังรูปที่ 4

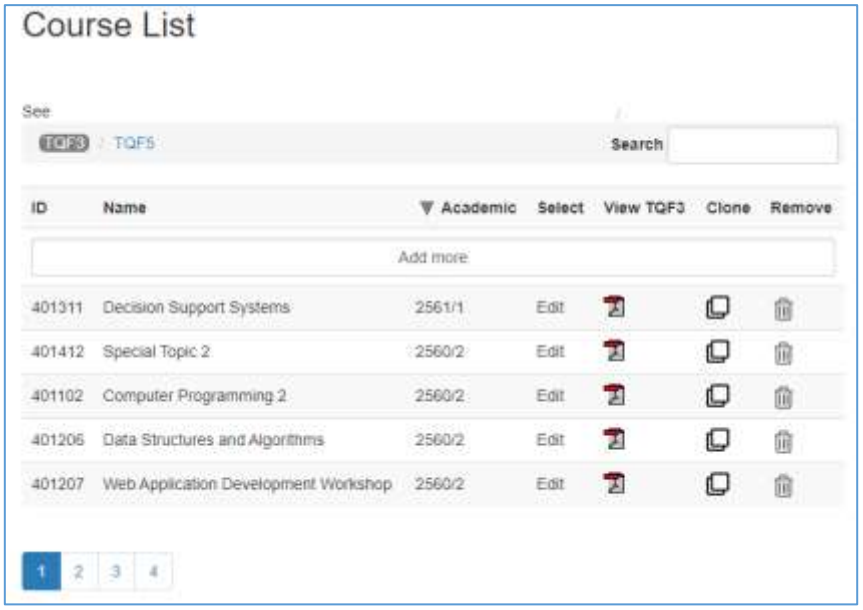


รูปที่ 4 โครงสร้างระบบไฟล์

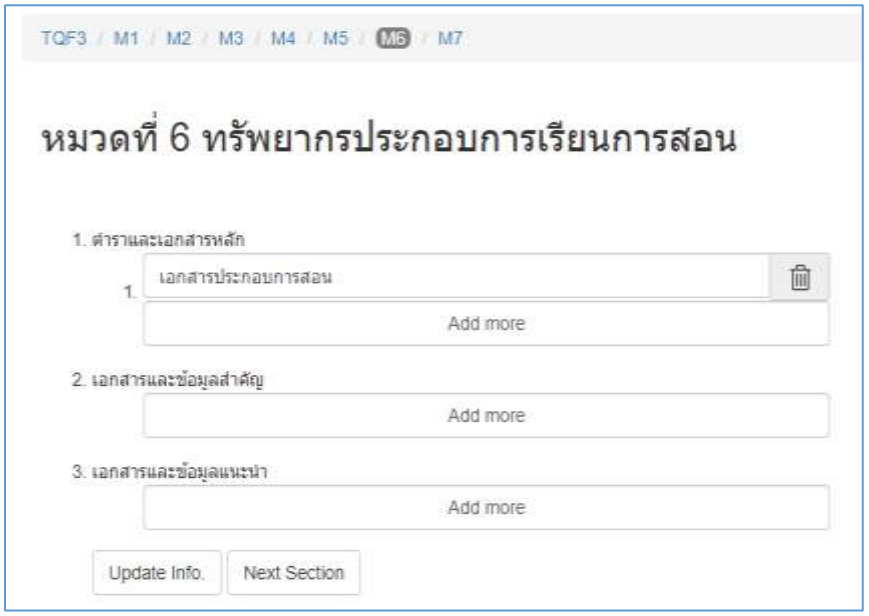
การออกแบบรูปแบบสถาปัตยกรรม ได้แบ่งแยกตามไฟล์งานและโฟลเดอร์ สามารถแก้ไขหรือเพิ่มเติมทำได้โดยไม่ต้องกังวลว่าจะกระทบกับส่วนงานอื่นมากนัก เช่น การเพิ่มหน้าแสดงผลหน้าหนึ่ง ก็สามารถเพิ่มไฟล์ใหม่ลงในโฟลเดอร์ views และเขียนเพิ่มคอนโทรลเลอร์ในโฟลเดอร์ controllers และหากมีธุรกรรมใดเพิ่มเติมก็เขียนเพิ่มไฟล์ ในโฟลเดอร์ buss โดยมีการทำงานกับฐานข้อมูลในรูปแบบมาตรฐานที่ไม่จำเป็นต้องแก้ไขอะไรในโฟลเดอร์ data

ความท้าทายของการพัฒนาระบบจึงไม่ขึ้นกับรูปแบบสถาปัตยกรรมอีกต่อไป แต่ให้ความสำคัญกับเนื้อหาการแสดงผลที่ทำให้ผู้ใช้รู้สึกใช้งานได้ง่ายและเป็นประโยชน์ เช่น ขั้นตอนทดสอบการใช้งาน มีคำแนะนำให้เพิ่มคำสั่ง ทำสำเนาเอกสารได้ เพราะเอกสารบางชิ้น แทบเหมือนกับของเดิม การแก้ไขทำเพียงการแก้ส่วนแสดงผล และเพิ่มไฟล์ธุรกรรม โดยมีผลกระทบกับส่วนอื่นน้อยมาก ผลที่ดีอีกอย่างหนึ่งคือ แต่ละรูปแบบสถาปัตยกรรมองค์กรมีหน้าที่รับผิดชอบที่ชัดเจน ทำให้สามารถแก้ไขดูแลรักษาได้ตรงจุดและตรงรูปแบบที่ใช้งาน

ตัวอย่างหน้าแสดงผลรวมของ TQF3 เป็นหน้าเว็บแสดงรายวิชาตามเอกสาร มคอ. 3 (TQF3) ที่แสดงเพียงส่วน ในหน้านี้สามารถดำเนินการแก้ไขเฉพาะรายการเอกสารได้ พิมพ์ได้ ทำสำเนาเป็นเอกสารใหม่ได้ ลบเอกสารได้ และส่วนประกอบการทำงานย่อยอื่นๆ สำหรับตัวอย่างหน้าหน้าบันทึกข้อมูล หมวด 6 ของ TQF3 เป็นหน้าเว็บแสดงรายละเอียดของเอกสาร มคอ.3 ฉบับหนึ่ง ที่เลือกเพื่อแก้ไข ในหมวดที่ 6 หน้าเว็บนี้สามารถเพิ่มเติมรายละเอียดได้ รวมทั้งลบรายละเอียดบางรายการได้ การเลือกหมวดสามารถเลือกได้โดยไม่จำเป็นต้องทำตามลำดับหมวด ยกเว้นการสร้างเอกสารใหม่ควรเลือกตามลำดับหมวด เพราะบางลำดับจะเป็นข้อมูลในลำดับหมวดต่อไป จะช่วยให้ประหยัดเวลาการกรอกข้อมูล ดังรูปที่ 5 และรูปที่ 6



รูปที่ 5 ตัวอย่างหน้าแสดงผลรวมของ มคอ.3 (TQF3)



รูปที่ 6 ตัวอย่างหน้าหน้าบันทึกข้อมูล หมวด 6 ของ TQF3

5.2 ผลการทดลองใช้ระบบสารสนเทศเพื่อการจัดการเอกสาร มคอ.3 และ มคอ.5

ผู้วิจัยดำเนินการทดลองใช้ระบบสารสนเทศที่พัฒนาขึ้นด้วยการทดสอบประเมินประสิทธิภาพการใช้งานของระบบจากผู้ใช้งาน โดยมีการทดสอบก่อนเรียนหลังการเรียนรู้ จากนั้นนำผลการเรียนรู้มาวิเคราะห์ด้วยค่าสถิติพื้นฐานเทียบกับเกณฑ์และสรุปผล

ตารางที่ 1 ผลการทดสอบประสิทธิภาพการใช้ระบบสารสนเทศเพื่อการจัดการเอกสาร มคอ.3 และ มคอ.5

รายการ	$\overline{x}$	SD.	ระดับความคิดเห็น
1. หมวด การเรียนรู้ที่ต้องการใช้งาน	4.3	0.51	มาก
2. หมวด การใช้งานง่าย เมื่อเทียบกับระบบเดิม (เอกสาร)	4.6	0.52	มากที่สุด
3. หมวด การเข้าถึงระบบ และการเรียกใช้ออนไลน์	4.8	0.43	มากที่สุด
โดยรวม	**4.56**	**0.48**	**มากที่สุด**

จากตารางที่ 1 ผลทางสถิติ พบว่าค่าโดยรวมอยู่ในระดับ (ดี) มากที่สุด คือ 4.56 และมีเบียงแบนมาตรฐาน 0.48 สำหรับค่าเฉลี่ยในหมวด การเรียนรู้ที่ต้องการใช้งานในครั้งแรก อาจเกิดจากความไม่เข้าใจต่อระบบใหม่ ซึ่งอาจต้องอบรมการใช้งานให้เข้าใจง่ายกว่านี้

6. อภิปรายผลการวิจัย

ผลการวิจัยแสดงให้เห็นว่าการพัฒนาเว็บแอปพลิเคชันด้วย AngularJS บนพื้นฐานของ Model-View-Controller (MVC) Architecture ช่วยแก้ปัญหาความซับซ้อนของระบบและช่วยสร้างแนวทางการทำงานร่วมกันที่เป็นมาตรฐานระหว่างกัน ด้วยแต่ละรูปแบบมีหน้าที่การทำงานเฉพาะด้าน สามารถแยกส่วนการทำงานแต่ละชั้นออกจากกันได้ช่วยให้นำฟังก์ชันหรือโมดูลกลับมาใช้ใหม่ได้และง่ายต่อการปรับปรุงหรือเพิ่มคุณสมบัติใหม่ๆ และรองรับการเปลี่ยนแปลงและขยายตัวจากพฤติกรรมการใช้งานส่งผลให้ระบบมีความยืดหยุ่นสูง ส่งผลให้การทดสอบการใช้งานระบบของผู้ใช้นั้น ผู้ใช้สามารถเข้าถึงระบบในระดับ (ดี) มากที่สุด ซึ่งสอดคล้องกับแนวเจตนาของผู้คิดค้นรูปแบบสถาปัตยกรรมนี้ [1]

7. ข้อเสนอแนะ

ระบบที่พัฒนาขึ้นในงานวิจัยนี้ใช้รูปแบบที่เหมาะสมสำหรับระบบขนาดเล็กถึงขนาดกลาง จึงเลือกใช้รูปแบบ Model-View-Controller (MVC) อย่างไรก็ตาม หากระบบมีขนาดใหญ่และซับซ้อนมากขึ้น ควรพิจารณาเปลี่ยนไปใช้รูปแบบ Domain Model ซึ่งสามารถจัดการกับตรรกะธุรกิจที่ซับซ้อนในระบบขนาดใหญ่ได้ดีกว่า รวมทั้งการเลือกใช้ Angular รุ่นใหม่ที่พัฒนาด้วยภาษา TypeScript เนื่องจาก TypeScript มีความสามารถในการตรวจสอบข้อผิดพลาดได้ตั้งแต่ขั้นตอนการเขียนโค้ด (Static Typing) ช่วยลดข้อผิดพลาด เพิ่มประสิทธิภาพ และทำให้ระบบรองรับการเปลี่ยนแปลงในระยะยาวได้ดีขึ้น

เอกสารอ้างอิง

[1] M. Fowler, Patterns of enterprise application architecture. Boston: Addison-Wesley Professional Publishers, 2002.

[2] R. Martin and M. Martin, *Agile principles, patterns, and practices in C#*. London: Pearson Education Publishers, 2006.

[3] R. Conery, S. Hanselman, P. Haack, and S. Guthrie, Microsoft application architecture guide 2nd edition (pattern & practices). California: Microsoft Press Publishers, 2009.

[4] P. Saint-Louis, M. C. Morency, and J. Lapalme, "Examination of explicit definitions of enterprise architecture," *International Journal of Engineering Business Management*, vol. 11, pp. 1–18, Jun. 2019.

[5] L. Halawi, R. McCarthy, and J. Farah, "Where we are with enterprise architecture," *Journal of Information Systems Applied Research*, vol. 12, no. 3, pp. 4–13, Dec. 2019.

[6] M. Fowler, "Who needs an architect?," *IEEE Software*, vol. 20, no. 5, pp. 11–13, Sep.–Oct. 2003.

[7] J. Nurmi, K. Penttinen, and V. Seppänen, "Examining enterprise architecture definitions: Implications from theory and practice," *Selected Papers of the IRIS*, no. 9, pp. 1–13, 2019.

[8] D. Venkatesan and S. Sridhar, "A rationale for the choice of enterprise architecture method and software technology in a software-driven enterprise," *International Journal of Business Information Systems*, vol. 32, no. 3, pp. 272–311, Oct. 2019.

[9] K. Svyatoslav, "TOGAF-based enterprise architecture practice: An exploratory case study," *Communications of the Association for Information Systems*, vol. 43, pp. 321–359, Sep. 2018.

[10] E. Atencio, M. Mancini, and G. Bustos, "Enterprise architecture approach for project-based organizations modeling, design, and analysis: An ontology-driven tool proposal," *Alexandria Engineering Journal*, vol. 98, pp. 312–327, Jul. 2024.

[11] C. V. Good, W. R. Merkel, and P. D. Kappa, Dictionary of education, 3rd ed. New York: McGraw-Hill Publishers, 1973.