รายงาน

เรื่อง MEDWARE

โดย

1.	63010082	นาย	กิตติภณ	สิงห์ชม
2.	63010183	นาย	ชนน	กุลกัตติมาส
3.	63010480	นางสาว	นฎคล	จันทราช
4.	63010629	นาย	พชร	หลาวเพ็ชร์
5.	63010885	นาย	วิทวัส	ดรบัณฑิตย์
6.	63010918	นาย	ศิวกร	น้อยสันโดด
7.	63010960	นาย	สวิตต์	ลิ้มเกียรติสถาพร
8.	63011052	นาย	อภิสิทธิ์	ภูกิ่งหิน
9.	63011063	นาย	อริน	สลับสี

เสนอ

ดร. ปริญญา เอกปริญญา

รายงานนี้เป็นส่วนหนึ่งของวิชาสถาปัตยกรรมและการออกแบบซอฟต์แวร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ภาคเรียนที่ 1 ปีการศึกษา 2565

คำนำ

รายงานเล่มนี้จัดทำขึ้นเพื่อเป็นส่วนหนึ่งของวิชาสถาปัตยกรรมและการออกแบบซอฟต์แวร์ เพื่อให้ได้ศึกษาหาความรู้ในเรื่อง Design Pattern & UML diagram และได้ศึกษาอย่างเข้าใจเพื่อเป็นประโยชน์กับการเรียน

คณะผู้จัดทำหวังว่า รายงานเล่มนี้จะเป็นประโยชน์กับผู้อ่าน หรือนักเรียน นักศึกษา ที่กำลังหาข้อมูลเรื่องนี้อยู่ หากมีข้อแนะนำหรือข้อผิดพลาดประการใด ผู้จัดทำขอน้อมรับไว้และขออภัยมา ณ ที่นี้ด้วย

คณะผู้จัดทำ

สารบัญ

ที่มาและจุดประสงค์ของชิ้นงาน

เนื่องจากในปัจจุบันในมีผู้ใช้งานโรงพยาบาลในแต่ละวันเป็นจำนวนมาก จึงเกิดความหนาแน่นในการเข้าใช้บริการ ทำให้เมื่อไปตรวจหรือไปตามนัดแล้วอาจเกิดความล่าช้ามาก และอาจทำให้เกิดความเสี่ยงเพราะความหนาแน่นของผู้เข้าใช้บริการจาก COVID-19 อีกทั้งยังสร้างภาระให้หมอและพยาบาลอย่างมาก นอกจากนี้ยังมีกรณีที่อาจมีธุระกะทันหันหรือว่ามีเหตุสำคัญทำให้ไม่สามารถไปตามนัดได้จึงต้องโทรติดต่อโรงพยาบาลเพื่อขอเลื่อนนัด แต่การโทรติดต่อโรงพยาบาลก็เป็นอีกหนึ่งขั้นตอนที่มีอาจเกิดข้อผิดพลาดได้ เพราะการพูดคุยทางโทรศัพท์ก็มีปัจจัยหลายอย่างที่ก่อให้เกิดการสื่อสารที่ผิดพลาด เช่น เสียงลม สัญญาณไม่ดี เป็นต้น นอกจากนี้การโทรเพื่อติดต่อโรงพยาบาลยังเป็นขั้นตอนที่มีความยุ่งยากเพราะอาจจะโทรไปแล้วสายไม่ว่าง ทำให้ต้องโทรซ้ำ หรือไม่ได้อยู่ในสถานที่ที่สะดวกคุยโทรศัพท์ นี่ก็เป็นหนึ่งในสาเหตุที่คนไข้หลายคนมักจะไม่ติดต่อเลื่อนนัดแต่จะปล่อยให้นัดเสียไปเลยเพื่อความสะดวกของตัวเอง

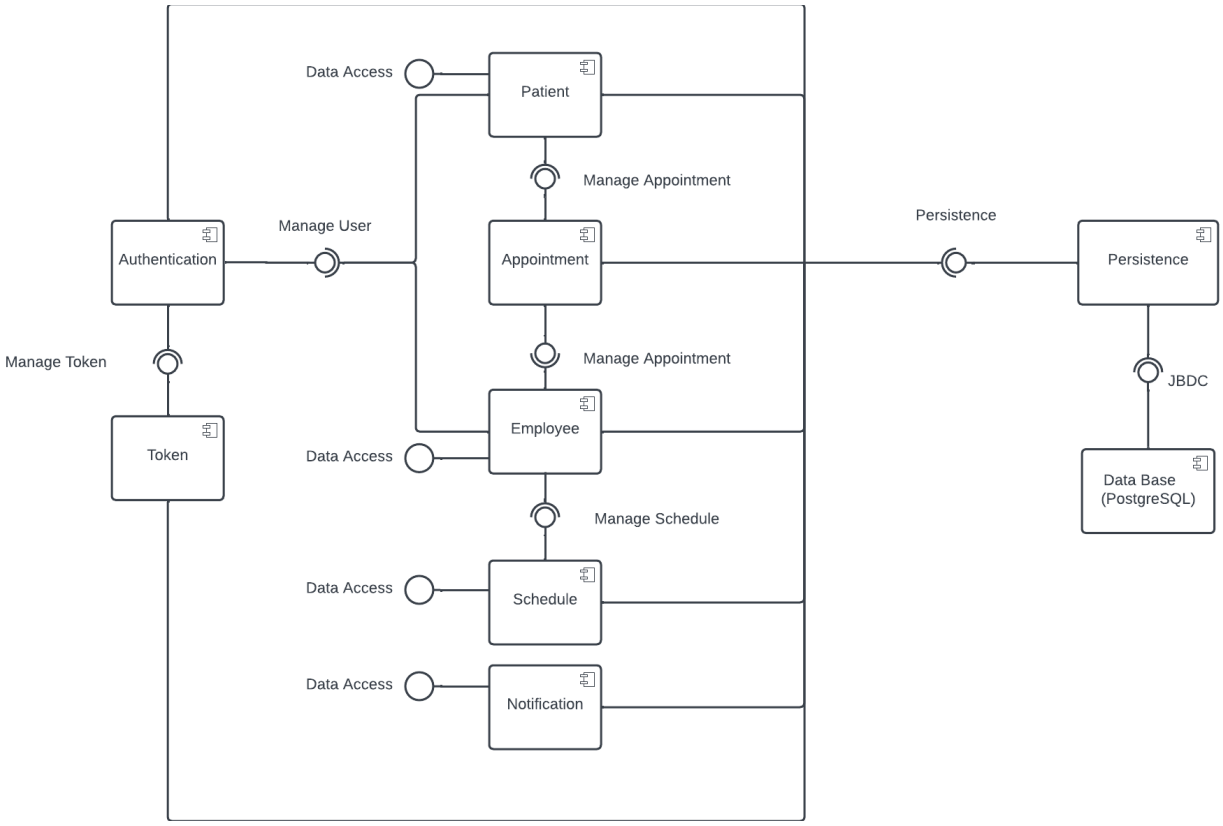
จากปัญหาที่กล่าวมาข้างต้น กลุ่มของพวกเราตัดสินใจที่จะสร้าง Mobile Application ประเภท HealthCare ที่มีฟังก์ชั่นในการจองคิวตรวจสุขภาพ บริจาคเลือด และจัดการนัดพบแพทย์ต่างๆ โดยมีฟังก์ชั่นที่ผู้ป่วยสามารถที่จะจองคิวเพื่อรับการตรวจสุขภาพและบริจาคเลือดล่วงหน้าได้ โดยไม่มีความจำเป็นที่จะต้อง walk-in เผชิญความเสี่ยงที่คิวเต็ม, คนหนาแน่นหรือต้องรอคิวนาน รวมถึงมีฟังก์ชั่นที่ผู้ป่วยสามารถเลื่อนคิวนัดพบแพทย์ได้อย่างอิสระภายใต้กรอบเวลาที่กำหนด และยังมีส่วนที่บุคลากรทางการแพทย์สามารถสร้างนัด เลื่อนนัด ยกเลิกนัดได้ด้วย พร้อมกับการแจ้งเตือนไปยังผู้ป่วยทันทีเมื่อนัดถูกปรับเปลี่ยน เพื่อแก้ไขปัญหาความยุ่งยากและความผิดพลาดในการติดต่อสื่อสารระหว่างผู้ป่วย-โรงพยาบาล, เพิ่มความสะดวกในการเลื่อนนัด, ลดภาระของบุคลากรที่ทำหน้าที่ในการจัดการการนัดหมายและลดจำนวนนัดเสียที่ผู้ป่วยไม่เข้าตามนัด

Software architecture

โดยสถาปัตยกรรมของ Medware มีสถาปัตกรรมหลักเป็น Architectural Style แบบ Representational State Transfer (REST) ซึ่งเป็นการแบ่งออกเป็น 2 ส่วนอย่างชัดเจน คือ Front-End และ Back-End ซึ่งการทำงานของ 2 ส่วนการทำงานนี้จะเป็น Client - Server ที่อยู่ระหว่าง Internet โดย Back-End จะทำหน้าที่รอรับ Request ตามที่ User สั่งให้ Front-End ทำงาน และทำหน้าที่เป็นตัวกลางระหว่าง Front-End และ Database โดยทำหน้าที่คำนวณและจัดการ Business Logic ประกอบด้วยการเก็บข้อมูล จัดการข้อมูล และให้บริการข้อมูลทั้งในเรื่องของ ข้อมูลผู้ป่วย ข้อมูลบุคคลากรในโรงพยาบาล ข้อมูลตารางเวลา ข้อมูลการนัดเวลา และข้อมูลเกี่ยวกับการแจ้งเตือนเพื่อเก็บ Log การแจ้งเตือน

โดยมีการนำหลักการของ Domain-Driven Design มาใช้ในการออกแบบสถาปัตยกรรมของ Medware อีกด้วย โดยเริ่มจากการแบ่ง Bounded Context จาก Domain หลักคือการเป็นตัวกลางในการอำนวยความสะดวกการจองนัดภายในโรงพยาบาลเพื่อลดปริมาณของผู้ป่วยที่ต้องรอขณะมาใช้บริการ โดยได้ทำการแบ่งเป็น 4 Bounded Context ประกอบด้วย Registration, Patient Appointment, Employee Appointment, Notification และได้ทำการแบ่งสมาชิกภายในกลุ่มทำงานตาม Bounded Context ที่ได้เลือกไว้เป็นทีมย่อยภายใน Bounded Context แต่ละ Bounded Context มีหน้าที่ของแต่ละ Sub System โดยที่ Registration ทำหน้าที่ในเรื่องของการลงทะเบียน เก็บข้อมูล และให้บริการข้อมูลของบุคคลทั้งหมด, Patient Appointment ทำหน้าที่เรื่องของการจัดการคิวนัดโดยจากผู้ป่วยที่มารับบริการ, Employee Appointment ทำหน้าที่เรื่องของการจัดการคิวนัดโดยบุคคลากรของโรงพยาบาล, Notification ทำหน้าที่เรื่องของการรับข้อมูล ของการแสดงการแจ้งเตือนจาก Front-End Application

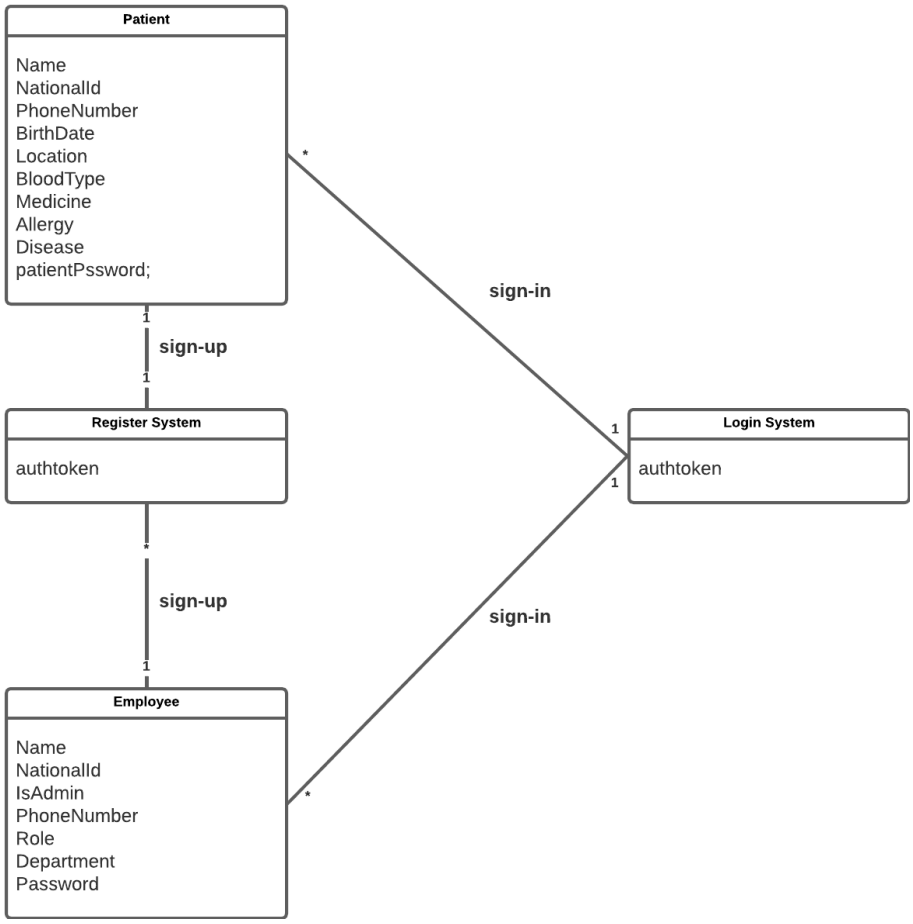
Software architecture (ต่อ)



Bounded context

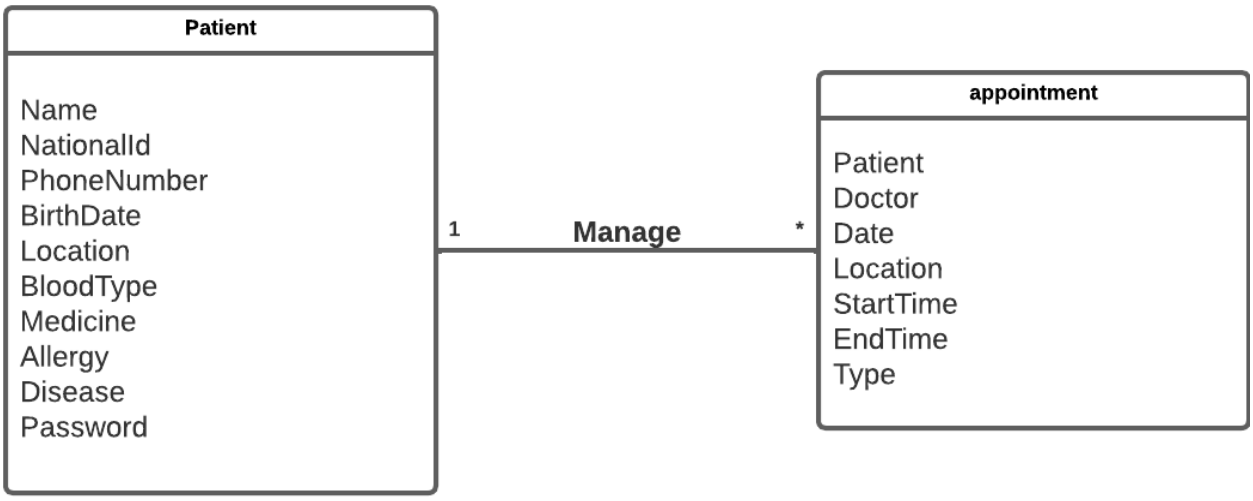
“MEDWARE” ประกอบไปด้วย 4 Bounded contexts

1. การลงทะเบียนและเข้าสู่ระบบของ User



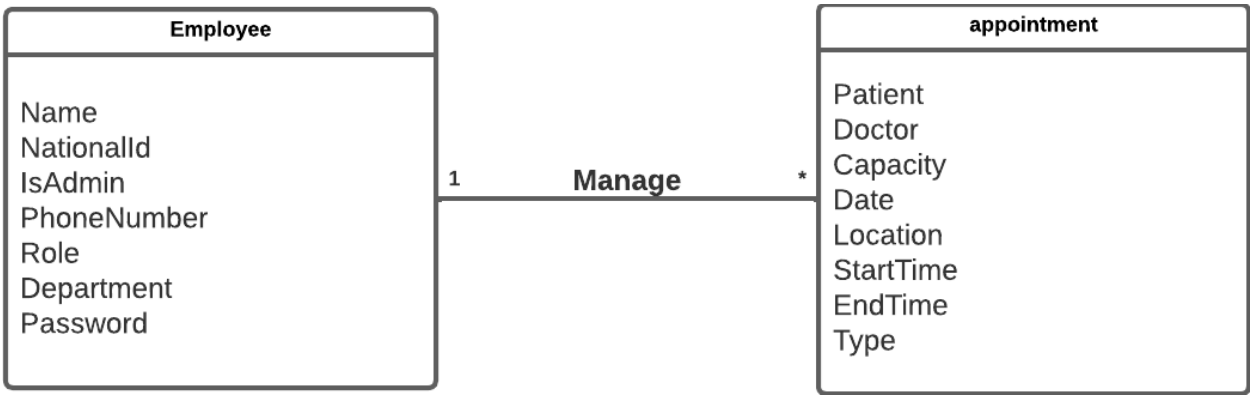
ภาพแสดง Domain model ของการลงทะเบียนและเข้าสู่ระบบของ User โดย User สามารถแบ่งได้ 2 ประเภท ได้แก่ คนไข้ (Patient) และบุคลากร (Employee) โดยใช้ข้อมูลส่วนตัวต่างๆในการลงทะเบียน และใช้ NationalId และ Password ในการเข้าสู่ระบบ

2. การจัดการAppointmentของUser<Patient>



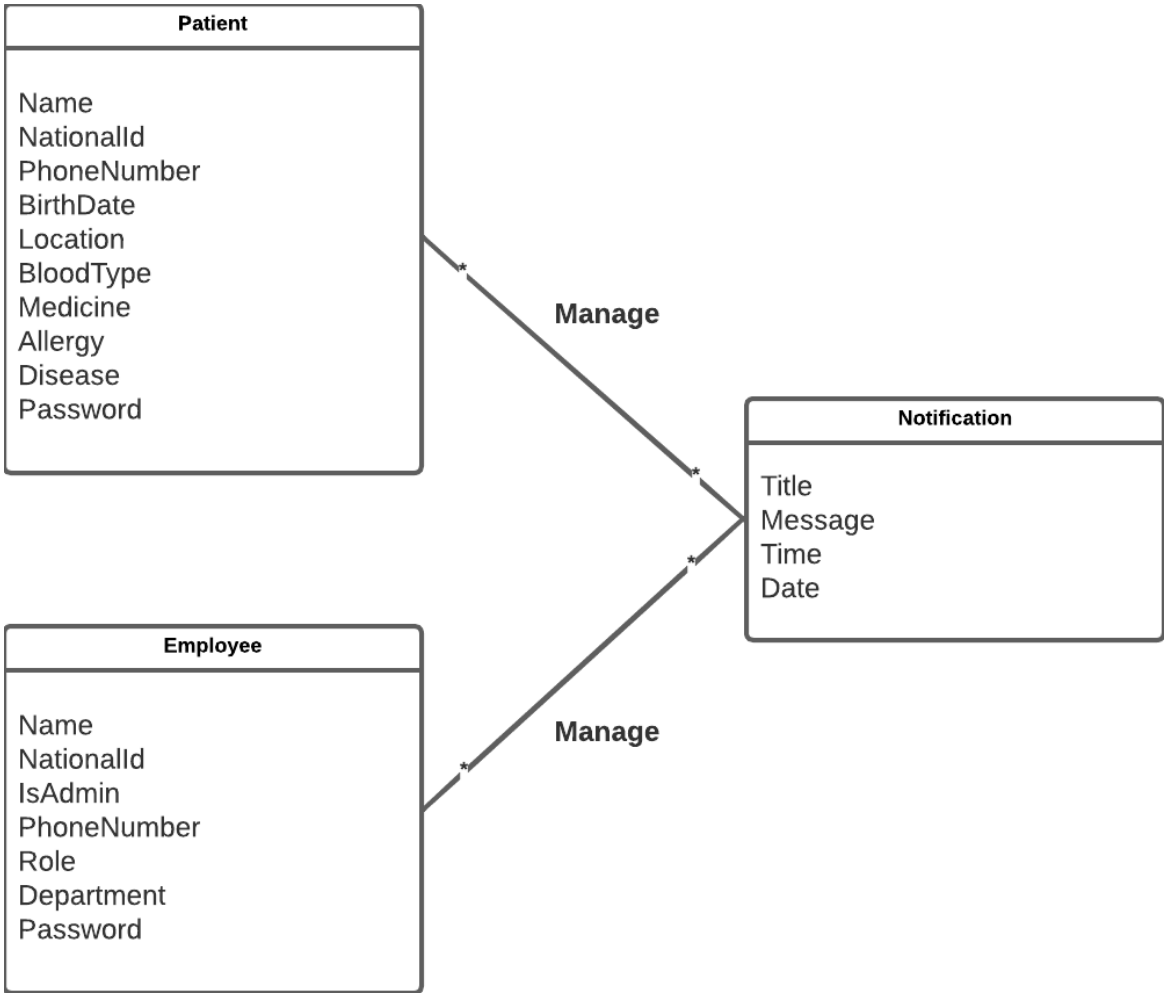
ภาพแสดง Domain model ของการจัดการการนัดหมาย (Appointment) ของคนไข้ (Patient) โดยคนไข้สามารถจัดการการนัดหมายได้ โดยสามารถเพิ่มนัดหมาย เลื่อนนัดหมายได้ภายในกรอบระยะเวลาที่กำหนดไว้

3. **การจัดการScheduleของUser<Employee>**



ภาพแสดง Domain model ของการจัดการนัดหมาย (Appointment) ของบุคลากร (Employee) โดยบุคลากรสามารถจัดการการนัดหมายได้ โดยสามารถเพิ่มนัดหมาย เลื่อนนัดหมาย ยกเลิกนัดหมาย เปลี่ยนหมอที่เข้าตรวจในนัดหมายนั้นๆได้

4. การแจ้งเตือน (notification)



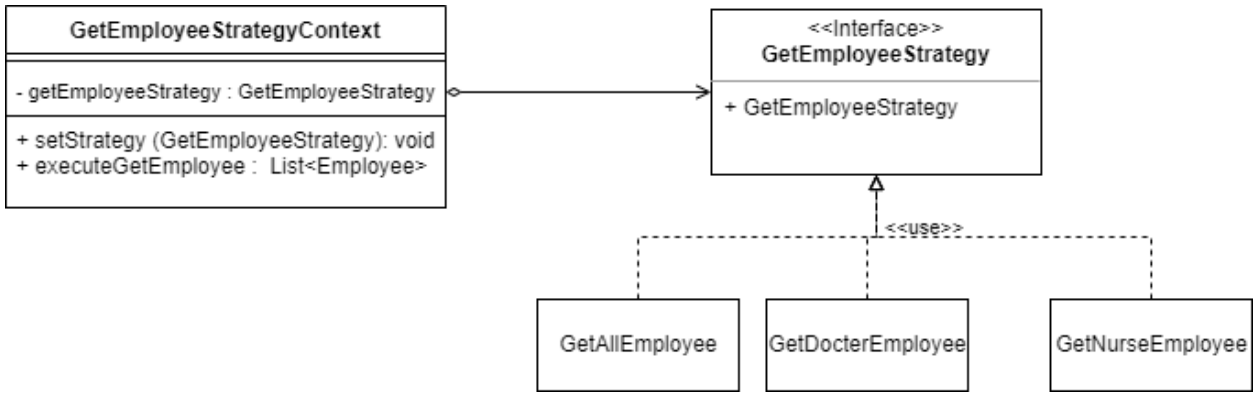
ภาพแสดง Domain model ของการแจ้งเตือน โดย User ทั้งคนไข้ (Patient) และบุคลากร (Employee) สามารถที่จะทำให้เกิดการแจ้งเตือนได้

Design Patterns

1. **Strategy**

ปัญหาที่เกิดขึ้นก่อนปรับใช้ Strategy คือการที่ Frontend ต้องการเรียกใช้การเสิร์ช Employee ในรูปแบบต่างๆ เช่น การเรียกรายชื่อ Employee ทั้งหมด, การเรียกเฉพาะ Employee ที่เป็นหมอก็จะต้องเรียกใช้ path แยกออกจากกันเป็นหลายๆ path เกิดความ

แต่เมื่อปรับใช้ Strategy แล้วทำให้ Frontend สามารถ custom การเรียกรายชื่อได้โดยไม่ต้องเปลี่ยน path ที่ใช้



ภาพแสดง Class Diagram ของ Strategy ที่ถูกนำมาปรับใช้ในงาน MEDWARE

```java
@Component
public class GetAllEmployee implements GetEmployeeStrategy {

    @Autowired
    private EmployeeRepository employeeRepository;

    public GetAllEmployee(EmployeeRepository employeeRepository) {
        this.employeeRepository = employeeRepository;
    }

    @Override
    public List<Employee> getEmployee() {
        return employeeRepository.findAll();
    }

}
```

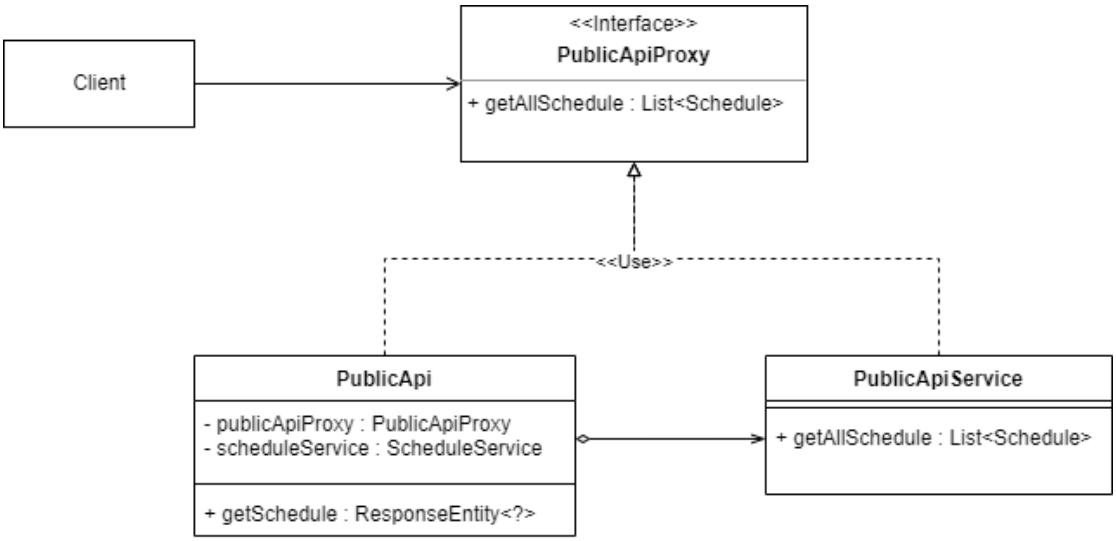
```java
public interface GetEmployeeStrategy {
    List<Employee> getEmployee();
}
```

```java
public class GetEmployeeStrategyContext {

    private GetEmployeeStrategy getEmployeeStrategy;

    public GetEmployeeStrategyContext() {
        this.setStrategy(getEmployeeStrategy);
    }

    public void setStrategy(GetEmployeeStrategy getEmployeeStrategy) {
        this.getEmployeeStrategy = getEmployeeStrategy;
    }

    public List<Employee> executeGetEmployee() {
        return this.getEmployeeStrategy.getEmployee();
    }

}
```

ภาพแสดงตัวอย่าง Code ในการปรับใช้ Strategy ในงาน MEDWARE

2. Proxy

ปัญหาที่เกิดขึ้นก่อนการปรับใช้ Proxy คือ api มีการจัดกระจายทำให้การแก้ไขและเรียกดูนั้นทำได้ยาก และควบคุมการเข้าถึงตัว serviceต่างๆด้วยinterfaceที่สร้างขึ้นทำให้การเรียกใช้servivceมีความปลอดภัยมากขึ้น



ภาพแสดง Class Diagram ของ Proxy ที่ถูกนำมาปรับใช้ในงาน MEDWARE

```java
@Service
public class PublicApiService implements PublicApiProxy {

    private ScheduleService scheduleService;

    public PublicApiService(ScheduleService scheduleService) {
        this.scheduleService = scheduleService;
    }

    public List<Schedule> getAllSchedule() {
        try {
            List<Schedule> data = scheduleService.getSchedule();
            return data;
        } catch (Exception e) {
            System.out.println(e);
            List<Schedule> tempList = new ArrayList<Schedule>();
            return tempList;
        }
    }

}
```

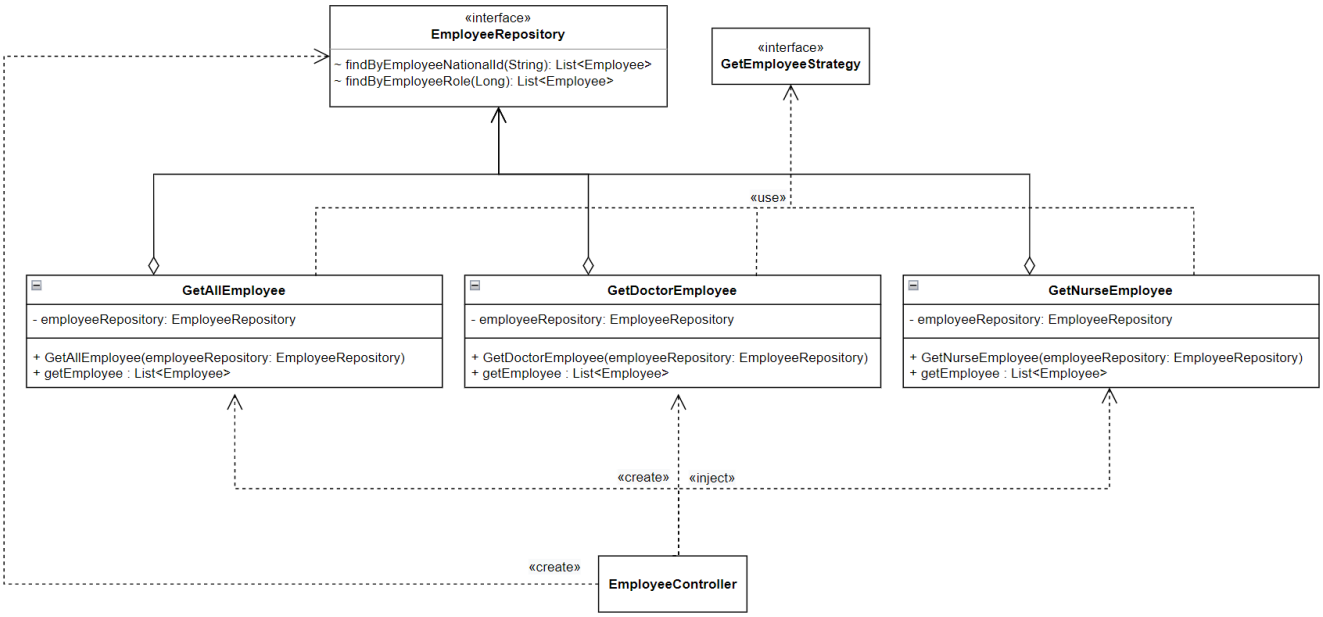
```java
public interface PublicApiProxy {
    List<Schedule> getAllSchedule();
}
```

```java
@EnableCaching
@RestController
public class PublicApi {

    @Autowired
    private PublicApiProxy publicApiProxy;

    @Autowired
    private ScheduleService scheduleService;

    @GetMapping(path = "/getSlotTime")
    public ResponseEntity<?> getSchedule() {
        try {
            List<Schedule> data = publicApiProxy.getAllSchedule();
            if (!(data != null && data.isEmpty())) {
                return ResponseEntity.ok().body(data);
            } else {
                return ResponseEntity.status(400).body("Not Found Data");
            }
        } catch (Exception e) {
            System.out.println(e);
            return ResponseEntity.status(500).body("server error");
        }
    }

}
```

ภาพแสดงตัวอย่าง code ในการปรับใช้ Proxy ในงาน MEDWARE

3. Dependency Injection

ปัญหาที่เกิดขึ้นก่อนจะปรับใช้ Dependency Injection คือเกิดการสร้าง Object โดยไม่จำเป็นในหลายๆครั้ง ทำให้เกิด coupling ขึ้นมากมายในโปรแกรม และเมื่อปรับใช้แล้วก็ลดการสร้าง object ที่ไม่จำเป็นลง



ภาพแสดง Class Diagram ของ Dependency Injection ที่ถูกใช้ในงาน MEDWARE

```java
@Component
public class GetAllEmployee implements GetEmployeeStrategy {

    @Autowired
    private EmployeeRepository employeeRepository;

    public GetAllEmployee(EmployeeRepository employeeRepository) {
        this.employeeRepository = employeeRepository;
    }

    @Override
    public List<Employee> getEmployee() {
        return employeeRepository.findAll();
    }

}
```

```java
@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {

    List<Employee> findByEmployeeNationalId(String employeeNationalId);

    List<Employee> findByEmployeeRole(Long employeeRoleId);

    Employee save(Optional<Employee> _employee);
```

```java
GetEmployeeStrategyContext getEmployeeStrategyContext = new GetEmployeeStrategyContext();

public List<Employee> getEmployee(int strategyType) {

    try {
        if (strategyType == 1) {
            getEmployeeStrategyContext.setStrategy(new GetAllEmployee(employeeRepository));
        } else if (strategyType == 2) {
            getEmployeeStrategyContext.setStrategy(new GetDoctorEmployee(employeeRepository));
        } else if (strategyType == 3) {
            getEmployeeStrategyContext.setStrategy(new GetNurseEmployee(employeeRepository));
        }
        System.out.println(getEmployeeStrategyContext.executeGetEmployee());
        return getEmployeeStrategyContext.executeGetEmployee();

    } catch (Exception e) {
        System.out.println(e);
        List<Employee> tempList = new ArrayList<Employee>();
        return tempList;
    }
```

ภาพแสดงตัวอย่าง code ในการปรับใช้ Dependency Injection ในงาน MEDWARE

Quality Attributes Scenario

Availability :
- Source of stimulus : User
- Stimulus : Unanticipated Message (up to 150 character message)
- Artifact : Process
- Environment : Runtime
- Response : Nothing
- Response Measure : No Downtime

Integrability :
- Source of stimulus : Developer
- Stimulus : Wishes to Integrated new component
- Artifact : Code
- Environment : Development Time
- Response : Integrated Complete
- Response Measure : 15 Line of Code Change

Modifiability :
- Source of stimulus : Developer
- Stimulus : Want to change Database
- Environment : Development Time
- Artifact : Code
- Response : Modification Complete
- Response Measure : in 1 hours

Modifiability :
- Source of stimulus : Developer
- Stimulus : Want to change UI
- Environment : Development Time
- Artifact : Code
- Response : Modification Complete
- Response Measure : In 1 week

Performance :
- Source of stimulus : User
- Stimulus : Normal request
- Environment : Runtime
- Artifact : System
- Response : get Response
- Response Measure : Average latency 417 millisecond

Security :
- Source of stimulus : System Attacker
- Stimulus : perform the SQL Injection
- Environment : Runtime
- Artifact : System
- Response : Auth Failed
- Response Measure : Data and services are protected from unauthorized access.

Usability

- Source of stimulus : End user
- Stimulus : Want to learn system feature
- Environment : Runtime
- Artifact : GUI
- Response : System provide needed feature
- Response measure : 15 minute in use

Usability

- Source of stimulus : End user
- Stimulus : use a system efficiently
- Environment : Runtime
- Artifact : GUI
- Response : User familiar with system
- Response measure : 30 minute in use