โครงงาน

เรื่อง X-Brain

จัดทำโดย

นางสาวนนทิชา	สุขเจริญ	รหัสนักศึกษา 63010484
นางสาวนภสร	ชาลานุมาศ	รหัสนักศึกษา 63010492
นายปฏิภาณ	ศรีธรณ์	รหัสนักศึกษา 63010555
นายปารมี	ชุมศรี	รหัสนักศึกษา 63010598
นายปุณณวิชญ์	พานิชผล	รหัสนักศึกษา 63010616
นางสาวพัณณ์ชิตา	ธีรพัฒนโรจน์	รหัสนักศึกษา 63010665
นายพีระภัทร์	เศรษฐพรนรา	รหัสนักศึกษา 63010702

เสนอ

ดร.ปริญญา เอกปริญญา

โครงงานนี้เป็นส่วนหนึ่งของวิชาสถาปัตยกรรมและการออกแบบซอฟต์แวร์

(Software Architecture and Design)

ภาคเรียนที่ 1 ปีการศึกษา 2565

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

คำนำ

รายงานเล่มนี้จัดทำขึ้นเพื่อเป็นส่วนหนึ่งของวิชาสถาปัตยกรรมและการออกแบบซอฟต์แวร์ เพื่อศึกษาหาความรู้อย่างเข้าใจและสามารถนำไปเป็นประโยชน์ในการทำงานได้

คณะผู้จัดทำหวังว่า รายงานเล่มนี้จะเป็นประโยชน์กับผู้อ่าน หรือผู้ที่สนใจข้อมูลนี้ หากมีข้อแนะนำหรือข้อผิดพลาดประการใด ผู้จัดทำขออภัยมา ณ ที่นี้ด้วย

คณะผู้จัดทำ

สารบัญ

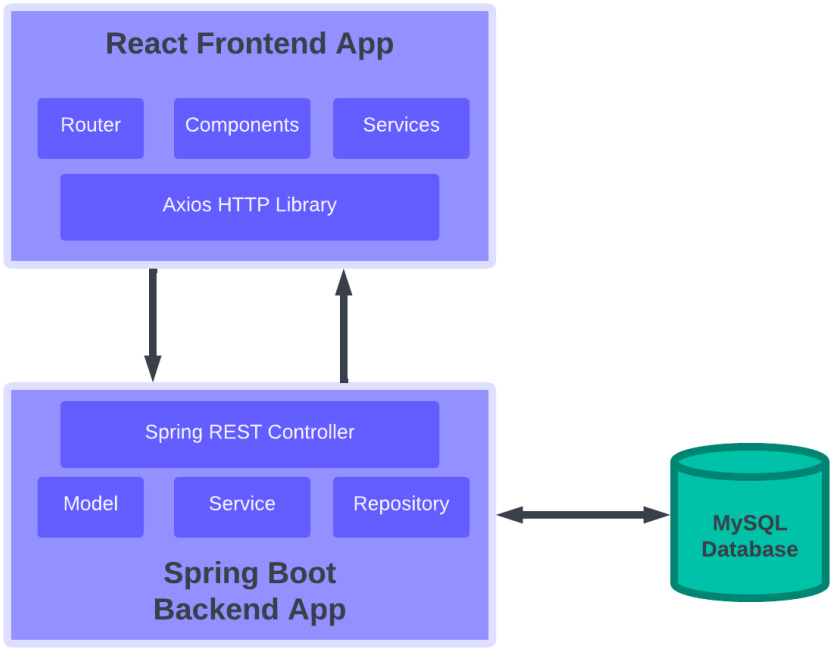
ที่มาและจุดประสงค์

เนื่องจากในปัจจุบันการศึกษาของไทยนั้นมีมากมายหลากหลายรูปแบบมีเนื้อหาวิชาและหลักการนำเสนอแตกต่างกันออกไป รวมทั้งยังมีการประยุกต์การใช้ทฤษฎีตัวอย่างต่างๆ ที่ไม่ได้อยู่ในบทเรียนทั้งหมด โดยนักเรียนหรือผู้ที่ต้องการที่จะศึกษาในเรื่องนั้นๆ มักจะพบกับคำถามหรือปัญหาที่ตนไม่สามารถแก้ไขได้ ผู้คนเหล่านี้จึงต้องการที่จะหาคำตอบ แต่ทว่าไม่สามารถที่จะนำคำถามหรือปัญหานั้นมาสอบถามหรือเผยแพร่ได้ เพราะ ขาดช่องทางในการสอบถามอย่างเป็นหลักเป็นแหล่ง สำหรับผู้ที่สนใจที่จะศึกษาเนื้อหาวิชาต่างๆ อย่างจริงจังหรือผู้ปกครองที่ต้องการหาผู้สอนพิเศษสำหรับบุตร-หลานที่มีความน่าเชื่อถือ ความปลอดภัยและสถานที่ในการสอนที่สามารถตรวจสอบได้ เพื่อที่จะทำให้มั่นใจในคุณภาพของการสอนก่อนสมัครเรียนกับอาจารย์สอนพิเศษนั้นๆ

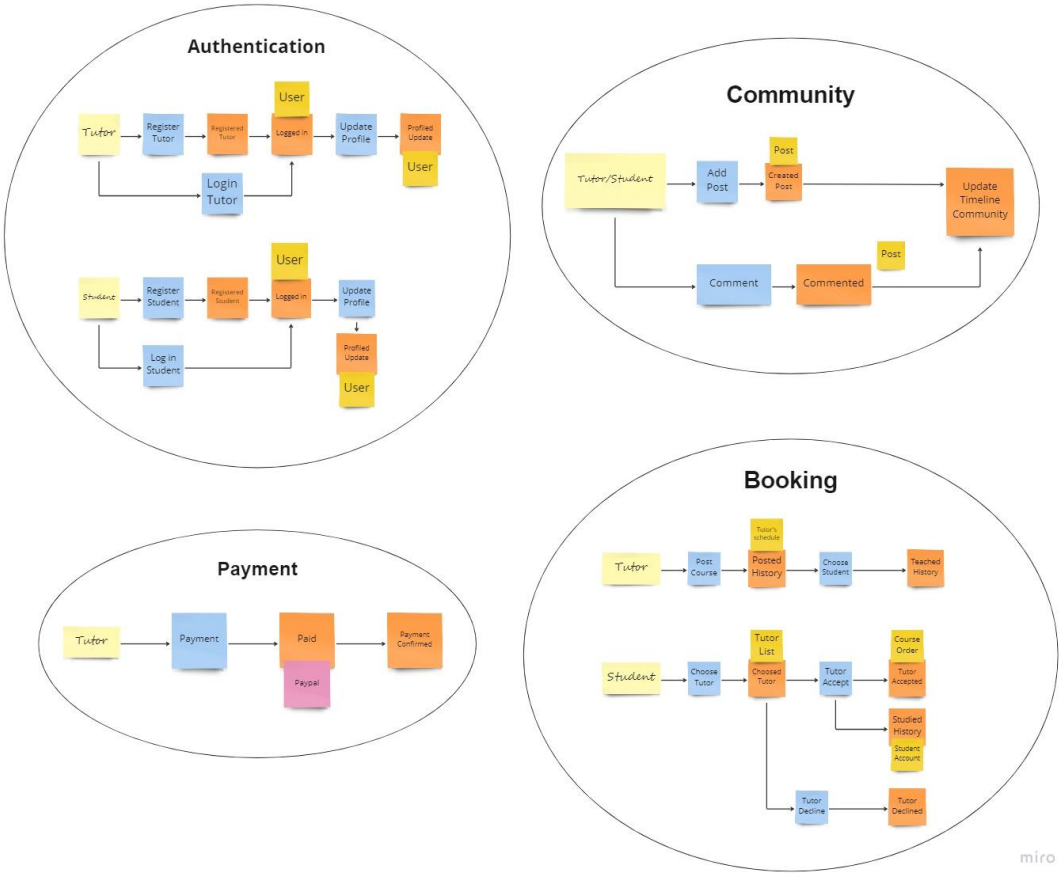
จากปัญหาข้างต้นจึงจัดทำเว็บแอพพลิเคชั่น X-Brain เป็นเว็บแอพพลิเคชั่นที่มาตอบโจทย์ปัญหาต่างๆ อำนวยความสะดวกในเรื่องการศึกษา โดยที่เราจะมีระบบ Booking เป็นสื่อกลางในการหาติวเตอร์สอนพิเศษ ซึ่งติวเตอร์สอนพิเศษสามารถกรอกรายละเอียดการสอน เลือกวิชาที่จะเปิดสอน สถานที่สะดวกสอน และวันเวลาที่สะดวกสอนได้ตามที่ต้องการได้ ส่วนของนักเรียนหรือนักศึกษาสามารถเลือกวิชาที่ต้องการเรียน และวันเวลาว่างที่ตรงกับติวเตอร์สอนพิเศษได้อีกทั้งระบบของเราจะมี Community ที่สามารถทำหน้าที่เป็นข่าวสาร คลังความรู้ที่ทุกคนจะมาแบ่งปันและ ถาม-ตอบปัญหาเรื่องเรียน รวมไปถึงเรื่องการซื้อ-ขายหรือแลกเปลี่ยนหนังสือเรียน

Software Architecture

สถาปัตยกรรมหลักของ X-Brain แบบ Representational State Transfer (REST) หรือ RESTful เป็นสถาปัตยกรรมซอฟต์แวร์ที่มีการกำหนดเงื่อนไขว่า API ควรทำงานเช่นไร โดยมีการแบ่งการทำงานเป็น 2 ส่วน ได้แก่ Front-End และ Back-End โดยมีโครงสร้างแบบ Client-Server คือการที่ผู้ใช้บริการส่งคำร้องติดต่อกับเซิร์ฟเวอร์ผ่านเครือข่ายคอมพิวเตอร์เป็นตัวกลาง เมื่อผู้ใช้งานสั่งการทำงานที่ Front-End จะส่งคำร้องไปที่ Back-end จากนั้น Back-End จะรับคำขอและทำหน้าที่เป็นสื่อกลางของการทำงาน Front-End กับ Database

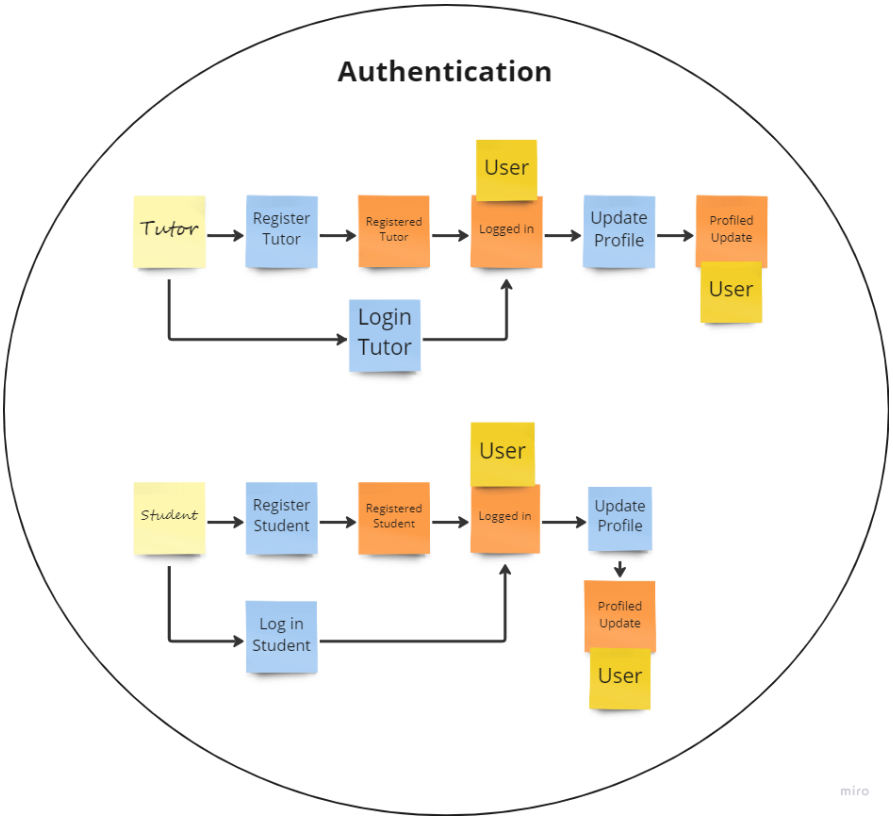


โดยจะมีการนำหลักการและวิธีการของ Domain-Driven Design มาใช้ในการทำงานและออกแบบสถาปัตยกรรมของ X-Brain ด้วย โดยเริ่มแรกคือการใช้ Event storming เพื่อลำดับเหตุการณ์ให้ง่ายต่อการทำงานของ X-Brain จากการเป็นเว็บแอพพลิเคชั่นที่ใช้อำนวยความสะดวกในเรื่องการศึกษา และเป็นสื่อกลางในการหาติวเตอร์สอนพิเศษ ซึ่งจะทำให้เราได้ Bounded Context แบ่งออกเป็น 4 Bounded Context



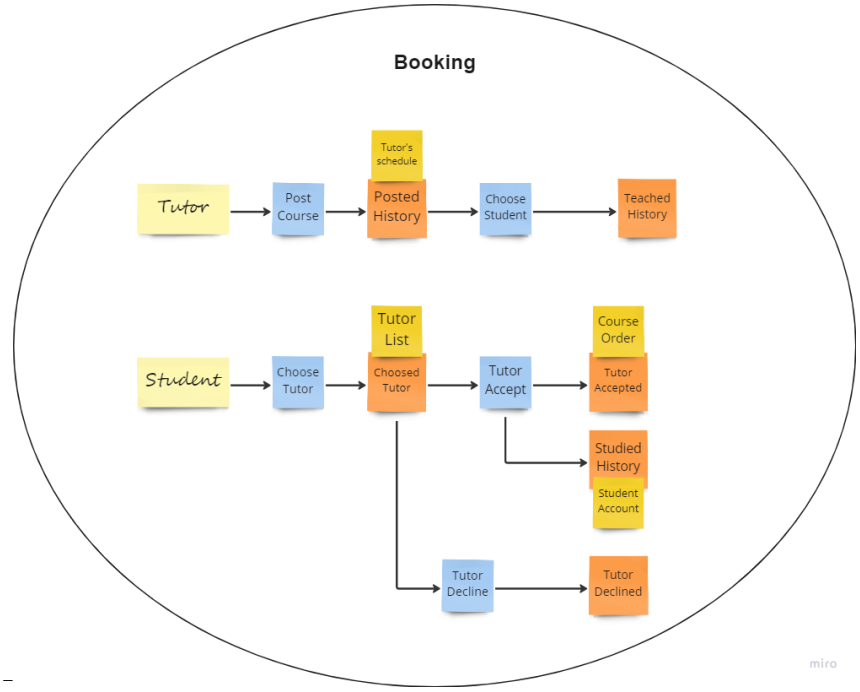
รูปที่ 1 Event storming ประกอบด้วย Authentication, Booking, Community และ Payment

Bounded context



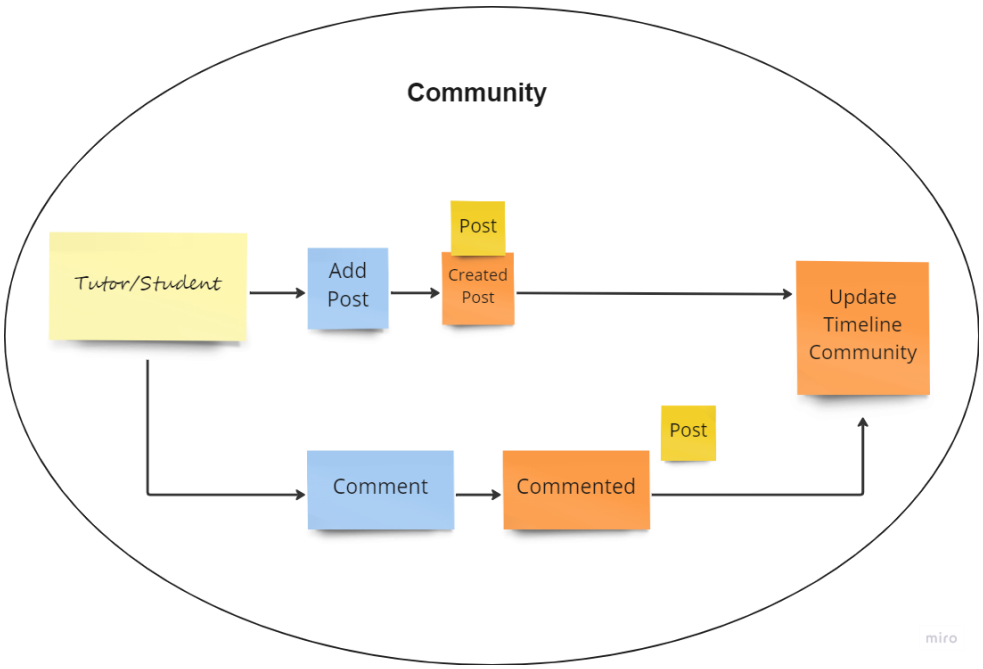
รูปที่ 2 Authentication

เป็นขอบเขตการทำงานที่เกี่ยวข้องกับ User ที่จะแบ่งเป็น 2 ประเภทคือ Student และ Tutor โดยจะเป็นการทำงานเกี่ยวกับด้านการลงทะเบียน เข้าสู่ระบบ และ อัพเดทโปรไฟล์ของแต่ละผู้ใช้งานนั้นๆ



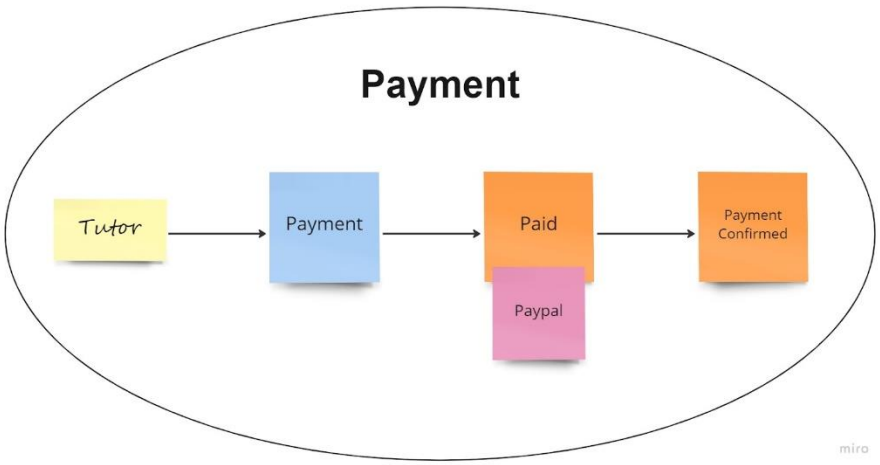
รูปที่ 3 Booking

เป็นขอบเขตการทำงานที่เกี่ยวข้องกับ ส่วนของ Student มีการกรอกข้อมูล จากนั้นส่งใบสมัครเพื่อที่จะสมัครเรียนโดยจะรอการยืนยันของใบสมัครของ Tutor ซึ่งผลลัพธ์จะอยู่หน้าประวัติการสมัคร และส่วนของ Tutor จะเริ่มจากการสร้างโพสต์สอนพิเศษ หน้าประวัติการโพสต์ทาง Tutor สามารถยอมรับหรือปฏิเสธ Student ที่เข้ามาสมัครได้



รูปที่ 4 Community

เป็นขอบเขตการทำงานที่เกี่ยวข้องกับการโพสต์สอบถามหรือพูดคุย โดยจะสามารถโพสต์และคอมเมนท์ตอบได้



รูปที่ 5 Payment

เป็นขอบเขตการทำงานที่เกี่ยวข้องกับการชำระเงินการโพสต์ของ Tutor โดยจะเป็นการชำระผ่านช่องทางของ Paypal

Design Patterns

1. Builder Pattern (Creational Design patterns)

การทำ Builder Pattern ทำให้สร้าง object ที่มีขั้นตอนในการสร้างที่ซับซ้อนให้ถูกสร้างได้ง่ายๆ ช่วยลดขั้นตอนการสร้างโค้ดที่ไม่จำเป็นรวมอยู่ที่เดียวกัน แต่สามารถสร้าง object ที่แตกต่างกันได้

- **ทำไมถึงใช้**

เป็นเพราะ การสร้าง User ในแต่ละ User นั้น จำเป็นต้องมีการใช้ข้อมูลที่เป็นพื้นฐานที่จะต้องใช้เพื่อเก็บอยู่แล้ว และมีการใช้ในเพื่อจุดประสงค์จุดเดียวกันด้วย ซึ่งนั่นก็คือ id, username, password, ด้วยเหตุผลนี้จึงควรที่จะต้องลดความยุ่งยากในการเรียกหรือสร้างมันลงไป

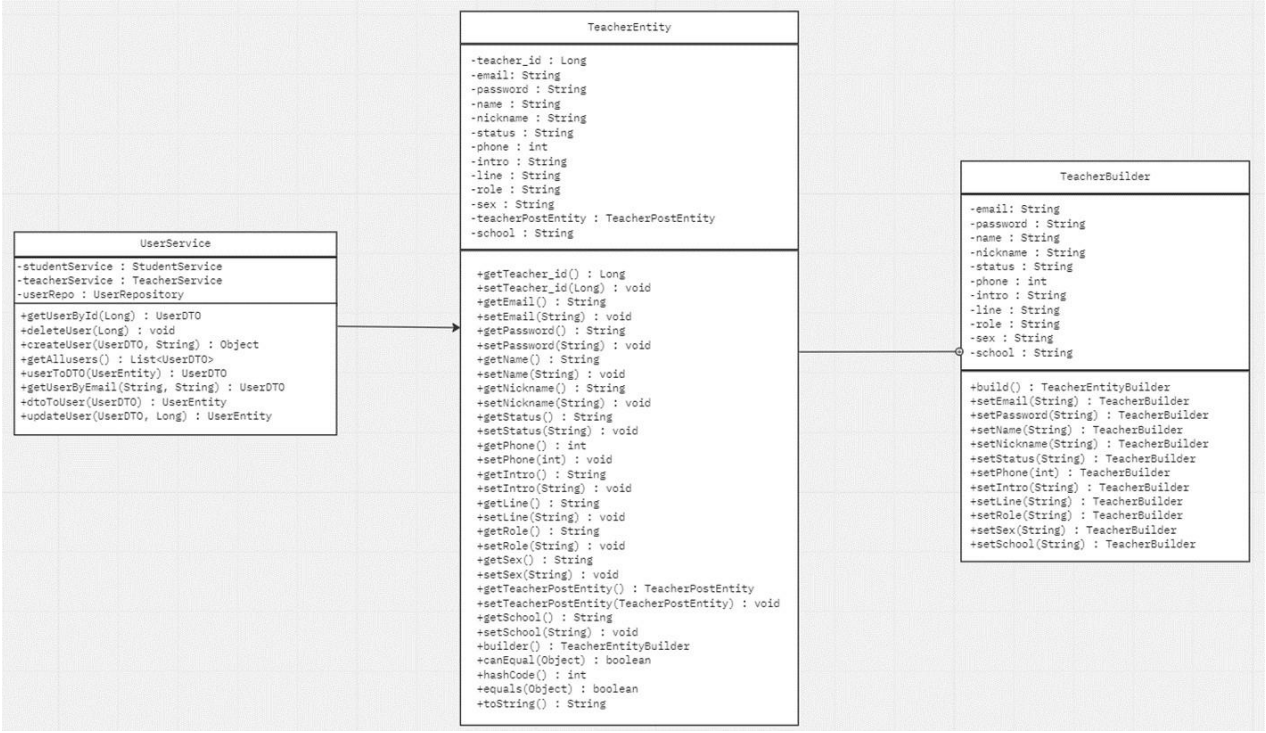
- **ใช้อย่างไร**

โดยเริ่มแรกเราจะสร้าง Class User ขึ้นมาและระบุข้อมูลหลักๆที่มีโอกาสที่จะปรับเปลี่ยนได้น้อยแต่ถูกเรียกทุกครั้งใส่ลงไป จากนั้น จะสร้าง DetailBuilder Class และ ContactBuilder Class เพื่อมาเป็น Builder class ที่ใส่ข้อมูลย่อยๆ ที่มีโอกาสถูกปรับเปลี่ยนและถูกเรียกบ่อยที่สุด รวมไปถึงข้อมูลใหม่ๆ ที่อยากจะใส่ลงไป

- **ใช้แล้วมีผลดีอย่างไร**

ลดปัญหาการเรียกใช้ class ซ้ำๆ, ลดความซับซ้อน, ความยาวที่จะสร้าง หรือ การที่มีข้อมูลที่เราไม่จำเป็นต้องเอามานั้นติดไปด้วยซึ่งก็จะลดการบวมของ parameters ลงไปได้

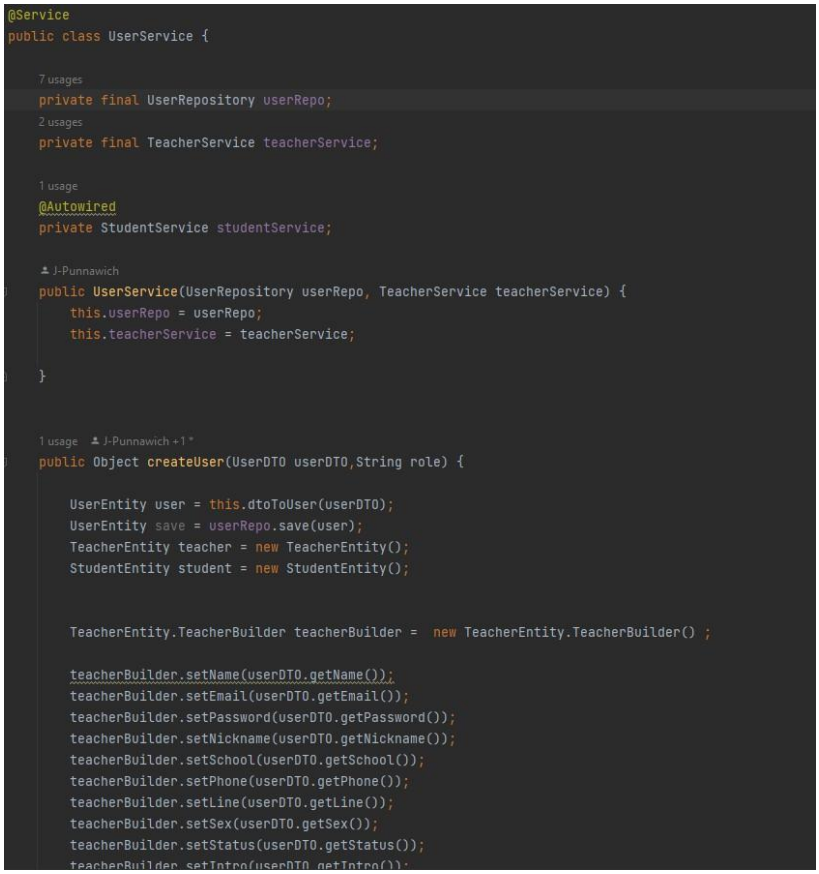
- **UML Class Diagram**



รูปที่ 6 แสดง Class Diagram ของ Builder Pattern ที่ถูกนำมาปรับใช้งาน

```java
public static class TeacherBuilder {
    2 usages
    private String name ;
    2 usages
    private String email ;
    2 usages
    private String password;
    2 usages
    private String school;
    2 usages
    private String nickname;
    2 usages
    private String sex;
    2 usages
    private String status;
    2 usages
    private String intro;
    2 usages
    private int phone;
    2 usages
    private String line;
    2 usages
    private String role;

    1 usage  Peerapat Sattapornnara
    public TeacherBuilder() {
    }

    Peerapat Sattapornnara
    public TeacherBuilder setName (String name) {
        this.name = name;
        return this;
    }
    Peerapat Sattapornnara
    public TeacherBuilder setEmail (String email) {
        this.email = email;
        return this;
    }
    Peerapat Sattapornnara
    public TeacherBuilder setPassword (String password) {
        this.password = password;
        return this;
    }
    1 usage  Peerapat Sattapornnara
    public TeacherBuilder setSchool (String school) {
        this.school = school;
        return this;
    }
    1 usage  Peerapat Sattapornnara
    public TeacherBuilder setNickname (String nickname) {
        this.nickname = nickname;
        return this;
    }
    1 usage  Peerapat Sattapornnara
    public TeacherBuilder setSex (String sex) {
        this.sex = sex;
        return this;
    }
    Peerapat Sattapornnara
    public TeacherBuilder setStatus(String status) {
        this.status = status;
        return this;
    }
    1 usage  Peerapat Sattapornnara
    public TeacherBuilder setIntro(String intro) {
        this.intro = intro;
        return this;
    }
    1 usage  Peerapat Sattapornnara
    public TeacherBuilder setPhone(int phone) {
        this.phone = phone;
        return this;
    }
    1 usage  Peerapat Sattapornnara
    public TeacherBuilder setLine(String line) {
        this.line = line;
        return this;
    }
    1 usage  Peerapat Sattapornnara
    public TeacherBuilder setRole(String role) {
        this.role = role;
        return this;
    }
```

```java
public class TeacherEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long teacher_id ;
    1 usage
    private String name ;
    1 usage
    private String email ;
    1 usage
    private String password;
    1 usage
    private String school;
    1 usage
    private String nickname;
    1 usage
    private String sex;
    1 usage
    private String status;
    1 usage
    private String intro;
    1 usage
    private int phone;
    1 usage
    private String line;

    1 usage
    private String role;


    @OneToOne(cascade = CascadeType.ALL , fetch = FetchType.LAZY)
    @JoinColumn(name = "post_id")
    @JsonIgnore
    private TeacherPostEntity teacherPostEntity ;
    1 usage  Peerapat Sattapornnara
    public TeacherEntity build() {
        return new TeacherEntity(   name,  email,  password,  school,  nickname,  sex,  status,  intro,  phone,  line,  role) ;
    }
}
```



```java
@Service
public class UserService {

    7 usages
    private final UserRepository userRepo;
    2 usages
    private final TeacherService teacherService;

    1 usage
    @Autowired
    private StudentService studentService;

    J-Punnawich
    public UserService(UserRepository userRepo, TeacherService teacherService) {
        this.userRepo = userRepo;
        this.teacherService = teacherService;

    }


    1 usage  J-Punnawich +1 *
    public Object createUser(UserDTO userDTO,String role) {

        UserEntity user = this.dtoToUser(userDTO);
        UserEntity save = userRepo.save(user);
        TeacherEntity teacher = new TeacherEntity();
        StudentEntity student = new StudentEntity();


        TeacherEntity.TeacherBuilder teacherBuilder =  new TeacherEntity.TeacherBuilder() ;

        teacherBuilder.setName(userDTO.getName());
        teacherBuilder.setEmail(userDTO.getEmail());
        teacherBuilder.setPassword(userDTO.getPassword());
        teacherBuilder.setNickname(userDTO.getNickname());
        teacherBuilder.setSchool(userDTO.getSchool());
        teacherBuilder.setPhone(userDTO.getPhone());
        teacherBuilder.setLine(userDTO.getLine());
        teacherBuilder.setSex(userDTO.getSex());
        teacherBuilder.setStatus(userDTO.getStatus());
        teacherBuilder.setIntro(userDTO.getIntro());
```

รูปที่ 7 ตัวอย่าง code

2. Strategy Pattern (Behavioral Design pattern)

เป็นการกำหนดกลุ่มของ Algorithm (Family of algorithm) ที่ซ้อน Algorithm นั้นๆไว้ และทำให้มันสามารถเปลี่ยนแปลงได้ Strategy ทำให้ Algorithm เปลี่ยนแปลงอย่างอิสระ จากการใช้งานของ Client สามารถเพิ่ม Strategy ใหม่ๆเข้าไปได้เรื่อย ๆหรือแก้ไข Strategy โดยที่ไม่มีผลกระทบกับ Context หรือ Strategy ตัวอื่น

- **ทำไมถึงใช้**

  เป็นเพราะมีการที่จะเพิ่ม parameter หรือ เพิ่มการสั่งงาน ในอนาคต หากเราไม่ทำอะไรแล้วมาเพิ่มเอาทีหลัง ก็อาจจะส่งผลกระทบที่ไม่ดีต่อการเรียกใช้งาน และ โครงสร้างโดยรวมได้

- **ใช้อย่างไร**

  เป็นการสร้าง interface getUserStratogy เพื่อมารองรับ getUserStrategContext ที่ต้องการที่จะการเรียกใช้งาน GetAllUser หรือ GetStudentUser หรือ GetTeacherUser ผ่านทาง interface นี้

- **ใช้แล้วมีผลดีอย่างไร**

  ทำให้ getUserStrategContext นั้นสามารถที่จะเพิ่ม คำสั่ง หรือ สั่งใช้งานคำสั่งอื่นๆได้ หลากหลายโดยที่ไม่มีการผิดพลาดหรือการปรับเปลี่ยนโครงสร้าง ของ Class เกิดขึ้น

- **UML Class Diagram**

รูปที่ 8 แสดง Class Diagram ของ Strategy Pattern ที่ถูกนำมาปรับใช้งาน

```
4 usages  1 implementation  Peerapat Sattapornnara
public interface getUserStrategy {
    1 implementation  Peerapat Sattapornnara
    List<UserEntity>  getUser () ;
}
```

```
public class getUserStrategyContext {
    3 usages
    private getUserStrategy getUserStrategy;

    public getUserStrategyContext(getUserStrategy getUserStrategy) {
        this.getUserStrategy = getUserStrategy ;
    }

    public void setGetUserStrategy (getUserStrategy getUserStrategy) {
        this.getUserStrategy = getUserStrategy ;
    }

    public List<UserEntity> useGetUser() {
        return this.getUserStrategy.getUser() ;
    }

}
```

```
public class getAllUsers implements getUserStrategy {
    1 usage
    @Autowired
    private UserRepository userRepository ;

    Peerapat Sattapornnara
    @Override
    public List<UserEntity> getUser() {
        return userRepository.findAll();
    }
```

รูปที่ 9 ตัวอย่าง code

3. Proxy Pattern (Structural Design patterns)

เป็นเหมือนตัวแทนหรือพ่อค้าคนกลางของ object ที่ต้องการเข้าถึง แต่ไม่ได้เข้าถึง object นั้นโดยตรงจริงๆ ซึ่งก็จะใช้ proxy มาใช้เข้าถึง object นั้นๆ แทน

- **ทำไมถึงใช้**

เนื่องจากการเข้าถึงข้อมูลโดยตรงนั้นทำให้แก้ไขและเรียกดูได้ยาก และ ไม่มีความปลอดภัยมากพอ หากเกิดการค้าง หรือการโหลดข้อมูลมีการดีเลย์ ทำให้ระบบส่วนอื่นแสดงผลผิดพลาดได้

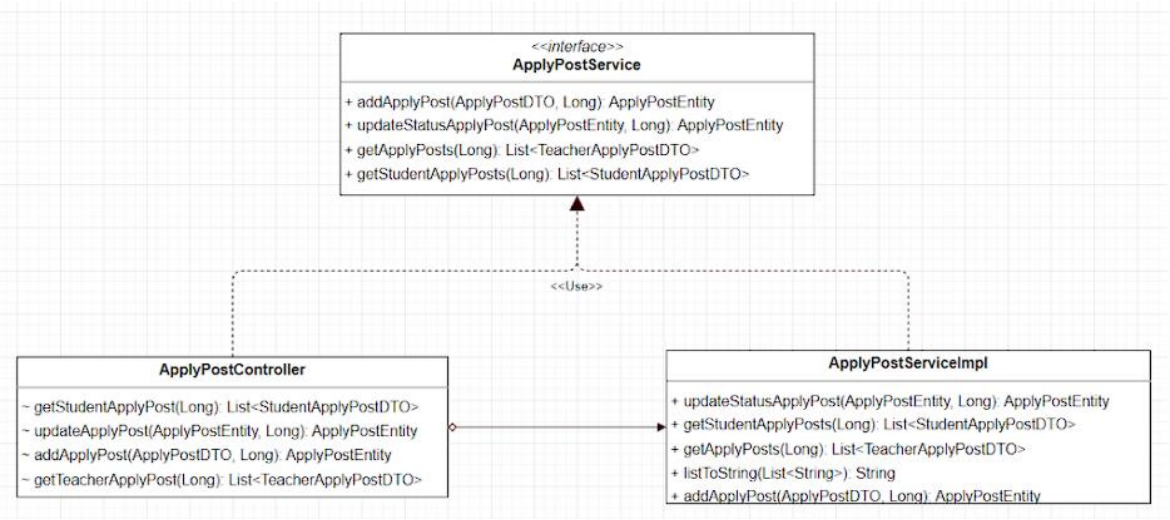
- **ใช้อย่างไร**

จะเป็นการสร้าง interface ApplyPostService ขึ้นมา เพื่อที่จะเรียกใช้ ApplyPostController Class เพื่อเข้าถึง ApplyPostServiceImpl ซึ่งจะไม่มีการเข้าถึงโดยตรง

- **ใช้แล้วมีผลดีอย่างไร**

สามารถช่วยควบคุมและเข้าถึง service ต่างๆ และทำให้มีความปลอดภัยมากขึ้น ในที่นี้จะเป็น การเข้าถึง ApplyPostServiceImpl ผ่าน interface ApplyPostService โดยติดต่อผ่าน ApplyPostController

- UML Class Diagram



รูปที่ 10 Class Diagram ของ Proxy Pattern ที่ถูกนำมาปรับใช้งาน

```java
public interface ApplyPostService {


    ApplyPostEntity addApplyPost (ApplyPostEntity applyPost) ;
```

```java
@CrossOrigin
@RestController
public class ApplyPostController {

    @Autowired
    ApplyPostService applyPostService ;
    @Autowired
    ApplyPostRepostity applyPostRepostity ;



    public ApplyPostController(ApplyPostService applyPostService, ApplyPostRepostity applyPostRepostity) {
        this.applyPostService = applyPostService;
        this.applyPostRepostity = applyPostRepostity;
    }

    @PostMapping(path = "/createApplyPost")
    ApplyPostEntity addApplyPost(@RequestBody ApplyPostEntity applyPostEntity){
        return applyPostService.addApplyPost(applyPostEntity);
    }
```

```java
@Service
public class ApplyPostServiceImpl implements ApplyPostService{
    2 usages
    @Autowired
    ApplyPostRepostity applyPostRepostity ;

    J-Punnawich
    @Autowired
    public ApplyPostServiceImpl(ApplyPostRepostity applyPostRepostity) { this.applyPostRepostity = applyPostRepostity; }

    J-Punnawich
    @Override
    public ApplyPostEntity addApplyPost(ApplyPostEntity applyPost) {

        return applyPostRepostity.save(applyPost) ;
    }
}
```

รูปที่ 11 ตัวอย่าง code

Quality Attribute Scenarios (QAS)

Availability ความพร้อมของระบบ :

Portion of Scenario	Possible Values
Source	ผู้ใช้งาน, การทำงานภายในและนอกของระบบ
Stimulus	ระบบล่ม หรือ การไม่ตอบสนองของเซิร์ฟเวอร์ / ค้าง / หลุด / ทำงานไม่ถูกต้อง
Artifact	ระบบเซิร์ฟเวอร์
Environment	ภายใต้เวลาการทำงานปกติ,
Response	แจ้งเตือนผ่านทาง E-mail
Response Measure	ต้องไม่มีการล่มของระบบ

Performance ประสิทธิภาพ :

Portion of Scenario	Possible Values
Source	ผู้ใช้งาน
Stimulus	การป้อนข้อมูลแบบปกติ
Artifact	ระบบการทำงาน
Environment	ภายใต้เวลาการทำงานปกติ
Response	ระบบรับข้อมูลจากการป้อน
Response Measure	เกิดการส่งข้อมูล (ภายในเวลา...)

Modifiability ความสามารถในการปรับปรุงแก้ไข :

Portion of Scenario	Possible Values
Source	ผู้พัฒนาระบบ หรือ Developer
Stimulus	ต้องการปรับปรุง / ลบ / แก้ไข / เปลี่ยนฟังก์ชั่น
Artifact	System & Runtime
Environment	ภายใต้เวลาการแก้ไขและปรับเปลี่ยน Code, ภายใต้เวลาการปิดการทำงานของระบบ
Response	ฟังก์ชั่นใช้งานได้ ไม่มีผลกระทบกับฟังก์ชั่นอื่นๆ
Response Measure	เวลาที่ใช้ในการปรับปรุง, ฟังก์ชั่นสามารถใช้งานได้, เมื่อเปิดการทำงานของระบบระบบสามารถทำงานได้

Usability การใช้งานง่าย :

Portion of Scenario	Possible Values
Source	ผู้ใช้งาน
Stimulus	สามารถใช้งานได้
Artifact	การใช้งาน, หน้า UI ที่ดูง่าย
Environment	ภายใต้เวลาการทำงานปกติ
Response	การเรียนรู้การใช้งานระบบและหน้าUI โดยผู้ใช้
Response Measure	เวลาที่ใช้ในการใช้งาน

Security ความปลอดภัย :

Portion of Scenario	Possible Values
Source	ผู้ใช้งานที่ไม่มีสิทธิ์การเข้าถึง
Stimulus	ต้องการเข้าถึงข้อมูลใน database
Artifact	Database
Environment	ภายใต้เวลาการทำงานปกติ
Response	แจ้งให้ทราบว่าผู้ใช้งานไม่มีสิทธิ์เข้าถึง
Response Measure	ข้อมูล database ไม่ถูกเข้าโดยผู้ไม่มีสิทธิ์เข้าถึง

Integrability การนำไปปรับประยุกต์ใช้กับระบบภายนอกได้ :

Portion of Scenario	Possible Values
Source	ผู้ใช้งาน
Stimulus	ต้องการชำระค่าบริการ
Artifact	PayPal (ระบบชำระเงินจากภายนอก)
Environment	ภายใต้เวลาการทำงานปกติ
Response	แจ้งการชำระเงินสำเร็จโดยใช้ PayPal
Response Measure	มี Statements ระบุการชำระเงินของ PayPal

Extensibility ความสามารถในการเพิ่มเติมความสามารถภายหลังได้ โดยไม่กระทบระบบเดิม :

Portion of Scenario	Possible Values
Source	ผู้พัฒนา
Stimulus	ต้องการเพิ่มฟีเจอร์ / ระบบอื่นๆ
Artifact	หน้า UI
Environment	ภายใต้เวลาการแก้ไขและปรับเปลี่ยน Code, ภายใต้เวลาการปิดการทำงานของระบบ
Response	เพิ่มฟีเจอร์ที่ต้องการของหน้า UI
Response Measure	สามารถเพิ่มฟีเจอร์ที่ต้องการที่หน้า UI ได้, เมื่อเปิดการทำงานของระบบระบบสามารถทำงานได้

Portion of Scenario	Possible Values
Source	ผู้พัฒนา
Stimulus	ต้องการเพิ่มฟีเจอร์ / ระบบอื่นๆ
Artifact	Database
Environment	ภายใต้เวลาการแก้ไขและปรับเปลี่ยน Code, ภายใต้เวลาการปิดการทำงานของระบบ
Response	เพิ่มฟีเจอร์ที่ต้องการของ Database
Response Measure	สามารถเพิ่มฟีเจอร์ที่ต้องการได้, เมื่อเปิดการทำงานของระบบระบบสามารถทำงานได้