



PIRANYA HUB

SINCE 2022

PRESENTATION

กลุ่มที่ 11 ปรันย่าชอบงบคุณ

ต้องการที่พัก นักถิ่น

PIRANYA HUB



กำลังมองหาที่พัก
ที่เหมาะสมกับคุณอยู่
ใช่มั้ย?

ให้เราได้เป็นส่วน
หนึ่งในการเลือก
ที่พักให้เหมาะสมกับ
ทุกสไตล์ของคุณ

PROPOSAL

PROBLEM:

เนื่องจากหอพักแควลาดกระบังนี้ มีมากมายหลายหอ แล้วเราจะสามารถทราบได้อย่างไรว่าในแต่ละหอพักมีรายละเอียดในเรื่องราคาย่างไร มีสิ่งอำนวยความสะดวกใดบ้าง เป็นหอพักที่เราต้องการหรือไม่ และหอพักนี้มีห้องว่างหรือไม่ ซึ่งการที่เราจะเดินไปตามตามหอพักเพื่อสอบถามรายละเอียดให้ครบถ้วนก็เป็นเรื่องที่ยากมาก





SOLUTION:

ซอฟต์แวร์ของเราระบุเป็นเว็บแอปพลิเคชัน
ที่สามารถกรองข้อมูลเกี่ยวกับหอพัก ซึ่งหอพักนั้นๆ
ก็สามารถประกาศโดยเจ้าของหอพัก และมีราย
ละเอียดต่างๆ เพื่อลดปัญหาข้างต้นให้กับผู้ที่
ต้องการจะหาหอพักอยู่

MODELING



ในขั้นแรก เราได้ทำ **EVENT STORMING** เพื่อที่จะดูภาพรวมของระบบ เพื่อที่จะแบ่ง **BOUNDED CONTEXT** ได้อย่างถูกต้องและครบถ้วน

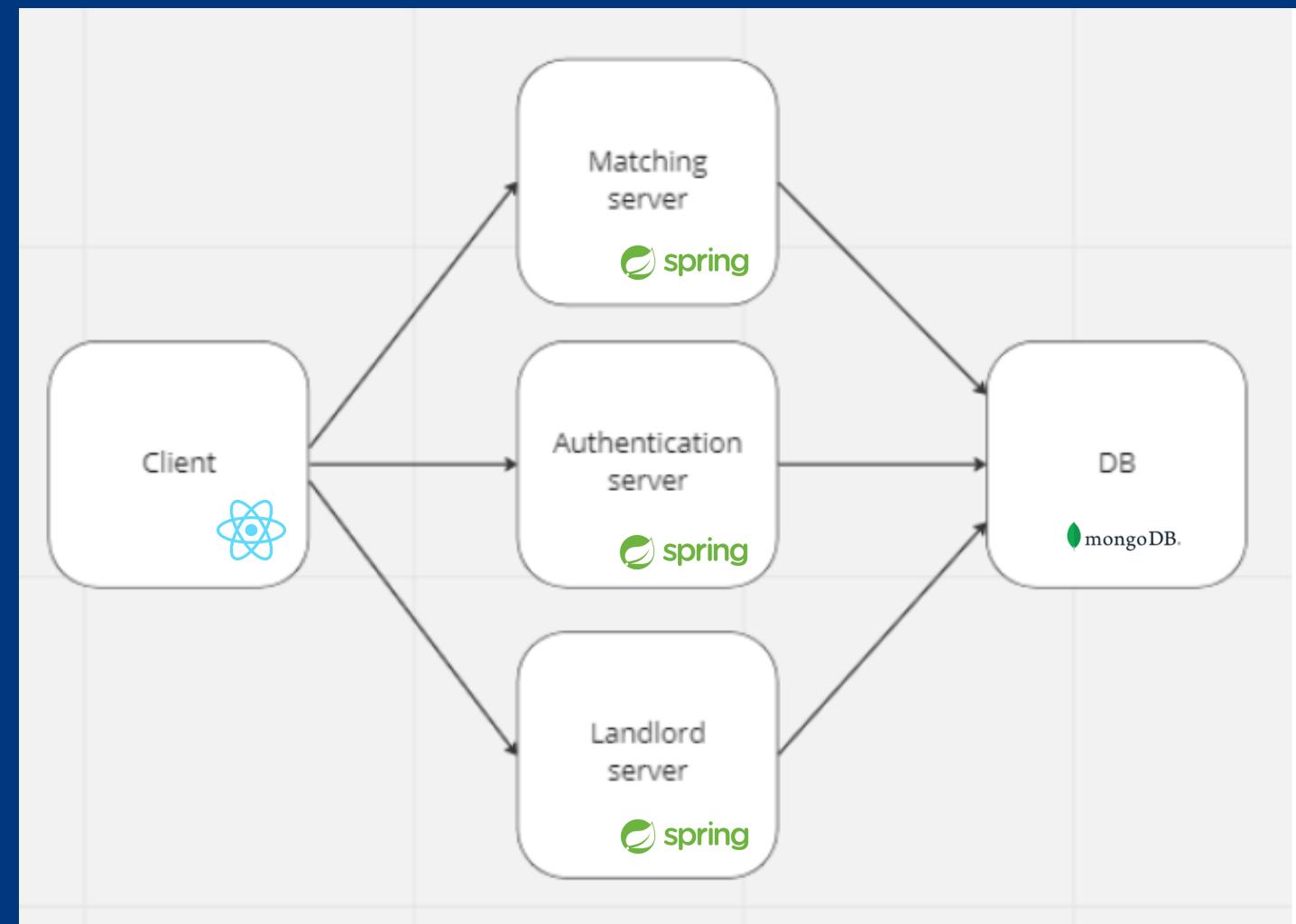
จากการทำ **EVENT STORMING** สามารถแบ่งได้เป็น 3 **BOUNDED CONTEXT**

1. AUTHENTICATION
2. LANDLORD
3. MATCHING

SOFTWARE ARCHITECTURE

ในการออกแบบ SOFTWARE กลุ่มของเราได้มีการนำ SOFTWARE ARCHITECTURE และ ARCHITECTURAL PATTERNS/STYLES ต่างๆ ที่ได้เรียนมาประยุกต์ใช้กับ PROJECT ของกลุ่มเราด้วย ได้แก่

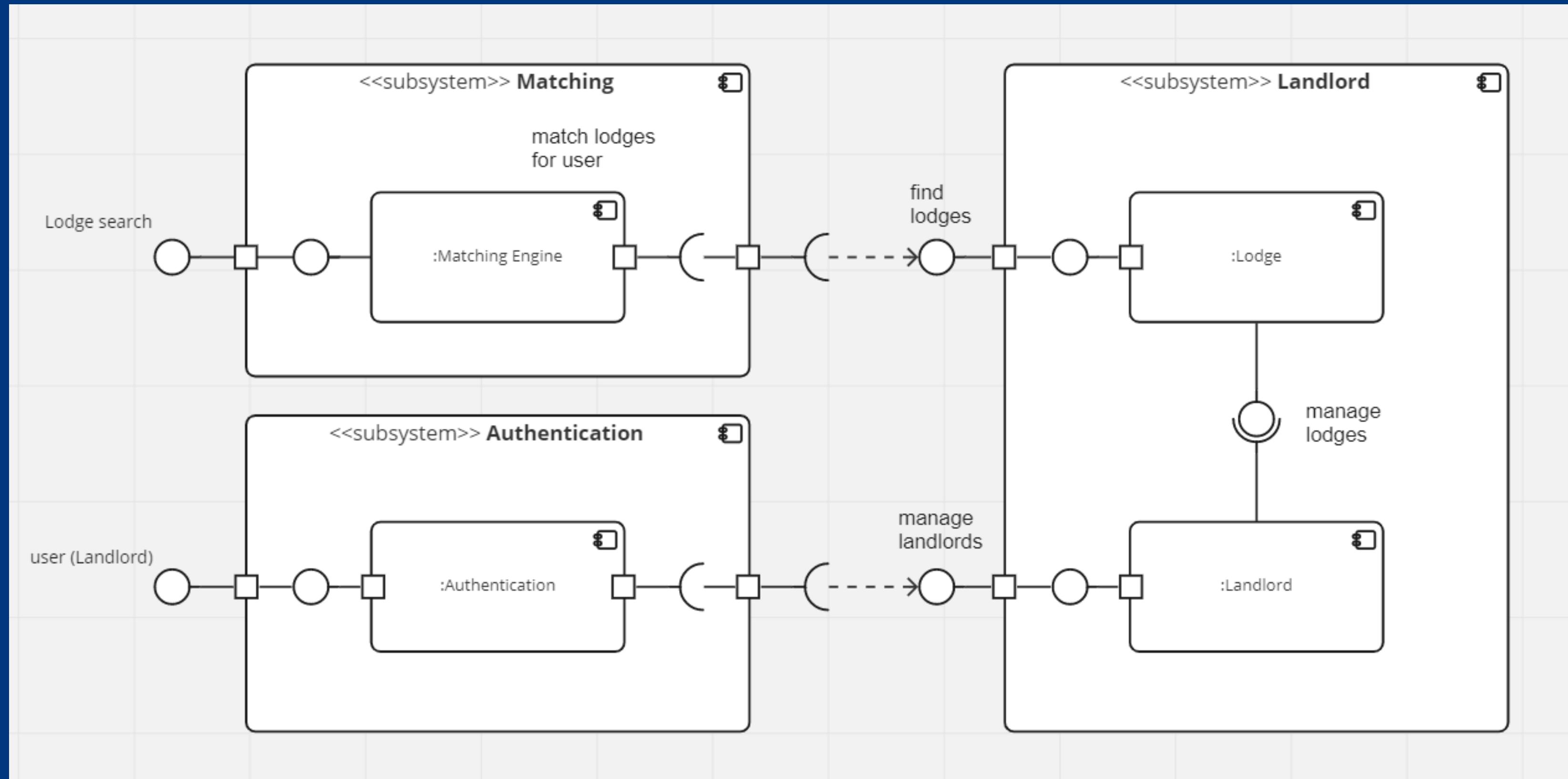
1. CLIENT-SERVER
2. REPRESENTATIONAL STATE TRANSFER (REST)



DIAGRAMS



UML COMPONENT DIAGRAM แสดงการ構成ของ SOFTWARE



DESIGN PATTERN

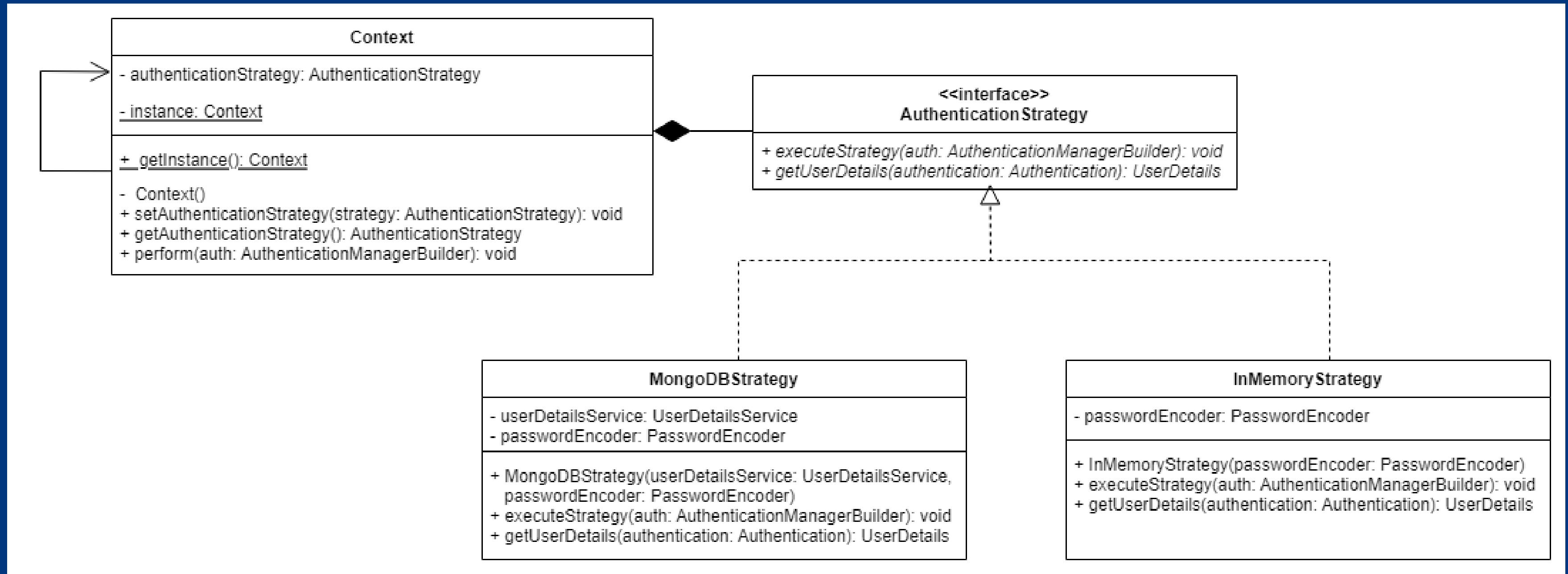
STRATEGY อันที่ 1

- Strategy Pattern ใช้เพื่อสร้างการทำงานหลายรูปแบบ เพื่อให้ client สามารถเลือกไปใช้งานตามความเหมาะสม และแยกการทำงานออกเป็นเรื่องๆ ขาดออกจากกันได้
- Strategy Pattern นำมาใช้ตอน Authentication เพื่อเลือกว่าจะ Authen จากฐานข้อมูลตัวไหน ตัวอย่างเช่น authen จาก mongoDB, authen จาก InMemory(local)



STRATEGY

// Class diagram



STRATEGY

// Strategy interface

```
● ● ●  
1 public interface AuthenticationStrategy {  
2     void executeStrategy(AuthenticationManagerBuilder auth) throws Exception;  
3     UserDetails getUserDetails(Authentication authentication);  
4 }
```

STRATEGY

// Strategies

```
● ● ●  
1 public class InMemoryStrategy implements AuthenticationStrategy {  
2     private PasswordEncoder passwordEncoder;  
3  
4     public InMemoryStrategy(PasswordEncoder passwordEncoder) {  
5         this.passwordEncoder = passwordEncoder;  
6     }  
7  
8     @Override  
9     public void executeStrategy(AuthenticationManagerBuilder auth) throws Exception {  
10         auth.inMemoryAuthentication().withUser("admin").password(passwordEncoder.encode("admin")).roles("USER");  
11     }  
12  
13    @Override  
14    public UserDetails getUserDetails(Authentication authentication) {  
15        return (UserDetails) authentication.getPrincipal();  
16    }  
17 }
```

STRATEGY

// Strategies



```
1 public class MongoDBStrategy implements AuthenticationStrategy {  
2     private final UserDetailsService userDetailsService;  
3     private final PasswordEncoder passwordEncoder;  
4  
5     public MongoDBStrategy(UserDetailsService userDetailsService, PasswordEncoder passwordEncoder) {  
6         this.userDetailsService = userDetailsService;  
7         this.passwordEncoder = passwordEncoder;  
8     }  
9  
10    @Override  
11    public void executeStrategy(AuthenticationManagerBuilder auth) throws Exception {  
12        auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);  
13    }  
14  
15    @Override  
16    public UserDetails getUserDetails(Authentication authentication) {  
17        return (UserDetailsImp) authentication.getPrincipal();  
18    }  
19 }
```

STRATEGY

// Context

```
● ● ●  
1 public class Context {  
2     private static Context instance;  
3     private AuthenticationStrategy authenticationStrategy;  
4  
5     private Context() {  
6     }  
7  
8     public void setAuthenticationStrategy(AuthenticationStrategy strategy){  
9         this.authenticationStrategy = strategy;  
10    }  
11  
12    public AuthenticationStrategy getAuthenticationStrategy() {  
13        return authenticationStrategy;  
14    }  
15  
16    public static Context getInstance(){  
17        if (instance == null){  
18            instance = new Context();  
19        }  
20        return instance;  
21    }  
22  
23    public void perform(AuthenticationManagerBuilder auth) throws Exception {  
24        this.authenticationStrategy.executeStrategy(auth);  
25    }  
26 }
```

STRATEGY

// ส่วนเรียกใช้

```
● ● ●

1 public class SecurityConfig extends WebSecurityConfigurerAdapter {
2     @Autowired
3     private final UserDetailsService userDetailsService;
4     @Autowired
5     private final PasswordEncoder passwordEncoder;
6     private Context context;
7
8     @Override
9     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
10         TypeOfAuth type = TypeOfAuth.MONGODB;
11         context = Context.getInstance();
12
13         if (type == TypeOfAuth.MONGODB) {
14             context.setAuthenticationStrategy(new MongoDBStrategy(userDetailsService, passwordEncoder));
15         } else if (type == TypeOfAuth.INMEMORY) {
16             context.setAuthenticationStrategy(new InMemoryStrategy(passwordEncoder));
17         } else if (type == TypeOfAuth.GOOGLE){
18             //
19         } else if (type == TypeOfAuth.FACEBOOK){
20             //
21         }
22         context.perform(auth);
23     }
```

STRATEGY อันที่ 2

- Design Pattern นี้ใช้เพื่อสร้างการทำงานหลายแบบ เพื่อให้ client เลือกไปใช้งานตามความเหมาะสม และแยกการทำงานออกเป็นเรื่องๆ ขาดจากกันได้
- Design Pattern นี้ใช้อย่างไร นำมาใช้ตอน Search ด้วยชื่อ เราไม่เคยอัลกอริทึมในการค้นหาก็ให้เลือกใช้ตามความเหมาะสม



STRATEGY ວິທີ 2

// STRATEGY INTERFACE



```
1 public interface FindByNameStrategy {  
2     public List<Lodge> executeData(String name);  
3 }
```

STRATEGY ວິທີ 2

// STRATEGIES

```
● ● ●  
1 public class ConcreteStrategyFindByNameNormal implements FindByNameStrategy{  
2     private LodgeRepository lodgeRepository;  
3  
4     public ConcreteStrategyFindByNameNormal(LodgeRepository lodgeRepository){  
5         this.lodgeRepository = lodgeRepository;  
6     }  
7  
8     @Override  
9     public List<Lodge> executeData(String name) {  
10         List<Lodge> data = lodgeRepository.findAll();  
11         List<Lodge> res = new ArrayList<>();  
12         for (int i = 0; i < data.size(); i++) {  
13             if(data.get(i).getInformation().getName().toLowerCase().indexOf(name.toLowerCase()) != -1) {  
14                 res.add(data.get(i));  
15             }  
16         }  
17         return res;  
18     }  
19 }
```

STRATEGY ວິທີ 2

// STRATEGIES



```
1 public class ConcreteStrategyFindByNameSuperAdvance implements FindByNameStrategy{
2     private LodgeRepository lodgeRepository;
3
4     public ConcreteStrategyFindByNameSuperAdvance(LodgeRepository lodgeRepository){
5         this.lodgeRepository = lodgeRepository;
6     }
7     @Override
8     public List<Lodge> executeData(String name) {
9         List<Lodge> data = lodgeRepository.findAll();
10        List<Lodge> res = new ArrayList<>();
11        String inp = name.toLowerCase().replaceAll("\\s+","");
12        for(int i = 0; i < data.size();i++)
13        {
14            if(data.get(i).getInformation().getName().replaceAll("\\s+","");
15            .contains(inp))
16            {
17                res.add(data.get(i));
18            }
19        }
20    }
21 }
22
```

STRATEGY ວິທີ 2

// CONTEXT



```
1 public class Context {  
2     private FindByNameStrategy strategy;  
3  
4     public void setStrategy(FindByNameStrategy strategy){  
5         this.strategy = strategy;  
6     }  
7  
8     public List<Lodge> executeStrategy(String name){  
9         return this.strategy.executeData(name);  
10    }  
11 }  
12
```

STRATEGY อันที่ 2

//ส่วนเรียกใช้

```
● ● ●  
1 public class MatchByNameNormal implements MatchingControllerGet <List<Lodge>> {  
2  
3     private Context context = new Context();  
4     @Autowired  
5     private LodgeRepository lodgeRepository;  
6     @Override  
7     @GetMapping("/findByName2/{name}")  
8     public List<Lodge> matchBy(@PathVariable("name") String str) {  
9         context.setStrategy(new ConcreteStrategyFindByNameNormal(lodgeRepository));  
10        return context.executeStrategy(str);  
11    }  
12 }
```

STRATEGY อันที่ 2

//ส่วนเรียกใช้

```
● ● ●  
1 public class MatchByNameNormal implements MatchingControllerGet <List<Lodge>> {  
2  
3     private Context context = new Context();  
4     @Autowired  
5     private LodgeRepository lodgeRepository;  
6     @Override  
7     @GetMapping("/findByName2/{name}")  
8     public List<Lodge> matchBy(@PathVariable("name") String str) {  
9         context.setStrategy(new ConcreteStrategyFindByNameNormal(lodgeRepository));  
10        return context.executeStrategy(str);  
11    }  
12 }
```

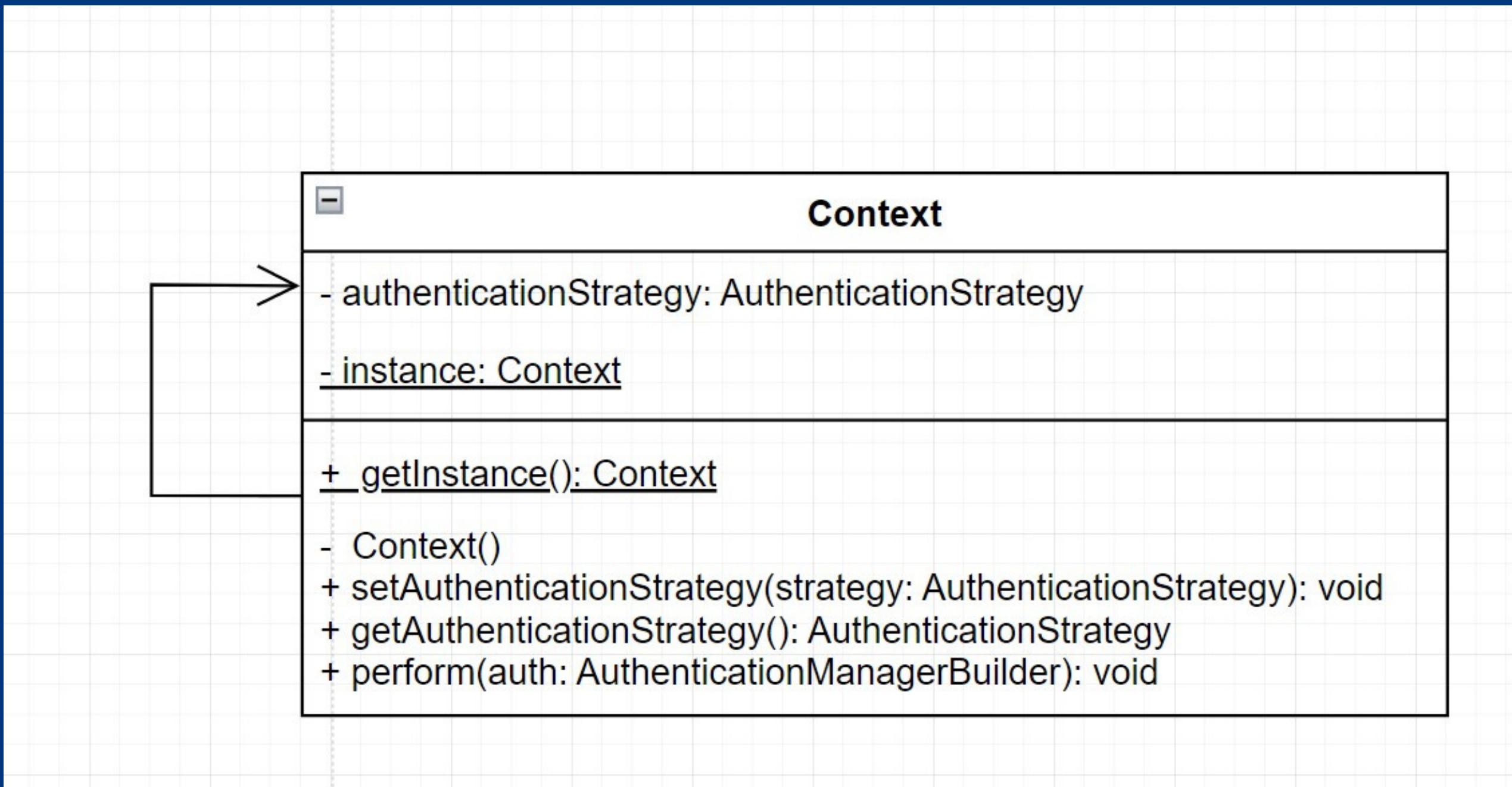
SINGLETON

- Singleton Pattern นี้แก้ปัญหาการอยากระสร้างคลาสที่สามารถนำไปสร้าง object ได้เพียงตัวเดียว
- Singleton Pattern นี้ใช้กับ Context ของ Strategy



SINGLETON

// Class diagram



SINGLETON

// Singleton

```
● ● ●  
1 public class Context {  
2     private static Context instance;  
3     private AuthenticationStrategy authenticationStrategy;  
4  
5     private Context() {  
6     }  
7  
8     public void setAuthenticationStrategy(AuthenticationStrategy strategy){  
9         this.authenticationStrategy = strategy;  
10    }  
11  
12    public AuthenticationStrategy getAuthenticationStrategy() {  
13        return authenticationStrategy;  
14    }  
15  
16    public static Context getInstance(){  
17        if (instance == null){  
18            instance = new Context();  
19        }  
20        return instance;  
21    }  
22  
23    public void perform(AuthenticationManagerBuilder auth) throws Exception {  
24        this.authenticationStrategy.executeStrategy(auth);  
25    }  
26 }
```

SINGLETON

// ส่วนที่เรียกใช้

```
● ● ●

1 public class SecurityConfig extends WebSecurityConfigurerAdapter {
2     .
3     .
4     .
5     private Context context;
6
7     @Override
8     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
9         TypeOfAuth type = TypeOfAuth.MONGODB;
10        context = Context.getInstance();
11        .
12        .
13        .
```

SINGLETON

// ส่วนที่เรียกใช้



```
1 @Override
2 protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response) {
3
4     Context context = Context.getInstance();
5     if (context.getAuthenticationStrategy() instanceof MongoDBStrategy) {
6         System.out.println("MongoDBStrategy Yeah man!!!");
7     } else if (context.getAuthenticationStrategy() instanceof InMemoryStrategy) {
8         System.out.println("InMemoryStrategy okee !!!");
9     }
10 }
```

FUNCTIONALITIES

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



QUALITY ATTRIBUTES

ด้าน USABILITY มีหน้า DASHBOARD ให้ผู้ใช้งานแก้ไขข้อมูลได้ง่าย
ไม่จำเป็นต้องกรอกใหม่หมด

SOURCE OF STIMULUS : USER

STIMULUS : ต้องการเรียนรู้ระบบ

ARTIFACTS : ระบุ

ENVIRONMENT : RUNTIME

RESPONSE : คาดการณ์ความต้องการของผู้ใช้

RESPONSE MEASURE : ความพึงพอใจของลูกค้า

ด้าน MODIFIABILITY DEV สามารถแก้ไขโค้ดในส่วน BACKEND
ได้โดยไม่กระทบส่วนอื่นๆ

SOURCE OF STIMULUS : DEVELOPER

STIMULUS : ต้องการเปลี่ยน UI

ARTIFACTS : UI

ENVIRONMENT : DESIGN TIME

RESPONSE : เปลี่ยนแปลงแล้วไม่กระทบส่วนอื่นๆ

RESPONSE MEASURE : ใช้เวลาบ่อยกว่า 2 ชั่วโมง



ด้าน SECURITY มีระบบที่ชื่อ AUTHEN ไว้ยืนยันตัวตน

SOURCE OF STIMULUS : ผู้ไม่ประสงค์ดี

STIMULUS : พยายามเข้ามาเปลี่ยนแปลงข้อมูลโดยไม่ได้รับอนุญาต

ARTIFACT : ระบบ

ENVIRONMENT : ONLINE

RESPONSE : LOG IN / ใช้งานไม่ได้

RESPONSE MEASURE : ไม่มีข้อมูลที่ได้รับผลกระทบ



สมาชิกในกลุ่ม

63010766

กูมิศักดิ์

แก้วสี

63010767

กูริช

จันทร์ประสาทรี

63010841

วรเทพ

เกียรติคงแสง

63010846

วรรณนัย

เมราเมลีอง

63010852

วรวิชญ์

รรร์มารักษ์วัฒนา

63011018

สุรพัศ

วงศ์ประไพพัตร์

63011075

อับดุลฮาคิม

มาหะ

63011414

อับดุลฮาคิม

มา努

THANK YOU



ส่งห้องคุณไปหาอาจารย์!!!