INT204 THEORY

Week 1

JSON : Java Srcipt Object Notation => สิ่งที่ใช้ในการแลกเปลี่ยนข้อมูลกัน / front ส่ง Json Back Back ส่ง Json หา front

- รูปแบบ Object ที่ java script ต้องใช้ (Backend ต้องส่ง JSON ไปให้ frontend)
- A lightweight data-interchange format => เป็น format ที่เหมาะแก่การแลกเปลี่ยนกับอุปกรณ์สมัยใหม่
- Easy for humans to read and write => สามารถอ่านเขียน JSON ด้วยคนได้
- Easy formachines to parse and generate => ง่ายแก่การวิเคราะห์และ generate
- ใช้ใน Javasript เมื่อ December 1999. Json text format ที่ Programmer เข้าใจง่าย

RESTful Resource => Backend ของการเข้าถึงข้อมูลเป็น resource ex. file pdf, file pic

- ประกอบด้วย URI HTTP Verb => 2 ตัวนี้รวมกัน เรียกว่า Endpoint / ทำเป็น CRUD
  - combine กัน ต้องห้ามซ้ำ

ex

URI	HTTP Verb	Description
api/offices	GET	Get all office
api/offices/1	GET	Get an office with id = 1
api/offices/1/employees	GET	Get all employee for office id = 1
api/offices	POST	Add new office
api/offices/1	PUT	Update an office with id = 1
api/offices/1	DELETE	Delete an office with id = 1

(ไม่ซ้ำ เพราะ method คนละแบบ)

GET => เป็นการดึงข้อมูล => ทั้ง All และตาม ID

POST => เป็นการเพิ่มข้อมูล

PUT => updat ข้อมูล

DELETE => delete ข้อมูล

REST API => REpresentational State Transfer

- API เพื่อให้ front มาเรียกไปใช้ Computer / มือถือ
- มี constraints ของมันอยู่ว่าต้องทำแบบไหน มีข้อบังคับบางอย่างอยู่
- เป็น architecture ที่อนุญาตให้เรา provides service ให้ฝั่ง client เข้ามา access and modifies resources ได้
- แต่ละ resource จะ: Identified ด้วย URI /global IDs => รับ id บางอย่างได้ ID จะส่งผ่าน path ได้เลย
- Restful จะ: represent (แทน) data แบบไหนก็ได้ => Text, JSON, XML แต่ส่ง JSON เป็นหลัก

Rules of REST API

- กฎที่ควรคำนึงในตอนสร้าง REST API endpoints
- endpoint ที่เหมาะ resource ต่างๆ ต้องเขียนด้วย noun (คำนาม)

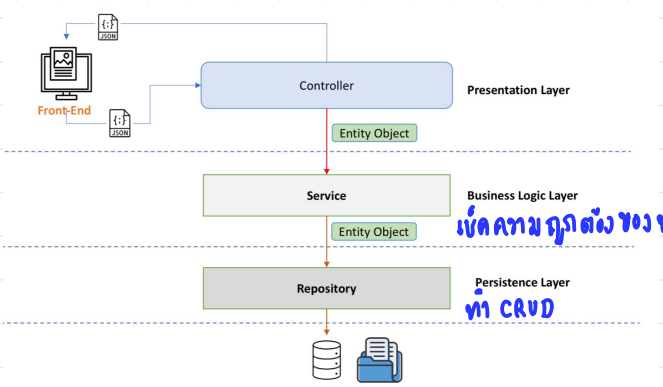
ex

URI	HTTP verb	Description
api/users	GET	Get all users
api/users/new	GET	Show form for adding new user
api/users	POST	Add a user
api/users/1	PUT	Update a user with id = 1
api/users/1/edit	GET	Show edit form for user with id = 1
api/users/1	DELETE	Delete a user with id = 1
api/users/1	GET	Get a user with id = 1

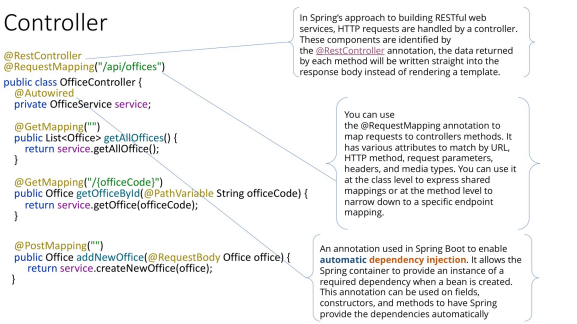
RESTful Principles and Constraints (หลักการ และ ข้อจำกัด)

- Restful Client-Server => มีครบ 2 ฝั่ง Client, Server => api service
- ต้องเป็น Stateless ฝั่ง backent จะไม่ยุ่งกับ state ของ app => มันใช้แค่เรื่องจัดการ resource Client เป็นคนจัดการ state, vue

- **Interface / Uniform Contract**
  - ทำให้ข้อตกลงระหว่าง client / server
  - POST - create resource
  - GET - retrieve resource
  - PUT - change state resource
  - DELETE - remove or delete resource
- **Code on demand** (Optional) ทำก็ได้ไม่ทำก็ได้
  - ฝั่ง backend สามารถมี endpoint ที่จะ generate JS ไปรันฝั่ง client ได้ ถ้าตกลงกัน
- **Cacheable** => back สามารถทำ cache
  - data เยอะๆ จาก db data ไม่ค่อยเปลี่ยน เหมาะทำ cache ไว้ เพื่อให้มัน cacheing บน server ได้เลย
- **Layered system** * REST ต้องแบ่งเป็น Layer ๆ
  - ห้ามถูกข้าม Layer



Controller ไว้คุย กับ front



ชื่อ TEST API

/

POSTMAN => API Platform ที่ใช้ในการ test API ช่วยให้ทำแต่ละขั้นตอนง่ายขึ้น เร็วขึ้น

Creating a Spring Boot REST API Project

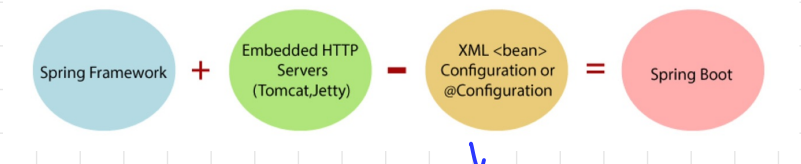
- Step 1: Initializing a Spring Boot Project
- Step 2: Connecting Spring Boot to the Database
- Step 3: Creating a User Model
- Step 4: Creating Repository Interface (Persistence Layer)
- Step 5: Creating Service Classes (Business Layer)
- **Step 6: Creating a Rest Controller** (Presentation Layer)
- Step 7: Compile, Build and Run
- **Step 8: Testing the APIs** POSTMAN

Spring boot => จะเป็น framework ช่วยให้เขียน backent (java)

ex. java ทำ Rest → sprin boot
python → ง่วกว่า

week 2

- เป็น Framework ที่ช่วยจัดการสิ่งที่เราต้องทำซ้ำๆ โดยที่เราไม่ต้องเขียน code
- เป็น project ที่สร้างขึ้นมาทีหลังครอบตัว **Spring framework** (เก่ากว่า) ข้างในก็คือของ Spring framework = IOC (In version of control หรือ DI (dependency Injection))
  โดย Spring boot ช่วยทำ auto config



↳ ที่เขียน mannual ออกไป

Spring Boot Features

- Create stand-alone app
- มี tomcat อยู่ด้านในไม่จำเป็นต้อง deploy WAR files
- สิ่งที่ provide มา มี dependency ที่เป็น 'starter' มาให้ครบตัว
- ทำ Automatically configure Spring and 3rd party libraries (ที่จะ: ไรที่เป็นอัตโนมัติ)
- มีเครื่องมือในการจัดการ ขึ้น. ที่แยกได้
- ไม่ได้ใช้เพื่อ generate code แต่: spring boot ไม่จำเป็นต้อง XML config อีกต่อไป

ทำไม ? ควรใช้ Spring boot Framework

- ไม่มี fw อื่นให้เลือกใช้สำหรับทำ REST
- ทำ dependency injection (DI) เป็น core สำคัญของ spring
- มีตัวจัดการ transaction db ให้เรา
- Integrat กับตัว JPA
- ลดเวลาทุน/แรงในการทำ App ลงได้มาก เพราะมี dependency ช่วยเราเยอะ

Spring Boot: Auto Config

- ขึ้น. spring framework มักทำ MVC แต่จะต้องมาเติมไปด้วย configuration ที่เราต้องเขียนเอง

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix"><value>/WEB-INF/views/</value></property>
    <property name="suffix"><value>.jsp</value> </property>
</bean>

<mvc:resources mapping="/webjars/**" location="/webjars/"/>
```

- Spring boot จะช่วยแก้ปัญหาเหล่านี้ให้เรา ไม่ต้องเขียนเอง เพราะ: auto config

Spring Boot : Starter Projects

- เข้าไปดูใน pom file

Spring Boot Starter Project Options

- spring-boot-starter-web-services - SOAP Web Services
- **spring-boot-starter-web - Web & RESTful applications**
- spring-boot-starter-test - Unit testing and Integration Testing
- spring-boot-starter-jdbc - Traditional JDBC
- spring-boot-starter-hateoas - Add HATEOAS features to your services
- **spring-boot-starter-security - Authentication and Authorization using Spring Security**
- **spring-boot-starter-data-jpa - Spring Data JPA with Hibernate**
- spring-boot-starter-cache - Enabling Spring Framework's caching support
- spring-boot-starter-data-rest - Expose Simple REST Services using Spring Data REST

Creating Spring Boot Projects

- ใช้ web http://start.spring.io สร้างเสร็จก็ zip
- Using the Spring Tool Suite (STS) ใช้ id แบบ Eclipse
- Using IDE Bundled tool.

สร้างเสร็จจะมาพร้อมกับ Maven Wrapper เอา command maven มาใส่ให้เรา เพื่อให้เทรันได้

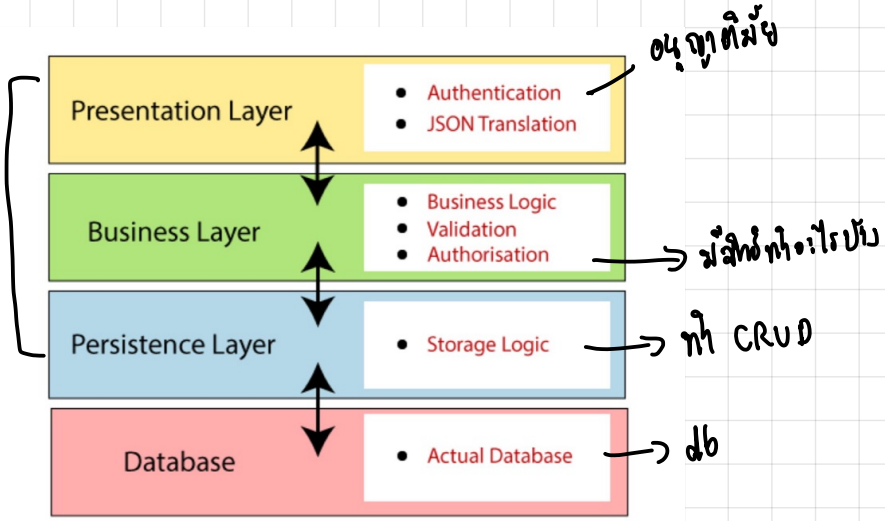
```
Maven Wrapper:
    mvnw dependency:tree
    mvnw spring-boot:run
```

maven project run ที่ไหนก็ได้ ไม่จำเป็นต้อง Intellij

Spring Boot Architecture

- แบ่งเป็น Layer จะแบ่งเป็น layer ตาม principle ของตัว REST

หลักการ

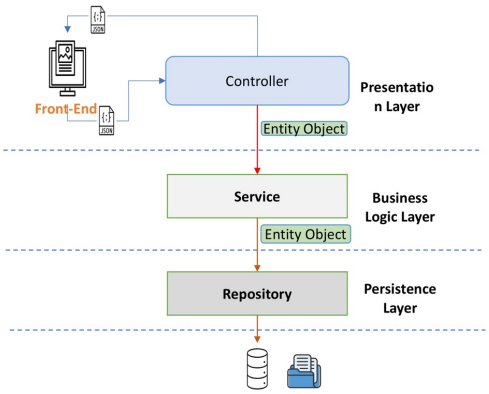


***

สร้าง API service เพื่อไม่ให้ลืม Layer

Spring Boot Flow Architecture

- Spring Boot uses all the modules of Spring-like Spring MVC, Spring Data, etc.
- Creates a data access layer and performs CRUD operation.
- The client makes the HTTP requests (GET or POST).
- The request goes to the controller, and the controller maps that request and handles it. After that, it calls the service logic if required.
- In the service layer, all the business logic performs. It performs the logic on the data that is mapped to JPA with model classes.
- A HTTP Response is returned to the user if no error occurred.



Spring Framework Annotations

week 3

- Basically, there are 6 types of annotation available in the whole spring framework.
  1. Spring Web Annotations เกี่ยวกับ Web / HTTP Protocal อยู่ใน presentation
  2. Spring Core Annotations
  3. Spring Boot Annotations
  4. Spring Scheduling Annotations
  5. Spring Data Annotations
  6. Spring Bean Annotations

1. Spring Web Annotations

- Present in the org.springframework.web.bind.annotation
- Some of the annotations that are available in this category are:
  - @RequestMapping
  - @RequestBody — json
  - @PathVariable
  - @RequestParam — ?
  - Response Handling Annotations
    - @ResponseBody @ExceptionHandler @ResponseStatus
  - @Controller
  - @RestController — ใช้เยอะ
  - @ModelAttribute
  - @CrossOrigin

ex. static

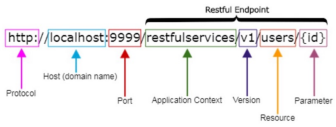
```
@RestController
@RequestMapping("/api/offices")
public class OfficeController {
   :
   :
   @GetMapping("/{officeCode}")
   public Office getOfficeById(@PathVariable String officeCode) {
      return service.getOffice(officeCode);
   }

   @PostMapping("")
   public Office addNewOffice(@RequestBody Office office) {
      return service.createNewOffice(office);
   }
```

path variable

อันใน ( ) ก็คือ PathVariable เมื่อ

อ้างถึงตัว Body ที่อยู่ใน request

REST API URI Naming Conventions and Best Practices



- Singleton and Collection Resources

```
/customers                  // is a collection resource
/customers/{id}             // is a singleton resource
```

- Sub-collection Resources

```
/customers/{id}/orders      // is a sub-collection resource
```

ไม่ควรทำเกิน 2 Level

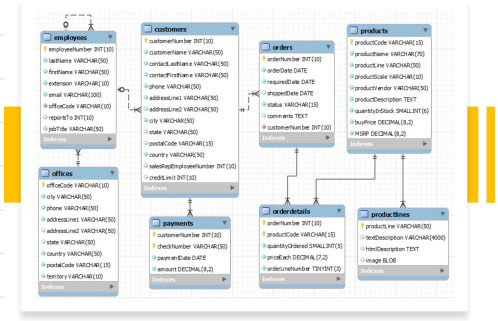
- Best Practices
  - https://medium.com/@nadinCodeHat/rest-api-naming-conventions-and-best-practices-1c4e781eb6a5
  - https://restfulapi.net/resource-naming

พื้นฐาน CRUD

ex. endpoint CUSTOMERS

/customers GET ALL customer
/Customers/{id} GET retrieve customer ตาม id ที่ระบุ
/Customers/{id} DELETE delete customer ตาม id
/Customers POST create customer
/customers/{id} PUT update customer ตาม id
/customers/{id}/orders GET
/customers/{id}/orders POST
/customers/{id}/orders DELETE
/customers/{id}/payments GET

* สามารถออกแบบ endpoint ได้ ขึ้นอยู่กับว่า front ต้องการบริการข้อมูลแบบไหนบ้าง



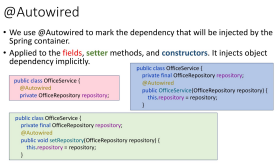
Excercise

Design REST Resources representation for the Classicmodels

2. Spring Core annotations

- Spring Core annotations are present in the 2 packages
  - org.springframework.beans.factory.annotation
  - org.springframework.context.annotation
- We can divide them into two broad categories:
  - DI-Related Annotations
    - @Autowired @Qualifier @Primary
    - @Bean @Lazy @Required
    - @Value @Scope
    - @Lookup, etc.
  - Context Configuration Annotations
    - @Profile @Import
    - @ImportResource @PropertySource, etc.

มี 3 ที่



เป็นการทำ dependency injection

fields injection
setter injection
constructors injection

จะเขียนง่าย

สร้าง object แต่ว่าสร้างที่ไหน

ของใช้ไปแล้ว เป็นการขอให้ spring boot ไปสร้าง object ของ data type ที่เราระบุ

3. Spring Boot Annotations

- @SpringBootApplication Combination of three annotations → เขียนอันนี้อันเดียวก็ได้
  - @EnableAutoConfiguration
  - @ComponentScan
  - @Configuration.

→ เขียนแยกเป็น 3 อันก็ได้

```
@SpringBootApplication

public class ClassicmodelServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(ClassicmodelServiceApplication.class, args);
    }
}
```

4. Spring Data Annotations => เน้นที่ Layer Repository

- Common Spring Data Annotations
  - @Transactional => จัดการ transaction ของ db
  - @Id
- Spring Data JPA Annotations
  - @Query

5. Spring Bean Annotations

- Some of the annotations that are available in this category are:
  - @ComponentScan
  - @Configuration
  - Stereotype Annotations
    - @Component
      - @Service
      - @Repository
      - @Controller

@Service: We specify a class with @Service to indicate that they're holding the business logic. Besides being used in the service layer, there isn't any other special use for this annotation. The utility classes can be marked as Service classes.
@Repository: We specify a class with @Repository to indicate that they're dealing with **CRUD operations**, usually, it's used with DAO (Data Access Object) or Repository implementations that deal with database tables.
@Controller: We specify a class with @Controller to indicate that they're front controllers and responsible to handle user requests and return the appropriate response. It is mostly used with REST Web Services.

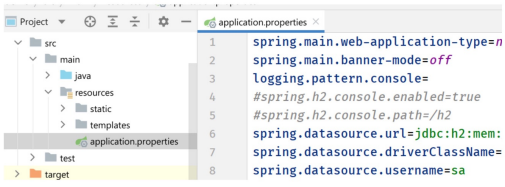
=> เน้นตัว Stereotype Annotations ให้เขียนระดับ class

เพื่อบอกว่า class นี้เป็น component ของ Spring boot ประกอบด้วย

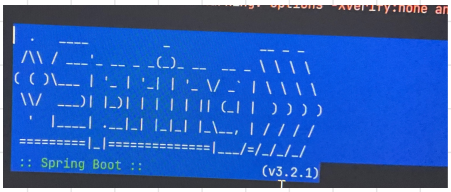
1. @Service
2. @Repository เช็ค CRUD
3. @Controller

Spring Boot Application Properties

- Spring Boot Framework comes with a built-in mechanism for application configuration using a file called application.properties.
- It is located inside the src/main/resources folder.
- The properties have default values.
- We can set a property(s) for the Spring Boot application.

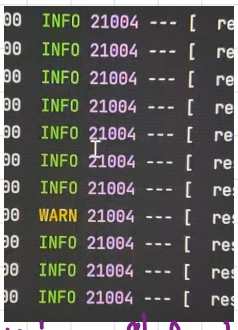


spring.main.banner-mode



ถ้าจะปิดก็ตั้งเป็น off

logging.level



* ข้อเสีย แบบนี้จะไม่มี trance อะไรมาให้ไม่เกี่ยวกับฝั่ง front

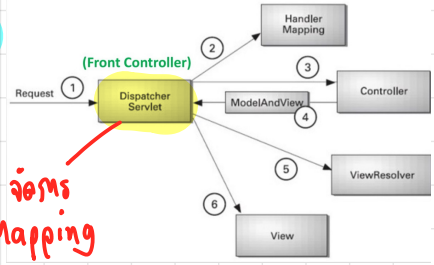
server.error.include-stacktrace

```
server.error.include-stacktrace=on_param
```

* อาจไม่ออก !!!

Spring Web MVC => มี spring MVC คอยจัดการให้เรา MVC ที่ทำงานขึ้น

Dispatcher Servlet and Flow of Spring Web MVC



Defining a Controller

- The DispatcherServlet delegates the request to the controllers to execute the functionality specific to it.
- The @Controller annotation indicates that a particular class serves the role of a controller.
- The @RequestMapping @GetMapping @PostMapping annotation is used to map a URL to either an entire class or a particular handler method.

```
@Controller
public class HelloController {
    @RequestMapping("/hello")
    public String printHello(ModelMap model) {
        model.addAttribute("message", "Hello Spring MVC Framework!");
        return "hello";
    }
}
```

Spring View Technology

เป็น plugins ที่จะมาทำให้เราสามารถเลือกได้ว่า เราจะใช้ View engins ตัวไหน

- JSP (Java Server Pages)
- Thymeleaf
- FreeMarker
- Groovy Markup Template Engine

Spring Data JPA

มี fw ที่ช่วยจัดการกับ CRUD ได้ง่ายขึ้น

มีทั้งแบบ relational และ แบบ no sql ใช้เหมือนกัน เราใช้ Sql

ทำ class ประเภท DAO (Data Access Object) => repository นั่นแหละ

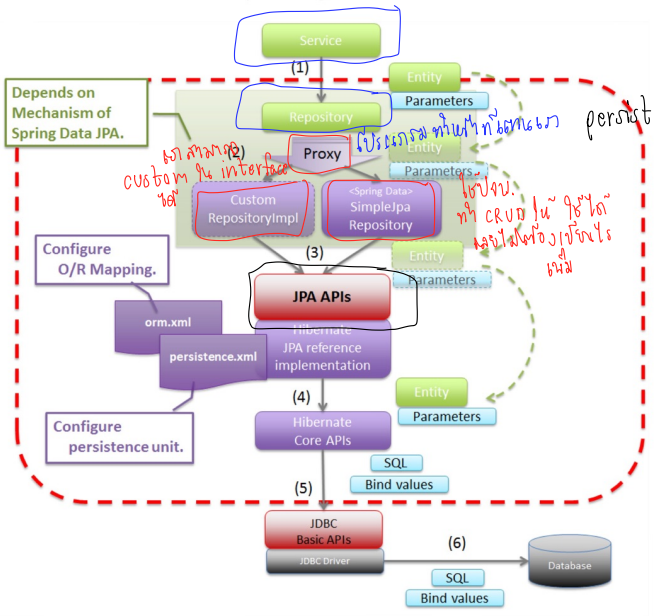
ถ้าเขียนของมันจะต้องเขียนซ้ำๆ boilerplate code

code ที่เขียนซ้ำๆ คล้ายๆกัน

ลดสิ่งนี้ด้วย Lombok

Spring Data JPA : extend ปุ๊ปใช้ได้ทันที

Basic Spring Data JPA Flow



persistence ธรรมดาไม่ต้องทำแล้ว proxy จะช่วย

JPA Repository Example

① ต้องการได้ Repository ในการจัดกับ Entity ตัวไหนก็ไปเขียน Interface

② Entity ที่จะจัดการ ต้องมี pk อยู่ในนั้น

ระบุไม่ชัดว่า เป็น repo ของ Entity ไหน

```
@Getter @Setter @NoArgsConstructor
@AllArgsConstructor @ToString
@Entity
public class Student {
    @Id
    private Integer id;
    private String name;
    private Double gpax;
}
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import sit.int204.demo.entities.Student;

public interface StudentRepository extends JpaRepository<Student, Integer> {
    List<Student> findByNameContainsOrGpaxBetweenOrderByGpaxDesc(
            String name, double low, double high);
}
```

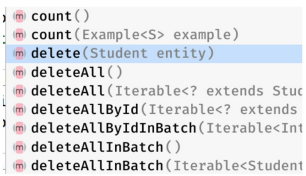
Query methods

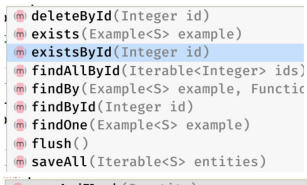
สามารถ custom ได้โดยการเขียน Query Methods

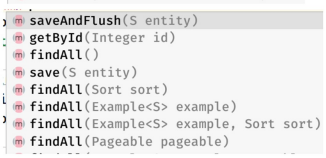
Jpa Repository default methods

```
public class AppController {
    @Autowired
    private final StudentRepository
studentRepository;
```

Query Method







week 4

Query Creation

การ Custom Query ลงไปใน repository

มี 2 วิธี

1. "Query Methods" => มีวิธีเขียนตาม pattern

```
public interface UserRepository extends Repository<User, Long> {
  List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);
}
```

ชื่อ Method จะบอกวิธี Query

2. "เขียน Query ลงไปเลยบอกเลยว่าจะให้ใช้คำสั่ง Query แบบไหน"

มี 2 แบบ
- JPA Query
- Native Query => ภาษา sql ของ db ตัวนั้นๆ

```
select u from User u where u.emailAddress = ?1 and u.lastname = ?2.
```

Query ส่งไปหา db

Supported keywords inside method names

Keyword	Sample	JPQL snippet
Distinct	findDistinctByLastnameAndFirstname	select distinct ... where x.lastname = ?1 and 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,findByFirstnameIs ,findByFirstnameEquals	... where x.firstname = ?1
Between	findByStartDateBetween	... where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1

เชื่อมเงื่อนไขมากกว่า 1

ตัวเลขตัวอักษร

Supported keywords inside method names (2)

Keyword	Sample	JPQL snippet
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	... where x.age >= ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull, Null	findByAge(Is)Null	... where x.age is null
IsNotNull, NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1

Supported keywords inside method names (3)

Keyword	Sample	JPQL snippet
StartingWith	findByFirstnameStartingWith	... where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	... where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	... where x.firstname like ?1 (parameter bound wrapped in %)
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

Query Method Example — ผสมชื่อ fild กับ keyword ที่เขาให้มา

```
public interface CustomerRepository extends JpaRepository<Customer, Integer> {
   public List<Customer> findAllByCustomerNameContaining(String name);
   public List<Customer> findAllByCityContainsOrderByCountry(String name);
   public List<Customer> findAllByCreditLimitBetween(Double lower, Double upper);
   public List<Customer> findAllByCustomerNameBetween(String lower, String upper);
}
```

JPA Named Queries

แบบ ① ตัว method จะสื่อเลยว่าจะให้ spring boot query ยังไง

แบบ ② เขียนชื่อตามใจเราเลย

```
public interface UserRepository extends JpaRepository<User, Long> {
   @Query("select u from User u where u.emailAddress = ?1")
   User findByEmailAddress(String emailAddress);
}
```

Native Query

ถ้าไม่ชอบ JPA Query สามารถเขียนเป็น native ได้เลย แต่ก็ยังไปว่า

- The @Query annotation allows for running native queries by setting the nativeQuery flag to true, as shown in the following example:
- Declare a native query at the query method using @Query

```
public interface UserRepository extends JpaRepository<User, Long> {
   @Query(value = "SELECT * FROM USERS WHERE EMAIL_ADDRESS = ?1", nativeQuery = true)
   User findByEmailAddress(String emailAddress);
}
```