



## Website KCU BLOG

เว็บไซต์แลกเปลี่ยนความคิดเห็น KCU

จัดทำโดย

นายณัฐวัฒน์ หมายบุญ	643020045-6
นายธีรภัทร แสงโชติ	643021233-0
นายปณัฏฐวุฒิ พรหมเมือง	643021239-8

อาจารย์ที่ปรึกษา : รศ. ดร.ปัญญาพล หอระตะ

รายงานเล่มนี้เป็นส่วนหนึ่งของการศึกษาวิชา SC313002 หลักการออกแบบพัฒนาซอฟต์แวร์

(PRINCIPLES OF SOFTWARE DESIGN AND DEVELOPMENT)

ภาคเรียนที่ 1 ปีการศึกษา 2566 สาขาวิทยาการคอมพิวเตอร์ วิทยาลัยการคอมพิวเตอร์ ภาคเรียนที่ 1 ปีการศึกษา 2566 สาขาวิทยาการคอมพิวเตอร์วิทยาลัยการคอมพิวเตอร์

มหาวิทยาลัยขอนแก่น (เดือน ตุลาคม พ.ศ. 2566

## คำนำ

โครงงานนี้เป็นส่วนหนึ่งของวิชา 313002 หลักการออกแบบพัฒนาซอฟต์แวร์ (PRINCIPLES OF SOFTWARE DESIGN AND DEVELOPMENT) วิทยาการคอมพิวเตอร์ วิทยาลัยการคอมพิวเตอร์ มหาวิทยาลัยขอนแก่น ซึ่งทางสาขาได้จัดหลักสูตรนี้ขึ้นโดยมีวัตถุประสงค์เพื่อให้นักศึกษานำความรู้มาประยุกต์ใช้ในการทำงานและทำให้เกิดประโยชน์ต่อไป ซึ่งโครงงานได้จัดทำเกี่ยวกับ เว็บไซต์แลกเปลี่ยนความคิดเห็น KKU โดยดำเนินการตามหลักการออกแบบพัฒนาซอฟต์แวร์อย่างถูกต้องตามวิธีการ

ผู้จัดทำโครงงานหวังเป็นอย่างยิ่งว่าโครงงานนี้จะเป็นประโยชน์กับผู้อ่าน หรือ นักเรียน นักศึกษา ที่กำลังหาข้อมูลเรื่องนี้อยู่ หากมีข้อเสนอแนะหรือข้อผิดพลาดประการใด ผู้จัดทำขอน้อมรับไว้และขออภัย มา ณ ที่นี้ด้วย

คณะผู้จัดทำ

## สารบัญ

### เรื่อง

### หน้า

คำนำ.....	ก
สารบัญ.....	๗
สารบัญ(ต่อ).....	ค
สารบัญรูปภาพ.....	ง
สารบัญรูปภาพ(ต่อ).....	จ
1.ที่มาและความสำคัญ .....	1
2.วัตถุประสงค์.....	1
3.ประโยชน์ที่คาดว่าจะได้รับ .....	1
4.ทฤษฎีและเครื่องมือที่เกี่ยวข้อง.....	1
4.1 ทฤษฎีที่เกี่ยวข้อง.....	1
4.1.1 Spring Boot .....	1
4.1.2 Strategy Pattern.....	1
4.1.3 Repository Pattern .....	2
4.1.4 Factory Pattern .....	2
4.1.5 Spring Dependency Injection .....	2
4.2 เครื่องมือที่เกี่ยวข้อง.....	2
4.2.1 โปรแกรม visual studio code.....	2
4.2.2 Mysql Workbench .....	2
4.2.3 โปรแกรม Microsoft Word.....	2
5. อธิบายฐานข้อมูล .....	3
5.1 Use case diagram .....	3
5.2 Class diagram .....	4

## สารบัญ

เรื่อง	หน้า
5.3 ER-diagram .....	5
5.4 UML Activity Diagram .....	6
6.อธิบายการสร้างระบบเว็บโดยใช้ Spring boot editor .....	6-21
7.User interface .....	22
7.1 login .....	22
7.2 Register .....	22
7.3 Dashboard Guest.....	23
7.4 Dashboard.....	23
7.5 Admin .....	24
7.6 Post .....	24
7.7 Your post .....	25
7.8 Edit profile .....	25
7.9 Look post.....	26
7.10 Comment.....	26
7.11 Your comment .....	27
อ้างอิง.....	28

## สารบัญรูปภาพ

รูปที่ 1 Use case diagram.....	3
รูปที่ 2 Class diagram.....	4
รูปที่ 3 ER-diagram.....	5
รูปที่ 4 UML Activity Diagram.....	6
รูปที่ 5 Class User .....	6
รูปที่ 6 Class Profile .....	7
รูปที่ 7 Class Role .....	7
รูปที่ 8 Class Post .....	8
รูปที่ 9 Class Image .....	8
รูปที่ 10 Class Comment.....	9
รูปที่ 11 Class Category.....	9
รูปที่ 12 Repository ทั้งหมด.....	10
รูปที่ 13 Class UserlistService .....	10
รูปที่ 14 Class UserlistService (ต่อ).....	11
รูปที่ 15 Class UserlistService (ต่อ).....	12
รูปที่ 16 Class UserlistService (ต่อ).....	12
รูปที่ 17 Class UserlistService (ต่อ).....	13
รูปที่ 19 Class UserlistService (ต่อ).....	14
รูปที่ 20 Class UserlistService (ต่อ).....	14
รูปที่ 21 Class UserServiceImpl.....	15
รูปที่ 22 Class UserServiceImpl (ต่อ).....	15
รูปที่ 23 Class UserLoginDTO .....	16
รูปที่ 24 Class UserRegisterDTO.....	16
รูปที่ 25 Class AdminController .....	17

## สารบัญรูปภาพ(ต่อ)

รูปที่ 26 Class DashboardControll .....	17
รูปที่ 27 Class DashboardControll (ต่อ).....	18
รูปที่ 28 Class DashboardControll (ต่อ).....	18
รูปที่ 29 Class GuestController.....	19
รูปที่ 30 Class LoginController.....	19
รูปที่ 31 Class RegisterController .....	20
รูปที่ 32 Class SpringSsecurityConfig.....	20
รูปที่ 33 Class CustomSuccessHanler .....	21
รูปที่ 34 User interface หน้า login .....	22
รูปที่ 35 User interface หน้า Register.....	22
รูปที่ 36 User interface หน้า Dashboard Guest.....	23
รูปที่ 37 User interface หน้า Dashboard .....	23
รูปที่ 38 User interface หน้า Admin .....	24
รูปที่ 39 User interface หน้า post.....	24
รูปที่ 40 User interface หน้า Your post .....	25
รูปที่ 41 User interface หน้า edit profile.....	25
รูปที่ 42 User interface หน้า look post.....	26
รูปที่ 43 User interface หน้า Comment.....	26
รูปที่ 44 User interface หน้า Your comment .....	27

## 1. ที่มาและความสำคัญ

ในปัจจุบันมหาวิทยาลัยขอนแก่นมีบุคลากรจำนวนมากทั้งนักศึกษา อาจารย์ พนักงาน และบุคคลภายนอก ทำให้การพูดคุยเกี่ยวกับหัวข้อต่างๆ มีมากตามไปด้วย ทั้งการตั้งคำถาม ซื่อขายของใช้ ประชาสัมพันธ์ และการพูดคุยส่วนใหญ่จะอยู่ในรูปแบบของแอปพลิเคชันโซเชียลมีเดีย ซึ่งมีมากมายต่างกันไป

จึงมีแนวคิดในการทำเว็บไซต์ที่รวบรวมหัวข้อต่างๆที่เกี่ยวข้องกับมหาวิทยาลัยขอนแก่น ที่สามารถพูดคุยตั้งกระทู้หัวข้อต่างๆเพื่อแลกเปลี่ยนความคิดเห็นมาไว้ที่เดียว เพื่อให้เกิดความสะดวกในการพูดคุยมากขึ้น

## 2. วัตถุประสงค์

1. เพื่อใช้เป็นสื่อในการศึกษาให้กับผู้ที่สนใจ เรื่อง Spring Boot ในการเขียน Web Blog
2. เพื่อพัฒนาการเขียน Spring Boot ใน Pattern ต่างๆ
3. เพื่อสร้าง Web Blog ในการใช้แลกเปลี่ยนความคิดเห็นของนักศึกษามหาวิทยาลัยขอนแก่น

## 3. ประโยชน์ที่คาดว่าจะได้รับ

1. เพิ่มความสะดวกในการสื่อสารพูดคุยเกี่ยวกับหัวข้อต่าง ๆ และแบ่งปันข้อมูลที่เกี่ยวข้องกับมหาวิทยาลัยขอนแก่น ที่เป็นที่น่าสนใจ
2. นักศึกษาและบุคลากรสามารถมาพูดคุย และแลกเปลี่ยนความคิดเห็นที่มีประโยชน์เพื่อพัฒนามหาวิทยาลัยและส่งเสริมความร่วมมือในการแก้ปัญหา
3. การประชาสัมพันธ์กิจกรรมต่างๆ ของมหาวิทยาลัย มีความทั่วถึงมากขึ้น

## 4. ทฤษฎีและเครื่องมือที่เกี่ยวข้อง

### 4.1 ทฤษฎีที่เกี่ยวข้อง

#### 4.1.1 Spring Boot

Spring Boot คือ Framework ใน Spring อันหนึ่ง สามารถช่วยทำให้สร้าง Web application หรือ Web service ได้ง่ายขึ้น เพราะ Spring Boot มี Auto Configuration ซึ่งช่วยลดความยุ่งยากในการกำหนดค่าต่างๆ และสามารถใช้งานได้ทันที เนื่องจาก Spring Boot มี Java Web Server ที่ built-in มาให้แล้ว

#### 4.1.2 Strategy Pattern

เป็นการสร้างการทำงานหลาย ๆ แบบ แยกออกเป็น class และมี interface ร่วมกัน ยกตัวอย่างเช่น การจัดเรียงข้อมูล ไม่ว่าจะเป็น merge sort, bubble sort, quick sort เราสามารถนำมาประยุกต์ใช้งานได้โดยการสร้างเพียง method เดียวและทำการกำหนด signature ให้เหมือนกัน

#### 4.1.3 Repository Pattern

คือการวางโครงสร้างของโค้ดรูปแบบหนึ่งเพื่อใช้แยก Logic สำหรับการเข้าถึงแหล่งข้อมูล (Data Source) ออกจาก Business layer โดยมี Repository Interface ทำหน้าที่เป็นตัวกลางในการติดต่อระหว่าง Business layer กับ Data source โดยจะดึงข้อมูลจาก Data source แล้วแปลงข้อมูลให้อยู่ในรูปแบบที่ทางฝั่ง Business layer สามารถนำไปใช้งานได้ทันที

#### 4.1.4 Factory Pattern

เป็นการกำหนด interface สำหรับสร้าง object แต่ให้ subclass สร้าง instance ได้เอง โดย Factory Method เป็นการยกหน้าที่การสร้าง instance ให้กับ subclass

#### 4.1.5 Spring Dependency Injection

เป็นเทคนิคหนึ่งในการทำ loose coupling dependency ให้กันระหว่าง class อย่างเช่น A class ต้องการเรียก B class , ตัว B class จะถือว่าเป็น dependency ของ A class. ส่วนนี้ก็เพื่อแยกการทำงานให้เล็กลง และ เมื่อต้องการ Function ที่ตัวเองไม่มีก็ทำการฉีด class อื่นเข้ามา

### 4.2 เครื่องมือที่เกี่ยวข้อง

#### 4.2.1 โปรแกรม visual studio code

คือโปรแกรมตัวหนึ่งที่เป็นเครื่องมือที่ช่วยพัฒนาซอฟต์แวร์และระบบต่างๆ ซึ่งสามารถติดต่อสื่อสารพูดคุยกับคอมพิวเตอร์ได้ในระดับหนึ่งแล้ว แต่ยังไม่สามารถพัฒนาเป็นระบบเองได้ เนื่องจากไม่ใคร่ซอฟต์แวร์ได้พัฒนาโปรแกรมและภาษาขึ้นมาควบคุมกันเพื่อให้ใช้งานได้ซึ่งกันและกัน ซึ่งนักโปรแกรมเมอร์จะนำเครื่องมือมาใช้ในการพัฒนาต่อยอดให้เกิดเป็นระบบต่างๆ หรือเป็นเว็บไซต์ และแอปพลิเคชันต่างๆ ได้

#### 4.2.2 Mysql Workbench

ระบบจัดการฐานข้อมูล หรือ Database Management System (DBMS) แบบข้อมูลเชิงสัมพันธ์ หรือ Relational Database Management System (RDBMS) ซึ่งเป็นระบบฐานข้อมูลที่จัดเก็บรวบรวมข้อมูลในรูปแบบตาราง โดยมีการแบ่งข้อมูลออกเป็นแถว (Row) และในแต่ละแถวแบ่งออกเป็นคอลัมน์ (Column) เพื่อเชื่อมโยงระหว่างข้อมูลในตารางกับข้อมูลในคอลัมน์ที่กำหนด

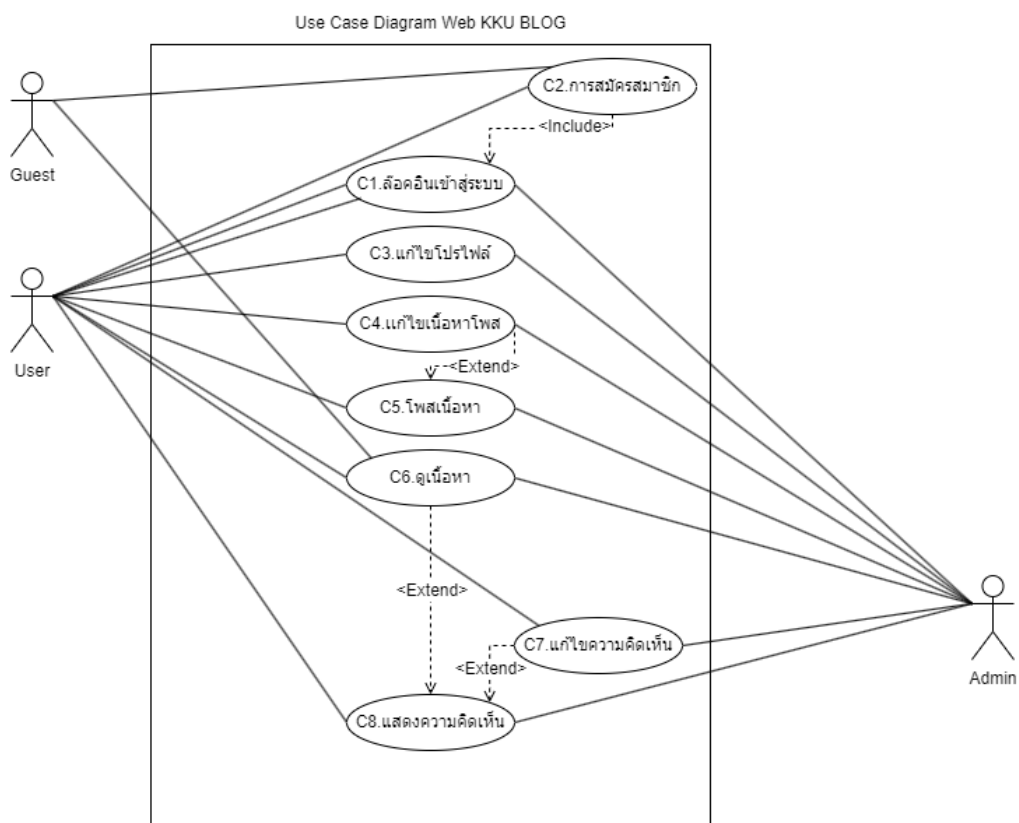
#### 4.2.3 โปรแกรม Microsoft Word

เป็นโปรแกรมประมวลผลคำเพื่องานการสร้างเอกสารได้อย่างมีประสิทธิภาพสะดวกและประหยัดเวลา เหมาะกับการพิมพ์เอกสารทุกประเภท เช่น จดหมาย ขອງจดหมาย บันทึกข้อความ รายงาน บทความ ประวัติย่อ และยังสามารถตรวจสอบ ทบทวน แก้ไข ปรับปรุง ความถูกต้องในการพิมพ์เอกสารได้อย่างง่ายดาย

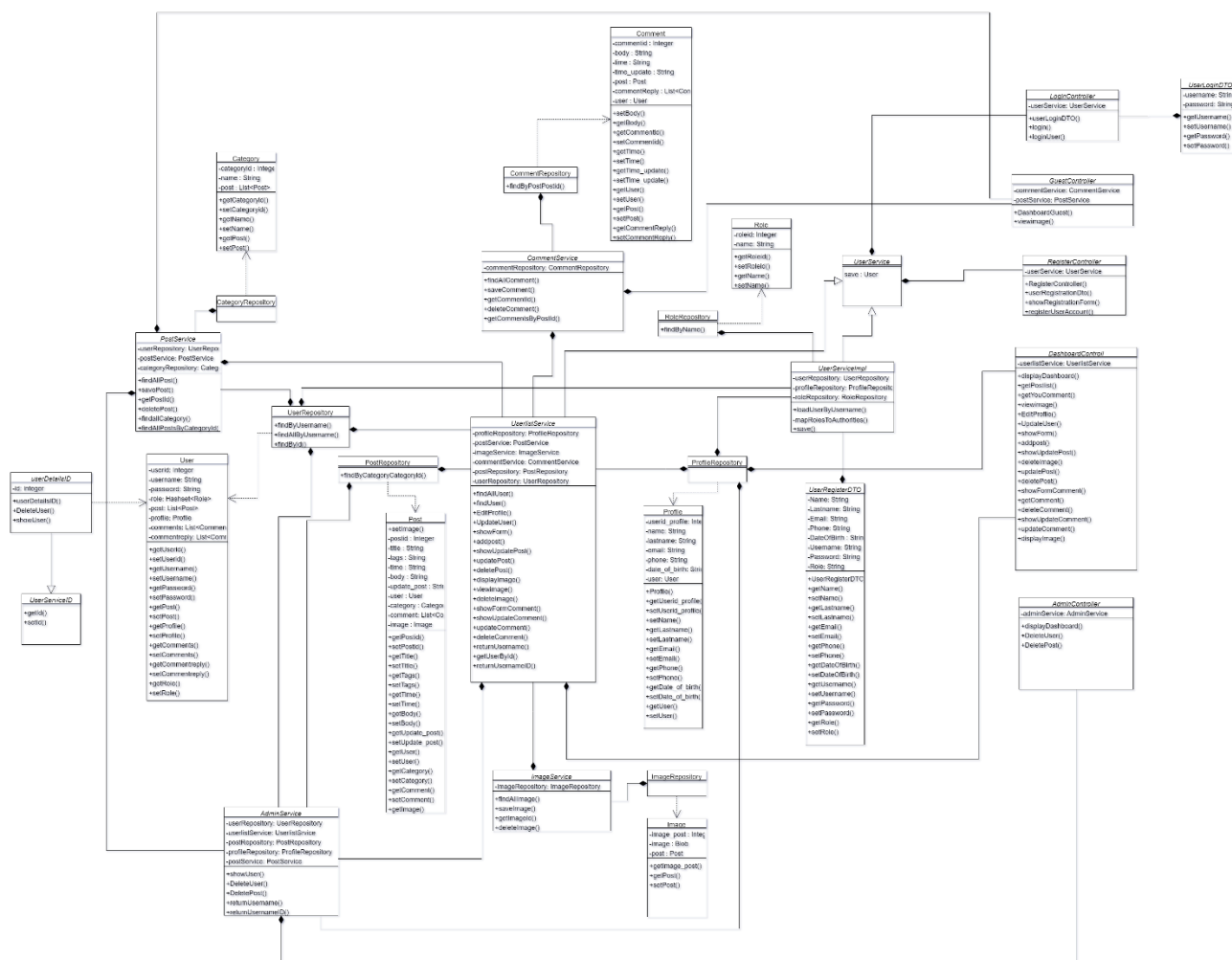


## 5. อธิบายฐานข้อมูล

### 5.1 Use case diagram

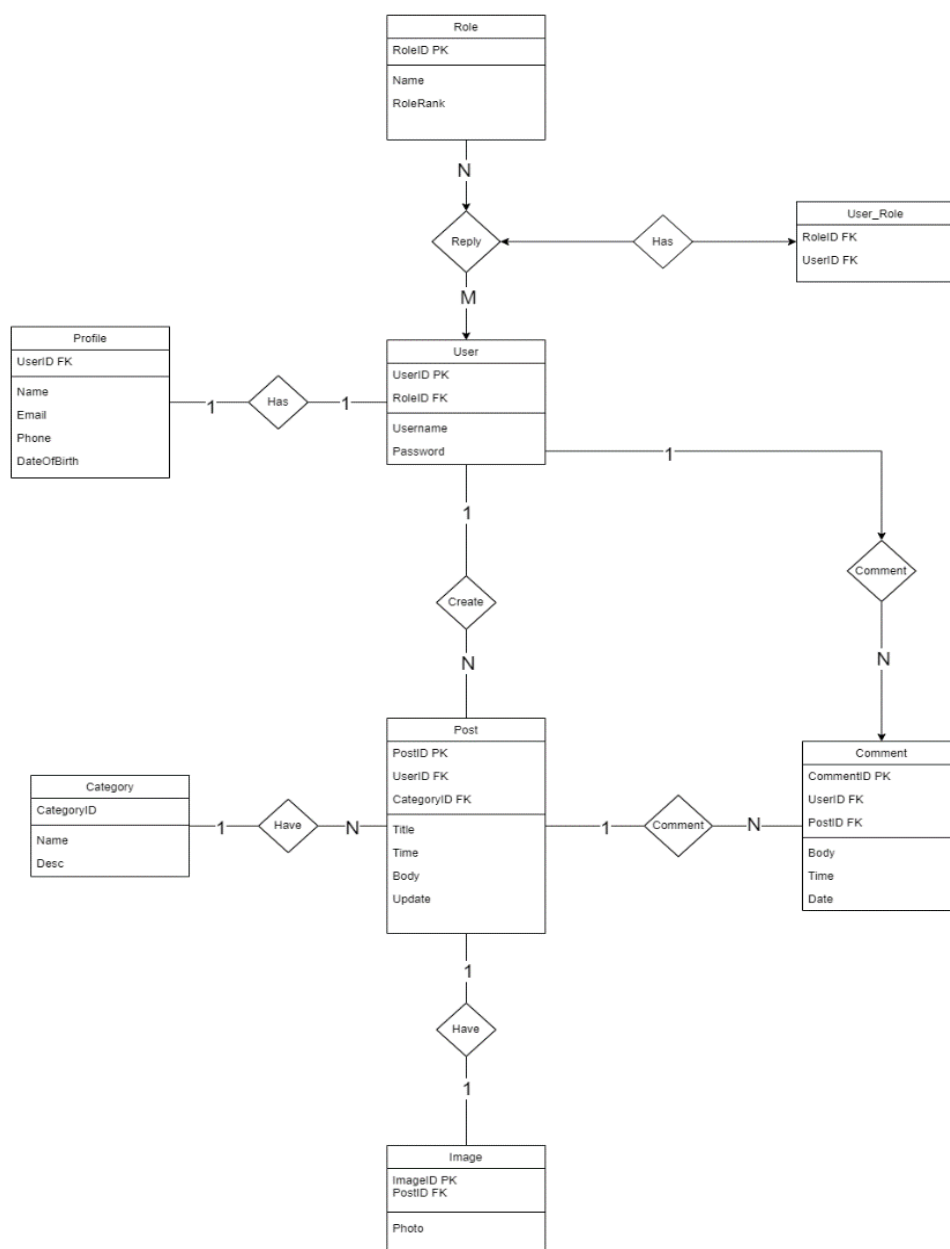


รูปที่ 1 Use case diagram



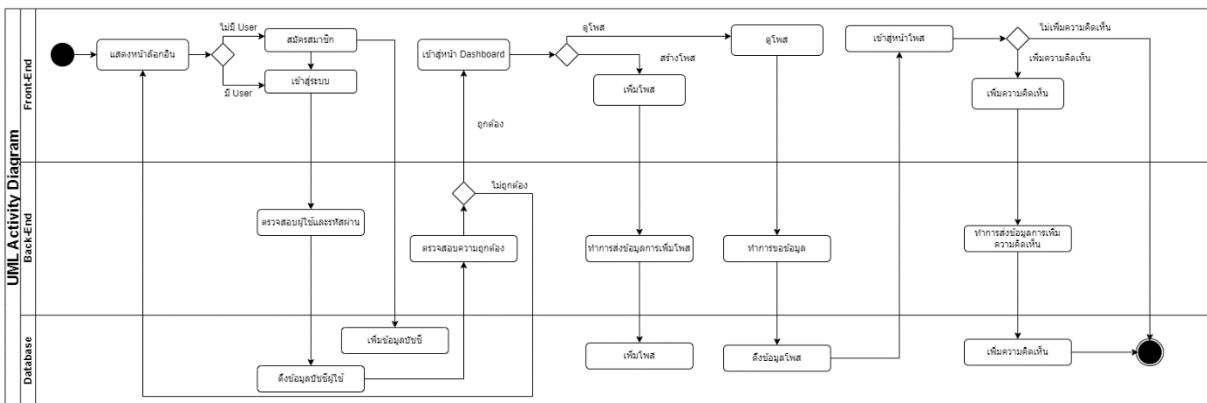
รูปที่ 2 Class diagram

## 5.3 ER-diagram



รูปที่ 3 ER-diagram

## 5.4 UML Activity Diagram



รูปที่ 4 UML Activity Diagram

## 6.อธิบายการสร้างระบบเว็บโดยใช้ Spring boot editor

เป็นการสร้างตัวรับข้อมูลจาก Database โดยชื่อ Entity คือ user มีการเชื่อม Many To Many กับ role และ One To Many กับ post , comment มีการเชื่อมแบบ One To One กับ Profile

```
@Entity
@Table(name = "user")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "UserID", unique = true, nullable = false)
    private int userID;
    @Column(name = "Username")
    private String username;
    @Column(name = "Password")
    private String password;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = "user_role", joinColumns = @JoinColumn(name = "UserID", referencedColumnName = "UserID"),
        inverseJoinColumns = @JoinColumn(name = "RoleID", referencedColumnName = "RoleID"))
    Set<Role> role = new HashSet<Role>();

    @OneToMany(targetEntity = Post.class, mappedBy = "user", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Post> post;

    @OneToOne(mappedBy = "user", cascade = CascadeType.ALL)
    @PrimaryKeyJoinColumn
    private Profile profile;

    @OneToMany(targetEntity = Comment.class, mappedBy = "user", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Comment> comments;

    @OneToMany(targetEntity = CommentReply.class, mappedBy = "user", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<CommentReply> commentReply;
}
```

รูปที่ 5 Class User

เป็นการสร้างตัวรับข้อมูลจาก Database โดยชื่อ Entity คือ profile มีการเชื่อม One To One กับ user

```
@Entity
@Table(name = "profile")
public class Profile {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "User_ID_Profile")
    private int userid_profile;
    @Column(name = "name")
    private String name;
    @Column(name = "Lastname")
    private String lastname;
    @Column(name = "Email")
    private String email;
    @Column(name = "Phone")
    private String phone;
    @Column(name = "date_of_birth")
    private String date_of_birth;

    @OneToOne
    @JoinColumn(name = "User_ID_Profile")
    private User user;

    public Profile() {
    }

    public int getUserid_profile() {
        return userid_profile;
    }

    public void setUserid_profile(int userid_profile) {
        this.userid_profile = userid_profile;
    }
}
```

รูปที่ 6 Class Profile

เป็นการสร้างตัวรับข้อมูลจาก Database โดยชื่อ Entity คือ role มีการเชื่อม

```
@Entity
@Table(name = "role")
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "roleid")
    private int roleid;
    @Column(name = "name")
    private String name;

    public int getRoleid() {
        return roleid;
    }

    public void setRoleid(int roleid) {
        this.roleid = roleid;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

รูปที่ 7 Class Role

เป็นการสร้างตัวรับข้อมูลจาก Database โดยชื่อ Entity คือ post มีการเชื่อม Many To One กับ user, category มีการเชื่อม One To Many กับ comment และ One To One กับ Image

```
@Entity
@Table(name = "post")
public class Post {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "PostID", unique = true, nullable = false)
    private int postId;
    @Column(name = "Title")
    private String title;
    @Column(name = "Tags")
    private String tags;
    @Column(name = "Time")
    private String time;
    @Column(name = "Body")
    private String body;
    @Column(name = "update_post")
    private String update_post;

    @ManyToOne(optional = false)
    @JoinColumn(name = "UserID")
    private User user;

    @ManyToOne(optional = false)
    @JoinColumn(name = "CategoryID")
    private Category category;

    @OneToMany(targetEntity = Comment.class, mappedBy = "post", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Comment> comment;

    @OneToOne(mappedBy = "post", cascade = CascadeType.REMOVE, orphanRemoval = true)
    @PrimaryKeyJoinColumn
    private Image image;

    public int getPostid() {
        return postId;
    }
}
```

รูปที่ 8 Class Post

เป็นการสร้างตัวรับข้อมูลจาก Database โดยชื่อ Entity คือ Image มีการเชื่อม One To One กับ post

```
@Entity
@Table(name = "image")
public class Image {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "image_post", unique = true, nullable = false)
    private int image_post;

    @Lob
    @Column(name = "image")
    private Blob image;

    @OneToOne
    @MapsId
    @JoinColumn(name = "image_post")
    private Post post;

    public int getImage_post() {
        return image_post;
    }

    public void setImage_post(int image_post) {
        this.image_post = image_post;
    }
}
```

รูปที่ 9 Class Image

เป็นการสร้างตัวรับข้อมูลจาก Database โดยชื่อ Entity คือ comment มีการเชื่อม Many To One กับ user, post

```
@Entity
@Table(name = "comment")
public class Comment {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "commentID")
    private int commentid;
    @Column(name = "Body")
    private String body;
    @Column(name = "Time")
    private String time;
    @Column(name = "time_update")
    private String time_update;

    @ManyToOne(optional = false)
    @JoinColumn(name="UserID")
    private User user;

    @ManyToOne(optional = false)
    @JoinColumn(name="PostID")
    private Post post;

    @OneToMany(targetEntity=CommentReply.class,mappedBy="comment",
    cascade = CascadeType.ALL,fetch = FetchType.LAZY)
    private List<CommentReply> commentReply;

    public int getCommentid() {
        return commentid;
    }

    public void setCommentid(int commentid) {
        this.commentid = commentid;
    }

    public String getBody() {
        return body;
    }
}
```

รูปที่ 10 Class Comment

เป็นการสร้างตัวรับข้อมูลจาก Database โดยชื่อ Entity คือ category มีการเชื่อม One To Many กับ post

```
@Entity
@Table(name = "category")
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "CategoryID")
    private Integer categoryid;
    @Column(name = "name")
    private String name;

    @OneToMany(targetEntity = Post.class, mappedBy = "category", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Post> post;

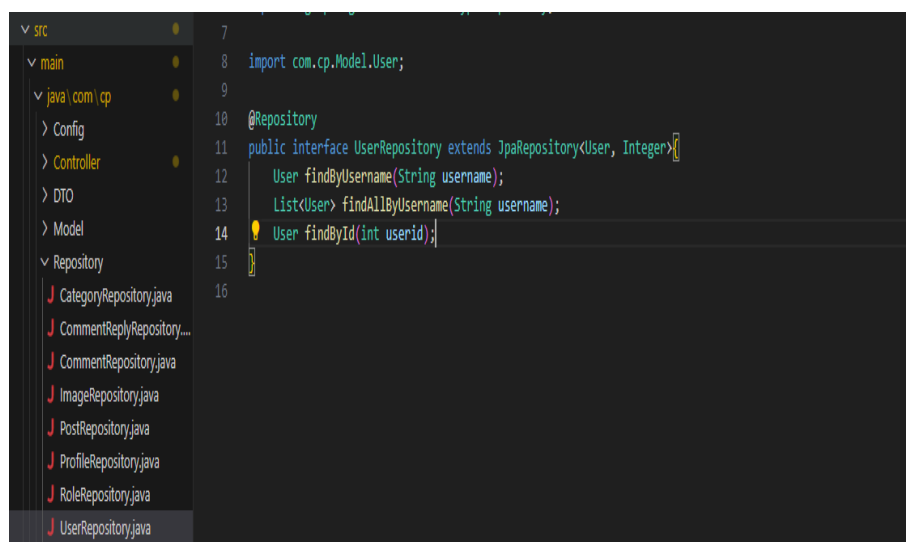
    public Integer getCategoryid() {
        return categoryid;
    }

    public void setCategoryid(Integer categoryid) {
        this.categoryid = categoryid;
    }

    public String getName() {
        return name;
    }
}
```

รูปที่ 11 Class Category

ทำการสร้าง Repository ทั้งหมดของทุก Class ที่มีการรับข้อมูลมาจาก Database เพื่อลบ แก้ไข อัปเดตข้อมูล



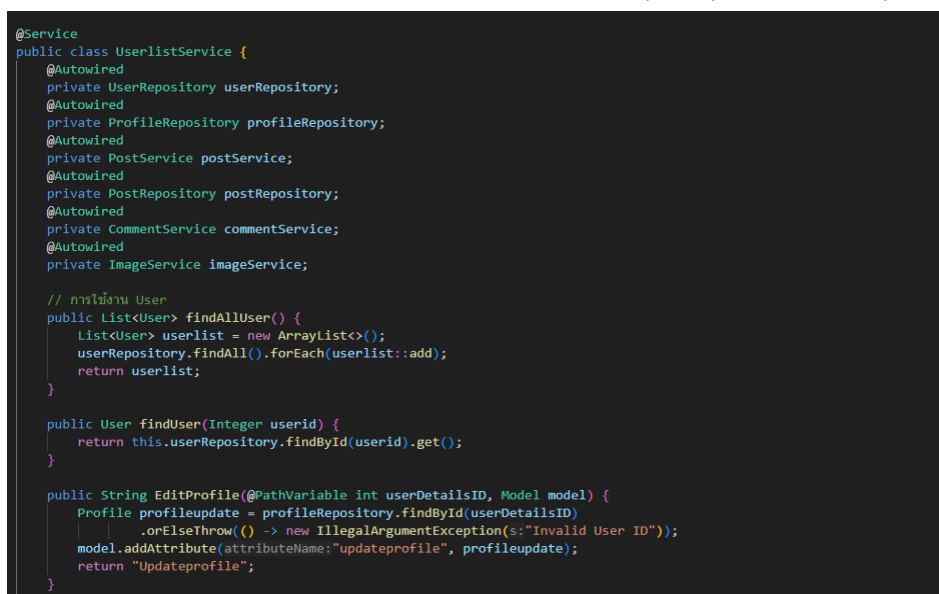
รูปที่ 12 Repository ทั้งหมด

ทำการดึงฟังก์ชันจาก Class UserRepository, ProfileRepository, PostService, PostRepository, CommentService, ImageService เพื่อจะใช้ฟังก์ชันนั้น ๆ

findAllUser จะทำการวนลูปเพื่อเก็บข้อมูล User ทั้งหมดแล้วส่งกลับค่าเป็น userList

findUser เป็นการใช้ userRepository ในการหาไอดีที่เราต้องการหาจาก Database

EditProfile ทำการรับค่า ID แล้วนำไปค้นหาใน Database แล้วส่งกลับ ID เป็น updateprofile ส่งไปที่ Updateprofile.html



รูปที่ 13 Class UserlistService



UpdateUser ใช้ในการอัปเดตข้อมูลที่มีการรับค่ามาจาก Updateprofile.html

displayDashboard เป็นการแสดงข้อมูล Username ของ User เก็บไว้ใน userDetails, UserID ของ User, Profile ของ User เก็บไว้ใน profile และ Category ทั้งหมด เก็บไว้ที่ category 1-6 แล้วทำการส่งค่าไปที่ Dashboard.html

```
public String UpdateUser(@PathVariable int userDetailsID, @ModelAttribute Profile updateprofile, Model Model) {
    User userupdate = userRepository.findById(userDetailsID);
    Profile profileUpdate = userupdate.getProfile();
    profileUpdate.setName(updateprofile.getName());
    profileUpdate.setEmail(updateprofile.getEmail());
    profileUpdate.setPhone(updateprofile.getPhone());
    profileUpdate.setDate_of_birth(updateprofile.getDate_of_birth());
    profileRepository.save(profileUpdate);
    return "redirect:/Dashboard";
}

// แสดงหน้าหลัก
@GetMapping("/Dashboard")
public String displayDashboard(Model model) {
    String user = returnUsername();
    model.addAttribute(attributeName:"userDetails", user);
    Integer userid = returnUsernameID();
    Profile profile = profileRepository.findById(userid)
        .orElseThrow(() -> new RuntimeException("Profile not found for user with ID " + userid));
    model.addAttribute(attributeName:"profile", profile);

    /*
     * List<Post> postlist = (ArrayList<Post>) postService.findAllPost();
     * model.addAttribute("postshow", postlist);
     */

    List<Post> category1 = postService.findAllPostsByCategoryId(categoryId:1);
    model.addAttribute(attributeName:"category1", category1);
    List<Post> category2 = postService.findAllPostsByCategoryId(categoryId:2);
    model.addAttribute(attributeName:"category2", category2);
    List<Post> category3 = postService.findAllPostsByCategoryId(categoryId:3);
    model.addAttribute(attributeName:"category3", category3);
    List<Post> category4 = postService.findAllPostsByCategoryId(categoryId:4);
    model.addAttribute(attributeName:"category4", category4);
    List<Post> category5 = postService.findAllPostsByCategoryId(categoryId:5);
    model.addAttribute(attributeName:"category5", category5);
    List<Post> category6 = postService.findAllPostsByCategoryId(categoryId:6);
    model.addAttribute(attributeName:"category6", category6);
    return "Dashboard";
}
```

รูปที่ 14 Class UserlistService (ต่อ)

getPostlist เป็นแสดง Post ของตัวเองทั้งหมด โดยการ ส่งกลับค่า username ไปเก็บที่ userDetails และส่งกลับ userid  
เจ้าของ User แล้วนำ userpost ไปวนลูปหา user ที่ตรงกัน แล้วนำไปเก็บที่ Postlist

getYouComment ทำการค้นหา PostID ของเจ้าของ User แล้วทำการวนลูปหาใน Comment ว่ามีส่วนไหนตรงกับ PostID  
เจ้าของบ้างแล้วนำไปเก็บที่ Commentlist หลังจากนั้นส่งไปที่ Yourcomment.html

showForm ทำการแสดงผลข้อมูลทั้งหมดที่มีอยู่ใน Entity Post แล้วส่งกลับไป Post.html

```
//userdetail/findAllPost
public String getPostlist(Model model) {
    String username = returnUsername();
    Integer userid = returnUserID();
    User user = getUserById(userid);
    List<Post> postlist = postService.findAllPost();
    List<Post> userpost = new ArrayList<>();
    for (Post post : postlist) {
        if (post.getUser() != null && post.getUser().equals(user))
            userpost.add(post);
    }

    model.addAttribute(attributeName:"Postlist", userpost);
    model.addAttribute(attributeName:"userDetails", username);
    return "Yourpost";
}

//userdetail/Comment
public String getYouComment(@PathVariable int postid, Model model) {
    Post post = postService.getPostId(postid);
    model.addAttribute(attributeName:"lookpost", post);
    Integer userid = returnUserID();
    User user = getUserById(userid);
    List<Comment> commentlist = commentService.getCommentsByPostId(postid);
    List<Comment> commentuser = new ArrayList<>();
    for (Comment comment : commentlist) {
        if (comment.getUser() != null && comment.getUser().equals(user))
            commentuser.add(comment);
    }
    model.addAttribute(attributeName:"Commentlist", commentuser);
    return "Yourcomment";
}

// การดูรูป Post
public String showForm(Model model) {
    Post post = new Post();
    model.addAttribute(attributeName:"post", post);
    List<Category> categorylist = (List<Category>) postService.findAllCategory();
    model.addAttribute(attributeName:"categorylist", categorylist);
    return "Post";
}
```

รูปที่ 15 Class UserlistService (ต่อ)

Addpost result.hasError เป็นการเช็คค่าถ้าเกิดไม่เจอข้อมูลที่ตามหาจะทำการส่งกลับหน้า Post และทำการส่งกลับค่า UserID  
เพื่อให้บอกว่า Post ที่สร้างใหม่เป็นเจ้าของ User ไหน รูปภาพก็ทำเหมือนกัน

showUpdatePost เป็นการส่งข้อมูล Post ที่ต้องการแก้ไขไปที่ Updatepost.html

```
public String addpost(@Validated Post post, @RequestParam("imageFile") MultipartFile imageFile,
    BindingResult result, Model model) throws IOException, ServletException, SQLException {
    if (result.hasErrors()) {
        List<Category> categorylist = (List<Category>) postService.findAllCategory();
        model.addAttribute(attributeName:"categorylist", categorylist);
        Integer userid = returnUserID();
        model.addAttribute(attributeName:"userid", userid);
        return "Post";
    }

    Integer userid = returnUserID();
    User user = getUserById(userid);
    post.setUser(user);
    postService.savePost(post);

    Image image = new Image();
    Blob blobImage = new SerialBlob(imageFile.getBytes());
    image.setImage(blobImage);
    image.setPost(post);
    imageService.saveImage(image);
    return "redirect:/Dashboard";
}

public String showUpdatePost(@PathVariable int postid, Model model) {
    Post postupdate = postRepository.findById(postid)
        .orElseThrow(() -> new IllegalArgumentException("Invalid Post ID" + postid));
    model.addAttribute(attributeName:"post", postupdate);
    List<Category> categorylist = (List<Category>) postService.findAllCategory();
    model.addAttribute(attributeName:"categorylist", categorylist);
    return "Updatepost";
}
```

รูปที่ 16 Class UserlistService (ต่อ)

updatePost เป็นส่วนในการเซฟข้อมูลทั้งหมดหลังจากที่ทำการแก้ไขแล้ว โดยตรงที่ if(!ImageFile.isEmpty) เป็นการเช็คว่ามีรูปภาพอยู่แล้วจะทำการแสดงเป็นภาพเดิม ส่วน Image == null คือถ้าไม่มีการเปลี่ยนแปลงจะใช้ภาพเดิม แต่หากมีการเปลี่ยนแปลงจะทำการใช้ภาพใหม่

DeletePost ทำการค้นหา PostID แล้วหลังจากนั้นทำการใช้ postService ในการลบ Post นั้น

```
public String updatePost(@PathVariable int postid, @ModelAttribute Post post,
    @RequestParam("imagefile") MultipartFile imageFile, BindingResult result, Model model)
    throws IOException, ServletException, SQLException {
    Post postup = postService.getPostId(postid);
    if (result.hasErrors()) {
        post.setPostId(postid);
        return "Updatepost";
    }
    postup.setTitle(post.getTitle());
    postup.setBody(post.getBody());
    postup.setTags(post.getTags());
    postup.setUpdate_post(post.getUpdate_post());
    postup.setCategory(post.getCategory());
    postRepository.save(postup);

    if (imageFile.isEmpty()) {
        try {
            Blob blobImage = new SerialBlob(imageFile.getBytes());
            Image images = postup.getImage();
            if (images == null) {
                images = new Image();
                images.setPost(postup);
            }
            images.setImage(blobImage);
            imageService.saveImage(images);
        } catch (IOException | SQLException e) {
            e.printStackTrace();
        }
    }
    return "redirect:/Dashboard/Yourpost";
}

public String deletePost(@PathVariable("postid") Integer postid, Model model) {
    Post post = postService.getPostId(postid);
    postService.deletePost(post);
    return "redirect:/Dashboard/Yourpost";
}
```

### รูปที่ 17 Class UserService (ต่อ)

displayImage เป็นส่วนในการใช้แสดงรูปภาพ

viewimage เป็นการค้นหา Postid ที่ต้องการมา แล้วทำการดึงข้อมูล Post ,Comment, Image ที่ตรงกันมาเพื่อแสดง

deleteImage เป็นการลบรูปภาพที่ต้องการลบใน Post นั้น

ShowFomeComment เป็นการส่งค่า PostID และ ทำการสร้าง Comment ใหม่ส่งกลับไป Comment.html

```
// Image
public ResponseEntity<byte[]> displayImage(@RequestParam("image_post") int image_post)
    throws IOException, SQLException {
    Image image = imageService.getImageId(image_post);
    byte[] imageBytes = null;
    imageBytes = image.getImage().getBytes(pos1, (int) image.getImage().length());
    return ResponseEntity.ok().contentType(MediaType.IMAGE_PNG).body(imageBytes);
}

// LookPost
public ModelAndView viewImage(@PathVariable("postid") Integer postid, Model model) {
    ModelAndView mv = new ModelAndView(viewName: "lookpost");
    Post post = postService.getPostId(postid);
    model.addAttribute(attributeName: "lookpost", post);
    Image image = post.getImage();
    mv.addObject(attributeName: "imageview", image);
    User userpost = post.getUser();
    model.addAttribute(attributeName: "postUser", userpost);
    List<Comment> comments = commentService.getCommentsByPostId(postid);
    model.addAttribute(attributeName: "comment", comments);
    return mv;
}

public String deleteImage(@PathVariable("imageid") Integer imageid, @PathVariable int postid, Model model) {
    Image image = imageService.getImageId(imageid);
    imageService.deleteImage(image);
    return "redirect:/Dashboard/edit_post/postid";
}

// Comment
public String showFomeComment(@PathVariable Integer postid, Model model) {
    Post postcommentid = postRepository.findById(postid)
        .orElseThrow(() -> new IllegalArgumentException("Invalid Post ID" + postid));
    model.addAttribute(attributeName: "post", postcommentid);
    Comment comment = new Comment();
    model.addAttribute(attributeName: "comment", comment);
    return "Comment";
}
```

### รูปที่ 18 Class UserService (ต่อ)

getComment เป็นการเซฟ Comment ที่ทำการเพิ่มเข้ามาจาก Comment.html หลังจากทำเสร็จแล้วจะส่งไปที่หน้า PostshowUpdateComment จะส่งข้อมูลที่ Comment ที่เรากำลังแก้ไขไปที่เว็บ Updatecomment.html updateComment หลังจากที่เราทำการแก้ไขข้อมูลเรียบร้อยแล้ว Updatecomment จะส่งข้อมูลมาที่ฟังก์ชันนี้ ฟังก์ชันนี้จะทำการบันทึกข้อมูลเอาไว้

DeleteComment ทำการค้นหา Comment ด้วย Commentid หลังจากนั้นจะทำการลบด้วย commentService.deleteComment

```
public String getComment(@Validated Comment comment, @PathVariable int postId,
    BindingResult result, Model model) throws IOException, ServletException, SQLException {
    if (result.hasErrors()) {
        Integer userId = returnUsernameID();
        model.addAttribute(attributeName "userId", userId);
        return null;
    }
    Integer userId = returnUsernameID();
    Post post = postService.getPostId(postId);
    User user = getUserById(userId);
    comment.setPost(post);
    comment.setUser(user);
    commentService.saveComment(comment);
    return "redirect://Dashboard/(postId)";
}

public String showUpdateComment(@PathVariable int commentId, @PathVariable int postId, Model model) {
    Comment commentupdate = commentService.getCommentId(commentId);
    model.addAttribute(attributeName "commentupdate", commentupdate);
    Post post = postService.getPostId(postId);
    model.addAttribute(attributeName "postcommentupdate", post);
    return "Updatecomment";
}

public String updateComment(@PathVariable int postId, @PathVariable int commentId, @ModelAttribute Comment comment,
    Model model) {
    Comment getcommentid = commentService.getCommentId(commentId);
    getcommentid.setBody(comment.getBody());
    getcommentid.setLine_update(comment.getLine_update());
    commentService.saveComment(getcommentid);
    return "redirect://Dashboard/(postId)/Yourcomment/";
}

public String deleteComment(@PathVariable("postId") Integer postId, @PathVariable("comment") Integer commentId,
    Model model) {
    Comment comment = commentService.getCommentId(commentId);
    commentService.deleteComment(comment);
    return "redirect://Dashboard/(postId)/Yourcomment/";
}
```

### รูปที่ 19 Class UserService (ต่อ)

returnUsername จะเป็นการค้นหา Username เจ้าของ User เพื่อนำมาแสดง

returnUsernameID จะเป็นการค้นหา UserID เจ้าของ User เพื่อนำมาแสดง

```
// การคืน Username user id
public String returnUsername() {
    SecurityContext securityContext = SecurityContextHolder.getContext();
    UserDetails user = (UserDetails) securityContext.getAuthentication().getPrincipal();
    User users = userRepository.findByUsername(user.getUsername());
    return users.getUsername();
}

public User getUserById(int userId) {
    return userRepository.findById(userId);
}

public int returnUsernameID() {
    SecurityContext securityContext = SecurityContextHolder.getContext();
    UserDetails userDetails = (UserDetails) securityContext.getAuthentication().getPrincipal();
    if (userDetails instanceof UserServiceID) {
        UserServiceID userDetailsService = (UserServiceID) userDetails;
        int userId = userDetailsService.getId();
        return userId;
    }
    return -1;
}
```

### รูปที่ 20 Class UserService (ต่อ)

loadUserByUsername จะทำการค้นหา Username เจ้าของ User แล้วเก็บไว้ใน userDetails หากไม่เจอจะทำการแจ้ง Error

```

@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private RoleRepository roleRepository;
    @Autowired
    private ProfileRepository profileRepository;

    private BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException(msg: "Invalid Username or Password");
        }
        UserServiceID userDetails = new UserDetailsID(
            user.getUsername(),
            user.getPassword(),
            mapRolesToAuthorities(user.getRole());
        userDetails.setId(user.getId());
        return userDetails;
    }
}

```

รูปที่ 21 Class UserServiceImpl

mapRolesToAuthorities จะทำการค้นหา Role ที่ตรงกับ User ที่ Login เข้ามาว่าตรงกับ User หรือ Admin

save จะเป็นการรับข้อมูลมาจาก UserRegisterDTO แล้วนำไปเพิ่ม User ใหม่ขึ้นมา

```

private Collection<? extends GrantedAuthority> mapRolesToAuthorities(Set<Role> roles) {
    return roles.stream().map(role -> new SimpleGrantedAuthority(role.getName())).collect(Collectors.toList());
}

@Override
public User save(UserRegisterDTO userRegisterDTO) {
    Role role = new Role();
    if (userRegisterDTO.getRole().equals(anObject: "USER"))
        role = roleRepository.findByName(name: "USER");
    else if (userRegisterDTO.getRole().equals(anObject: "ADMIN"))
        role = roleRepository.findByName(name: "ADMIN");
    User user = new User();
    user.setUsername(userRegisterDTO.getUsername());
    user.setPassword(passwordEncoder.encode(userRegisterDTO.getPassword()));
    user.setRole(role);
    user = userRepository.save(user);
    Profile profile = new Profile();
    profile.setName(userRegisterDTO.getName());
    profile.setLastname(userRegisterDTO.getLastname());
    profile.setEmail(userRegisterDTO.getEmail());
    profile.setPhone(userRegisterDTO.getPhone());
    profile.setDate_of_birth(userRegisterDTO.getDateOfBirth());
    profile.setUser(user);
    profile = profileRepository.save(profile);
    return user;
}

```

รูปที่ 22 Class UserServiceImpl (ต่อ)

ใช้ในการตรวจสอบ Username และ Password ในการ Login ว่าตรงกันหรือไม่

```
public class UserLoginDTO {
    private String username;
    private String password;

    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

รูปที่ 23 Class UserLoginDTO

ใช้ในการรับข้อมูลที่ใส่เข้ามาหลังจากนั้น UserServiceImpl.save จะนำไปใช้ในการสร้าง User ใหม่

```
public class UserRegisterDTO {
    private String Name;
    private String Lastname;
    private String Email;
    private String Phone;
    private String DateOfBirth;
    private String Username;
    private String Password;
    private String Role;

    public UserRegisterDTO() {
        super();
    }

    public UserRegisterDTO(String Role) {
        super();
        this.Role = Role;
    }

    public String getName() {
        return Name;
    }

    public void setName(String name) {
        Name = name;
    }

    public String getLastName() {
        return Lastname;
    }

    public void setLastName(String lastname) {
        Lastname = lastname;
    }

    public String getEmail() {
        return Email;
    }
}
```

รูปที่ 24 Class UserRegisterDTO

จะเป็นการดึงฟังก์ชันต่างๆ จาก AdminService มาใช้

```
@Controller
public class AdminController {
    @Autowired
    AdminService adminService;

    @GetMapping("/Admin")
    public String displayDashboard(Model model){
        return adminService.displayDashboard(model);
    }

    @GetMapping("/Admin/DeleteUser/{userid}")
    public String DeleteUser(@PathVariable int userid, Model model){
        return adminService.DeleteUser(userid, model);
    }

    @GetMapping("/Admin/DeletePost/{postid}")
    public String DeletePost(@PathVariable int postid, Model model){
        return adminService.DeletePost(postid, model);
    }
}
```

รูปที่ 25 Class AdminController

จะเป็นการดึงฟังก์ชันต่างๆ จาก UserlistService มาใช้

```
@Controller
public class DashboardControll {
    @Autowired
    private UserlistService userlistService;

    // หน้าหลัก
    @GetMapping("/")
    public String displayDashboard(Model model) {
        return userlistService.displayDashboard(model);
    }

    // หน้าแสดงโพสของฉัน
    @GetMapping("/Dashboard/Yourpost")
    public String getPostlist(Model model) {
        return userlistService.getPostlist(model);
    }

    // หน้าแสดงคอมเม้นท์ของโพส
    @GetMapping("/Dashboard/{postid}/Youcomment/")
    public String getYouComment(@PathVariable int postid, Model model) {
        return userlistService.getYouComment(postid, model);
    }

    @GetMapping("/Dashboard/{postid}")
    public ModelAndView viewimage(@PathVariable("postid") Integer postid, Model model) {
        return userlistService.viewimage(postid, model);
    }

    @GetMapping("/Dashboard/UpdateProfile/{userDetailsID}")
    public String EditProfile(@PathVariable int userDetailsID, Model model) {
        return userlistService.EditProfile(userDetailsID, model);
    }

    @PostMapping("/Dashboard/UpdateProfile/{userDetailsID}")
    public String UpdateUser(@PathVariable int userDetailsID, @ModelAttribute Profile updateprofile, Model model) {
        return userlistService.UpdateUser(userDetailsID, updateprofile, model);
    }
}
```

รูปที่ 26 Class DashboardControll

```

@GetMapping("/Dashboard/new_post")
public String showForm(Model model) {
    return userListService.showForm(model);
}

@PostMapping("/Dashboard/new_post")
public String addPost(@Validated Post post, @RequestParam("imageFile") MultipartFile imageFile,
    BindingResult result, Model model)
    throws IOException, ServletException, SQLException {
    return userListService.addPost(post, imageFile, result, model);
}

@GetMapping("/Dashboard/edit_post/{postId}")
public String showUpdatePost(@PathVariable int postId, Model model) {
    return userListService.showUpdatePost(postId, model);
}

@GetMapping("/Dashboard/{postId}/edit_post/delete_image/{imageId}")
public String deleteImage(@PathVariable("imageId") Integer imageId, @PathVariable int postId, Model model) {
    return userListService.deleteImage(imageId, postId, model);
}

@PostMapping("/Dashboard/edit_post/{postId}")
public String updatePost(@PathVariable int postId, @ModelAttribute Post post,
    @RequestParam("imageFile") MultipartFile imageFile, BindingResult result, Model model)
    throws IOException, ServletException, SQLException {
    return userListService.updatePost(postId, post, imageFile, result, model);
}

@GetMapping("/Dashboard/delete_post/{postId}")
public String deletePost(@PathVariable("postId") Integer postId, Model model) {
    return userListService.deletePost(postId, model);
}

@GetMapping("/Dashboard/{postId}/Comment")
public String showFormComment(@PathVariable("postId") Integer postId, Model model) {
    return userListService.showFormComment(postId, model);
}

```

รูปที่ 27 Class DashboardControll (ต่อ)

```

@GetMapping("/Dashboard/{postId}/Comment")
public String showFormComment(@PathVariable("postId") Integer postId, Model model) {
    return userListService.showFormComment(postId, model);
}

@PostMapping("/Dashboard/{postId}/Comment")
public String getComment(@Validated Comment comment, @PathVariable int postId,
    BindingResult result, Model model) throws IOException, ServletException, SQLException {
    return userListService.getComment(comment, postId, result, model);
}

@GetMapping("/Dashboard/{postId}/delete_comment/{commentId}")
public String deleteComment(@PathVariable("postId") Integer postId, @PathVariable("commentId") Integer commentId,
    Model model) {
    return userListService.deleteComment(postId, commentId, model);
}

@GetMapping("/Dashboard/{postId}/edit_comment/{commentId}")
public String showUpdateComment(@PathVariable int commentId, @PathVariable int postId, Model model) {
    return userListService.showUpdateComment(commentId, postId, model);
}

@PostMapping("/Dashboard/{postId}/edit_comment/{commentId}")
public String updateComment(@PathVariable int postId, @PathVariable int commentId, @ModelAttribute Comment comment,
    Model model) {
    return userListService.updateComment(postId, commentId, comment, model);
}

@GetMapping("/display")
public ResponseEntity<byte[]> displayImage(@RequestParam("image_post") int image_post)
    throws IOException, SQLException {
    return userListService.displayImage(image_post);
}

```

รูปที่ 28 Class DashboardControll (ต่อ)



จะเป็นการนำฟังก์ชันของ PostService และ CommentService มาใช้ในการดู Post และ Comment

```
@Controller
public class GuestController {
    @Autowired
    private PostService postService;
    @Autowired
    private CommentService commentService;

    @GetMapping("/Dashboard/Guest")
    public String DashboardGuest(Model model) {
        List<Post> category1 = postService.findAllPostsByCategoryId(categoryId:1);
        model.addAttribute(attributeName:"category1", category1);
        List<Post> category2 = postService.findAllPostsByCategoryId(categoryId:2);
        model.addAttribute(attributeName:"category2", category2);
        List<Post> category3 = postService.findAllPostsByCategoryId(categoryId:3);
        model.addAttribute(attributeName:"category3", category3);
        List<Post> category4 = postService.findAllPostsByCategoryId(categoryId:4);
        model.addAttribute(attributeName:"category4", category4);
        List<Post> category5 = postService.findAllPostsByCategoryId(categoryId:5);
        model.addAttribute(attributeName:"category5", category5);
        List<Post> category6 = postService.findAllPostsByCategoryId(categoryId:6);
        model.addAttribute(attributeName:"category6", category6);
        return "Guest";
    }

    @GetMapping("/Dashboard/{postId}/Guest")
    public ModelAndView viewImage(@PathVariable("postId") Integer postId, Model model) {
        ModelAndView mv = new ModelAndView(viewName:"lookpostGuest");
        Post post = postService.getPostId(postId);
        model.addAttribute(attributeName:"lookpost", post);
        Image image = post.getImage();
        mv.addObject(attributeName:"Imageview", image);
        User userpost = post.getUser();
        model.addAttribute(attributeName:"postUser", userpost);
        List<Comment> comments = commentService.getCommentsByPostId(postId);
        model.addAttribute(attributeName:"Comment", comments);
        return mv;
    }
}
```

รูปที่ 29 Class GuestController

ใช้ UserLoginDTO ในการรับค่าที่เขียนเข้ามา หลังจากนั้นจะใช้ฟังก์ชันจาก Class UserService ในการตรวจสอบข้อมูลว่า Username และ Password ตรงกันกับฐานข้อมูลหรือไม่

```
@Controller
@RequestMapping("/login")
public class LoginController {
    @Autowired
    private UserService userService;

    @ModelAttribute("user")
    public UserLoginDTO userLoginDTO(){
        return new UserLoginDTO();
    }

    @GetMapping
    public String login(){
        return "Login";
    }

    @PostMapping
    public void loginUser(@ModelAttribute("user")
        UserLoginDTO userLoginDTO){
        userService.loadUserByUsername(userLoginDTO.getUsername());
    }
}
```

รูปที่ 30 Class LoginController

ใช้ userRegisterDTO ในการรับค่าเข้ามาหลังจากนั้นใช้ฟังก์ชันจาก Class UserService ในการบันทึกลงในฐานข้อมูลของ User

```
@Controller
@RequestMapping("/register")
public class RegisterController {

    private UserService userService;

    public RegisterController(UserService userService) {
        super();
        this.userService = userService;
    }

    @ModelAttribute("user")
    public UserRegisterDTO userRegistrationDto() {
        return new UserRegisterDTO();
    }

    @GetMapping
    public String showRegistrationForm() {
        return "Register";
    }

    @PostMapping
    public String registerUserAccount(@ModelAttribute("user") UserRegisterDTO registerDTO) {
        userService.save(registerDTO);
        return "redirect:/login";
    }
}
```

รูปที่ 31 Class RegisterController

เป็นการให้สิทธิ์ว่าใครสามารถเข้าหน้าเว็บไหนได้บ้าง โดยส่วนที่จะเข้าได้ทุกคนคือ register, dashboardGuest และหากไม่ login แต่เข้าไปที่หน้าอื่นจะทำการดึงไปที่หน้า Login แต่หาก Login แล้วจะสามารถเข้าได้ทุกหน้าโดยจะไม่ดึงไปที่หน้า Login

```
@SuppressWarnings("deprecation")
@Configuration
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserServiceImpl customUserServiceImpl;

    @Autowired
    AuthenticationSuccessHandler successHandler;

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider auth = new DaoAuthenticationProvider();
        auth.setUserDetailsService(customUserServiceImpl);
        auth.setPasswordEncoder(passwordEncoder());
        return auth;
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(authenticationProvider());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers(...antPatterns: "/register", "/Dashboard/Guest", "/Dashboard/(postid)/Guest", "/display").permitAll() // Allow unau
            .anyRequest().authenticated()
            .and().formLogin().loginPage("/login").successHandler(successHandler).permitAll()
            .and().logout().invalidateHttpSession(true).clearAuthentication(true)
            .logoutRequestMatcher(new AntPathRequestMatcher(pattern: "/logout")).logoutSuccessUrl("/login?logout")
            .permitAll();
    }
}
```

รูปที่ 32 Class SpringSecurityConfig

หลังจากที่เรา Login เข้ามาแล้วจะทำการตรวจสอบว่า User ตรงกับ User หรือ Admin หากตรงกับ User จะนำไปที่หน้า Dashboard ถ้าตรงกับ Admin จะนำไปที่หน้า Admin

```
@Component
public class CustomSuccessHanler implements AuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
        Authentication authentication) throws IOException, ServletException {

        String redirectUrl = null;

        Collection<? extends GrantedAuthority> authorities = authentication.getAuthorities();
        for (GrantedAuthority grantedAuthority : authorities){
            if (grantedAuthority.getAuthority().equals(anObject:"USER")){
                redirectUrl = "/Dashboard";
                break;
            }else if (grantedAuthority.getAuthority().equals(anObject:"ADMIN")){
                redirectUrl = "/Admin";
                break;
            }
        }
        if (redirectUrl == null){
            throw new IllegalStateException();
        }
        new DefaultRedirectStrategy().sendRedirect(request, response, redirectUrl);
    }
}
```

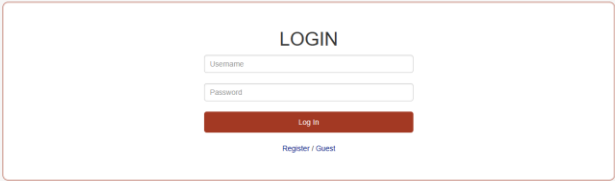
รูปที่ 33 Class CustomSuccessHanler

## 7.User interface

ระบบการทำงานของ User interface

### 7.1 login

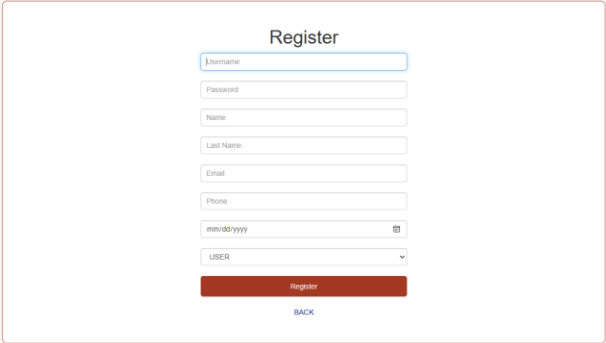
การทำงานเพื่อให้สมัคร login จาก username และ password ที่สมัครไว้



รูปที่ 34 User interface หน้า login

### 7.2 Register

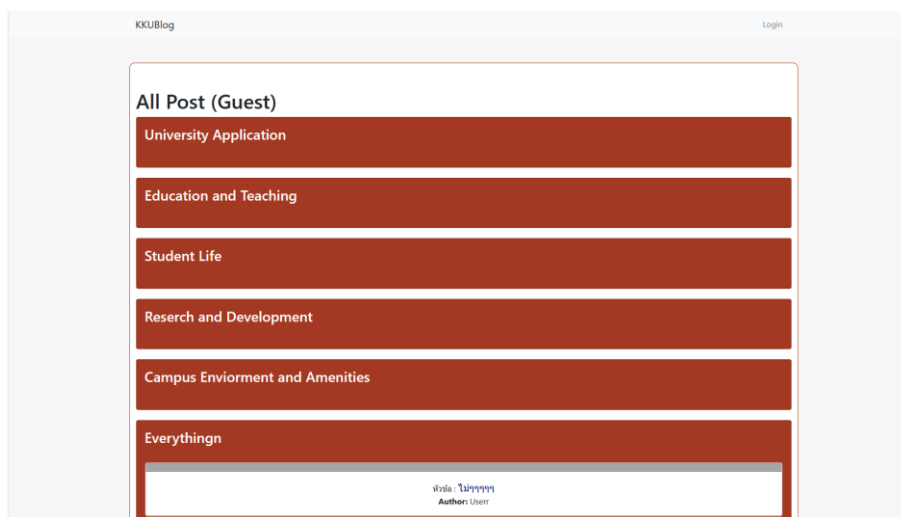
การทำงานเพื่อให้ผู้เยี่ยมชมที่ต้องการสมัครสมาชิกกรอกข้อมูลเพื่อสมัคร



รูปที่ 35 User interface หน้า Register

### 7.3 Dashboard Guest

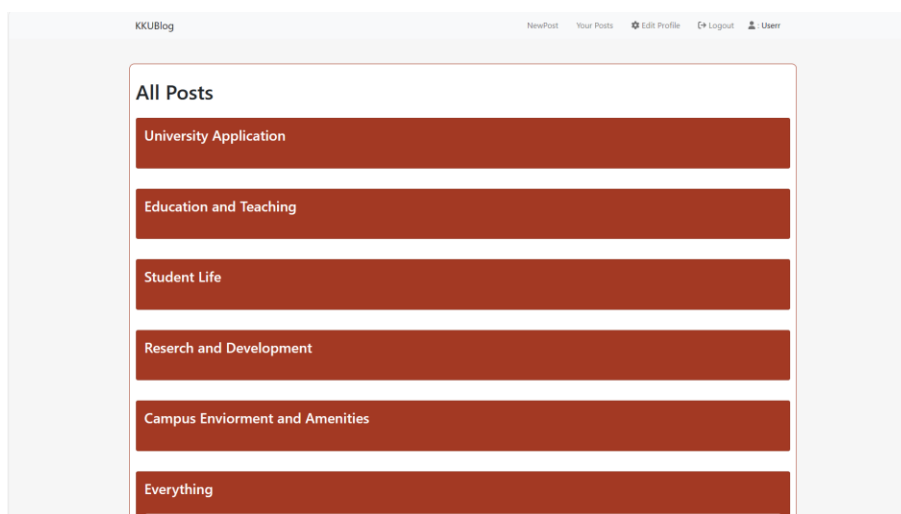
Dashboard Guest เป็นหน้ารวมกระทู้ต่างๆ สำหรับผู้ที่เป็นผู้เยี่ยมชม ไม่ได้ login เป็นสมาชิก



รูปที่ 36 User interface หน้า Dashboard Guest

### 7.4 Dashboard

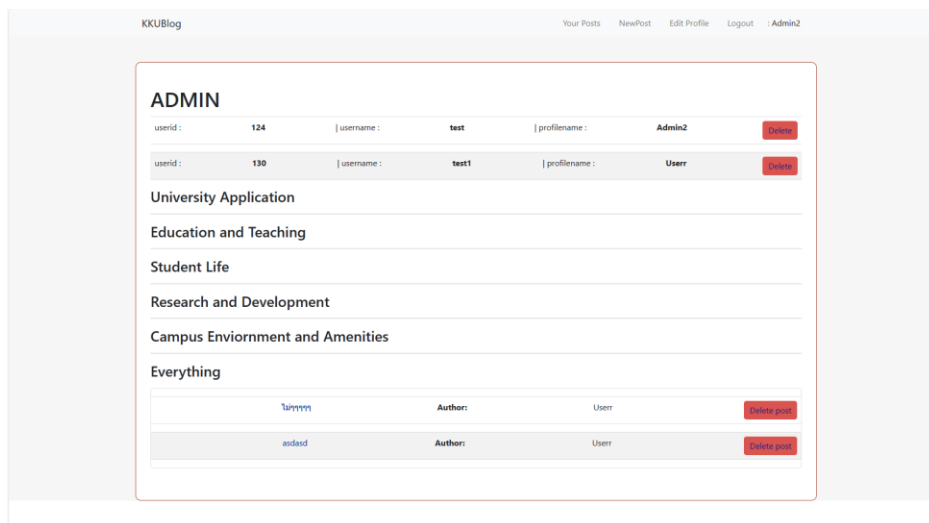
Dashboard เป็นหน้ารวมกระทู้ต่างๆ สำหรับสมาชิกที่ login



รูปที่ 37 User interface หน้า Dashboard

## 7.5 Admin

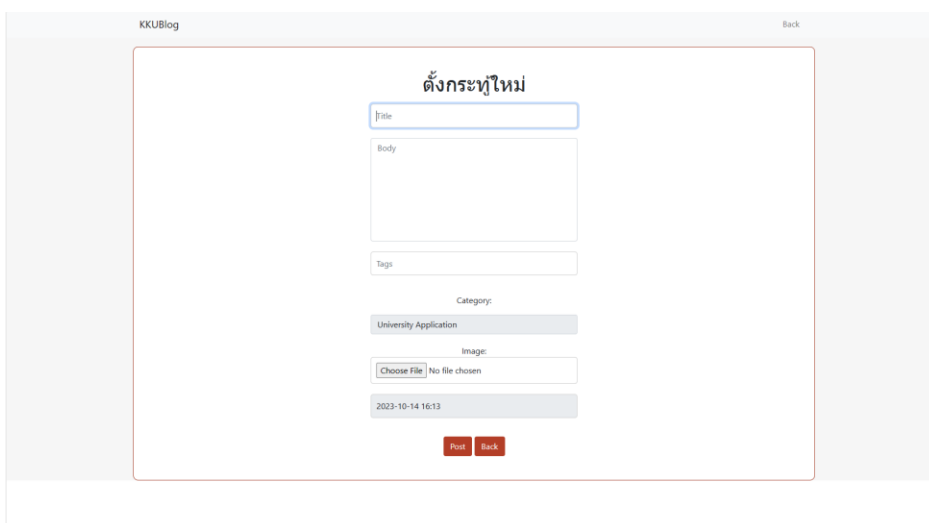
Admin เป็นหน้าสำหรับผู้ที่มีสิทธิ์การเข้าถึงข้อมูลของกระทู้และข้อมูลของผู้ใช้งานทั่วไปเพื่อจัดการกับข้อมูลนั้น



รูปที่ 38 User interface หน้า Admin

## 7.6 Post

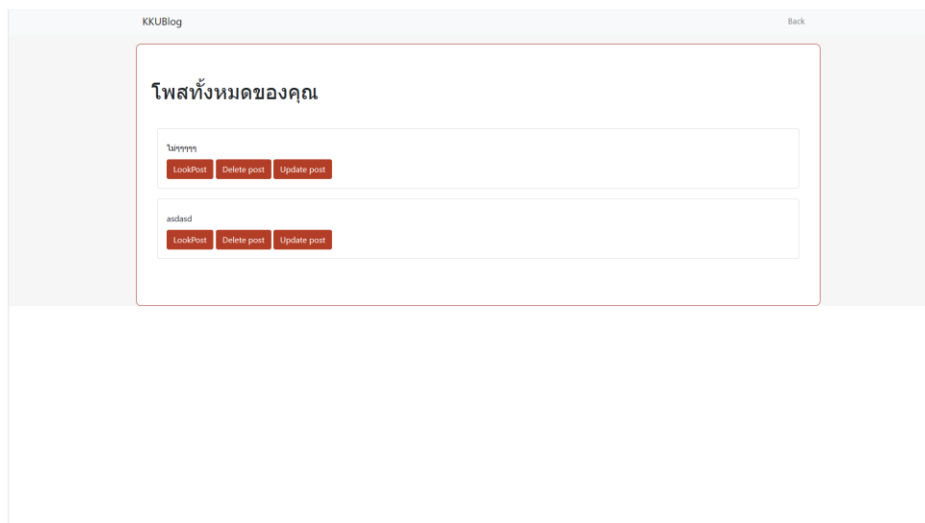
Post เป็นหน้าสำหรับการตั้งกระทู้ใหม่โดยใส่หัวข้อ เนื้อ และประเภทของกระทู้ที่ต้องการ



รูปที่ 39 User interface หน้า post

## 7.7 Your post

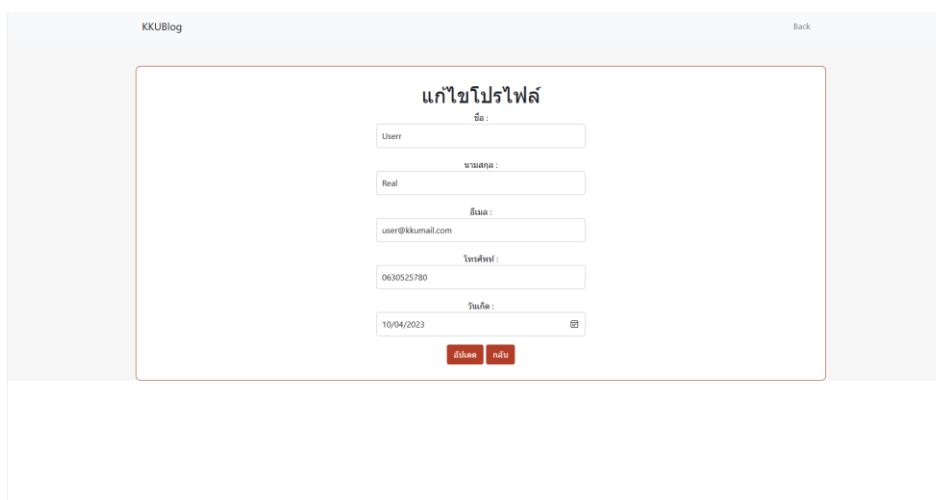
Your post เป็นหน้าที่แสดงกระทู้ทั้งหมดของผู้ใช้ ที่ได้ทำการตั้งกระทู้ไป และสามารถลบ เข้าดู และแก้ไขกระทู้ได้



รูปที่ 40 User interface หน้า Your post

## 7.8 Edit profile

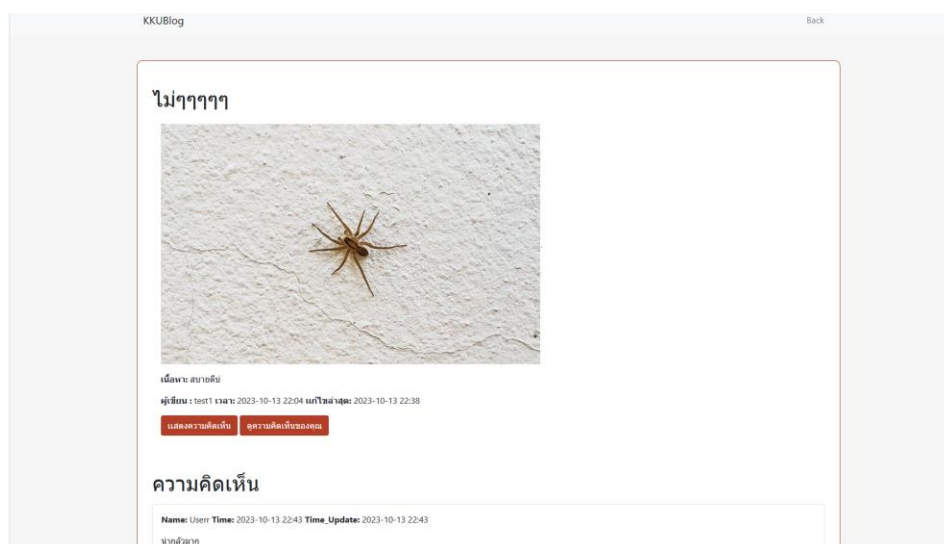
Edit profile เป็นหน้าที่สำหรับแก้ไขข้อมูลของผู้ใช้



รูปที่ 41 User interface หน้า edit profile

## 7.9 Look post

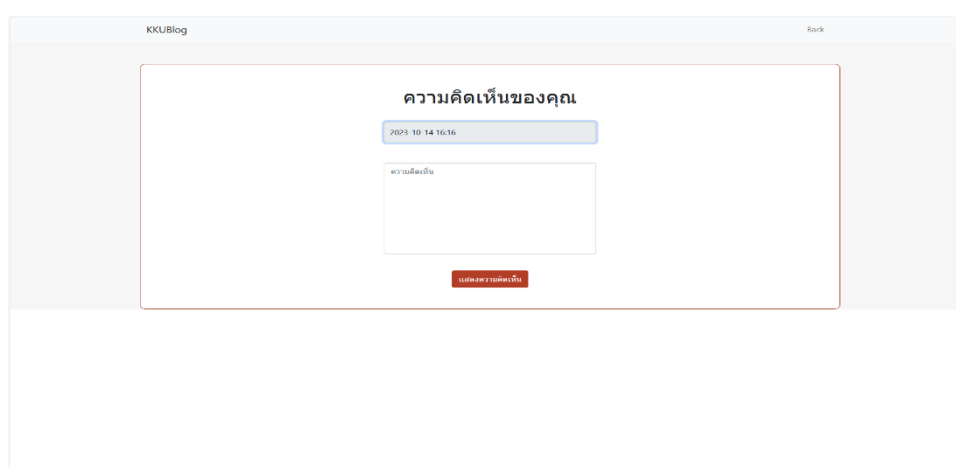
Look post เป็นหน้าสำหรับ ดูรายละเอียดของกระทู้ต่างๆ ที่แสดงหัวข้อ เนื้อหา และความคิดเห็นของกระทู้ สามารถแสดงความคิดเห็นต่อกระทู้ต่างๆได้.



รูปที่ 42 User interface หน้า look post

## 7.10 Comment

Comment เป็นหน้าที่สามารถแสดงความคิดเห็นของกระทู้ต่างๆ ได้

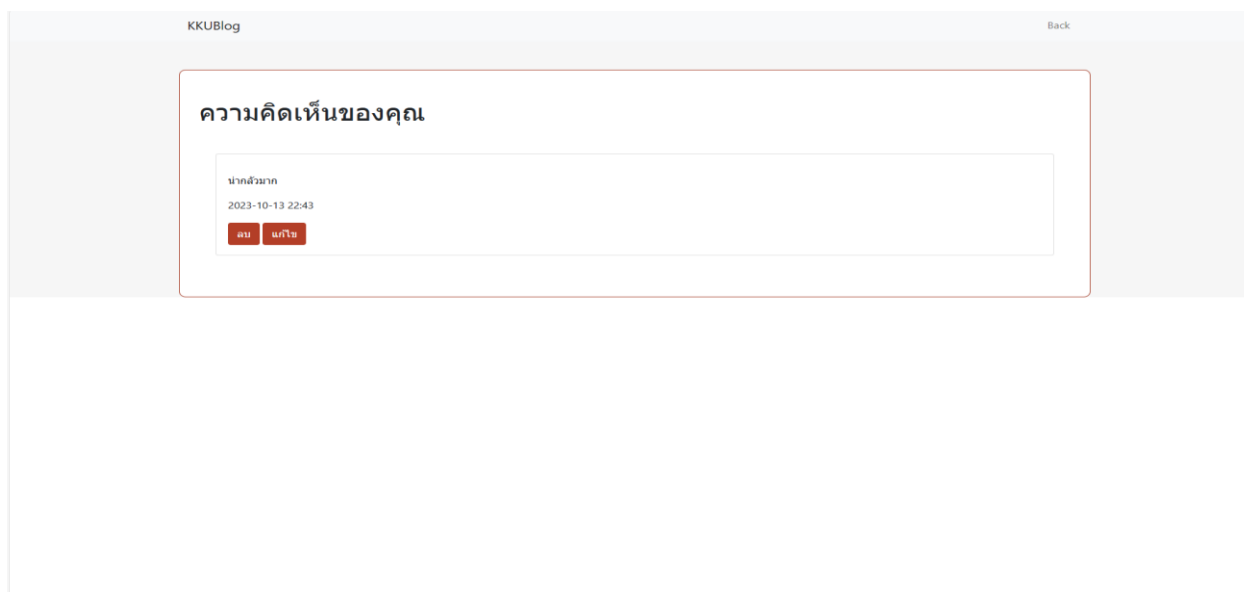


รูปที่ 43 User interface หน้า Comment



### 7.11 Your comment

Your comment เป็นหน้าสำหรับดูความคิดเห็นทั้งหมดที่ผู้ใช้นั้นได้แสดงความคิดเห็นไป และสามารถลบและแก้ไขความคิดเห็นได้



รูปที่ 44 User interface หน้า Your comment

## อ้างอิง

- Teerawat Amornrattanakij.(2563). Spring Boot มีไว้ทำอะไร?. สืบค้น 16 ตุลาคม 2566 จาก <https://medium.com/@Teerawat.amo/spring-boot-มีไว้ทำอะไร-c1d84a7796d7>
- Chantachat Choedsuwan.(2565). Designs patterns in GoLang. สืบค้น 16 ตุลาคม 2566 จาก <https://hugeman.co/designs-patterns-in-golang/#:~:text=Strategy,ให้เหมือนกันนั่นเอง>
- Nutron.(2560). Repository Pattern เป็นยังไงหนอ แล้วจะใช้อย่างไร. สืบค้น 15 ตุลาคม 2566 จาก <https://medium.com/@nutron/repository-pattern-c66f1cb37f2a>
- Phayao Boonon.(2562). Design Pattern 101 — Factory Pattern. สืบค้น 15 ตุลาคม 2566 จาก <https://phayao.medium.com/design-pattern-101-factory-pattern-a0a3f89cfc23>
- Theerut Bunkhanphol.(2563). เกี่ยวกับ Spring Dependency Injection. สืบค้น 15 ตุลาคม 2566 จาก <https://theerut-bun.medium.com/เกี่ยวกับ-spring-dependency-injection-3695e449a6f2>
- วิกิพีเดีย สารานุกรมเสรี.(2566).ไมโครซอฟท์ วิวอลสตูดิโอ. สืบค้น 14 ตุลาคม 2566 จาก [https://th.wikipedia.org/wiki/ไมโครซอฟท์\\_วิวอลสตูดิโอ](https://th.wikipedia.org/wiki/ไมโครซอฟท์_วิวอลสตูดิโอ)
- Thanatcha Veeravattanayothin.(2566). MySQL คือ อะไร ? โปรแกรมจัดการฐานข้อมูล Open Source ยอดนิยม !. สืบค้น 14 ตุลาคม 2566 จาก <https://blog.openlandscape.cloud/mysql>
- Thawatchai pradu. (2563). ไมโครซอฟท์ เวิร์ด (โปรแกรมประมวลผลคำเพื่องานเอกสาร) | Microsoft Word. สืบค้น 14 ตุลาคม 2566 จาก <https://library.wu.ac.th/km/ไมโครซอฟท์-เวิร์ด-โปรแกรม/>