

SERVER SIDE "INT204"

MVC Web app architectures use spring framework

- [] modern : ໄຟ້ໄຫວ່າ data (JSON) ສ່ວນ FE
- [] spa : single page Application
- [] ແບກ front ແລະ Back ອອງຈາກນີ້

"WEB APP ARCHITECTURE"

ຄວາມອຸນຕະແບບເວັບໄຟ້ຈົກກະຣາກ
request ໃປຖື້ງ server ໄທ້ອັນທຶນ
(ດັນໃນທີ່ client ໄກສໍາທ່ານໄດ້
ແລ້ວຕ່ອນບໍ່ສ່ວນ server ທີ່ເຄີຍ)



INTRODUCTION SPRING BOOT"



CONTAINER SERVICES

- ຊັບຕາມຈົກກະຣາກ component ກ່ອງໃປຢູ່ container

- ຕົ້ນ Life - cycle (ຖັນເພື່ອໃຫ້ new obj / ໃຫດເອົາເປີດ ມີໂດຍ close ຮັງ)

- ຈົກກະ dependency

- ດັນໃນ component ກ່ອງຕົ້ນກັນ

- configuration (ດ່ານ mapping ວິທີຜົນຕົ້ນເຮັດຕົວຫານາ)

MIDDLEWARE SERVICE

- Transaction management

- Security

- Thread management

- Object + resource pooling

- remote access for components

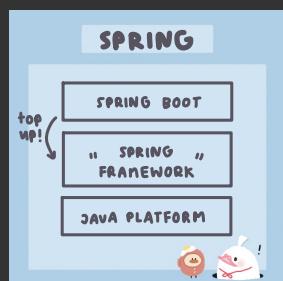
AOP (Aspect Oriented Programming)

- ແນວດຕາການຮ່ວມປະກາດໂປຣແກຣມ / ກຽວຂ້ອງນີ້ oop cross-cutting concerns

- ແນວດຕາການປະກາດ concerns (ແນວດຕາການເປັນຫຼັງທີ່ແນກຕາມຈົກກະຣາກນີ້ໄດ້) ດ້ວຍນາງ services ກັນ

- ແນວດຕາການ transaction , authentication , logging , security etc.

- ທ່ານ AOP : ເກມເຮົາແກ່ໄດ້ອົມ calling ໄປແກ້ໄນ xml file (iven flow ນີ້ກ່າວໜູນກວາ) ມີນໍາ maintenance
ເທົ່ານັ້ນ



SPRING dependency Injection or Inversion of Control (IoC)

= ຊ່ວນໃຫ້ທ່າແນນ loosely coupled (ນລວມ) flexible

SPRING BOOT (project build on spring framework) + spring boot SF

= ເນື່ອ configutations ມີຫຼັກຂອງໃຫຍ່ເປົ້າໃຈ (minimal or zero)

= ແກ້ໄຂ deploy = run ໄດ້ໄລ + ສະຕາກອນການເປັນ micro service



SPRING FRAMEWORK

= manually change almost all configuration

Features

- lightweight : ຢຸ່າດເປົ້າໃໝ່ (1MB)

- Inversion of Control (IoC) : ອົບນາງ how to it should be created ໂດຍກ່ຽວຂ້ອງ service ໄປສ່ວນ configuration file

- Aspect oriented (AOP)

- container : ຕົ້ນ life cycle + configuration

- MVC Framework

- Transaction Management

container (Modules)

- spring AOP
- spring ORM
- spring web
- spring DAO
- spring context
- spring web MVC
- spring core

dependency Injection

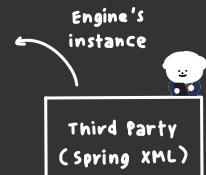
SPRING INVERSION OF CONTROL (IoC / DI)

WITH OUT DI

```
Car
Engine engine;
getEngine() {
    engine= new Engine();
}
```

WITH DI

```
Car
Engine engine;
```



TYPE (ໃຫ້ວິຊາ Attribute)

1. setter injection

new obj ກ່ອນແລະດ້ວຍໃຫ້ setter ພົບປົກທີ່ກ່າວ

2. constructor Injection

ຈົ່າກາປະເຈົ້າຕົ້ນຕ່າງໆ

3. Field Injection (annotation only)

ໃຫ້ວິຊາກ່າວສ່າງ obj ແລະໃຫ້ກາງຈົດເຫັນລົງບໍ່ນ

By reflections

INVERSION OF CONTROL



OOD : SOLID

- Single-responsibility principle : หน้ากากการตั้งค่าให้ object รับผิดชอบเรื่องเดียว
- Open-close principle : เป็นไปได้ obj ของบ้านสามารถรักษาตัว + ไม่สามารถปรับแก้
- Liskov Substitution principle
- Interface Segregation principle
- Dependency Inversion principle *



DEPENDENCY INJECTION (constructor + setter injection)

```
<!-- Constructor Injection -->
<bean id="car" class="sit.int204.lab01.beans.Car"> primitive+String-based
    <!-- Literal Injection กรณี value ต้องมีความต้องการที่ ต้องเป็น -->
    <constructor>arg name="brand" value="Benz E250D" />
    <constructor>arg name="chasisNumber" value="ZM25487ER" />
    <!-- Object Injection กรณี ref ต้องสืบทอดกัน -->
    <constructor>arg name="engine" ref="2KD-FTV" />
</bean>

<bean id="1KD-FTV" class="sit.int204.lab01.beans.DieselEngine">
    <constructor>arg value="2988" type="int" />
</bean>

<!-- คือเมื่อเขียนบัญชีแล้วก็สามารถร่างตัวเอง เมื่อต้องการใช้ตัวเองได้ -->
<bean id="2KD-FTV" class="sit.int204.lab01.beans.GasolineEngine">
    <constructor>arg value="1200" type="int" />
</bean>

<!-- Setter Injection ตัวเราเองจะมาใช้ตัวเองเพื่อยื่นเข้า constructor-arg เป็น property
ให้ใน class ที่เราต้องการใช้ตัวเอง setter ตัวเอง -->
<bean id="carX" class="sit.int204.lab01.beans.Car">
    <property name="brand" value="BMW X7" />
    <property name="chasisNumber" value="KE4557" />
    <property name="engine" ref="1KD-FTV" />
</bean>
```

```
public class CarApp {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(configLocation: "ApplicationContext.xml");
        Car car = (Car) context.getBean(s: "car");
        car.start();
        System.out.println(car);
        System.out.println("-----");
        Car carX = (Car) context.getBean(s: "carX");
        carX.start();
        System.out.println(carX);
    }
}
```

↓ run

Turn On - Gasoline Engine
 Car: ZM25487ER - Benz E250D, 1200

Turn On - Diesel Engine
 Car: KE4557 - BMW X7, 2988

* วิธีการ inject collection value 1 ล้อ (List , Map , Set)

ล สร้าง property ทั่วไปที่ list > value

(กรณีของ Java ให้ทำการเปลี่ยนแปลงข้อมูลใน class หรือหันมือ)

DIFFERENCE BTW CONSTRUCTOR & SETTER INJECTION

- constructor ไม่สามารถทำ partial dependency ได้
- กรณี override จะลบ setter (หันเพื่อ: setter override constructor)
- change value ของ setter = flexible มากกว่า

1. constructor เป็นตัว

2. setter

SPRING FRAMEWORK (IoC = core von spring framework)



IoC (Inversion of control)

- = dependency injection : inversion of control pattern
- = สร้าง instance + obj ขึ้นมาให้
- = ช่วยทำให้เป็นแบบ loose : ไม่ต้องเขียนโปรแกรมเพื่อเชื่อม component
 - + ก็มีการแก้ไขข้อมูลของ component ได้โดยการแก้ไข configuration file 1 ตัว (1 ไม้ต้องแก้ไข file code)
- หน้าที่ (main) : new obj = สร้าง instantiate
 - : configue obj (ดู file config)
 - : Assemble IoC obj ทุกอย่างที่มีอยู่ในหน้าที่ทำงาน
- ประเมิน
 1. BeanFactory Interface
 2. ApplicationContext Interface (สำคัญ) + รองรับความสามารถหลายอย่าง + ทำให้หัวใจของบุรุษมากขึ้น

AUTOWIRING IN SPRING (autowire)

- = feature ที่ต้อง inject obj ให้กับตัวมือของบุรุษ: ใจพูดกันบ่อ: ใจ
- = ต้องได้ทั้ง 2 แบบ constructor + setter
- = ใจกับ primitive ||: string value บ่อ: ใจ (ใจเดียวกับ obj)
- = ใจดี : เขียนง่ายดีสุดๆ // ใจเสื้อ : ควบคุมใจได้
- Modes
 1. No (default) : ไม่มีการให้ autowiring
 2. byName : ถูกจัดชั้น (ต้องมี setter)
 3. byType : ถูกจัดชั้น ซึ่งต้องมีตัว obj เดียว (bean เดียว) (ต้องมี setter)
 4. constructor : เข้าใจ constructor ที่มีแล้วจะตรวจสอบ bean ว่ามีตัวไหนตรงกัน (เจอกันที่ชื่อของกัน)
 5. autodetect : X ไม่มีแล้ว

byType

```
<!-- byType นี้ต้อง engine  minden bean id="oneKd" ที่มี Type engine ให้มีหน้าที่
      แต่ต้องไม่มี bean ที่ชื่อเป็น engine ให้ต้อง error
<bean id="car" class="sit.int204.lab01.beans.Car" autowire="byType">
    <property name="brand" value="BMW X7" />
    <property name="chasisNumber" value="KE4557" />
</bean>

<bean id="oneKd" class="sit.int204.lab01.beans.DieselEngine">
    <constructor>arg value="1111" type="int" />
</bean>
```

constructor

```
<!-- constructor -->
<bean id="car" class="sit.int204.lab01.beans.Car" autowire="constructor" />

<bean id="oneKd" class="sit.int204.lab01.beans.DieselEngine">
    <property name="capacity" value="120" />
</bean>

<bean id="brand" class="java.lang.String">
    <constructor>arg value="Honda CRV" />
</bean>

<bean id="chasisNumber" class="java.lang.String">
    <constructor>arg value="BY-CONST-1234" />
</bean>
```

หากมี map 2 ตัว

ก ต้องไม่กรองเลือกกัน autowire

SPRING BEAN SCOPE (scope)

- * 1. (App ทั่วไป) Singleton : (default) สร้าง obj รูปเดียว ให้เรา get ก็จะได้ obj (instance) ตัวเดียว
- * 2. (App ทั่วไป) prototype : สร้างตัวในแต่ละรอบ ทุกครั้งที่มีการ get
- 3. request : ครอบเท่าที่ request ดำเนินส่งผล
- 4. session : ตาม HTTP Session
- 5. application : ครอบ App ที่ servletContext
- 6. websocket : session network (ครอบคนใช้เว็บร่วมกัน)

SIGLETON SCOPE
carA ID : 2035070981
carB ID : 2035070981

PROTOTYPE SCOPE
carA ID : 2035070981
carB ID : 1219161283

```
<!-- ผู้อ่านแบบ scope="prototype" -->
<bean id="carZ" class="sit.int204.lab01.beans.Car" autowire="constructor" scope="singleton"/>
```

/ หมวด core JAVA (JSE) design pattern

DESIGN PATTERNS FOR CREATE OBJ (creational design pattern)

1. Singleton : only one (สร้างตัวเดียว)
2. Prototype : cloning (สร้างใหม่รื้อปู)
3. Factory Method : สร้าง class ที่มีหน้าชัดเจน (ฟ์factory class)
4. Abstract Factory : สร้าง obj with theme
5. Builder : สร้าง complex obj w/ many options

static object : ผู้ตัวเดียวทั่ว App
private constructor : ไม่รับ private ไม่ใช้ new ได้
static Method : นำ obj ผ่านตัวนี้

```
public class Counter {
    private static final Counter INSTANCE = new Counter();
    private int count;

    private Counter() {
    }

    public static Counter getInstance() {
        return INSTANCE;
    }

    public void increment() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

ที่ 3 types 1. static factory ที่ return instance ของตัวเอง : singleton (return ตัว obj เอง)

↓

```
<!-- เป็นแบบ singleton ไม่สามารถที่จะแก้ไขตัวเดิม (getInstance ที่แบบ singleton) -->
<bean id="counter" class="sit.int204.lab01.beans.Counter" factory-method="getInstance" />
```

2. static factory ที่ return instance ของ class อื่น : e.g. persistance

3. non-static factory ที่ return instance ของ class อื่น : ต้องสร้าง obj ใหม่ e.g. EntityManagerFactory

```
<!-- แบบ static แต่ต้อง obj มากที่สุดแทน -->
<bean id="c" class="sit.int204.lab01.beans.GeometricShapeFactory" factory-method="getCircle" />
<bean id="r" class="sit.int204.lab01.beans.GeometricShapeFactory" factory-method="getRectangle" />

<!-- แบบ non-static ต้องหัวใจ obj geometricShapeFactory รับมาตอนเชื่อมนาทีต่อไป -->
<bean id="geometricShapeFactory" class="sit.int204.lab01.beans.GeometricShapeFactory" />
<bean id="circle" factory-bean="geometricShapeFactory" class="sit.int204.lab01.beans.Circle" />
<bean id="rectangle" factory-bean="geometricShapeFactory" class="sit.int204.lab01.beans.Rectangle">
    <property name="height" value="2.5" />
    <property name="width" value="3.2" />
</bean>
```

```
GeometricShape rectangle = (GeometricShape) context.getBean( "rectangle" );
System.out.println(rectangle);
GeometricShape circle = (GeometricShape) context.getBean( "circle" );
System.out.println(circle);
```

Rectangle:(3.2 X 2.5)->8.0
Circle:(1.0)->3.141592653589793

```
GeometricShape.java
public interface GeometricShape {
    public double getArea();
}

Circle.java
public class Circle implements GeometricShape {
    private double radius = 1.0;
    @Override
    public double getArea() {
        return radius * radius * Math.PI;
    }
    @Override
    public String toString() {
        return "Circle(" + radius + ")->" + getArea();
    }
}

Rectangle.java
public class Rectangle implements GeometricShape {
    private double width = 1.0;
    private double height = 1.0;
    @Override
    public double getArea() {
        return width * height;
    }
    @Override
    public String toString() {
        return "Rectangle(" + width + " X " + height + ")->" + getArea();
    }
    public void setWidth(double width) {
        this.width = width;
    }
    public void setHeight(double height) {
        this.height = height;
    }
}
```

```
GeometricShapeFactory.java
public class GeometricShapeFactory {
    public static GeometricShape getCircle() {
        return new Circle();
    }

    public static GeometricShape getRectangle() {
        return new Rectangle();
    }

    public GeometricShape getGeometricShape(Class<? extends GeometricShape> shapeClass) {
        GeometricShape object = null;

        try {
            object = (GeometricShape) shapeClass.getDeclaredConstructor().newInstance();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return object;
    }
}
```

LAZY INITIALIZATION : เอาไว้ที่หลัง เพื่อห้าม resource อย่างนั้นประยุกต์กิจกรรม e.g. กำหนดค่าให้เป็น null ก่อนเพื่อป้องกันมิจฉาชั่งสร้าง

SPRING BOOT = spring framework + HTTP server + Configuration

- = project ผู้ใช้ framework spring framework
- = เพิ่ม configuration (improve config) : รันต์ framework
- = HTTP: RESTful , Web App

FEATURE

- ไม่ต้อง deploy บน web server (Embed Tomcat)
- ฟีเจอร์ dependency
- Auto configure (e.g. MySQL ก็จะ default ชื่อ: configure ให้)
- Provide externalized configuration
- X code generate , X XML configuration

WHY USE SPRING BOOT !!

- ฟีเจอร์ dependency injection (1 ตัว obj ไม่ต้องว่ำべทั้งหมด)
- ฟีเจอร์ db transaction
- integration ที่สุด Java framework ที่มี (JPA, ORM etc.)
- ↓ cost ↓ time

SPRING BOOT : STARTER PROJECTS (ตัวอย่าง maven ของไฟล์ pom.xml)

- set vox dependency
- OPTION
 - web-service
 - web *
 - test
 - jdbc
 - data-rest
 - hateoas : เรียก url ผ่าน server
 - security . Authen *
 - data-jpa *
 - cache



CREATE SPRING BOOT

1. use spring Initializr (file zip)
2. use spring tool suite (STS)
3. use IDE Bundled tool

SPRING BOOT ARCHITECTURE

= เป็นแบบ layer Architecture (รับผิดชอบเฉพาะชั้นต่อไป + สามารถ communicate ระหว่าง layer ที่ต่อไป)

- LAYER
1. presentation layer : HTTP request , การแปลง JSON obj เป็น JAVA , Authen
 2. Business layer : service class
 3. Persistence layer : ดู ORM (storage logic)
 4. Database layer : crud (create, retrieve, update, delete)

- FLOW
1. คนที่มาผ่านทาง presentation = controller (รับ request)
 2. ฟีเจอร์ service ของธุรกิจ Business logic (ตัวรับ dependency injection (ห้อง repository ต่างๆ + ไลบรารีที่บันทึกต่อ))
 3. ดึงข้อมูลจาก DB Model (entity) ใช้ JPA / spring data ดึงข้อมูลจาก db

SPRING BOOT LAYER ARCH. VS MVC

- presentation = model+view+controller
- Service class = Business logic
- Repository class = Persistence

SPRING BOOT ANNOTATIONS

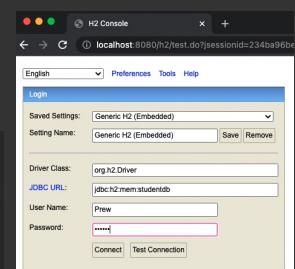
- @SpringBootApplication : รวม 3 ตัว @EnableConfiguration, @ComponentScan, @Configuration [Auto configure]
- Controller : @Required, @Autowired, @Configuration, @Bean
- Stereotype (class-level annotation) งานที่จะแบบ layer spring
 - > @Component: bean ปกติ (ห้อง)
 - > @Controller : งานแรกเขียน annotation method
 - > @Service
 - > @Repository

- @Requestmapping : รับทั้ง get + post
- @Getmapping : รับทั้ง get
- @Postmapping : รับ post

```
# ตั้งค่า default สำหรับ H2
spring.h2.console.path=/h2
spring.datasource.url=jdbc:h2:mem:studentdb
spring.datasource.password=abc123
spring.datasource.username=Prew
```

SPRING BOOT : APPLICATION PROJECT

- file ส่วนใหญ่ configue // กำหนดค่าที่จะเป็นตัว default ให้



SPRING WEB MVC

- = project នៃវា in spring framework
 - = flexible + loose coupled
 - = ឱសរីប្រាក់ Aspect នៃវាចំណែកអំពីគ្មាន
 - = ដំឡើង : dispatcherServlet // front controller
 - ↳ នៅក្នុង 1. ពី request ទៅផែនក្នុង
 - FLOW
 - in Handler Mapping រកវា និមួយន៍ controller នេះ
 - ដោលក្នុង controller នេះ
 - នៅក្នុង controller រក view
 - នៅក្នុង view រក viewResolver (ដែលរាយការណ៍ និង + ក. ឯកសារ)
 - ទូទៅ view

SPRING VIEW TECHNOLOGY

- Java Server Pages
 - Thymeleaf
 - Freemarker
 - Groovy Markup Template Engine

SPRING DATA JPA

- = access data in persistence + transfer data (CRUD, query methods)
 - = Boilerplate code : code ซ้ำๆ กัน很多 กัน But หาวาก็ปน. Roberts method พัฒนาให้ทำแล้วง (out-of-the-box : ใช้งานได้ทันที)

Spring RESTful API



↳ use w/ frontEnd : web service

WEB SERVICE (1)

WEB SERVICE (2) (սերվիսներ)

- = ทำให้ reuse Business logic ได้ (พร้อมเดินใช้งานรวด) + ต่างภาษาที่เรียนรู้ได้ + ไม่ต้องเขียนโค้ดซ้ำซ้อน
 - = e.g. App A (ต้องการข้อมูลลูกค้าเบื้องต้น): JAVA 
App B (ต้องการรายละเอียดลูกค้า): python

WEB SERVICE (3) เปลี่ยนจาก SOAP ໄກ REST API

- = ที่เปลี่ยน API:
 1. เริ่มต้น device เป็นที่ e.g. phone ที่ใช้ com ใจ : ลักษณะเป็น กี SOAP เมื่อมาเข้าร่วม
 2. REST ของรับการท้า วอน API ที่ให้สั่งตากัน frontEnd มากกว่า
 - = หากแยกเป็นสองส่วนจะเป็นแบบที่ 2 คือ gateway แล้ว

MICROSERVICE : បុគ្គលិក service (វិធាន៖ service រួមចំណែកដោយខ្លួន, reuse រាយ)

e.g., 1 App នີ້ service A ດ້ວຍນີ້. ຈົດການ db A // service B ອີ່ລວງເຕັມຂອງຈົດການ db B

RESTful API : REST API (REpresentational State Transfer)



= App programming interface (ເປັນມາດຕະຖານ)

= ຮູບຮັບການຮັບຮັດ resources ໂດຍອ່ານໆທ່ານ url

ຂໍ້ຕຳຫຼາດ 1. RESTful Client - Server : Client ອີ່ນອິນ້າ (request) server ອີ່ນອິນ້າ (reject ແລ້ວກົມໂອ=ໄຮັດ)

2. stateless : server ມີນ state (X stored context) : client ຕ້ອງຈົດການໃອ!

3. cacheable : resource ສູງໃນ cache ແລ້ວ, ນີ້ໄວ້ (server) : ຖ້າ client ໄລັງວາໃດໆຈະປະກິດກົດຈຳກັດໄວ້

4. Layered system : layer Architecture

5. Interface / uniform contract . ນີ້ໄວ້ໃຫ້ HTTP protocol ລ່າງໃນ ເຊັ່ນ resource ແລ້ວໃຫ້ການໃຊ້ກົດຈຳກັດຈຳກັດໄວ້ (Method)

6. Code on demand (optional) : server ສູງ code ມີໃນ client run

RULE 1. ເລົາອ່າງຟື້ນ resource ໄປນີ້ນີ້ noun (ແນະນຳປິ່ນພາຫຼວງນໍ e.g. API / users)

2. verb ເປັນ Action (GET, PUT, POST, DELETE)

3. ໃນດຽວໃຈ Method ສະບັບນັກ ເພື່ອໃຫ້ຄົວໜົວກັນ

- POST : ສ້າງ resource ໃນ server
- GET : ອ່ານ resource ໃນ server
- PUT : update resource
- DELETE : delete resource

BUILDING A SPRING BOOT REST API



1. initializing spring boot project

2. connect db (user, driver, username, password, url)

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=abc123
spring.datasource.url=jdbc:mysql://localhost:3306/classicmodels

# ໃຫ້ລັບລົບໃຫ້ມີການນູ້ນັ້ນໃຫຍ້ເວັບໄວ້ເປັນເປົ້າ
spring.jpa.hibernate.ddl-auto:none

# ໃຫ້ລັບລົບໃຫຍ້ເວັບໄວ້ເປັນເປົ້າ
# 1. ດີເລີນ @Column(name) ໃຫຍ້ເວັບໄວ້ເປົ້າ
# 2. ດີເລີນ @Table
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```

3. create models (soureentity)

Employee.java

```
// JsonIgnore ຈະທ່າງການເລີນຕົວທີ່ເພື່ອໄຟໄຟໃຫ້ໄຟຟ້ວດ JSON ມີນາດທີ່ໄຫຼຸມຈາກເກີນໄປ ເພື່ອມີນະຫາຍ້ອມລັບສ້າງອຳນວຍໄປນາ
// ອ່ານວ່າອັນນີ້ມີນັກຈີ່ມີຢູ່ໂປກດີຂອງ Employee ແຕ່ມີເອົາຕາວເຊື່ອ reportsTo ມາ
@JsonIgnore
@ManyToOne
@JoinColumn(name = "reportsTo")
private Employee employees;
```

4. create repo class (ເພີ້ມໃຫ້CRUD)

```
public interface OfficeRepository extends JpaRepository<Office, String> { }
```

5. create rest controller (ສະບັບ @RestController + ກຳນົດ resource @RequestMapping (" /api / ຫຼື "))

OfficeController.java

```
@RestController // ໃຫ້ anno(@) ກໍາທັນດວກວ່າອັນນີ້ເປັນ Rest Controller
@RequestMapping(" /api / offices ") // ເລົາເຮັດໃຫ້ຈຳກັດຜ່ານ /api/offices
public class OfficeController {
    @Autowired
    private OfficeRepository repository;
```

6. compile+Build+run

7. test APIs

GET localhost:8080/api/offices

Params	Authorization	Headers (6)	Body	Pre-request Script	Tests	Settings
Body	Cookies	Headers (5)	Test Results			
Pretty	Raw	Preview	Visualize	JSON		

```
1 [
2   [
3     {
4       "id": "1",
5       "city": "San Francisco",
6       "phone": "+1 650 219 4782",
7       "addressLine1": "100 Market Street",
8       "addressLine2": "Suite 300",
9       "state": "CA",
10      "country": "USA",
11      "postalCode": "94080",
12      "territory": "NA"
13   ]
14 ]
```

GET localhost:8080/api/offices/1

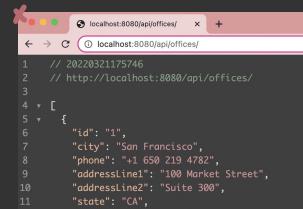
Params	Authorization	Headers (6)	Body	Pre-request Script	Tests	Settings
Body	Cookies	Headers (5)	Test Results			
Pretty	Raw	Preview	Visualize	JSON		

```
1 [
2   {
3     "id": "1",
4     "city": "Glasgow",
5     "phone": "+44 20 7877 2041",
6     "addressLine1": "25 Old Broad Street",
7     "addressLine2": "Level 8",
8     "state": null,
9     "country": "UK",
10    "postalCode": "EC2N 1HN",
11    "territory": "EMEA"
12  ]
13 ]
```

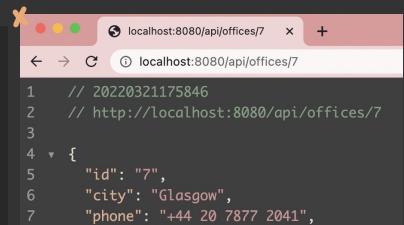
E.G. SPRING BOOT + REST API

(ព័ត៌មានបញ្ជីការងារ + កើតឡើងកិច្ចការនៃការទូទាត់ port 8000 ។ នៅថ្ងៃទី០១)

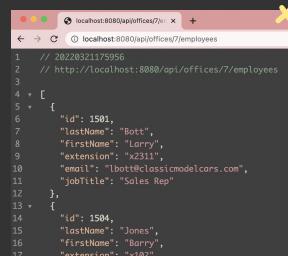
```
// ខ្សោយការណ៍ដែលបានរាយការណ៍ជាការងារ
@GetMapping("")
public List<Office> getOffices() {
    return repository.findAll(); // ប្រើប្រាស់repo ដើម្បីទូទាត់findAll មានការស្វែងរកទាំងអស់
}
```



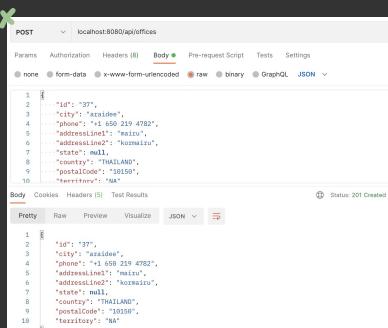
```
// ខ្សោយការណ៍ដែលបានរាយការណ៍ជាការងារ
@GetMapping("/{officeCode}")
// @PathVariable String officeCode មេនុយត្រូវបានស្វែងរកទាំងអស់
// ការកែតិចិត្តថាអាម៉ូនផ្លូវការណ៍នេះមានការរាយការណ៍ជាការងារ
public Office getOffice(@PathVariable String officeCode) {
    return repository.findById(officeCode).orElseThrow(
        () -> new RuntimeException(officeCode + " does not exist !!!"));
}
```



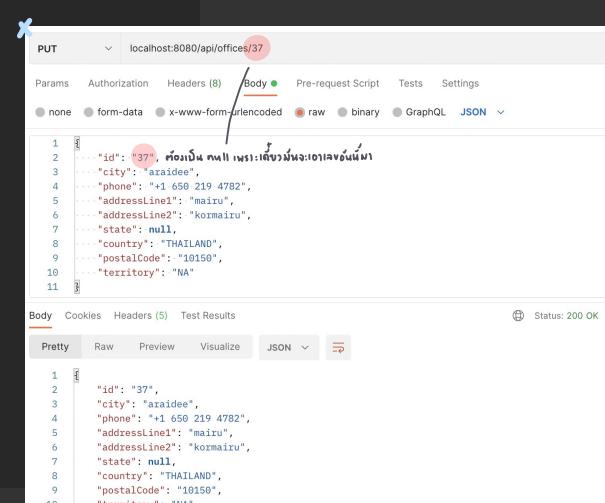
```
// ខ្សោយការណ៍ដែលបានរាយការណ៍ជាការងារ
@GetMapping("/{officeCode}/employees")
public Set<Employee> getOfficeEmployees(@PathVariable String officeCode) {
    Office office = repository.findById(officeCode).orElseThrow(
        () -> new ResponseStatusException(HttpStatus.NOT_FOUND, officeCode + " does not exist !!!"));
    return office.getEmployees();
}
```



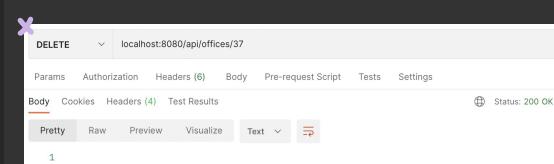
```
// ខ្សោយការណ៍ដែលបានរាយការណ៍ជាការងារ
@PostMapping("")
@ResponseStatus(HttpStatus.CREATED) // 201 success
public Office create(@RequestBody Office newOffice) { // តើនេះ JSON ដើម្បី add នៅក្នុង
    return repository.saveAndFlush(newOffice); // បានកើតឡើងការងារនៃការ add នៅក្នុងហើយ
}
```



```
get និងលើកស្រួល set ( នាមពីរការងារ កំណត់រាយការណ៍ដែលត្រូវបានរាយការណ៍ជាការងារ )  
// ខ្សោយការណ៍ដែលបានរាយការណ៍ជាការងារ
@GetMapping("/{officeCode}")
// តាមរាយការណ៍នេះក្នុងការងារនៃការ update នឹងត្រូវបានរាយការណ៍ជាការងារ
public Office update(@RequestBody Office updateOffice, @PathVariable String officeCode) {
    Office office = repository.findById(officeCode).map(o->mapOffice(o, updateOffice)).orElseGet(() ->
    {
        updateOffice.setId(officeCode);
        return updateOffice;
    });
    return repository.saveAndFlush(office);
}  
  
// តាម mapOffice នៃការងារនេះ នឹងត្រូវបានរាយការណ៍ជាការងារ
private Office mapOffice(Office existingOffice, Office updateOffice) {
    existingOffice.setAddressLine1(updateOffice.getAddressLine1());
    existingOffice.setAddressLine2(updateOffice.getAddressLine2());
    existingOffice.setCity(updateOffice.getCity());
    existingOffice.setCountry(updateOffice.getCountry());
    existingOffice.setPhone(updateOffice.getPhone());
    existingOffice.setPostalCode(updateOffice.getPostalCode());
    existingOffice.setState(updateOffice.getState());
    existingOffice.setTerritory(updateOffice.getTerritory());
    return existingOffice;
}
```



```
// ខ្សោយការណ៍ដែលបានរាយការណ៍ជាការងារ
@DeleteMapping("/{officeCode}")
public void delete(@PathVariable String officeCode) {
    repository.findById(officeCode).orElseThrow(() ->
        new RuntimeException(officeCode + " does not exist !!!"));
    repository.deleteById(officeCode);
}
```



SPRING DATA JPA (Pagination + Sorting)

```
@RestController
@RequestMapping("/api/customers")
public class CustomerController {
    private final CustomerRepository repository;

    @Autowired // นี่ injection ผ่าน injection ที่ constructor แนว
    public CustomerController(CustomerRepository repository) { this.repository = repository; }
```

SPRING DATA SORT + ORDER (เรียง sort แบบต่อๆ กัน)

L sort แบบ single, Multiple 1 ต่อ 1 กำหนดพิจารณา sort (asc, desc) > มาจาก findAll (default เป็น sort - มาก asc)

```
// Sort + Order object example
@GetMapping("testSortAndOrder")
public List<Customer> getCustomer() {
    return repository.findAll(Sort.by("customerName").descending().and(Sort.by("creditLimit")));
}

// จากด้านบน เราสามารถสร้าง List ของ order มาช่วยทำได้ด้วย
@GetMapping("/testListOrder")
public List<Customer> getCustomerOne() {
    List<Order> orders = new ArrayList<>();
    Order customer1 = new Order(Sort.Direction.DESC, property: "customerName");
    orders.add(customer1);
    Order customer2 = new Order(Sort.Direction.ASC, property: "creditLimit");
    orders.add(customer2);
    return repository.findAll(Sort.of(orders));
}
```

- แบบอีก way order มากขึ้น

JPA REPOSITORY WITH PAGINATION (pagination)

- = findAll (Pageable pageable) : แบ่งเป็น page หนึ่ง pageSize ต่อหนึ่ง : return เป็น page
- = PageRequest ใช้สร้าง pageable 1 ต่อ 1

PAGE + SORT

```
@GetMapping("") // ส่งแบบเป็น page จะได้เมื่อต้องไปหน้าเพราJson อาจจะเอื้อไป
public Page<Customer> getAllCustomers (
    // ค่า default จะใช้ก็ถ้าไม่มีค่าส่งมา ก็จะใช้ค่า default [?ด้วย] ส่งแบบ http protocol
    @RequestParam(defaultValue = "id") String sortBy,
    @RequestParam(defaultValue = "0") Integer page,
    @RequestParam(defaultValue = "10") Integer pageSize ) {
    Sort sort = Sort.by(sortBy);
    // กำหนด page และตัวพารามที่ต้อง sort ให้ หากไม่กำหนดพิจารณา sort ตาม id
    return repository.findAll(PageRequest.of(page, pageSize, sort));
}
```

```
@GetMapping("")
// เมื่อเข้า Page<Customer> เน้น List<Customer> แล้วห้ามเพิ่ม .getContent()
// นี่จะเข้าเพื่อขอของ customer ที่เราต้องการมาและ ที่ไม่แสดงข้อมูลของ page ในตอนท้ายแล้ว
public List<Customer> getAllCustomertwo (
    // ค่า default จะใช้ก็ถ้าไม่มีค่าส่งมา ก็จะใช้ค่า default [?ด้วย] ส่งแบบ http protocol
    @RequestParam(defaultValue = "id") String sortBy,
    @RequestParam(defaultValue = "0") Integer page,
    @RequestParam(defaultValue = "10") Integer pageSize ) {
    Sort sort = Sort.by(sortBy);
    // กำหนด page และตัวพารามที่ต้อง sort ให้ หากไม่กำหนดพิจารณา sort ตาม id
    return repository.findAll(PageRequest.of(page, pageSize, sort)).getContent();
}
```

CREATE QUERY METHOD

- = spring 9: แบบเป็น query ให้ตั้งหนังสือ



KEYWORD

Keyword	Sample	JQL snippet
Distinct	findByDistinctLastnameAndFirstname	... where x.lastname = ?1 ... where x.firstname = ?2
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Is, Equals	findByFirstname, findByLastname	... where x.lastname = ?1 ... where x.firstname = ?1
Between	findByStartdateBetween	... where x.startdate >= ?1 and x.startdate <= ?2
LessThan	findByAgeLessThan	... where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1

Keyword	Sample	JQL snippet
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
GreaterThanOrEqualTo	findByAgeGreaterThanOrEqualTo	... where x.age >= ?1
After	findByStartdateAfter	... where x.startdate > ?1
Before	findByStartdateBefore	... where x.startdate < ?1
IsNull, Null	findByAgeIsNull	... where x.age is null
NotIsnull, NotNull	findByAgeNotIsNotNull	... where x.age is not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1

Keyword	Sample	JQL snippet
StartingWith	findByFirstnameStartingWith	... where x.lastname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	... where x.lastname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	... where x.lastname like ?1 (parameter bound with appended %)
Orderby	findByAgeOrderbyLastnameDesc	... where x.age > ?1 order by x.lastname desc
NotIn	findByAgeNotIn(Collection<Age> ages)	... where x.age not in ?1
True	findByActiveTrue()	... where x.active = true
False	findByActiveFalse()	... where x.active = false

```
public interface CustomerRepository extends JpaRepository<Customer, Integer> {
    // ให้รูปแบบ 2 ต่อ upper และ lower
    List<Customer> findAllByCreditLimitBetween(BigDecimal lower, BigDecimal upper);
    // List<Customer> findAllByCreditLimitGreaterThanOrCityStartsWith(BigDecimal limit, String city);
    List<Customer> findAllByCustomerNameBetween(String lower, String upper);

    @GetMapping("credit/{lower}/{upper}")
    public List<Customer> getCustomerByCredit (
        @PathVariable BigDecimal lower,
        @PathVariable BigDecimal upper ) {
        return repository.findAllByCreditlimitBetween(lower, upper);
        // return repository.findAllByCreditlimitBetween(new BigDecimal(20000), new BigDecimal(30000));
    }

    @GetMapping("test/{lower}/{upper}")
    public List<Customer> getCustomer(
        @PathVariable String lower,
        @PathVariable String upper ) {
        return repository.findAllByCustomerNameBetween(lower, upper);
        // return repository.findAllByCustomerNameBetween(new String("b"), new String("c"));
    }
}
```

การใช้ Between

Integer : between 100, 200
L >= 100 and <= 200

String : between b, c
L >= b and <= c

กำหนดค่าในบัญชีว่าเราเรียกง่ายๆ

book	{	between = book b,c * lws: cat,custom, custom center > c c
cat		
center		

กิ between b,c = book, cat

* lws: ca ห้องกว่า cb

JPA NAMED QUERIES

- = ใจสัตต์และไม่ต้องรู้ SQL @Query (value = "sql", nativeQuery = true)

```
public interface UserRepository extends JpaRepository<User, Long> {
    @Query(value = "SELECT * FROM USERS WHERE LASTNAME = ?1",
    countQuery = "SELECT count(*) FROM USERS WHERE LASTNAME = ?1", nativeQuery = true)
    Page<User> findByLastname(String lastname, Pageable pageable);
```

↳ 用於跑入 code SQL

SPRING RESTFUL API DTO

DATA TRANSFER OBJECT (DTO)

(ຄົວເລກ = field ທີ່ຈະເຈັດຈາກ Entity)

= DTO ຂະໜາວຈາກ Entity ເພີ້ມໃຫ້ສາມາກສ່ວນຂອງລາຍລັບການແປນ e.g. Entity Customer → mapper → customer type1 → customer type2
L ສ່ວນແບບ POJO (getter setter)

= DTO ນັບອາການ, Entity ອັການນັກນຳ data ໄທ FE ທີ່ເລີ່ມ e.g. Entity Customer → Entity Employee → Entity Payment → Mapper → CustomerInfo (DTO)

= ບົດ : ຂໍ້ມູນປະໂຫຍດ (ບາງຂຶ້ນຈະເຮັດັກ json ignore) DTO ຂໍ້ມູນໂຄງການ

: Help ລັດຖະບານການເຮັດກຳຂ່າງຈຸດ

SERVICE LAYER

```
@Service // ດຳເນີນໄວ້ເປົ້າໃນ service ຊື່ເປັນພື້ນຕົວອັນ controller + ຕິດ repo ມາໃຊ້ຈານ
public class CustomerService {
    @Autowired
    private CustomerRepository repository;

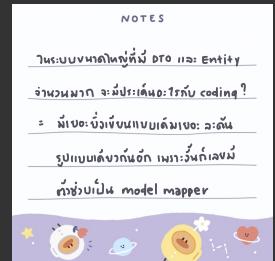
    // ລັດຖະບານ id ວ່າເຮົາອາກໃຫ້ຕັ້ນໃຫ້ເຈັດຈາກ
    public SimpleCustomerDTO getSimpleCustomerId(Integer id) {
        return repository.findById(id).map(customer -> convertEntityToDto(customer))
            .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, id+"ເບີ້ມີ"));
    }

    // ລັດຖະບານໃຫ້ຕັ້ນພື້ນຕົວອັນ
    private SimpleCustomerDTO convertEntityToDto(Customer customer) {
        SimpleCustomerDTO simpleCustomerDTO = new SimpleCustomerDTO();
        simpleCustomerDTO.setCustomerName(customer.getCustomerName());
        simpleCustomerDTO.setPhone(customer.getPhone());
        simpleCustomerDTO.setCity(customer.getCity());
        simpleCustomerDTO.setCountry(customer.getCountry());
        simpleCustomerDTO.setSalesPerson(customer.getSalesRepEmployee().getFirstName()
            + " " + customer.getSalesRepEmployee().getLastName());
        return simpleCustomerDTO;
    }
}
```

```
SimpleCustomerDTO.java
public class SimpleCustomerDTO {
    private String customerName;
    private String phone;
    private String city;
    private String country;
    private String salesPerson;
}

@RestController
@RequestMapping("/api/customers")
public class CustomerController {
    @Autowired // ເຊື້ອີ້ນໃຫ້ service
    private CustomerService customerService;

    @GetMapping("/{id}")
    public SimpleCustomerDTO getCustomerById(@PathVariable Integer id) {
        return customerService.getSimpleCustomerId(id);
    }
}
```



MODEL MAPPER LIBRARY

- ເນັດ້ວັນເບັນ boilerplate + ທ່າ vice-versa 1:1 (ແປວຈາກ Entity - DTO ແລະ DTO - Entity)
- automatically + default behavior (ນັກພ້າງກົດຕົກໃນແບບອອນນາວ)

```
ApplicationConfig.java
@Configuration
public class ApplicationConfig {
    @Bean
    public ModelMapper modelMapper() {
        return new ModelMapper();
    }
}

CustomerService.java
@Authorized
private ModelMapper modelMapper;
// ມີມີກຳລັງ ModelMapper ນັ້ນ
public SimpleCustomerDTO getSimpleCustomerId(Integer id) {
    Customer customer = repository.findById(id)
        .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, id+"ເບີ້ມີ"));
    // ມີມີກຳລັງ entity customer ໄກສອນນີ້ DTO ດີກຕ່າງແມ່ນໄດ້
    return modelMapper.map(customer, SimpleCustomerDTO.class);
}
```

```
localhost:8080/api/customers14
1 // 20220323002338
2 // http://localhost:8080/api/customers/124
3
4 +
5 "customerName": "Mini Gifts Distributors Ltd.",
6 "phone": "4155551450",
7 "city": "San Rafael",
8 "country": "USA",
9 "salesPerson": "Leslie Jennings"
10 }
```

DEEP MAPPINGS

- ຄານສ່ວນຮັບກົດ class ອື່ນ e.g. customer ບໍ່ saleRepEmployee ທີ່ເຂື້ອນກົດ Employee

1. ທ່າມແບບຮັບ obj ຈັດ
2. ທ່າມແບບຈຳ field 1:1 (ທ່າມຕົວຮັບ obj) ກ່າວຈຳກັບໂອ
3. ສ່ວນ field ໃຫ້ຈຳການເວັບຂອງລາຍງານຮຽກກຳລັດ

```
SimpleEmployeeDTO.java
public class SimpleEmployeeDTO {
    private String lastName;
    private String firstName;

    // ແບບທີ 3 ດອຍເນີ້ນເພີ້ມອັນຫຼວຍ
    public String getName() {
        return firstName + " " + lastName;
    }
}
```

```
SimpleCustomerDTO.java
public class SimpleCustomerDTO {
    private String customerName;
    private String phone;
    private String city;
    private String country;
    // ມີມີ 1
    private SimpleEmployeeDTO salesRepEmployee;

    // ມີມີ 2
    private String salesRepEmployeeFirstName;
    private String salesRepEmployeeLastName;

    // ມີມີ 3
    @JsonIgnore // ເພີ້ມໃຫ້ຈຳການເວັບຂອງລາຍງານຮຽກກຳລັດ ເພີ້ມໃຫ້ຈຳການເວັບຂອງລາຍງານຮຽກກຳລັດ
    private SimpleEmployeeDTO salesRepEmployee;
    public String getSalesPerson() { // ສືບຕົວເພີ້ມໃຫ້ຈຳການ get SalesPerson
        return salesRepEmployee==null ? "-" : salesRepEmployee.getName();
    }
}
```

```
localhost:8080/api/customers124
1 // 20220323014202
2 // http://localhost:8080/api/customers/114
3
4 +
5 "customerName": "Australian Collectors, Co.",
6 "phone": "03 9520 4555",
7 "city": "Melbourne",
8 "country": "Australia",
9 "salesPerson": null
10 }
```

```
localhost:8080/api/customers124
1 // 20220323015451
2 // http://localhost:8080/api/customers/124
3
4 +
5 "customerName": "Mini Gifts Distributors Ltd.",
6 "phone": "4155551450",
7 "city": "San Rafael",
8 "country": "USA",
9 "salesRepEmployeeFirst": "Leslie", IIUU 1
10 "salesRepEmployeeLast": "Jennings"
11 }
```

```
localhost:8080/api/customers124
1 // 20220323015456
2 // http://localhost:8080/api/customers/124
3
4 +
5 "customerName": "Mini Gifts Distributors Ltd.",
6 "phone": "4155551450",
7 "city": "San Rafael",
8 "country": "USA",
9 "salesRepEmployeeFirst": "Leslie", IIUU 2
10 "salesRepEmployeeLast": "Jennings"
11 }
```

```
localhost:8080/api/customers124
1 // 20220323021807
2 // http://localhost:8080/api/customers/124
3
4 +
5 "customerName": "Mini Gifts Distributors Ltd.",
6 "phone": "4155551450",
7 "city": "San Rafael",
8 "country": "USA", IIUU 3
9 "salesPerson": "Leslie Jennings"
10 }
```

MAPPING LISTS WITH MODEL MAPPER + DTO - ENTITY

= ແປວຂຶ້ນຈຳກັບ list ໄກສອນໂປົວຕົວໄດ້ໃນຮູບແບບນົບສອນ stream ແລ້ວຄົວນາ return ເປັນ list ຈັດ

ຕາມກຳ GENERAL-PURPOSE PARAMETERIZED METHOD ຈະໄດ້ນັກໂອງເປັນໄດ້ຕ້ອງກ່າວ:

SINGLETON LISTMAPPER : constructor ອື່ນ private ແລະ ນຳໃຫຍ່ method Instance

```
ApplicationConfig.java
@Bean
public ListMapper listMapper() { return ListMapper.getInstance(); }

ListMapper.java
public class ListMapper {
    private static final ListMapper listMapper = new ListMapper();
    private ListMapper() {} // private constructor
    public <T> T mapList(List<?> source, Class<T> targetClass, ModelMapper modelMapper) {
        return source.stream().map(item -> modelMapper.map(item, targetClass)).collect(Collectors.toList());
    }

    public static ListMapper getInstance() {
        return listMapper;
    }
}
```

```
EmployeeService.java
@Service
public class EmployeeService {
    @Autowired
    private EmployeeRepository repository;
    @Autowired
    private ModelMapper modelMapper;
    @Autowired
    private ListMapper listMapper;

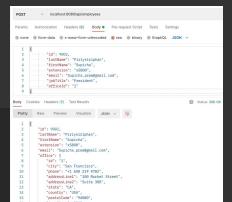
    public Employee save(SimpleEmployeeDTO employee) {
        Employee employeeFromDB = repository.findById(employee.getId());
        employeeFromDB.setName(employee.getName());
        employeeFromDB.setJobTitle(employee.getJobTitle());
        employeeFromDB.setOfficeId(employee.getOfficeId());
        return repository.saveAndFlush(employeeFromDB);
    }

    public List<SimpleEmployeeDTO> getAllEmployees() {
        return listMapper.mapList(repository.findAll(), SimpleEmployeeDTO.class);
    }
}
```

```
EmployeeController.java
@RestController
@RequestMapping(value = "/api/employees")
public class EmployeeController {
    @Autowired
    private EmployeeService employeeService;

    @GetMapping(value = "/")
    public List<SimpleEmployeeDTO> getAllEmployees() {
        return employeeService.getAllEmployees();
    }

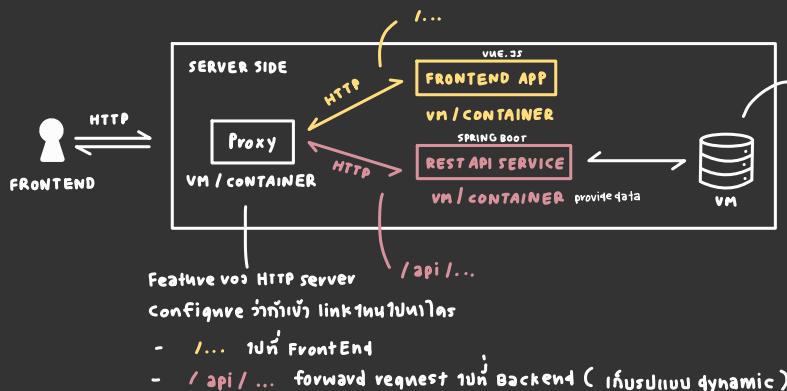
    @PostMapping(value = "/")
    // ແກ້ໄຂຈຳກັບ list ຢັງໃຫ້ຈຳກັບ
    public Employee create(@RequestBody SimpleEmployeeDTO newEmployee) {
        return employeeService.save(newEmployee);
    }
}
```



SPRING RESTFUL API FILE SERVICES

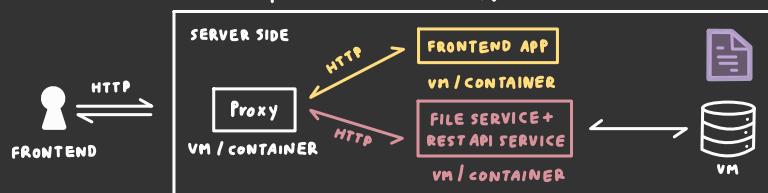


SPA RUNNING ENVIRONMENT

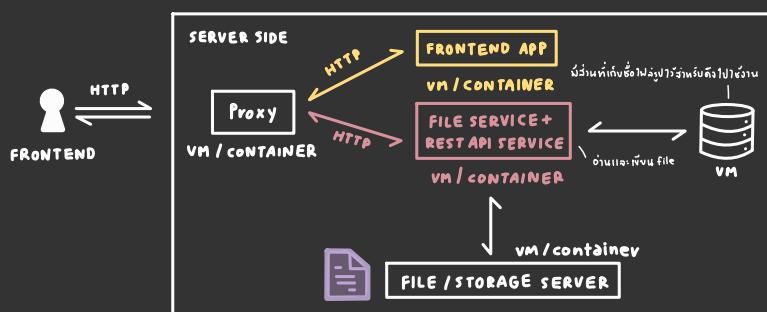


FILE STORAGE ARCHITECTURES

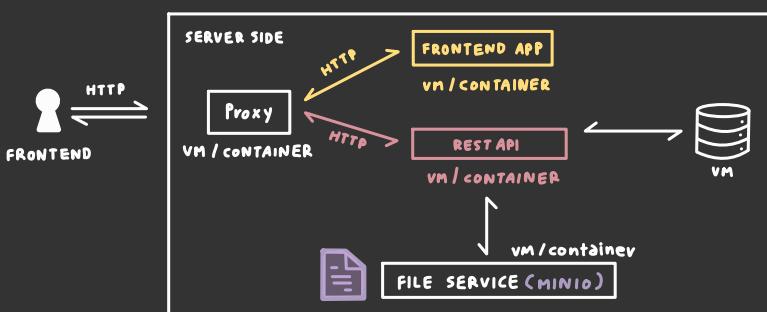
- เก็บ file ให้ดู แต่พื้นที่บุ่งขนาด เฟรด: ให้สามารถถูรูปมาด้วยรูปเป็น Binary + ปรับสัดส่วนตามต้อง



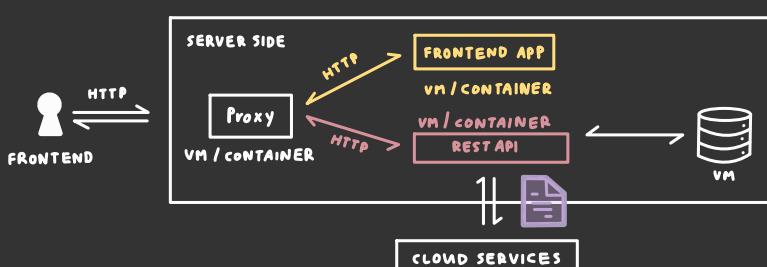
- เก็บรูปใน BackEnd (ที่ folder นั้นใน BackEnd) ผู้ดูแลในการเขียนรูปลงใน (อยู่ใน BackEnd) *



- แบบสำเร็จรูป (MINIO) ช่วยในการจัดการ บนจะหาซึ่งปะ: ส่งกลับมาให้เรา



- จะต้องเป็นที่ cloud service ต้องเขียน API กัน



FILE OPERATION

- upload : client ส่งไฟล์แบบ multipart (FormData) // Server ต้องข้อมูลไฟล์จาก request และ save ไฟล์
- download : รับไฟล์ binary ของ file ไปดูแล ก็ต้องดำเนินการลงใน storage และ ส่งกลับไป
- delete

FILE STORAGE IIUH 2

1. กันต์ multipart property

```
# Multipart (MultipartProperties)
# ค่าที่ multipart upload
spring.servlet.multipart.enabled=true

# ค่า buffer จำกัดขนาดของไฟล์ที่ไม่สามารถอ่านได้ (Threshold)
spring.servlet.multipart.file-size-threshold=4KB

# Max file size
spring.servlet.multipart.max-file-size=10MB

# Max Request size
spring.servlet.multipart.max-request-size=80MB

# Path ที่จะเก็บไฟล์
file.upload-dir=/Users/spreewii/Desktop/public/classicmodels/uploads
```

2. ห้าม property ที่อยู่ในรูปแบบ Pojo class

```
@Getter
@Setter
// บันทึกว่าเราจะเอาไฟล์ขึ้นด้วย configure properties ที่ชื่นตั้งค่ายก้าว file เขียนตามชื่อนี้
@ConfigurationProperties(prefix = "file")
public class FileStorageProperties {
    private String uploadDir;
}
```

```
// ลงทะเบียนให้ spring รู้ว่ามี class นี้เป็น property class
@EnableConfigurationProperties({
    FileStorageProperties.class
})
```

3. ห้าม service

```
// รับฟังก์ชันนี้ - เรื่องไฟล์
@Service
public class FileService {
    private final Path fileStorageLocation;

    @Autowired // เรียกไฟล์ folder ที่เรา
    public FileService(FileStorageProperties fileStorageProperties) {
        this.fileStorageLocation = Paths.get(fileStorageProperties.getUploadDir()).toAbsolutePath().normalize();
        try {
            Files.createDirectories(this.fileStorageLocation);
        } catch (Exception ex) {
            throw new RuntimeException("Failed to create directory "+ex);
        }
    }

    // method store สำหรับบันทึกไฟล์ ที่เรียกใช้เมื่อเรา upload ไฟล์
    public String store(MultipartFile file) {
        // ตรวจสอบไฟล์ none
        String fileName = StringUtils.cleanPath(file.getOriginalFilename());
        try {
            // เช็คไฟล์ name ด้วย invalid character ด้วย
            if (fileName.contains("..")) {
                throw new RuntimeExeption("File name contains invalid character: " + fileName);
            }
            // สร้าง path ที่จะบันทึกไฟล์ (ถ้ามี location ที่เดียวกัน ก็จะใช้)
            Path targetLocation = this.fileStorageLocation.resolve(fileName);
            Files.copy(file.getInputStream(), targetLocation, StandardCopyOption.REPLACE_EXISTING);
            return fileName;
        } catch (IOException ex) {
            throw new RuntimeException("Failed to store file " + fileName + " " + ex);
        }
    }
}
```

```
// method loadFileAsResource สำหรับโหลดไฟล์
public Resource loadFileAsResource(String fileName) {
    Path filePath = this.fileStorageLocation.resolve(fileName).normalize();
    Resource resource = new UrlResource(filePath.toUri());
    if (resource.exists()) {
        return resource;
    } else {
        throw new RuntimeException("File " + fileName + " does not exist");
    }
} catch (MalformedURLException ex) {
    throw new RuntimeException("File " + fileName + " " + ex);
}
```

4. ห้าม controller

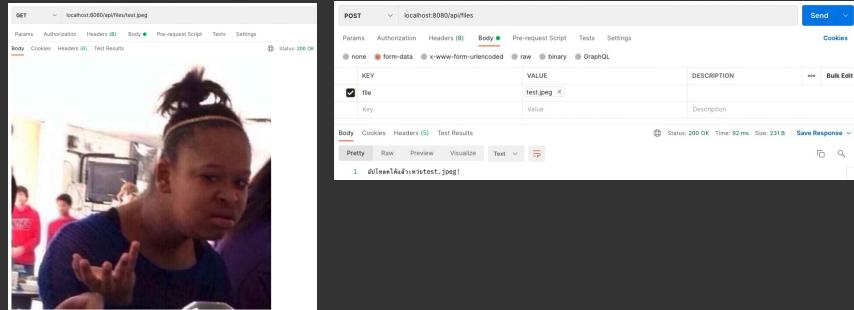
```
@RestController
@RequestMapping(value = "/api/files")
public class FileController {
    private final FileService fileService;

    @Autowired
    public FileController(FileService fileService) {
        this.fileService = fileService;
    }

    @GetMapping("/{fileName:.+}")
    @ResponseBody
    public ResponseEntity<Resource> serveFile(@PathVariable String fileName) {
        Resource file = fileService.loadFileAsResource(fileName);
        return ResponseEntity.ok().contentType(MediaType.IMAGE_JPEG).body(file);
    }

    @PostMapping("")
    public String fileUpload(@RequestParam("file") MultipartFile file) {
        fileService.store(file);
        return "File uploaded successfully " + file.getOriginalFilename() + "!";
    }
}
```

5. test

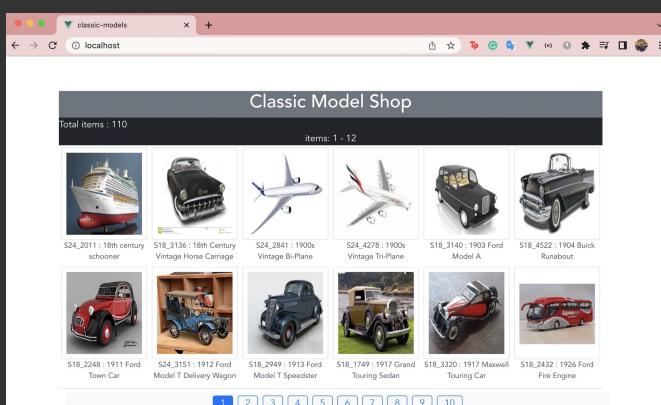


การเอาไฟล์ FRONT+BACK BY AJ 'PICHET'

"Back น์ entity, DTO, Repo, service,
controller, property,
file configure
L ร่อนๆ path folder guy"

"FRONT น์ vue project
L ร่อนๆ"

```
sudo npm install -g @vue/cli
sudo vue create <> projectName>> -g
cd <> projectName<>
sudo npm run serve
```



L + sudo i bootstrap jquery popper.js @popperjs/core axios

L ห่างๆ กัน request API ช่วงบ่ดการการเรียก API