รายงาน

Software Design Project

จัดทำโดย

63010766	ภูมิศักดิ์	แก้วสี
63010767	ภูริช	จันทร์ประสิทธิ์
63010841	วรภพ	เกียรติคงแสง
63010846	วรรธนัย	เมธาเมลือง
63010852	วรวิชญ์	ธรรมารักษ์วัฒนะ
63011018	สุรพัศ	วงศ์ประไพพักตร์
63011075	อับดุลฮากิม	มาหะ
63011414	อับดุลฮากีม	มามุ

เสนอ

ดร.ปริญญา เอกปริญญา

รายงานนี้เป็นส่วนหนึ่งของการเรียนวิชา 01076024

SOFTWARE ARCHITECTURE AND DESIGN ภาคเรียนที่ 1

ปีการศึกษา 2565 สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

คำนำ

เนื่องจากหอพักแถวลาดกระบังนั้น มีมากมายหลายร้อยหอ แล้วเราจะสามารถทราบได้อย่างไรว่าในแต่ละหอพักมีรายละเอียดในเรื่องราคาอย่างไร มีสิ่งอำนวยความสะดวกใดบ้าง เป็นหอพักที่เราต้องการหรือไม่ และหอพักนั้นมีห้องว่างหรือเปล่า ซึ่งการที่เราจะเดินไปถามตามหอพักเพื่อสอบถามรายละเอียดให้ครบทุกที่ มันก็เป็นเรื่องที่ยากมาก กลุ่มของพวกเราจึงได้คิดที่จะทำเว็บแอปพลิเคชันที่จะช่วยให้ทุกๆ คนที่อยู่ในแถบลาดกระบัง หรือนักศึกษาที่พึ่งเข้ามาใหม่ สามารถหาที่พักต่างๆ ได้อย่างง่ายดายตามสิ่งทีตัวเองต้องการ ซึ่งกลุ่มของพวกเราได้อยู่แถวลาดกระบังมาประมาณ 2 ปี และยังไม่เคยเห็นว่ามีเว็บแอปพลิเคชันไหน ที่จะช่วยอำนวยความสะดวกในด้านนี้ พวกเราจึงคิดที่จะสร้างเว็บแอปพลิเคชันนี้ขึ้นมา เพื่อให้คนที่ต้องการอยู่ในลาดกระบังนั้นได้ใช้แบบจริง ๆ ซึ่งถ้าหากเป็นที่นิยมมาก พวกเราก็อาจจะไปต่อยอดในการใช้กับพื้นที่อื่นๆ เพื่ออำนวยความสะดวกให้กับผู้คนที่มีความต้องการที่จะหาที่พักภายในละแวกนั้นๆ

สารบัญ

Proposal

Business Sector: Hotel & Lodging

Problem:

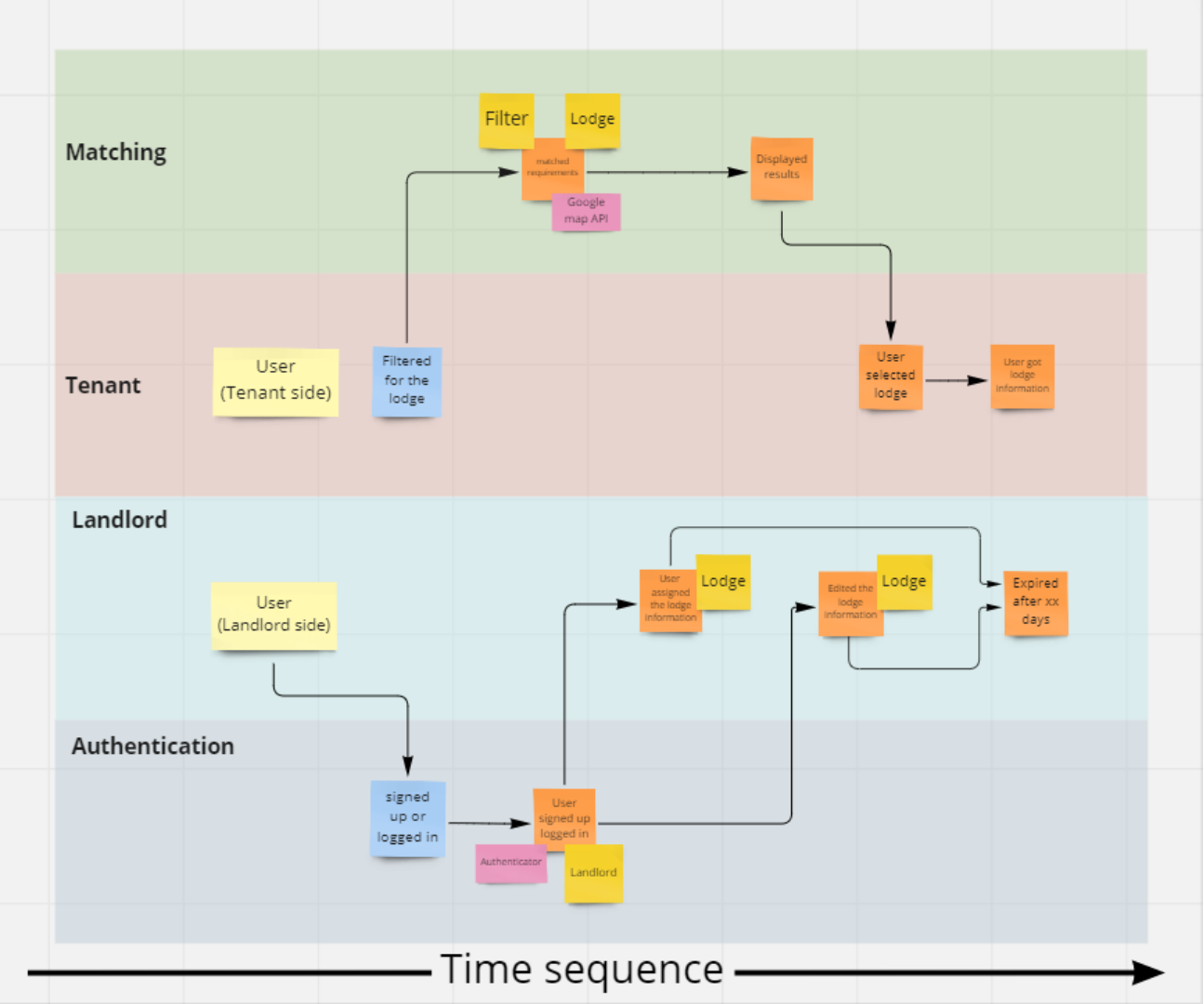
เนื่องจากหอพักแถวลาดกระบังนั้น มีมากมายหลายร้อยหอ แล้วเราจะสามารถทราบได้อย่างไรว่าในแต่ละหอพักมีรายละเอียดในเรื่องราคาอย่างไร มีสิ่งอำนวยความสะดวกใดบ้าง เป็นหอพักที่เราต้องการหรือไม่ และหอพักนั้นมีห้องว่างหรือเปล่า ซึ่งการที่เราจะเดินไปถามตามหอพักเพื่อสอบถามรายละเอียดให้ครบทุกที่ มันก็เป็นเรื่องที่ยากมาก กลุ่มของพวกเราจึงได้คิดที่จะทำเว็บแอปพลิเคชันที่จะช่วยให้ทุกๆ คนที่อยู่ในแถบลาดกระบัง หรือนักศึกษาที่พึ่งเข้ามาใหม่ สามารถหาที่พักต่างๆ ได้อย่างง่ายดายตามสิ่งทีตัวเองต้องการ ซึ่งกลุ่มของพวกเราได้อยู่แถวลาดกระบังมาประมาณ 2 ปี และยังไม่เคยเห็นว่ามีเว็บแอปพลิเคชันไหน ที่จะช่วยอำนวยความสะดวกในด้านนี้ พวกเราจึงคิดที่จะสร้างเว็บแอปพลิเคชันนี้ขึ้นมา เพื่อให้คนที่ต้องการอยู่ในลาดกระบังนั้นได้ใช้แบบจริง ๆ ซึ่งถ้าหากเป็นที่นิยมมาก พวกเราก็อาจจะไปต่อยอดในการใช้กับพื้นที่อื่นๆ เพื่ออำนวยความสะดวกให้กับผู้คนที่มีความต้องการที่จะหาที่พักภายในละแวกนั้นๆ

Software solution:

คำจำกัดความของซอฟแวร์ของเราสั้น ๆ คือ เป็นเว็บที่สามารถกรองข้อมูลของหอพักที่ผู้ใช้ต้องการ ไม่ว่าจะเป็นราคา ระยะห่างจากมหาวิทยาลัย สิ่งอำนวยความสะดวกต่าง ๆของหอพัก และในแต่ละหอพักมีห้องว่างอยู่หรือไม่ ซอฟแวร์ของเราจะช่วยแก้ปัญหาในการหาห้องพักบริเวณ สจล.ได้ โดยตัวโปรแกรมจะเป็นเว็บแอปพลิเคชันที่สามารถกรองข้อมูลได้ตามที่กล่าวข้างต้น ซึ่งหอพักนั้น ๆ ก็สามารถถูกอัพเดตโดยตัวแทน หรือเจ้าของหอว่าว่างหรือไม่ มีรายละเอียด รูป ค่าเช่า สถานที่ เบอร์โทรศัพท์ติดต่อ เพื่อลดปัญหาให้กับผู้ที่ต้องการจะหาหอพักอยู่แต่หอพักนั้นมีมากมาย และหลากหลายให้เราเลือกอยู่ ข้อดีอีกอย่างคือตัวเว็บนั้น รวบรวมข้อมูลติดต่อเบื้องต้นทุกอย่างของหอพักไว้เรียบร้อยแล้ว ทำให้เราไม่ต้องไปคอยตามหาเบอร์ติดต่อให้ยุ่งยาก และได้หอพักตามที่ต้องการจริงๆ

Software Design Project : Modeling

ในขั้นแรก เราได้ทำ Event Storming เพื่อที่จะดูภาพรวมของระบบ เพื่อที่จะแบ่ง Bounded context ได้อย่างถูกต้องและครบถ้วน



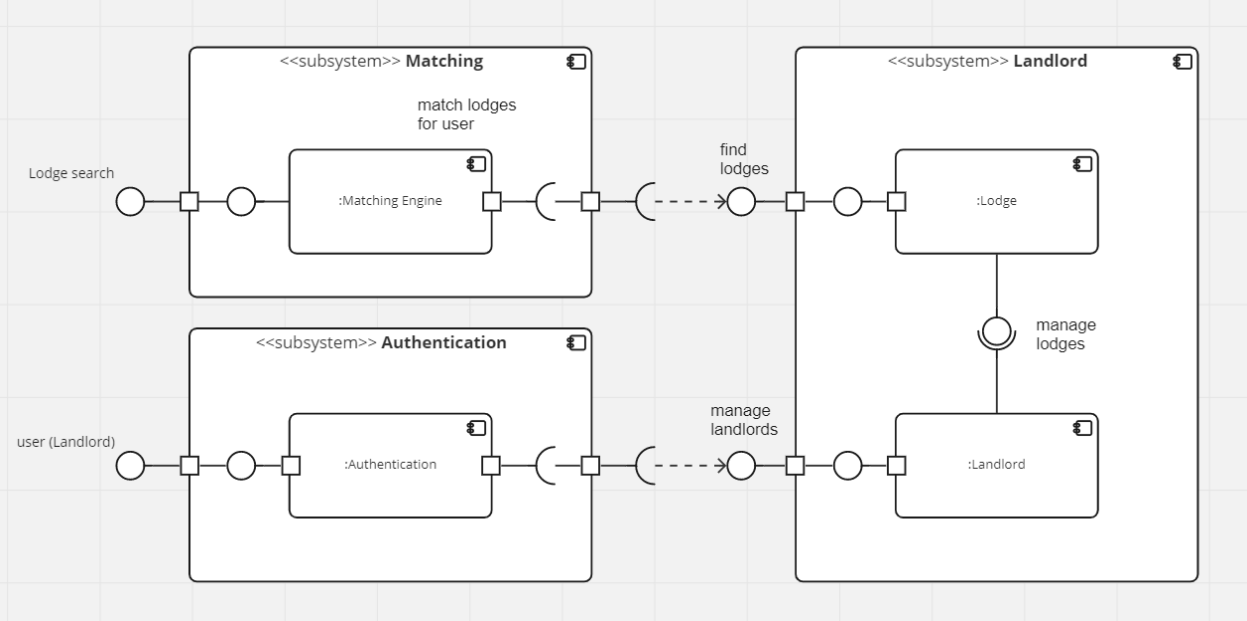
รูปที่ 1 Event Storming

จากการทำ Event Storming สามารถแบ่งได้เป็น 3 bounded context

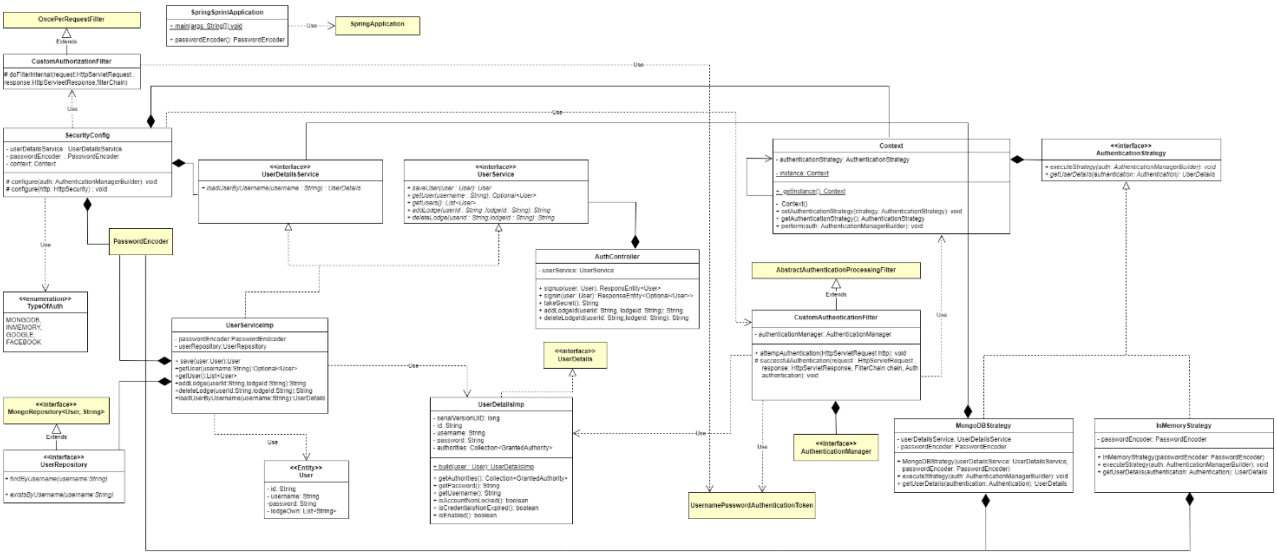
1. Authentication

2. Landlord

3. Matching

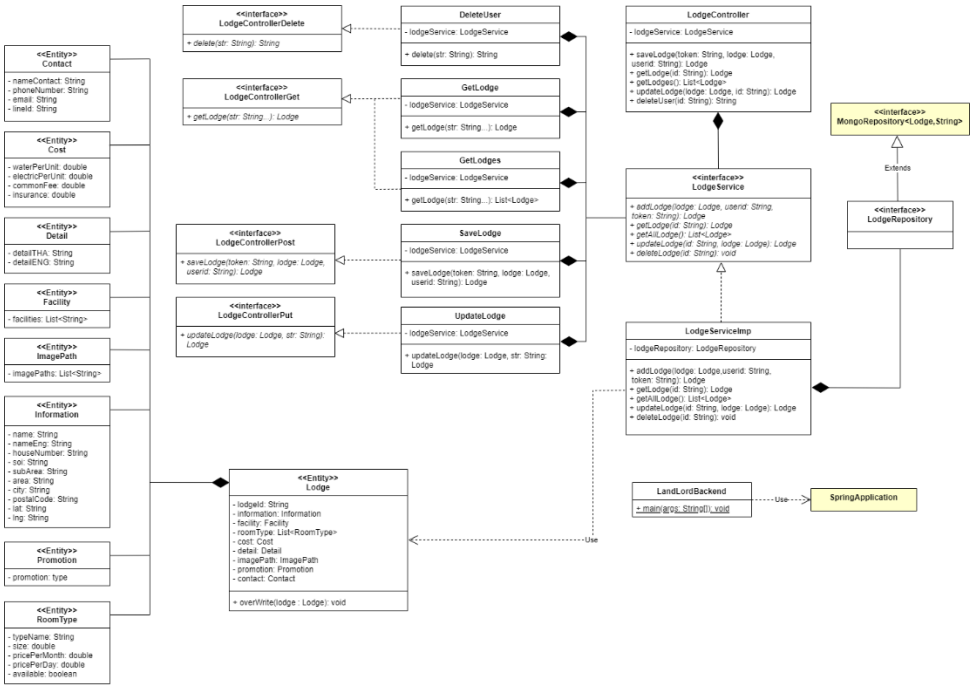


รูปที่ 2 UML Component Diagram แสดงภาพรวมของ software



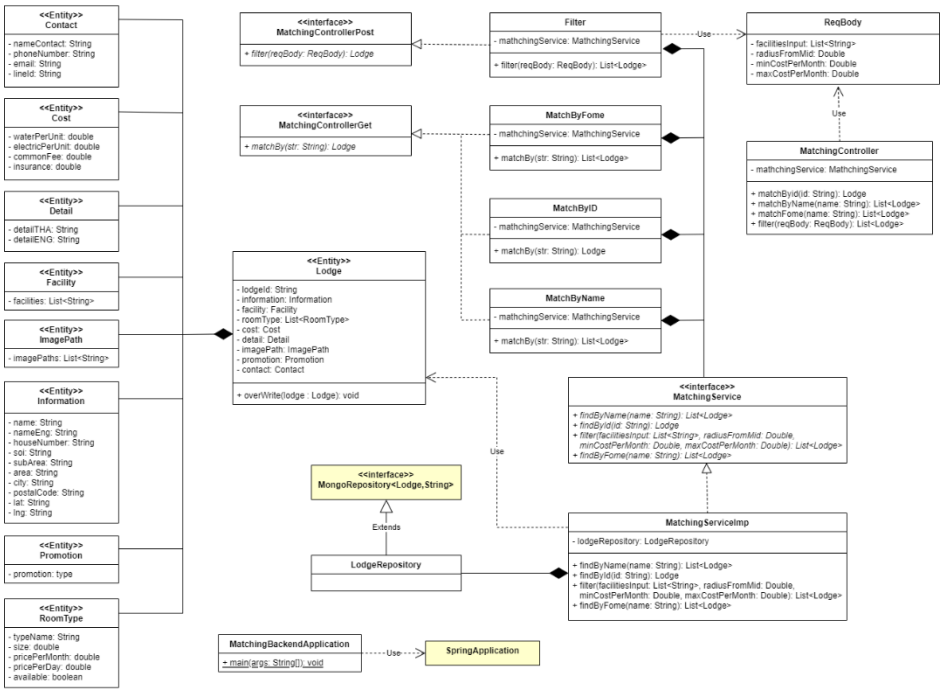
รูปที่ 3 UML Diagram ของ Authentication System

https://github.com/poomsakk/piranya-hub/blob/main/diagram/class_diagram_auth.png



รูปที่ 4 UML Diagram ของ Landlord System

https://github.com/poomsakk/piranya-hub/blob/main/diagram/class_diagram_landlord.png



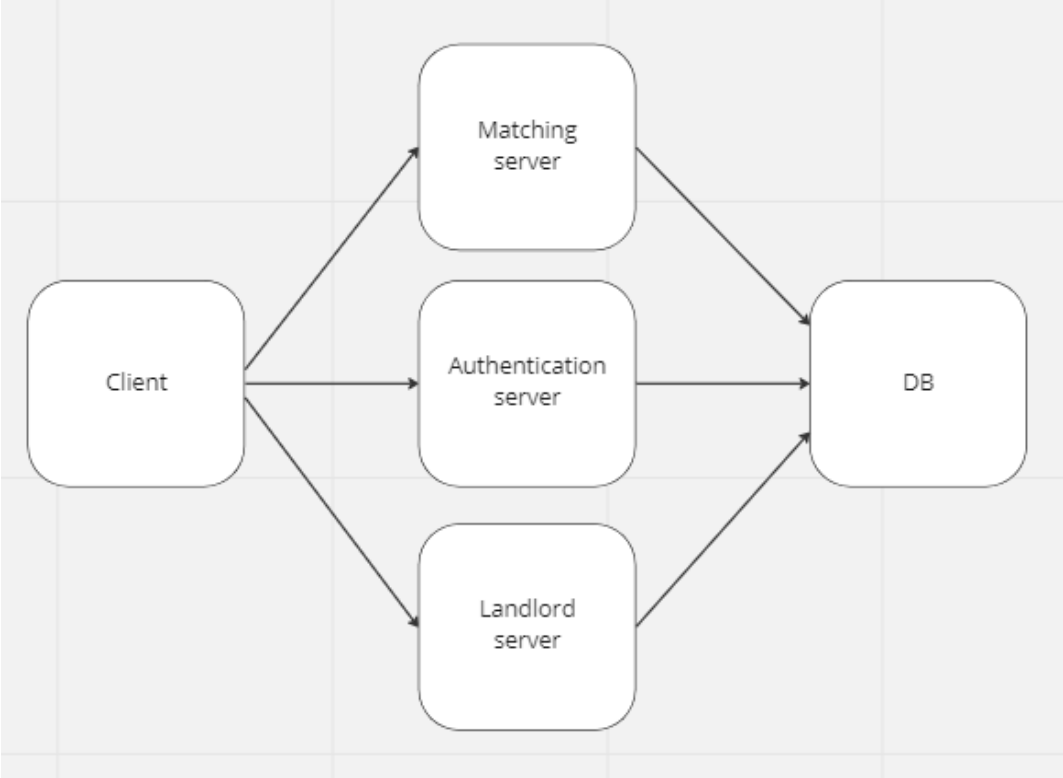
รูปที่ 5 UML Diagram ของ Matching System

https://github.com/poomsakk/piranya-hub/blob/main/diagram/class_diagram_matching.png

Software Architecture

ในการออกแบบ software กลุ่มของเราได้มีการนำ Software architecture และ Architectural Patterns/Styles ต่างๆ ที่ได้เรียนมาประยุกต์ใช้กับ project ของกลุ่มเราด้วย ได้แก่

1. Client-Server



รูปที่ 6 Client-Server diagram

2. Representational State Transfer (REST)

Software Design Project : Quality

Web Application ของเรามี Quality Attribute หลายด้าน เพื่อแสดงว่าว่าระบบของเราทำงานได้ดีมากน้อยแค่ไหน ซึ่งสามารถวัดผลได้ตาม Quality Attribute Scenario ดังนี้

Scenario 1: Availability

Source of stimulus : software

Stimulus : crash

Artifacts : Process

Environment : Normal operation

Response : ใช้งานไม่ได้ชั่วคราวขณะที่กำลังซ่อม

Response Measure : downtime ไม่เกิน 2 ชั่วโมง

Scenario 2: Integrability

Source of stimulus : ผู้มีส่วนได้ส่วนเสียกับระบบ

Stimulus : รวม component

Artifacts : ทั้งระบบ

Environment : Integration

Response : เปลี่ยนแปลงได้โดยไม่มีข้อผิดพลาด

Response Measure : ใช้เวลาไม่เกิน 2 ชั่วโมง

Scenario 3: Modifiability

Source of stimulus : Developer

Stimulus : ต้องการเปลี่ยน UI

Artifacts : UI

Environment : design time

Response : เปลี่ยนแปลงได้สำเร็จ

Response Measure : ใช้เวลาน้อยกว่า 2 ชั่วโมง

Scenario 4: Performance

Source of stimulus : user

Stimulus : ส่ง request

Artifacts : ระบบ

Environment : โหมดปกติ

Response : ระบบตอบกลับ response

Response Measure : ตอบกลับไม่เกิน 2 วินาที

Scenario 5: Security

Source of stimulus : ผู้ไม่ประสงค์ดี

Stimulus : พยายามเข้ามาเปลี่ยนแปลงข้อมูลโดยไม่ได้รับอนุญาต

Artifact : ระบบ

Environment : online

Response : log in / ใช้งานไม่ได้

Response Measure : ไม่มีข้อมูลที่ได้รับผลกระทบ

Scenario 6: Testability

Source of stimulus : developer

Stimulus : เขียน code เสร็จ

Artifacts : code

Environment : ช่วงกำลังพัฒนา

Response : ทำการ test ตามลำดับ

Response Measure : test เสร็จใน 2 ชั่วโมง

Scenario 7: Usability

Source of stimulus : User

Stimulus : ต้องการเรียนรู้ระบบ

Artifacts : ระบบ

Environment : runtime

Response : คาดการณ์ความต่องการของผู้ใช้

Response Measure : ความพึงพอใจของลูกค้า

Scenario 8: Availability

Source of stimulus : software

Stimulus : เกิด fault

Artifacts : process

Environment : ช่วงเวลาปกติ

Response : แจ้งไปยังระบบ

Response Measure : สามารถตรวจพบ fault ได้ภายใน 1 นาที

Software Design Project : Design Pattern

Software ของเราได้มีการนำหลักการของ Design Pattern เข้ามาใช้ใน project นี้ด้วย เพื่อเพิ่ม function ใหม่ๆเข้าไปในระบบได้ง่ายขึ้น โดย Design Pattern ที่เรานำมาใช้ ได้แก่

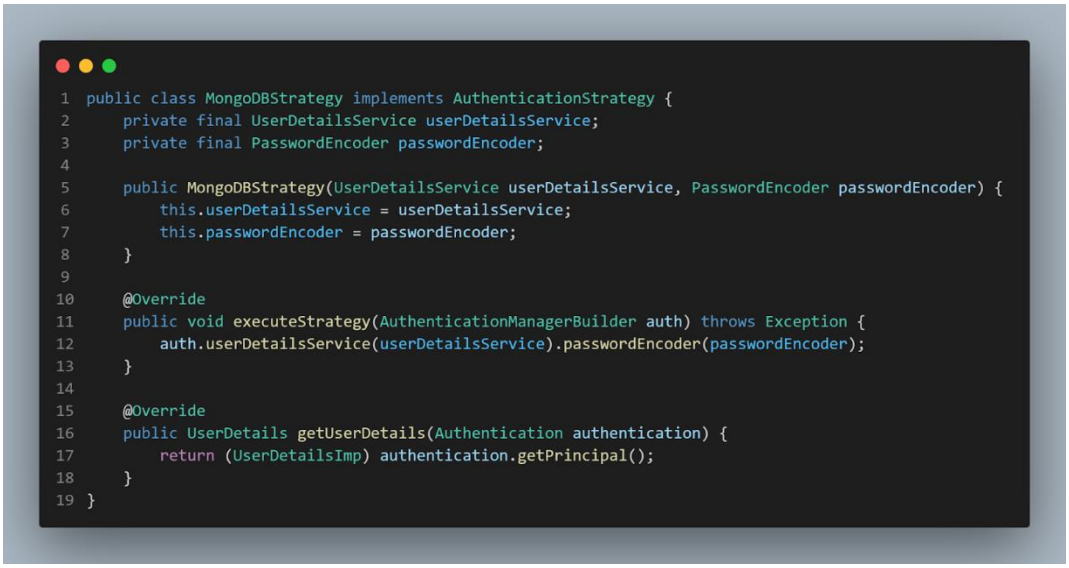
1. **Strategy pattern (ส่วนที่ 1)**
    - Design Pattern นี้ใช้เพื่อสร้างการทำงานหลายๆแบบ เพื่อให้ client เลือกไปใช้งานตามความเหมาะสม และแยกการทำงานออกเป็นเรื่องๆขาดจากกันได้
    - Design Pattern นี้ใช้อย่างไร นำมาใช้ตอน Authentication เพื่อเลือกว่าจะ Authen จากฐานข้อมูลตัวไหน ตัวอย่างเช่น authen จาก mongoDB, authen จาก Inmemory(local)
    - ตัวอย่างการใช้งาน

```java
public interface AuthenticationStrategy {
    void executeStrategy(AuthenticationManagerBuilder auth) throws Exception;
    UserDetails getUserDetails(Authentication authentication);
}
```

รูปที่ 7 Strategy interface (Strategy pattern ส่วนที่ 1)

```java
public class InMemoryStrategy implements AuthenticationStrategy {
    private PasswordEncoder passwordEncoder;

    public InMemoryStrategy(PasswordEncoder passwordEncoder) {
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    public void executeStrategy(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication().withUser("admin").password(passwordEncoder.encode("admin")).roles("USER");
    }

    @Override
    public UserDetails getUserDetails(Authentication authentication) {
        return (UserDetails) authentication.getPrincipal();
    }
}
```

รูปที่ 8 Strategies(1) (Strategy pattern ส่วนที่ 1)



```java
public class MongoDBStrategy implements AuthenticationStrategy {
    private final UserDetailsService userDetailsService;
    private final PasswordEncoder passwordEncoder;

    public MongoDBStrategy(UserDetailsService userDetailsService, PasswordEncoder passwordEncoder) {
        this.userDetailsService = userDetailsService;
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    public void executeStrategy(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);
    }

    @Override
    public UserDetails getUserDetails(Authentication authentication) {
        return (UserDetailsImp) authentication.getPrincipal();
    }
}
```

รูปที่ 9 Strategies(2) (Strategy pattern ส่วนที่ 1)

```java
public class Context {
    private static Context instance;
    private AuthenticationStrategy authenticationStrategy;

    private Context() {
    }

    public void setAuthenticationStrategy(AuthenticationStrategy strategy){
        this.authenticationStrategy = strategy;
    }

    public AuthenticationStrategy getAuthenticationStrategy() {
        return authenticationStrategy;
    }

    public static Context getInstance(){
        if (instance == null){
            instance = new Context();
        }
        return instance;
    }

    public void perform(AuthenticationManagerBuilder auth) throws Exception {
        this.authenticationStrategy.executeStrategy(auth);
    }
}
```

รูปที่ 10 Context (Strategy pattern ส่วนที่ 1)

```java
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private final UserDetailsService userDetailsService;
    @Autowired
    private final PasswordEncoder passwordEncoder;
    private Context context;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        TypeOfAuth type = TypeOfAuth.MONGODB;
        context = Context.getInstance();

        if (type == TypeOfAuth.MONGODB) {
            context.setAuthenticationStrategy(new MongoDBStrategy(userDetailsService, passwordEncoder));
        } else if (type == TypeOfAuth.INMEMORY) {
            context.setAuthenticationStrategy(new InMemoryStrategy(passwordEncoder));
        } else if (type == TypeOfAuth.GOOGLE){
           //
        } else if (type == TypeOfAuth.FACEBOOK){
           //
        }
        context.perform(auth);
    }
```

รูปที่ 11 ส่วนเรียนใช้ (Strategy pattern ส่วนที่ 1)

2. **Strategy pattern (ส่วนที่ 2)**
    - Design Pattern นี้ใช้เพื่อสร้างการทำงานหลายๆแบบ เพื่อให้ client เลือกไปใช้งานตามความเหมาะสม และแยกการทำงานออกเป็นเรื่องๆขาดจากกันได้
    - Design Pattern นี้ใช้อย่างไร นำมาใช้ตอน Search ด้วยชื่อ เรามีหลายอัลกอริทึมในการค้นหาก็ให้เลือกใช้ตามความเหมาะสม
    - ตัวอย่างการใช้งาน

```java
public interface FindByNameStrategy {
    public List<Lodge> executeData(String name);
}
```

รูปที่ 12 Strategy interface (Strategy pattern ส่วนที่ 2)

```java
public class ConcreteStrategyFindByNameNormal implements FindByNameStrategy{
    private LodgeRepository lodgeRepository;

    public ConcreteStrategyFindByNameNormal(LodgeRepository lodgeRepository){
        this.lodgeRepository = lodgeRepository;
    }

    @Override
    public List<Lodge> executeData(String name) {
        List<Lodge> data = lodgeRepository.findAll();
        List<Lodge> res = new ArrayList<>();
        for (int i = 0; i < data.size(); i++) {
            if(data.get(i).getInformation().getName().toLowerCase().indexOf(name.toLowerCase()) != -1) {
                res.add(data.get(i));
            }
        }
        return res;
    }
}
```

รูปที่ 13 Strategies(1) (Strategy pattern ส่วนที่ 2)

```java
public class ConcreteStrategyFindByNameSuperAdvance implements FindByNameStrategy{
    private LodgeRepository lodgeRepository;

    public ConcreteStrategyFindByNameSuperAdvance(LodgeRepository lodgeRepository){
        this.lodgeRepository = lodgeRepository;
    }
    @Override
    public List<Lodge> executeData(String name) {
        List<Lodge> data = lodgeRepository.findAll();
        List<Lodge> res = new ArrayList<>();
        String inp = name.toLowerCase().replaceAll("\\s+","");
        for(int i = 0; i < data.size();i++)
        {
            if(data.get(i).getInformation().getName().replaceAll("\\s+","").contains(inp))
            {
                res.add(data.get(i));
            }
        }
        return res;
    }
}

```

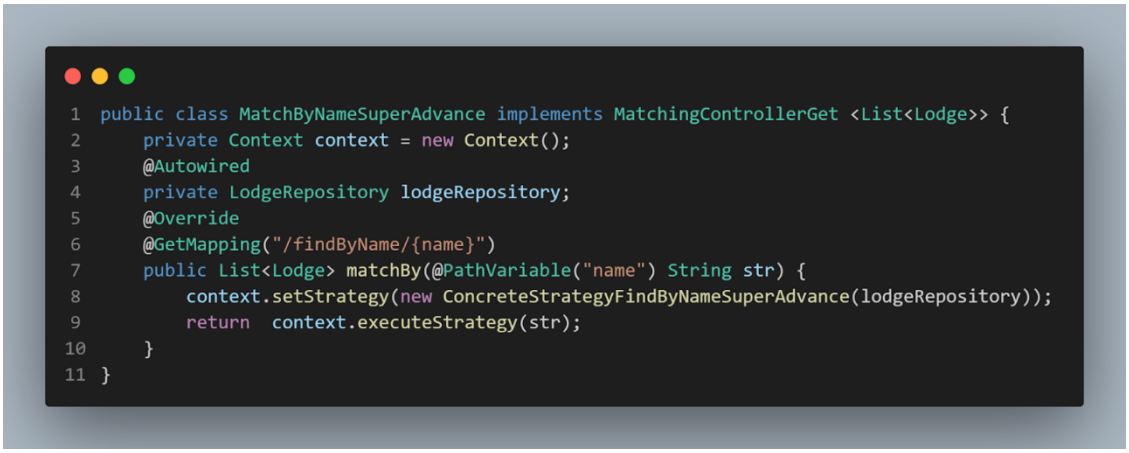
รูปที่ 14 Strategies(2) (Strategy pattern ส่วนที่ 2)

```java
public class Context {
    private FindByNameStrategy strategy;

    public void setStrategy(FindByNameStrategy strategy){
        this.strategy = strategy;
    }

    public List<Lodge> executeStrategy(String name){
        return this.strategy.executeData(name);
    }
}

```

รูปที่ 15 Strategies(3) (Strategy pattern ส่วนที่ 2)

```java
public class MatchByNameNormal implements MatchingControllerGet <List<Lodge>> {

    private Context context = new Context();
    @Autowired
    private LodgeRepository lodgeRepository;
    @Override
    @GetMapping("/findByName2/{name}")
    public List<Lodge> matchBy(@PathVariable("name") String str) {
        context.setStrategy(new ConcreteStrategyFindByNameNormal(lodgeRepository));
        return  context.executeStrategy(str);
    }
}
```

รูปที่ 16 ส่วนเรียกใช้(1) (Strategy pattern ส่วนที่ 2)



```java
public class MatchByNameSuperAdvance implements MatchingControllerGet <List<Lodge>> {
    private Context context = new Context();
    @Autowired
    private LodgeRepository lodgeRepository;
    @Override
    @GetMapping("/findByName/{name}")
    public List<Lodge> matchBy(@PathVariable("name") String str) {
        context.setStrategy(new ConcreteStrategyFindByNameSuperAdvance(lodgeRepository));
        return  context.executeStrategy(str);
    }
}
```

รูปที่ 17 ส่วนเรียกใช้(2) (Strategy pattern ส่วนที่ 2)

3. Singleton

- Design Pattern นี้แก้ปัญหาการอยากจะสร้างคลาสที่สามารถนำไปสร้าง object ได้เพียงตัวเดียว
- Design Pattern นี้ใช้กับ Context ของ strategy ด้านข้อทีแล้ว
- ตัวอย่างการใช้งาน

```
public class Context {
    private static Context instance;
    private AuthenticationStrategy authenticationStrategy;

    private Context() {
    }

    public void setAuthenticationStrategy(AuthenticationStrategy strategy){
        this.authenticationStrategy = strategy;
    }

    public AuthenticationStrategy getAuthenticationStrategy() {
        return authenticationStrategy;
    }

    public static Context getInstance(){
        if (instance == null){
            instance = new Context();
        }
        return instance;
    }

    public void perform(AuthenticationManagerBuilder auth) throws Exception {
        this.authenticationStrategy.executeStrategy(auth);
    }
}
```

รูปที่ 18 Singleton

```
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    .
    .
    .
    private Context context;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        TypeOfAuth type = TypeOfAuth.MONGODB;
        context = Context.getInstance();
        .
        .
        .
```

รูปที่ 19 ส่วนเรียกใช้(1) (Singleton)

```java
@Override
protected void successfulAuthentication(HttpServletRequest request, HttpServletRespon

    Context context = Context.getInstance();
    if (context.getAuthenticationStrategy() instanceof MongoDBStrategy) {
        System.out.println("MongoDBStrategy Yeah man!!");
    } else if (context.getAuthenticationStrategy() instanceof InMemoryStrategy) {
        System.out.println("InMemoryStrategy okee !!");
    }
```

รูปที่ 20 ส่วนเรียกใช้(2) (Singleton)

Software Design Project : Functionality

- ระบบ Log in / Register
- Filter หอพักได้ตามที่ต้องการ
- รู้ได้ว่าหอพักแต่ละที่มีห้องว่างหรือไม่
- ลงประกาศหอพัก
- แก้ไขรายละเอียดของหอพัก
- ระบบ Authentication
- Search ตามชื่อหอพัก

แหล่งอ้างอิง

- https://github.com/poomsakk/piranya-hub