KKU SCIENCE JOURNAL
Journal Home Page : https://ph01.tci-thaijo.org/index.php/KKUSciJ
Published by the Faculty of Science, Khon Kaen University, Thailand

ดีไซน์แพตเทิร์นเพื่อเพิ่มความปลอดภัย โดยการจัดเก็บรหัสผ่านที่เข้ารหัสด้วยฟังก์ชันแฮชหลายรูปแบบ

Design Patterns to Enhance Security by Storing Passwords Encryption using Multiple Hashing Functions

นฤพล สุวรรณวิจิตร[1] สมเกียรติ ช่อเหมือน[1*] และ วรเชษฐ์ อุทธา[1]

Naruapon Suwanwijit[1], Somkiat Chormuan[1*] and Worachet Uttha[1]

[1]สาขาวิชาวิศวกรรมซอฟต์แวร์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏนครปฐม จังหวัดนครปฐม 73000

[1]Software Engineering, Faculty of Science and Technology, Nakhon Pathom Rajabhat University, Nakhon Pathom, 73000, Thailand

บทคัดย่อ

การรักษาความปลอดภัยของข้อมูลส่วนบุคคลที่เก็บอยู่ในฐานข้อมูลของเว็บแอปพลิเคชันเป็นสิ่งที่สำคัญ ในปัจจุบันการสร้างความปลอดภัยให้กับข้อมูลส่วนบุคคลโดยเฉพาะรหัสผ่านมีการนำฟังก์ชันแฮชมาใช้ ซึ่งฟังก์ชันแฮชเป็นอัลกอริทึมทางคณิตศาสตร์ที่ใช้ในการเข้ารหัสข้อมูลแบบทางเดียวไม่สามารถถอดรหัสกลับเป็นข้อมูลต้นฉบับได้ แต่อาศัยการเปรียบเทียบในการตรวจสอบความถูกต้องของข้อมูล โดยทั่วไปแล้วแต่ละเว็บแอปพลิเคชันจะเลือกใช้ฟังก์ชันแฮชเพียงฟังก์ชันหรืออัลกอริทึมเดียวในการเข้ารหัสผ่านของผู้ใช้ ไม่ได้ออกแบบซอฟต์แวร์ให้มีความสามารถในการปรับเปลี่ยนอัลกอริทึมได้โดยง่าย ผู้วิจัยเห็นว่าดีไซน์แพตเทิร์นนั้นถูกนำมาใช้ในการออกแบบซอฟต์แวร์ที่ดี ซึ่ง Strategy Pattern เป็นหนึ่งในแพตเทิร์นที่สำคัญของดีไซน์แพตเทิร์นที่สามารถนำมาประยุกต์ใช้ในการออกแบบซอฟต์แวร์ ที่มีอัลกอริทึมให้เลือกหลากหลายและรองรับการปรับเปลี่ยนอัลกอริทึมได้อย่างอิสระ เพื่อให้เหมาะสมในแต่ละสถานะการณ์ ในงานวิจัยนี้ผู้วิจัยจึงมีความสนใจที่จะนำดีไซน์แพตเทิร์น ทั้ง Strategy pattern และ Factory method pattern มาประยุกต์ในออกแบบและพัฒนาซอฟต์แวร์ในส่วนของฟังก์ชันแฮชให้มีความหลากหลายในการเข้ารหัส และการเลือกสร้าง instance ของฟังก์ชันแฮช ผลจากการวิจัยพบว่าระบบมีความยืดหยุ่นในการเปลี่ยนแปลงและเพิ่มเติมฟังก์ชันแฮชใหม่ๆ เพื่อจัดการกับการจัดเก็บรหัสผ่านผู้ใช้ด้วยฟังก์ชันแฮชหลายรูปแบบ และยังทำให้รหัสผ่านมีความปลอดภัยมากยิ่งขึ้นโดยการเพิ่มซับซ้อนในการเจาะระบบเมื่อเทียบกับฟังก์ชันแฮชแบบเดียว

ABSTRACT

The protection of personal information stored in the database of a web application is critical. Currently, securing personal information, especially passwords, has been introduced with the hash function. The hash function is a mathematical algorithm used to encrypt data in a single way that cannot be easily decrypted into original data. However, comparisons are used to verify the correctness of the data. In general,

*Corresponding Author, E-mail: tko@webmail.npru.ac.th

Received: date: 6 March 2024 | Revised date: 15 May 2024 | Accepted date: 19 May 2024

when encrypting a user's password, each web application will select a unique hash function or algorithm. The software has not been designed to provide the ability to easily modify the algorithm. The researchers found that the design pattern was used in good software design, with the "Strategy Pattern" being one of the key patterns of design patterns that can be applied in software design with a wide range of algorithms to choose from and support the freely modified algorithm to suit each situation. In this research, the researchers were interested in applying design patterns, both "Strategy Pattern" and "Factory Method Pattern," in the design and development of software in the section of hash functions, providing a wide range of encryption and instance selection of the hash function. The research results indicated that the system was flexible in changing and adding new hash functions to handle user password storage with multiple hash functions and also made passwords much more secure by adding complexity to system penetration compared to a single hash function.

คำสำคัญ: การออกแบบซอฟต์แวร์ ดีไซน์แพตเทิร์น การพัฒนาซอฟต์แวร์ ความปลอดภัยเว็บ ฟังก์ชันแฮช
Keywords: Software Design, Design Patterns, Software Development, Web Security, Hash Function

บทนำ

โดยทั่วไปวิธีการจัดการรหัสผ่านด้วยฟังก์ชันแฮชเป็นที่ยอมรับในการรักษาความปลอดภัยของข้อมูลและเครือข่ายคอมพิวเตอร์ เนื่องจากสามารถเพิ่มความปลอดภัยในการเก็บรักษารหัสผ่าน ซึ่งเป็นวิธีการพื้นฐานที่ถูกใช้งานอย่างแพร่หลายในการรักษาความปลอดภัยของข้อมูลและระบบต่างๆ ทำให้ได้รหัสผ่านที่มีประสิทธิภาพในการป้องกัน สำหรับการจัดการรหัสผ่านที่ดี อาทิเช่น การใช้ฟังก์ชันแฮชในการจัดเก็บรหัสผ่าน การตั้งรหัสผ่านที่ซับซ้อน การเปลี่ยนรหัสผ่านเป็นประจำ และการไม่ใช้รหัสผ่านเดียวกันกับหลายบริการจึงมีจำเป็นต้องในการใช้งานในปัจจุบัน หากจัดการรหัสผ่านได้ไม่ดีพอ อาจส่งผลต่อการรักษาความมั่นคงปลอดภัยของสารสนเทศ ทั้ง 3 ด้านประกอบด้วย 1) ด้านความลับ (Confidentiality) 2) ด้านความถูกต้องสมบูรณ์ (Integrity) และ 3) ด้านความพร้อมใช้งาน (Availability) ดังนั้นจึงควรให้ความสำคัญกับการจัดเก็บรหัสผ่านที่ดี ซึ่งไม่ควรจัดเก็บในรูปแบบข้อความธรรมดา (Plain text) หรืออาจใช้วิธีการเข้ารหัสแบบย้อนกลับ ไม่ว่าจะเป็น การเข้ารหัสแบบสมมาตร (Symmetric encryption) หรือการเข้ารหัสแบบอสมมาตร (Asymmetric encryption) ซึ่งเป็นวิธีการเข้ารหัสแบบทางเดียว (One-way encryption) ที่ใช้การเข้ารหัสแบบแฮช (Hash value) ในการแปลงข้อมูลโดยใช้ฟังก์ชันแฮชที่ไม่สามารถถอดรหัสกลับไปยังรูปแบบเดิมได้ ช่วยให้รหัสผ่านมีความปลอดภัยเพิ่มขึ้น แต่ยังมีอีกหลายปัจจัยในการเลือกใช้ฟังก์ชันแฮช อาทิเช่น ความซับซ้อนของฟังก์ชันแฮช ประสิทธิภาพของฮาร์ดแวร์ ขนาดของข้อมูล การใช้งานฟังก์ชันแฮชทั้งแบบมี SALT และไม่มี SALT รวมถึงการปรับแต่งแฮชอัลกอริทึม เป็นต้น ในการนำฟังก์ชันแฮชแบบมี SALT มาใช้งานจะช่วยให้รหัสผ่านมีค่าแฮชที่แตกต่างกัน จนทำให้การสุ่มหรือเดารหัสผ่านทำได้ยากขึ้น ส่วนการใช้เทคนิคในการโจมตีรหัสผ่านที่ถูกแฮชด้วยการใช้ตารางที่เตรียมไว้ล่วงหน้า (Rainbow tables) จะไม่สามารถทำได้ แต่จำเป็นต้องจัดก็บทั้ง SALT และ Hash value และต้องใช้ระยะเวลาในการประมวลผลที่มากขึ้น แต่สามารถยอมรับได้เมื่อแลกกับความปลอดภัยที่เพิ่มขึ้นด้วยเช่นกัน โดยทั่วไปแล้วนิยมใช้ฟังก์ชันแฮชเดียวทั้งระบบเนื่องจากง่ายในการจัดการ

Ntantogian *et al*. (2019) แสดงให้เห็นถึงระบบการจัดการเนื้อหาแบบโอเพนซอร์สที่นิยม ได้ใช้การเข้ารหัสฟังก์ชันแฮชทั้งแบบ SHA512 BCRYPT MD5 SHA1 เป็นต้น และยังมีปัจจัยอื่นๆ ที่ทำให้ระบบมีความปลอดภัย เช่น การสุ่มข้อมูลเติมเข้าไปก่อนที่จะทำการแฮช (SALT) การกำหนดจำนวนรอบของขั้นตอนการแฮช และจำนวนขั้นต่ำของการตั้งค่ารหัสผ่าน ซึ่งสามารถช่วยเพิ่มประสิทธิภาพในการป้องกันการโจมตีได้เพิ่มขึ้น และ Roman (2019) ได้เสนอแนวทางการรักษา

ความปลอดภัยของรหัสผ่านด้วยการประยุกต์ใช้เทคนิคการเข้ารหัสที่มีประสิทธิภาพสูงร่วมกับกระบวนการสร้างข้อมูลสุ่มที่ซับซ้อน ซึ่งจะช่วยเพิ่มระดับความปลอดภัยและลดความเสี่ยงจากการถูกเจาะระบบ

การเลือกใช้ฟังก์ชันแฮช ควรคำนึงถึงความต้องการและบริบทของการใช้งาน ซึ่งในเวลานี้ฟังก์ชันแฮชถือว่ามีความปลอดภัย แต่อาจจะไม่ปลอดภัยในอนาคต ดังนั้นจึงควรมีการตรวจสอบและปรับปรุงอัลกอริทึมอย่างสม่ำเสมอ สำหรับการปรับปรุงอัลกอริทึมถือเป็นข้อจำกัดในการพัฒนาซอฟต์แวร์ โดยทั่วไปแล้วจะไม่ได้ออกแบบให้รองรับในการเปลี่ยนแปลงหรือปรับปรุงอัลกอริทึมในการเข้ารหัส แต่การนำดีไซน์แพตเทิร์นต่างๆ มาประยุกต์ใช้ในการออกแบบซอฟต์แวร์ จะช่วยให้ระบบสามารถปรับปรุงหรือขยายได้ง่ายขึ้น โดยเฉพาะปัญหาในการเปลี่ยนแปลงหรือปรับปรุงอัลกอริทึมของฟังก์ชันแฮช จากการศึกษางานวิจัย Rashidi (2012) ที่ได้นำเสนอเกี่ยวกับการนำ Strategy pattern มาใช้จัดการกับการเลือกอัลกอริทึมของฟังก์ชันแฮชแบบไดนามิก แต่ยังขาดการประยุกต์ใช้ที่ชัดเจน และอาจเกิดปัญหาเมื่อนำ Strategy pattern มาใช้งานจริงได้ ผู้วิจัยจึงมีแนวคิดในการประยุกต์ใช้ดีไซน์แพตเทิร์นอื่นๆ เข้ามาใช้ในการออกแบบซอฟต์แวร์ และนำเสนอรูปแบบการพัฒนาเว็บแอปพลิเคชันที่มีคุณสมบัติเชิงฟังก์ชันในแง่ของการบำรุงรักษาและความปลอดภัยของระบบ

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

1. การออกแบบซอฟต์แวร์ และหลักการออกแบบ

การออกแบบซอฟต์แวร์ เป็นกระบวนการในการสร้างแผนหรือแบบผังสำหรับโครงสร้างและการจัดระเบียบของระบบซอฟต์แวร์ ซึ่งจะเกี่ยวข้องกับการตัดสินใจเกี่ยวกับสถาปัตยกรรมโดยรวม โมดูล อินเทอร์เฟซ และอัลกอริทึมที่จะถูกใช้ในระบบ (Ramasamy *et al*., 2015)

การเขียนโปรแกรมเชิงวัตถุ (Object-oriented programming: OOP) เป็นวิธีการสำหรับการสร้างแบบจำลองและออกแบบระบบซอฟต์แวร์ ซึ่งประกอบด้วยแนวคิดพื้นฐานของการหุ้มห่อข้อมูล (Encapsulation) การทำให้ข้อมูลเป็นนามธรรม (Abstraction) การสืบทอด (Inheritance) และการหลายรูป (Polymorphism) วิธีการเหล่านี้ถือเป็นหลักการออกแบบ (Design principles) และ การใช้กฎของดีมิเตอร์ (Law of Demeter: LoD) เป็นกฎแบบง่ายๆ สำหรับการออกแบบระบบที่เน้นการเชื่อมโยงของวัตถุ (Zotos, 2007) และใช้ในการออกแบบโปรแกรมที่เน้นการลดความขึ้นต่อกัน (Dependencies) ของอ็อบเจกต์หรือคลาสต่างๆ ในระบบซอฟต์แวร์ ซึ่งถูกนำเสนอในรูปของ "Principle of Least Knowledge" ที่ควรปฏิบัติตามในการเขียนโค้ด โดยอ็อบเจกต์หนึ่งควรเรียกใช้เมธอดของอ็อบเจกต์อื่นที่ได้รับมาโดยตรงอย่างเดียว และไม่ควรไปเรียกใช้เมธอดของอ็อบเจกต์ที่เป็นผลลัพธ์จากการเรียกเมธอดของอ็อบเจกต์อื่น ซึ่งเป็นเป้าหมายของ LoD คือการสร้างระบบที่ยืดหยุ่น ง่ายต่อการดูแลรักษา และเป็นโมดูลาร์ โดยลดความขึ้นต่อกันของคลาสหรืออ็อบเจกต์ในระบบ

Watts (2023) กล่าวถึงหลักการออกแบบ SOLID ที่ได้รับความนิยมในการพัฒนาซอฟต์แวร์แบบวัตถุ (Object-oriented software development) ประกอบไปด้วย 5 ประการ ได้แก่ 1) หลักการความรับผิดชอบเดี่ยว (Single responsibility principle) 2) หลักการเปิด-ปิด (Open/Closed principle) 3) หลักการแทนที่ของลิสคอฟ (Liskov substitution principle) 4) หลักการแยกส่วนต่อประสาน (Interface segregation principle) และ 5) หลักการกลับด้านของการขึ้นต่อกัน (Dependency inversion principle) ซึ่งถูกนำไปใช้โดยวิศวกรซอฟต์แวร์อย่างแพร่หลายและเกิดประโยชน์ต่อนักพัฒนาซอฟต์แวร์ โดยนำหลักการออกแบบ SOLID ไปใช้พัฒนาระบบที่มีคุณสมบัติที่ดีขึ้นในด้านต่างๆ ทั้งการบำรุงรักษา การขยายความสามารถ การทดสอบ และการนำกลับมาใช้ใหม่ ช่วยให้การออกแบบการเชื่อมโยงของวัตถุ ลดการขึ้นต่อกันและเพิ่มความสามารถในการบำรุงรักษา (Osman and Ömer, 2018) รวมถึงการเกาะติดและการเชื่อมโยง (Cohesion and coupling) ซึ่งมีบทบาทสำคัญในการกำหนดคุณภาพของระบบ ทั้งทางด้านความน่าเชื่อถือ ความสามารถในการบำรุงรักษา และความพร้อมใช้งาน (Jha *et al*., 2014) ได้นำดีไซน์แพตเทิร์นเข้ามาร่วมในการออกแบบที่ดีและสามารถ

เข้ากันได้ ทำให้นักพัฒนานำความรู้และประสบการณ์จากบุคคลอื่นมาใช้ในการแก้ไขปัญหาด้านการออกแบบที่คล้ายคลึงกันได้ (Bijlsma *et al*., 2022)

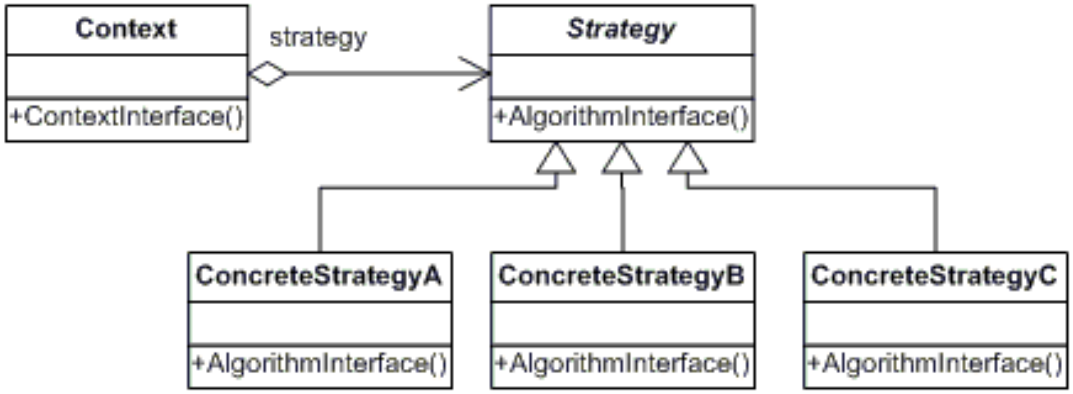
2. ดีไซน์แพตเทิร์น

ดีไซน์แพตเทิร์น (Design pattern) เป็นการแก้ปัญหาที่มีรูปแบบหรือแบบแผนของการออกแบบซอฟต์แวร์ ซึ่งเป็นวิธีที่เรียบง่ายและสามารถปรับใช้ได้กับปัญหาที่เกิดขึ้นบ่อยๆ ในการพัฒนาซอฟต์แวร์ โดยการนำดีไซน์แพตเทิร์นไปใช้ออกแบบ ช่วยให้อ่านโค้ดได้ง่าย และสามารถแก้ไขหรือขยายงานได้อย่างยืดหยุ่น ถือเป็นวิธีการแก้ปัญหาที่สามารถนำแบบแผนเดิมมาช่วยในการออกแบบซอฟต์แวร์ที่ได้รับการพิสูจน์ว่ามีประสิทธิภาพในทางปฏิบัติ วิธีในการแก้ไขปัญหาในการออกแบบด้วยดีไซน์แพตเทิร์นที่เกิดขึ้นบ่อยๆ ถือเป็นการปรับปรุงคุณภาพและความสามารถในการบำรุงรักษาระบบซอฟต์แวร์ให้ดีขึ้นได้ ในการจัดระเบียบโค้ด ทำให้โค้ดมีโครงสร้างที่ชัดเจน ยืดหยุ่น และเข้าใจง่าย (Bijlsma *et al*., 2022) ทำให้การพัฒนาซอฟต์แวร์ตามดีไซน์แพตเทิร์นที่ออกแบบจำลองโครงสร้างผ่านแผนภาพ UML ที่สามารถแปลงเป็นชุดคำสั่งด้วยการเขียนโปรแกรมเชิงวัตถุได้ตามความต้องการ ช่วยลดต้นทุนในการพัฒนาซอฟต์แวร์และบำรุงรักษา เกิดความน่าเชื่อถือในกระบวนการผลิตซอฟต์แวร์ (Al-Hawari, 2022) ได้ในระยะเวลาที่สั้นลง ถือเป็นการเรียนรู้ในการออกแบบซอฟต์แวร์จากประสบการณ์ของผู้เชี่ยวชาญที่มีแบบแผนชัดเจน (Ramasamy *et al*., 2015)

ดีไซน์แพตเทิร์นที่นำมาใช้ในงานวิจัยนี้ ประกอบด้วย Strategy pattern และ Factory method pattern ซึ่งมีรายละเอียด โดยสังเขปดังนี้

2.1 Strategy Pattern

ดีไซน์แพตเทิร์นที่นำกลยุทธมาใช้ในการสร้างอ็อบเจกต์แทนที่อัลกอริทึมที่ต่างกัน ทำให้สามารถเปลี่ยนแปลงอัลกอริทึมหรือการดำเนินการทำงานได้โดยไม่ต้องแก้ไขโค้ดที่เรียกใช้งานอัลกอริทึมหรือการดำเนินการได้ดังรูปที่ 1



รูปที่ 1 UML class diagram for Strategy pattern (Khosravi and Guéhéneuc, 2017)

จากรูปที่ 1 แสดงแผนภาพคลาสไดอะแกรมของ Strategy design pattern ซึ่งมีการกำหนดเมธอดภายใน Interface ของคลาส Strategy ซึ่งทำการ Implement อัลกอลิทึมของเมธอดที่มีความต้องการแตกต่างกันของ AlgorithmInterface()

2.2 Factory Method Pattern

ดีไซน์แพตเทิร์นที่ใช้แยกส่วนของการสร้างอ็อบเจกต์ออกจากคลาสที่เรียกใช้อ็อบเจกต์ โดยมีเมธอดภายในคลาสทำหน้าที่สร้างอ็อบเจกต์ใหม่ ทำให้การสร้างอ็อบเจกต์มีความซับซ้อน แต่สามารถซ่อนไว้ภายในเมธอดที่ทำหน้าที่สร้างอ็อบเจกต์และถูกเรียกใช้งานได้โดยที่ไม่ต้องทราบรายละเอียด สามารถใช้งานได้อย่างแพร่หลายและมีประสิทธิภาพ เช่นไลบรารีของภาษา Java ช่วยให้การพัฒนาคลาสที่มีการผูกแน่นลดลง ซึ่งง่ายต่อการดูแลรักษาและปรับปรุงชุดคำสั่ง นอกจากนี้ยังทำให้

การออกแบบเกิดความยืดหยุ่นและสามารถเพิ่มผลิตภัณฑ์ใหม่เข้าไปในโปรแกรมโดยไม่ทำลายชุดคำสั่งเดิม ด้วยการเพิ่มคลาสย่อย (Subclasses) จำนวนมาก และกล่าวถึงข้อดีและข้อเสียของดีไซน์แพตเทิร์นในการประยุกต์ใช้งาน (Temaj, 2020)

งานวิจัยนี้ได้นำดีไซต์แพทเทิร์นทั้ง 2 แบบมาประยุกต์ในการออกแบบกระบวนการจัดการรหัสผ่านโดยใช้ฟังชันแฮชหลากหลายรูปแบบ ซึ่งช่วยให้การแก้ไขปรับปรุงซอฟต์แวร์มีความยืดหยุ่นสำหรับการรักษาความปลอดภัยด้วยรหัสผ่าน

3. ฟังก์ชันแฮช

ฟังก์ชันแฮช (Hash function) คือฟังก์ชันที่รับข้อมูลนำเข้าที่เป็นข้อความธรรมดา และสร้างข้อความของตัวอักษรขนาดคงที่ (ค่าแฮช) เป็นข้อมูลส่งออก ถูกใช้โดยทั่วไปในระบบการเข้ารหัสรหัสผ่านเพื่อแปลงรหัสผ่านเป็นรูปแบบที่ปลอดภัยขึ้นสำหรับการจัดเก็บและการเปรียบเทียบ ในบริบทของการจัดเก็บรหัสผ่านประกอบด้วยสามพารามิเตอร์ ประกอบด้วย ฟังก์ชันแฮช การวนซ้ำ และ SALT เป็นพารามิเตอร์หลัก จำนวนการวนซ้ำระบุจำนวนของการดำเนินการฟังก์ชันแฮชต่อเนื่องเพื่อคำนวณค่าแฮชถูกใช้เพื่อลดความเร็วของการโจมตีรหัสผ่าน ฟังก์ชันแฮชบางตัว เช่น PBKDF2 BCRYPT SCRYPT และ Argon2 ใช้การวนซ้ำเป็นค่าเริ่มต้นเพื่อเพิ่มเวลาในการคำนวณและเพิ่มความปลอดภัย ซึ่ง SCRYPT และ Argon2 เป็นฟังก์ชันแฮชที่ใช้หน่วยความจำอย่างมาก ทำให้การใช้งานมีต้นทุนสูงขึ้นและจำกัดจำนวนของการทำงานแบบขนานที่ผู้โจมตีสามารถใช้ได้ (Ntantogian *et al*., 2019)

การประยุกต์ใช้งานฟังก์ชันแฮชมีการใช้งานที่หลากหลาย เช่น การตรวจสอบความถูกต้องของข้อมูล (Data integrity) การเข้ารหัสของรหัสผ่าน และลายเซ็นอิเล็กทรอนิกส์ เป็นต้น ฟังก์ชันแฮชที่นิยม เช่น MD5 SHA-1 SHA-256 SHA-512 และอื่นๆ ในการใช้งานควรเลือกใช้อัลกอริทึมที่มีความปลอดภัยตามความจำเป็นและประเภทของงาน

BCRYPT เป็นฟังก์ชันแฮชเข้ารหัสที่มีความซับซ้อนอย่างมาก มีการนำไปใช้สำหรับเก็บรหัสผ่าน และสามารถกำหนดค่าที่ใช้เพื่อกำหนดระดับความซับซ้อนของการแฮชพาสเวิร์ด (Cost) ได้ ยิ่งค่า Cost ที่สูง การคำนวณจะยิ่งยุ่งยากและใช้เวลามากขึ้น ซึ่งค่า Cost สามารถปรับได้ให้เหมาะสมกับบริบทของระบบ ทำให้มีความปลอดภัยมากกว่าฟังก์ชันแฮชอื่นๆ เพราะการโจมตีด้วย Brute-Force Attack ต้องใช้เวลามากขึ้น

4. งานวิจัยที่เกี่ยวข้อง

งานวิจัย Verma and Liu (2003) ประยุกต์ใช้ดีไซน์แพตเทิร์น ในการนำมาปรับปรุงโค้ดเก่า (Legacy code) กรณีศึกษาซอฟต์แวร์ที่ใช้ในการสร้างโครงข่าย (Mesh generation software) พบว่าการนำดีไซน์แพตเทิร์น ช่วยเพิ่มความยืดหยุ่น (Flexibility) ความสามารถในการขยาย (Extensibility) และความง่ายในการดูแลรักษา (Maintainability) โดยที่ไม่ส่งผลกระทบต่อประสิทธิภาพการทำงานของซอฟต์แวร์

งานวิจัย Vaghela and Pithva (2016) เน้นที่เฟรมเวิร์กที่ออกแบบอิงตามซอฟต์แวร์ดีไซน์แพตเทิร์น สำหรับโมดูลการตรวจสอบความถูกต้อง โดยใช้เฟรมเวิร์กการเข้าสู่ระบบที่อิงตามดีไซน์แพตเทิร์น นักพัฒนาสามารถจัดการกับความต้องการในอนาคตได้ง่าย มีความยืดหยุ่นและความสามารถในการขยายตัวของแอปพลิเคชัน

ดีไซน์แพตเทิร์นถูกประเมินจากประสิทธิภาพในการแก้ปัญหาเฉพาะ และการปรับปรุงคุณภาพซอฟต์แวร์ และถูกประเมินในเชิงของความยืดหยุ่น การนำกลับมาใช้ซ้ำ ความสามารถในการบำรุงรักษา และการขยายตัว การประเมินรูปแบบการออกแบบเกี่ยวข้องกับการวิเคราะห์ผลกระทบของดีไซน์แพตเทิร์น ต่อความง่ายในการอ่านชุดคำสั่ง ความเป็นโมดูลและการขยายตัว รวมไปถึงการพิจารณาความสามารถในการใช้ชุดคำสั่งซ้ำและลดเวลาในการพัฒนา (Bijlsma *et al*., 2022)

หลักการ Open/Close กำหนดให้ คลาส (Classes) โมดูล (Modules) และฟังก์ชัน (Functions) ควรจะเปิดให้สามารถขยายความสามารถได้แต่ปิดไม่ให้แก้ไข องค์ประกอบหนึ่งๆ สามารถให้พฤติกรรมถูกขยายได้โดยไม่ต้องแก้ไขซอร์สโค้ดเดิม หรือคลาสควรจะสามารถขยายได้ง่ายๆ โดยไม่ต้องแก้ไขคลาสเอง ในกรณีที่ข้อกำหนดเปลี่ยนแปลง สามารถขยายพฤติกรรมของโมดูลเหล่านั้นโดยการเพิ่มโค้ดใหม่ ไม่ใช่โดยการเปลี่ยนแปลงโค้ดเก่าที่ทำงานได้แล้ว (Osman and Ömer, 2018)

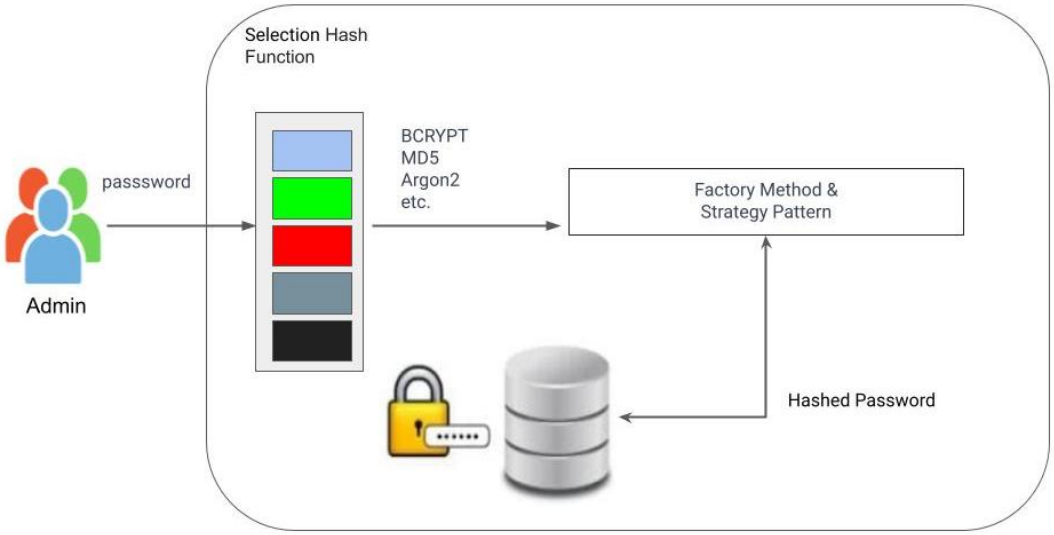
งานวิจัย Shri and Ravikumar (2018) ได้แสดงให้เห็นถึงความจำเป็นของความปลอดภัยในระดับแอปพลิเคชันในสภาพแวดล้อมของคลาวด์สาธารณะ และเสนอการใช้ฟังก์ชันแฮช BCRYPT เพื่อเพิ่มความปลอดภัย ซึ่ง BCRYPT มีการเข้ารหัสที่มีความซับซ้อนถูกนำไปใช้ในการเก็บรหัสผ่าน โดยที่ยังมีความยืดหยุ่นรองรับอนาคตและเพิ่มความปลอดภัย ด้วยการอนุญาตให้ผู้ใช้เลือกจำนวนรอบสำหรับการสร้างข้อมูลสุ่ม

ในงานวิจัยการเพิ่มแฮชฟังก์ชันชนิดใหม่เป็นการนำหลักการ SOLID มาใช้ เช่น หลักการ Open/Close ในการรองรับแฮชฟังก์ชันใหม่โดยที่ไม่ต้องแก้ไขคลาสฟังก์ชันแฮชเก่าที่มีอยู่แล้ว และประเมินประสิทธิภาพโดยทำการทดสอบฟังก์ชันการทำงานทั้งในรูปแบบการทดสอบระดับหน่วย (Unit testing) และการทดสอบระบบ (System testing) ในการทดลองผู้วิจัยได้กำหนดการเข้ารหัสผ่าน 2 แบบ คือ Type1 และ Type2 ซึ่ง Type1 คือ การเข้ารหัสแบบ BCRYPT และ Type2 การเข้ารหัสแบบ MD5 และได้ดำเนินการเพิ่ม Type3 ซึ่งมีการเข้ารหัสแบบ Argon2 เข้าไปเพื่อให้รองรับการเข้ารหัสรูปแบบใหม่

วิธีการดำเนินการวิจัย

1. แนวคิดและขั้นตอนในการดำเนินการวิจัย

ในงานวิจัยการได้ดำเนินการตาม SDLC Model ในการพัฒนาระบบ โมเดล SDLC เป็นแนวทางอย่างเป็นระบบในการพัฒนาซอฟต์แวร์จากเริ่มต้นไปจนจบ มีการรวมกันของหลายๆ ขั้นตอนที่แตกต่างกัน เช่น วิเคราะห์ ออกแบบ พัฒนา ทดสอบและบำรุงรักษาระบบ เพื่อให้มั่นใจว่าการพัฒนาและการใช้งานประสบความสำเร็จ ซึ่งโมเดล SDLC อาจแตกต่างกันไปตามโครงการและองค์กร (Patel, 2023)



รูปที่ 3 แนวคิดในการจัดเก็บรหัสผ่านให้มีความปลอดภัยด้วยการใช้ Design pattern

จากรูปที่ 3 เป็นการเลือกฟังก์ชันแฮชที่เหมาะสมและการใช้ดีไซน์แพตเทิร์นเพื่อสร้างระบบที่มีความยืดหยุ่นและปลอดภัยในการจัดเก็บรหัสผ่านที่มีความหลากหลายของแฮชฟังก์ชัน โดยที่ Factory Method Pattern ใช้ในการเลือกสร้างวัตถุของประเภทของแฮชฟังก์ชันจากพารามิเตอร์ที่ผู้ใช้เลือก ส่วน Strategy pattern ใช้ในการเข้ารหัสหรือตรวจสอบความถูกต้องตามแฮชฟังก์ชันที่ได้เลือกไว้

2. รายละเอียดการดำเนินการวิจัย

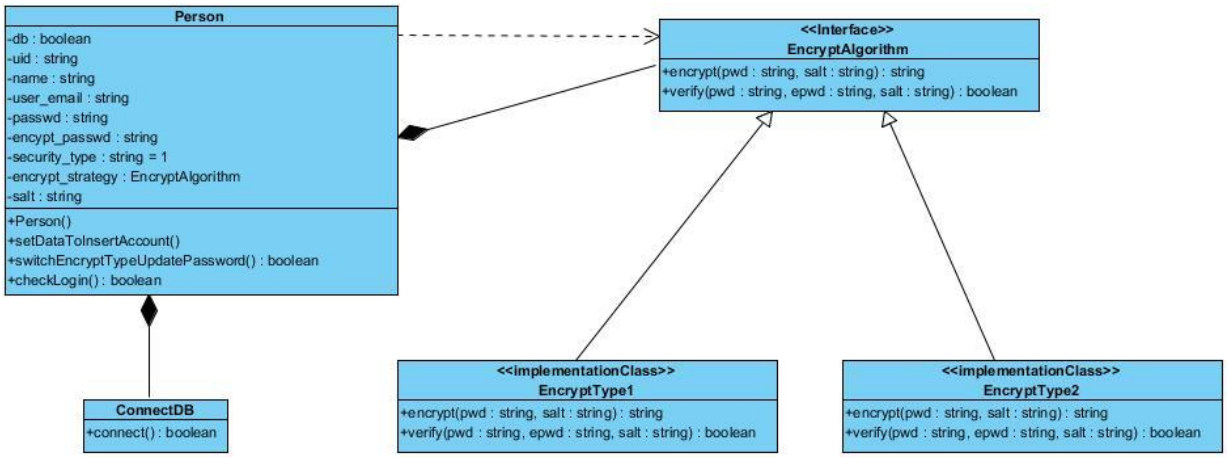
2.1 การวิเคราะห์ความต้องการของระบบ

ตารางที่ 1 ความต้องการทางด้านฟังก์ชัน

Function requirement ID	Description	Use-Case
FR-01	เพิ่มผู้ใช้และกำหนดประเภทความปลอดภัยของ Hash	Add user and Select security type
FR-02	ตรวจสอบผู้ใช้และรหัสผ่าน	Check user validate
FR-03	เปลี่ยนรหัสผ่านและประเภทความปลอดภัยของ Hash	Change password and Security

2.2 การออกแบบคลาส ฐานข้อมูล และออกแบบหน้าเว็บ

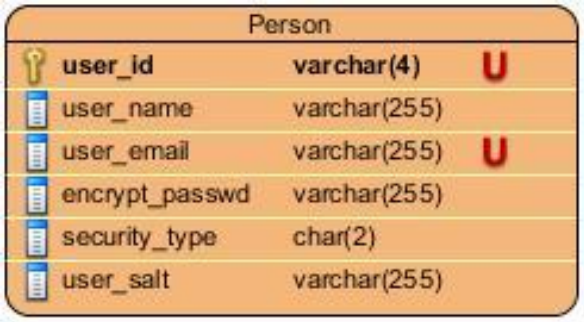
ผู้วิจัยได้ดำเนินการออกแบบระบบด้วยคลาสไดอะแกรม ER-Diagram ซีเควนไดอะแกรม และออกแบบหน้าเว็บเบื้องต้นด้วย Wireframe โดยมีรายละเอียดของคลาส ดังรูปที่ 4



รูปที่ 4 คลาสไดอะแกรมของระบบ

2.3 การออกแบบฐานข้อมูล

การออกแบบฐานข้อมูลได้สร้าง ER-Diagram เพื่อใช้ในการจัดเก็บข้อมูลผู้ใช้ ดังรูปที่ 5



รูปที่ 5 แผนภาพ ER-Diagram

2.4 การพัฒนาซอฟต์แวร์

2.4.1 การติดตั้งเครื่องมือและสภาพแวดล้อม ผู้วิจัยได้ทำการติดตั้งสภาพแวดล้อมต่างๆ ภายใต้ระบบปฏิบัติการ Windows 10 ตามที่กำหนดไว้ในขั้นตอนของการวิจัย ประกอบด้วย ภาษา PHP ภาษา Python เว็บเซิร์ฟเวอร์ Apache เครื่องมือในการเขียนชุดคำสั่ง Visual studio code เครื่องมือในการทดสอบระดับหน่วย PHPUnit โปรแกรมการจัดการฐานข้อมูล MySQL Database และ เครื่องมือในการทดสอบระบบ (Selenium)

2.4.2 การพัฒนาฐานข้อมูล โครงสร้างของตารางในฐานข้อมูล ที่มีรายละเอียดในการจัดเก็บข้อมูล ประกอบด้วย รหัสผู้ใช้ ชื่อผู้ใช้ อีเมล์ รหัสผ่านที่มีการเข้ารหัส ประเภทของแฮชฟังก์ชันซึ่งแทนด้วยตัวเลขทั้งนี้เพื่อความปลอดภัย และ ค่า SALT เพื่อใช้ประกอบในการเข้ารหัสและการตรวจสอบความถูกต้อง ซึ่ง SALT ที่จัดเก็บมีลักษณะเป็นแบบพลวัตร (Dynamic) มีการสุ่มในการสร้างเข้ารหัสในครั้งแรกและมีการสุ่มขึ้นมาใหม่ทุกครั้งที่มีการเปลี่ยนแปลงวิธีการเข้ารหัสของผู้ใช้

user_id ▲ 1	user_name	user_email	encrypt_passwd	security_type	user_salt
1111	user1	user1@sciencereserach.com	$2y$10$qQZUcmaLxfrwI8EILmUGA.o443th1pPRlPaHyvPYTe4...	1	79623406
2222	user2	user2@sciencereserach.com	$2y$10$5H2B.x3HS1QYcihMrZVZJON6OYGsn0FtjFauZYCqHgm...	1	0735a4f2
3333	user3	user3@sciencereserach.com	$2y$10$45d6x1uJTWgazqOgvBHlp.HXz0B5ZeQ1BEXnizjmq5B...	1	3fd529a3
4444	user4	user4@sciencereserach.com	c6bcdbca094a9329db181e446c147a2a	2	a29c7277
5555	user5	user5@sciencereserach.com	$2y$10$7Sj98KSBME55BkSnUjATIOA/oS6bX9m0wnTn.FClchF...	1	0075957f
6666	user6	user6@sciencereserach.com	$2y$10$/q1HFWEZNw.qynzsYnhuV.8ytTIZwT5.E.Z8r1Y3LjP...	1	10e1b1de

รูปที่ 6 ข้อมูลในฐานข้อมูล

จากรูปที่ 6 จะเห็นได้ว่าในส่วนของรหัสผ่านมีการผ่านแฮชฟังก์ชันก่อนการจัดเก็บ และ SALT ที่ถูกสร้างขึ้นนั้นไม่ได้มีความซ้ำซ้อนกัน ทำให้กรณีที่ผู้ใช้ตั้งรหัสผ่านเดียวกันและใช้แฮชฟังก์ชันเดียวกัน แต่ผลลัพธ์ของรหัสที่มีการเข้ารหัสจะไม่เหมือนกัน ซึ่งช่วยให้ระบบมีความปลอดภัยเพิ่มขึ้นอีกด้วย

2.4.3 สร้างคลาสด้วยภาษา PHP ตามที่ออกแบบ คลาสที่ในส่วนของการกำหนดฟังก์ชันแฮช ประกอบด้วย EncryptAlgorithm และ EncryptType1 และ EncryptType2 ทั้งสองประเภท มีการใช้แฮชฟังก์ชันที่แตกต่างกัน โดย EncryptType1 มีการใช้ฟังก์ชันแฮชแบบ BCRYPTส่วน EncryptType2 มีการใช้ฟังก์ชันแฮชแบบ MD5 รายละเอียดของชุดคำสั่งดังรายการชุดคำสั่งที่ 1 - 2

```
interface EncryptAlgorithm{
    public function encrypt($pwd,$salt);
    public function verify($pwd,$epwd,$salt);
}
```

รายการชุดคำสั่งที่ 1 Interface EncryptAlgorithm

```
class EncryptType1 implements EncryptAlgorithm{
    public function encrypt($pwd,$salt){
        $pwd_encrypt = password_hash($pwd.$salt, PASSWORD_DEFAULT);
        return $pwd_encrypt;
    }
    public function verify($pwd,$epwd,$salt){
        if (password_verify($pwd.$salt,$epwd)){
            return true;
         } else {
            return false;
         }
    }
}
```

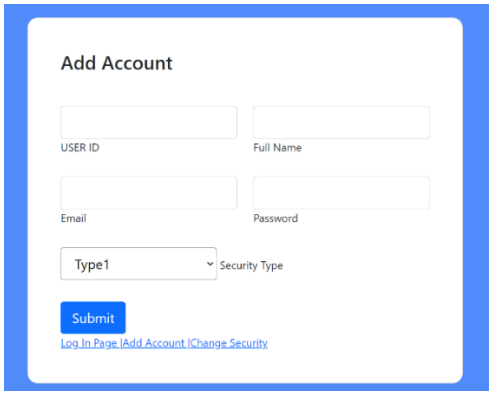
รายการชุดคำสั่งที่ 2 Class EncryptType1

```
Function verifyEncrypt
    Get the security type
    If the security type is "1"
        Set encryption strategy to EncryptType1
        Verify using EncryptType1 with password, encrypted password, and salt
        Return the verification result
    Else if the security type is "2"
        Set encryption strategy to EncryptType2
        Verify using EncryptType2 with password, encrypted password, and salt
        Return the verification result
    Else
        Return false (indicating failure or unsupported security type)
End Function
```

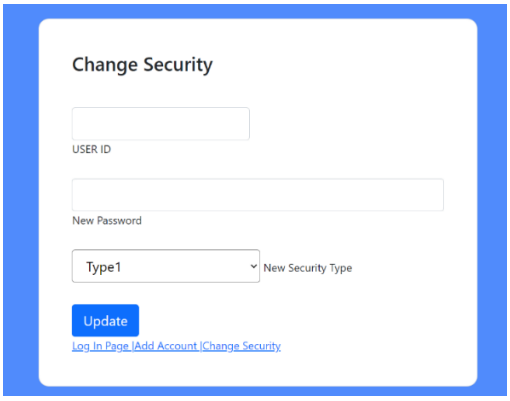
รหัสเทียมที่ 1 การกำหนดกลยุทธ์การเข้ารหัสด้วย Factory Method

จากรหัสเทียมที่ 1 เป็นการอธิบายการทำงานของฟังก์ชันที่ใช้ในการตรวจสอบและยืนยันการเข้ารหัสรหัสผ่าน โดยมีการใช้กลยุทธ์การเข้ารหัสที่แตกต่างกันตามประเภทของความปลอดภัยที่ระบุไว้ ฟังก์ชันนี้จะตรวจสอบประเภทความปลอดภัยและตามด้วยการตรวจสอบกลยุทธ์การเข้ารหัส (EncryptType1 หรือ EncryptType2) จากนั้นจะทำการยืนยันหรือตรวจสอบการเข้ารหัสรหัสผ่านโดยใช้ข้อมูลรหัสผ่าน รหัสผ่านที่เข้ารหัสแล้วและ SALT หากไม่มีการระบุประเภทความปลอดภัยที่รองรับฟังก์ชันนี้จะคืนค่า false หมายความว่า การตรวจสอบไม่สำเร็จหรือไม่รองรับประเภทความปลอดภัยที่กำหนด

2.4.4 สร้างหน้าเว็บไซต์ตามที่ได้ออกแบบ หน้าจอต่างๆ ที่ได้พัฒนาตาม Wireframes ที่ได้ออกแบบไว้ตามรูปที่ 7 และ รูปที่ 8



รูปที่ 7 หน้าเพิ่มผู้ใช้



รูปที่ 8 หน้าเปลี่ยนแปลงประเภทของความปลอดภัย

2.4.5 การดำเนินการทดสอบระดับหน่วย หลังจากที่ได้ดำเนินการพัฒนาตามดีไซน์แพตเทิร์น แล้วจึงดำเนินการทดสอบการทำงานระดับหน่วยโดยใช้เฟรมเวิร์ก PHPUnit ภายในคลาส Person ประกอบด้วยฟังก์ชันดังต่อไปนี้

1. ฟังก์ชันตรวจสอบผู้ใช้ (checkLogin)
2. ฟังก์ชันการเพิ่มผู้ใช้ (setDataToInsertAccount)
3. ฟังก์ชันการเปลี่ยนรหัสผ่านและความปลอดภัย (switchEncryptTypeUpdatePassword)

2.4.6 การดำเนินการปรับเปลี่ยนอัลกอริทึม ในการเพิ่มฟังก์ชันแฮชใหม่ๆ ทำได้โดยการสร้างคลาส EncryptionType ที่มีการอิมพลีเมนต์อินเตอร์เฟซตามที่กำหนดไว้ใน EncryptAlgorithm ประกอบด้วยการเข้ารหัส (encrypt) และตรวจสอบความถูกต้องของรหัส (verify) เพื่อให้ระบบรองรับการเข้ารหัสโดยฟังก์ชันแฮชใหม่ได้ตามแบบแผน โดยแก้ไขเพิ่มเติมในส่วนของ Factory method ให้เลือกสร้างอินสแตนท์ของฟังก์ชันแฮชใหม่ได้ และส่วนของ UI นั้น ทำการปรับแก้ให้สามารถเลือกประเภทการเข้ารหัสแบบใหม่ ผู้วิจัยได้ดำเนินการทดสอบซ้ำ (Sanity testing) ซึ่งเป็นการทดสอบในส่วนที่มีการแก้ไขในส่วนของการทดสอบระดับหน่วย และการทดสอบระบบ ส่วนของการทดสอบระบบได้ทดสอบฟังก์ชันในการเพิ่มผู้ใช้ ตรวจสอบสิทธิ์ และการปรับเปลี่ยนความปลอดภัย เพื่อให้เกิดความเชื่อมั่นในการใช้งานระบบ ในการวิจัยได้ทำการสร้างฟังก์ชันแฮชใหม่ (EncryptType3) ที่เป็นฟังก์ชันแฮชแบบ Argon2 ดังรายการชุดคำสั่งที่ 6

```
class EncryptType3 implements EncryptAlgorithm{
    public function encrypt($pwd,$salt){
        $pwd_encrypt = password_hash($pwd.$salt, PASSWORD_ARGON2I);
        return $pwd_encrypt;
    }
    public function verify($pwd,$epwd,$salt){
        if (password_verify($pwd.$salt,$epwd)){
            return true;
         } else {
            return false;
         }
    }
}
```

รายการชุดคำสั่งที่ 6 Class EncryptType3 ฟังก์ชันแฮชแบบ Argon2

จาก Class EncryptType3 เป็นการดำเนินการสร้างรูปแบบการเข้ารหัสและการตรวจสอบความถูกต้อง โดยใช้แฮชฟังก์ชันในรูปแบบของ Argon2 ทำให้การเพิ่มฟังก์ชันแฮชและเลือกใช้การเข้ารหัสแบบใหม่ได้ในระบบ

ผลการวิจัยและวิจารณ์ผล

ผู้วิจัยได้ดำเนินการสร้างระบบและเพิ่มผู้ใช้เข้าไปในระบบทีละ 6 รายการ ตามชนิดของการเข้ารหัสทั้ง 3 และในกรณีที่ 4 ได้ดำเนินการเพิ่มผู้ใช้ที่มีการเข้ารหัสแต่ละประเภทอย่างละ 2 รายการ และดำเนินการทดสอบในการเข้าระบบโดยดำเนินการเขียนสคริปต์การทดสอบแบบอัตโนมัติด้วย Selenium โดยใช้ภาษา Python ในการทดสอบระบบ ตามรูปที่ 9 หน้าตรวจสอบผู้ใช้และรหัสผ่าน และได้บันทึกเวลาที่ใช้ในการเข้าระบบจำนวน 3 ครั้งและหาเฉลี่ยของเวลาที่ใช้ในการประมวลผล

ตารางที่ 2 การประมวลผลการทดสอบระบบในการเข้าระบบ

Encryption Type	เวลาประมวลผล (วินาที)			
(Run 6 tests)	ครั้งที่ 1	ครั้งที่ 2	ครั้งที่ 3	ค่าเฉลี่ย
1. EncryptType1	45.456	45.587	45.966	45.669
2. EncryptType2	44.948	45.235	45.351	45.178
3. EncryptType3	47.980	47.612	47.768	47.786
4. EncryptType1,2,3	46.215	46.008	46.522	46.248

จากตารางที่ 2 พบว่า การเข้าระบบด้วยการเข้ารหัสแบบ MD5 ใช้เวลาน้อยที่สุด คือ มีค่าเฉลี่ยที่ 45.178 วินาที และการเข้ารหัสแบบ Argon2 ใช้เวลามากที่สุด คือ มีค่าเฉลี่ยที่ 47.786 วินาที และการจัดเก็บรหัสผ่านที่มีความหลากหลายของแฮชฟังก์ชัน ค่าเฉลี่ยที่ 46.248 วินาที ทั้งนี้เพราะว่า MD5 มีความซับซ้อนในการเข้ารหัสน้อยกว่าแบบ BCRYPT และ ARGON2 ซึ่งมีความปลอดภัยมากกว่า แม้ว่า BCRYPT และ ARGON2 จะใช้เวลาในการทำงานมากกว่าแต่ก็สามารถเพิ่มความปลอดภัยให้กับรหัสผ่านได้ดีขึ้น เนื่องจากผู้ที่ไม่หวังดีก็ต้องใช้เวลาในการโจมตีมากขึ้นตามไปด้วยเช่นกัน

ตามหลักการออกแบบที่ดีทำให้บำรุงรักษาซอฟต์แวร์ได้ง่าย โดยไม่ส่งผลกระทบในการแก้ไขหลายๆ จุดของโปรแกรม และซึ่งเป็นไปตามหลักการ SOLID ดังนี้ หลักการ Single Responsibility ในคลาส EncryptType ต่างๆ จะทำหน้าที่ในการเข้ารหัส และตรวจสอบความถูกต้องของรหัสผ่านเพียงแฮชฟังก์ชันเดียว หลักการ Open-Closed มีความยืดหยุ่นและรองรับการเพิ่มแฮชฟังก์ชันประเภทอื่นๆ ได้ในอนาคต โดยการสร้างคลาสที่นำ Interface encrypt algorithm โดยที่ไม่ต้องไปแก้ไขรายละเอียดของชุดคำสั่งเดิมในคลาสก่อนหน้า ซึ่งเป็นไปตามหลักการที่ว่า โมดูลหรือฟังก์ชันควรต้องเปิดให้ขยายได้แต่ปิดสำหรับการแก้ไข และหลักการ Interface segregation จะเห็นได้ว่าทั้ง 2 เมธอด เป็นเมธอดที่สำคัญในเข้ารหัสและตรวจสอบความถูกต้องรหัสผ่าน จากผลการวิจัยเห็นได้ว่าการออกแบบโดยใช้ Factory method และ Strategy pattern ทำให้นักพัฒนาสามารถเพิ่มเติมฟังก์ชันเข้ารหัสและตรวจสอบความถูกต้องได้ง่าย และสอดคล้องกับวิธีการแก้ไขปัญหาที่ดีไซน์แพตเทิร์นนั้นถูกคิดขึ้นมา มีการยึดเหนี่ยวระหว่างคลาส Person และ อินเตอร์เฟส EncryptAlgorithm ที่เป็นแบบ Loose coupling ในแง่ของความปลอดภัยจะเห็นได้ว่าในฐานข้อมูลของระบบไม่ได้จัดเก็บรหัสผ่านในรูปแบบของแฮชฟังก์ชันเพียงรูปแบบเดียว ซึ่งจะทำให้เกิดความซับซ้อนในการที่จะต้องใช้เวลาและความพยายามในการถอดรหัสมากขึ้นกว่าการใช้แฮชเพียงแบบเดียว

จากการวิจัยที่ได้ดำเนินการนั้นเป็นไปตามวัตถุประสงค์ของการวิจัยในแง่ของการบำรุงรักษาระบบ เช่น การเพิ่มฟังก์ชันแฮชใหม่ๆ และ Security การปรับเปลี่ยนรูปแบบของฟังก์ชันแฮชได้ตลอดเวลา และความปลอดภัยที่เพิ่มขึ้นในแง่ของการเพิ่มความซับซ้อนให้กับระบบ

สรุปผลการวิจัย

ดีไซน์แพตเทิร์นที่นำมาใช้ในงานวิจัยนี้เพื่อให้ซอฟต์แวร์มีความง่ายในการดูแลรักษาระบบ เช่น การเปลี่ยนฟังก์ชันแฮชและรองรับฟังก์ชันแฮชใหม่ๆ ที่จะนำมาใช้ สามารถทำความเข้าใจในการทำงานของชุดคำสั่งได้ง่ายขึ้น เป็นต้น และการเพิ่มความปลอดภัย มุ่งเน้นไปที่การรักษาความปลอดภัยที่มีการเข้ารหัสโดยการใช้ฟังก์ชันแฮชหลายรูปแบบเป็นหลัก แต่การทำให้ระบบมีความปลอดภัย ยังมีประเด็นอื่นๆ ที่ต้องนำมาประยุกต์ใช้เพื่อให้เกิดความปลอดภัยเพิ่มมากขึ้น เช่น การกำหนดความยาวขั้นต่ำของรหัสผ่าน การตั้งรหัสผ่านให้มีความซับซ้อน การใช้ค่า SALT ที่มีจำนวนบิตที่สูงขึ้น กระบวนการในการเข้ารหัสให้มีความซับซ้อนขึ้น หรือการใช้การประมวลผลภาพมาใช้ในการป้องกัน เช่น การให้ใส่ตัวอักษรจากภาพที่สุ่มออกมา หรือการเลือกรูปภาพจากข้อความที่กำหนดก็ยิ่งทำให้ระบบมีความปลอดภัยที่สูงขึ้น

ในการปรับปรุงชุดคำสั่งที่ผู้วิจัยได้ทำการทดลองนั้น ยังสามารถเพิ่มเติมดีไซน์แพตเทิร์นอื่นๆ มาใช้เพิ่มเติมได้ อาทิเช่น Singleton pattern factory pattern Proxy pattern Decorator pattern หรือ Observer pattern แต่ทั้งนี้ผู้พัฒนาจำเป็นจะต้องมีความเข้าใจในดีไซน์แพตเทิร์นที่จะนำมาใช้เป็นอย่างดีซึ่งจะช่วยให้ซอฟต์แวร์ที่มีคุณภาพ

เอกสารอ้างอิง

Al-Hawari, F. (2022). Software design patterns for data management features in web-based information systems. Journal of King Saud University - Computer and Information Sciences 34(10): 10028 – 10043. doi: 10.1016/j.jksuci.2022.10.003.

Bijlsma, L.A., Kok, A.J.F., Passier, H.J.M., Pootjes, H.J. and Stuurman, S. (2022). Evaluation of design pattern alternatives in Java. Software: Practice and Experience 52(5): 1305 – 1315. doi: 10.1002/spe.3061.

Jha, P.C., Bali, V., Narula, S. and Kalra, M. (2014). Optimal component selection based on cohesion & coupling for component based software system under build-or-buy scheme. Journal of Computational Science 5(2): 233 – 242. doi: 10.1016/j.jocs.2013.07.003.

Khosravi, K. and Guéhéneuc, Y-G. (2017). A Quality Model for Design Patterns. Source: https://www.researchgate.net/publication/249885094_A_Quality_Model_for_Design_Patterns. Retrieved from 28 October 2023.

Ntantogian, C., Malliaros, S. and Xenakis, C. (2019). Evaluation of password hashing schemes in open source web platforms. Computers & Security 84(2): 206 – 224. doi: 10.1016/j.cose.2019.03.011.

Osman, T. and Ömer, T. (2018). An Experimental Evaluation of The Effect of SOLID Principles to Microsoft vs Code Metrics. AJIT-e: Online Academic Journal of Information Technology 9(34): 7 - 24. doi: 10.5824/1309-1581.2018.4.001.x.

Patel, H. (2023). An insight on software development lifecycle (SDLC) process models. Advance doi: 10.31124/advance.22354453.v1.

Ramasamy, S., Jekese, G. and Hwata, C. (2015). Impact of Object Oriented Design Patterns on Software Development. International Journal of Scientific and Engineering Research 3(2): 6.

Rashidi, H. (2012). Using the Strategy Pattern to select encryption algorithms dynamically in application softwares. Source: https://www.semanticscholar.org/paper/890587e99f753ba42e510071e50e24bd6ddce654. Retrieved from 28 October 2023.

Roman, A.L. (2019). Ameliorating Password Security Authentication Using BCRYPT Algorithm with Dynamic Salt Generation. Journal of Advanced Research in Dynamical and Control Systems 11. 1240 - 1245. doi: 10.5373/JARDCS/V11SP12/20193331.

Shri, R.N. and Ravikumar, B. (2018). Enhancement of public cloud, application security, using Bcrypt algorithm. International Journal of Scientific Research in Computer Science, Engineering and Information Technology 3(3): 1029 – 1032. doi: 10.32628/CSEIT1833362.

Temaj, G. (2020). Factory Design Pattern. Source: https://www.researchgate.net/publication/350611051_Factory_Design_Pattern. Retrieved from 28 October 2023.

Vaghela, R.K. and Pithva, K.A. (2016). Software design pattern approach to develop login framework. In: Proceedings of the 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India. 1013 – 1017.

Verma, C. and Liu, L. (2003). Re-engineering legacy code with design patterns : A case study in mesh generation software. Source: https://www.semanticscholar.org/paper/1aa8ba2e3cd596f3c501e0966530ab24c6bb190e. Retrieved from 28 October 2023.

Watts, S. (2023). The importance of SOLID design principles. BMC Blogs. Source: https://www.bmc.com/blogs/solid-design-principles/. Retrieved from 27 October 2023.

Zotos, K. (2007). Object-oriented design principles in mathematics. Applied Mathematics and Computation 188(2): 1430 – 1436. doi: 10.1016/j.amc.2006.11.009.

❑❑❑❑❑