MongoDB Document Relationship + JUnit Testing

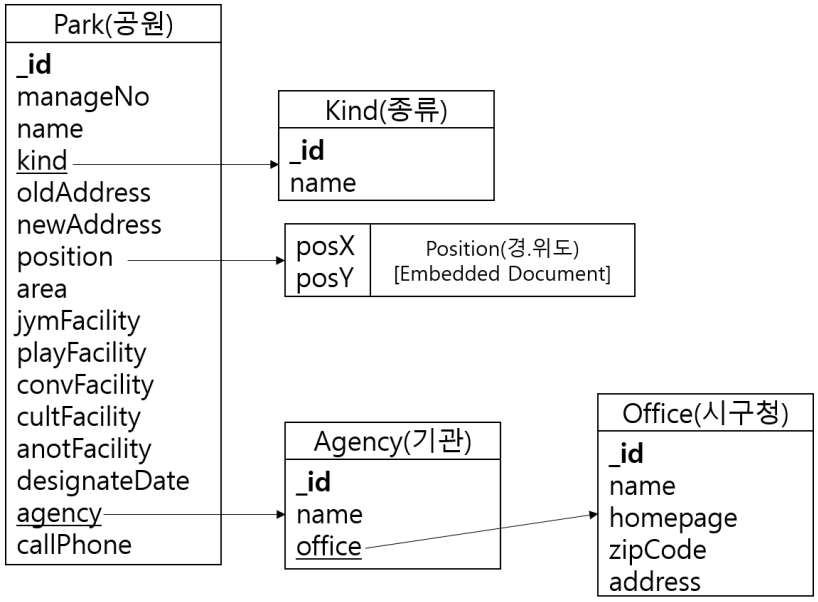
소프트웨어공학과 / 201332001 강인성 / hogu9401@gmail.com

0. 데이터베이스 구상과 정의

이번에는 지난 시간에 Singled Document에 대해서 REST API를 작성한 과정과 JUnit + Mockito Mock MVC를 이용해서 테스팅이 어떻게 이뤄지는지에 대해서 작성을 해 봤지만 이번 시간에는 NoSQL에서 Document 간의 관계성을 정의해서 REST API를 만들면서 동시에 단위 테스팅을 하는 소스코드에 대해서 작성을 할 것이다. 이번 시간에 쓰는 데이터베이스는 아래와 같이 성남시 공공데이터 API 중에서 성남시 도시공원 목록 정보에서 따온 엑셀 파일에 있는 데이터들을 활용해서 아래와 같은 구상으로 데이터를 쓸 예정이다. 아래에는 현재 데이터 중에 올라온 데이터 중 일부로서 변수 이름은 오른쪽과 같이 이용을 할 것이다.

관리번호(park.manageNo)	41131-00001
공원이름(park.name)	태평공원
공원구분(kind.name)	기타
지번주소(park.oldAddress)	경기도 성남시 수정구 복정동 산71-1
도로명주소(park.newAddress)	
위도(park.position.posX)	37.4615567
경도(park.position.posY)	127.1409336
공원면적(park.area)	96328
운동시설(park.gymFacility)	배드민턴장, 체력단련장
유희시설(park.playFacility)	
편의시설(park.convFacility)	화장실
교양시설(park.cultFacility)	
기타시설(park.anotFacility)	
지정고시일(park.designateDate)	1972-11-03
관리기관명(agency.name)	경기도 성남시 공원과
연락처(park.callPhone)	031-729-4264

NoSQL에서 쓰인 데이터베이스의 구성은 아래와 같이 구상할 수 있는데 이를 구상하여 어떻게 관계성을 접목하는지에 대해서 구상을 해 보도록 하겠다.



[그림] seongnam_city_park DB 구상도

seongnam_city_park.agency

DOCUMENTS 6 | TOTAL SIZE 880B | AVG. SIZE 147B | INDEXES 2 | TOTAL SIZE 72.0KB | AVG. SIZE 36.0KB

Documents | Explain Plan | Indexes

FILTER { field: 'value' } OPTIONS FIND

INSERT DOCUMENT VIEW LIST TABLE Displaying documents 1 - 6 of 6

```
_id: ObjectId("5ab5fd8e4421072258a1bd3c")
name: "경기도 성남시 공원과"
office: DBRef(office, 5abafca45870f22c1829be51, seongnam_city_park)

_id: ObjectId("5ab5fe284421072258a1bd3d")
name: "경기도 성남시 녹지과"
office: DBRef(office, 5abafca45870f22c1829be51, seongnam_city_park)

_id: ObjectId("5ab5fe3c4421072258a1bd3e")
name: "경기도 성남시 분당구 녹지공원과"
office: DBRef(office, 5ab5ffc94421072258a1bd43, seongnam_city_park)

_id: ObjectId("5ab5fe6d4421072258a1bd3f")
name: "경기도 성남시 수정구 환경위생과"
office: DBRef(office, 5ab6002c4421072258a1bd44, seongnam_city_park)

_id: ObjectId("5ab5fea24421072258a1bd40")
name: "경기도 성남시 중원구 공원과"
office: DBRef(office, 5ab6005c4421072258a1bd45, seongnam_city_park)

_id: ObjectId("5ab5fefa4421072258a1bd41")
name: "경기도 성남시 중원구 환경위생과"
office: DBRef(office, 5ab6005c4421072258a1bd45, seongnam_city_park)
```

seongnam_city_park.office

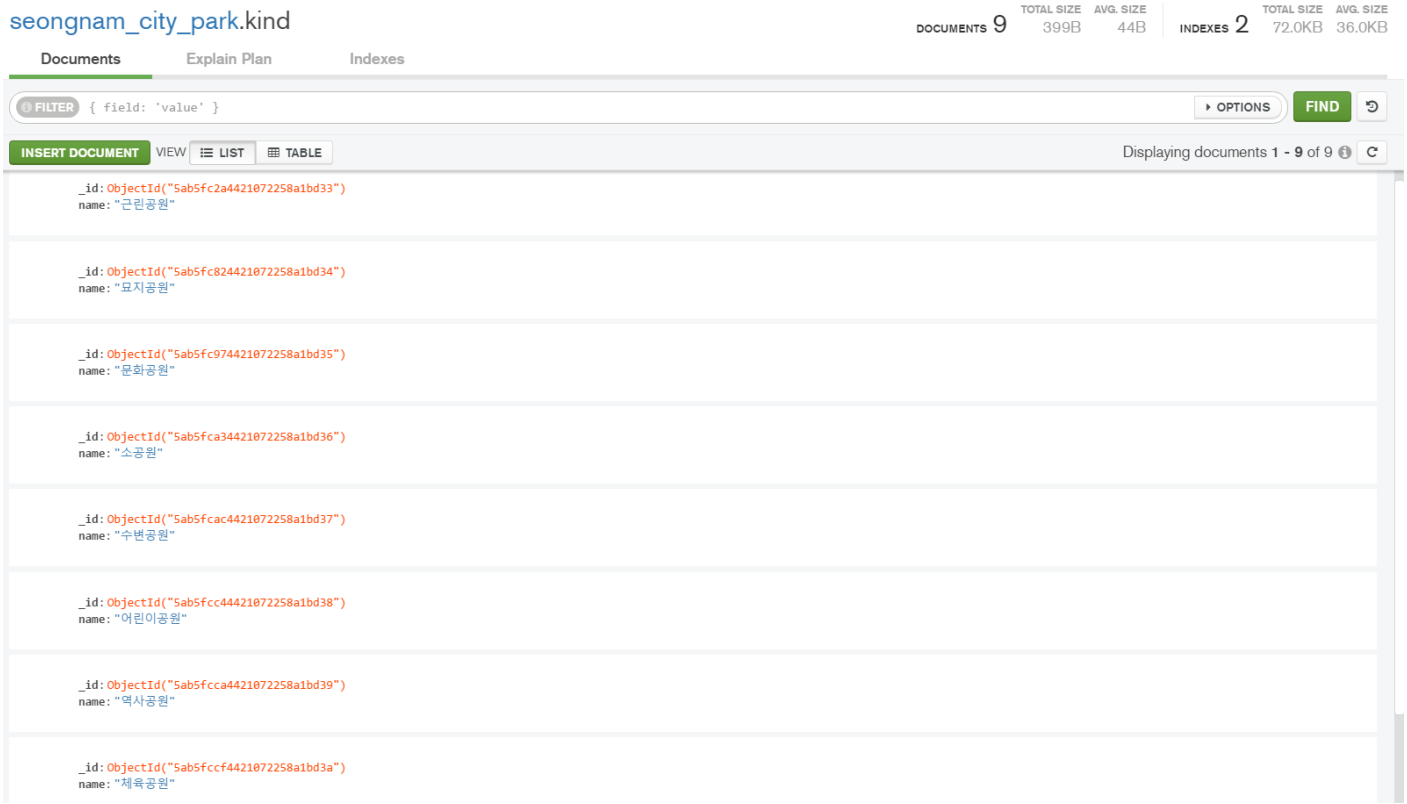
DOCUMENTS 4 | TOTAL SIZE 781B | AVG. SIZE 195B | INDEXES 1 | TOTAL SIZE 36.0KB | AVG. SIZE 36.0KB

Documents | Explain Plan | Indexes

FILTER { field: 'value' } OPTIONS FIND

INSERT DOCUMENT VIEW LIST TABLE Displaying documents 1 - 4 of 4

```
_id: ObjectId("5ab5ffc94421072258a1bd43")
name: "성남시 분당구청"
homepage: "http://www.bundang-gu.go.kr/main/index.asp"
zipCode: "13594"
address: "경기도 성남시 분당구 분당로 50 (수내동)"

_id: ObjectId("5ab6002c4421072258a1bd44")
name: "성남시 수정구청"
homepage: "http://www.sujeong-gu.go.kr/main/index.asp"
zipCode: "13259"
address: "경기도 성남시 수정구 수정로 283 (신흥동)"

_id: ObjectId("5ab6005c4421072258a1bd45")
name: "성남시 중원구청"
homepage: "http://www.jungwongu.go.kr/main/index.asp"
zipCode: "13371"
address: "경기도 성남시 중원구 제일로 36 (성남동)"

_id: ObjectId("5abafca45870f22c1829be51")
name: "성남시청"
homepage: "http://www.seongnam.go.kr/index.do"
zipCode: "13437"
address: "경기도 성남시 중원구 성남대로 997(여수동)"
```



각각 Agency, Office, Kind에 Document들을 임시로 추가를 하고 시작을 해야 효율적으로 작동할 수 있다. 엑셀 파일에서 제공하는 종류와 기관에 대해서는 피벗 테이블을 이용해서 어떠한 데이터들이 각각 존재하는지에 대해서 인지를 하고 난 이후에 이용을 하면 Document들을 추가하는데 어려움이 없을 것이다.

1. REST API 생성

A. Domain 클래스 생성

Domain 클래스들에 대해서는 net.kang.domain 패키지 내부에 존재하니 참고하길 바란다. 여기서 모든 기능들에 대해서 다루게 된다면 내용이 길어져서 새로 나오는 개념들 이외에는 GitHub에 올려 있는 주석 처리를 통한 설명으로 대체하고, **일부 중요한 개념들에 대해서만 따로 가져와서 설명을 하는 것으로 대체하겠다.**

```
package net.kang.domain;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.CompoundIndex;
import org.springframework.data.mongodb.core.index.CompoundIndexes;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;

import lombok.Data;
@Data
@Document(collection="agency")
@CompoundIndexes({
    @CompoundIndex(name = "name_unique", def = "{'name' : 1}", unique = true)
}) // 기관 이름을 유일하게 설정을 해서 findOne이 가능하게끔 하기 위해 이를 생성.
public class Agency { // 기관. 본인이 직접 만듬.
        @Id
        String id;
```

```
        String name; // 기관 이름
        @DBRef(db="seongnam_city_park", lazy=false)
        Office office; // 시-구청
}
```

[mongoDB_JPA_Shared_Example_01 > src > main > java > net > kang > domain > Agency.java]

@DBRef(db="데이터베이스_이름", lazy=true/false)

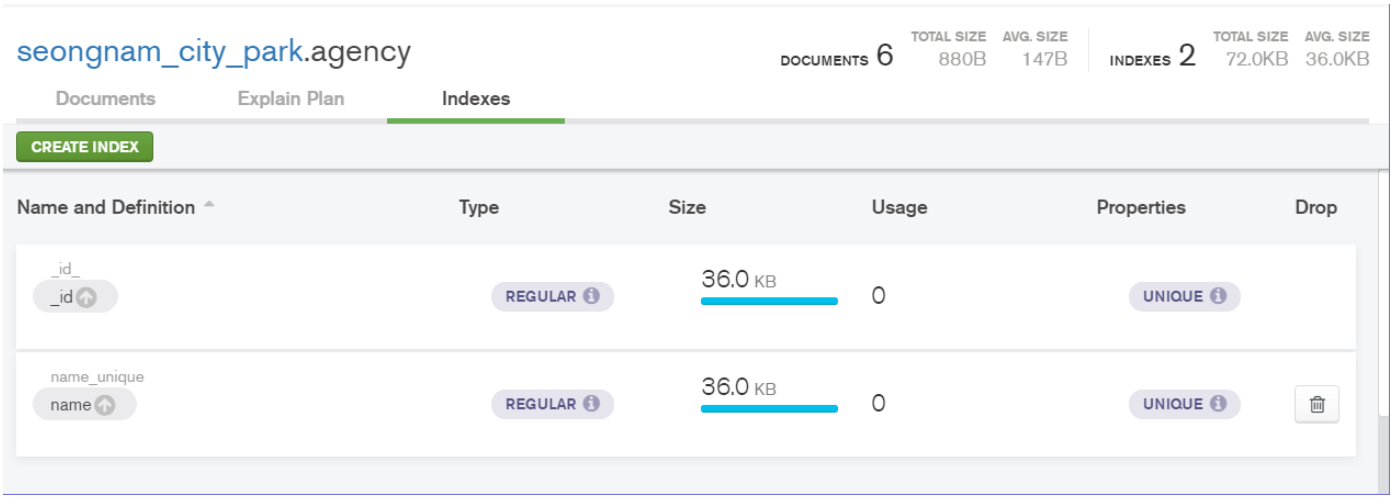
이는 agency Collection에 들어있는 Document 중에서 office 객체에 대해서 관계성을 부여할 수 있을 때 이를 이용해서 같은 도메인 패키지에 있는 Office를 접목할 수 있도록 형성을 하는 어노테이션으로 볼 수 있다. 여기서 db를 비워 두면 현재 데이터베이스 내에 있는 Collection을 참조할 수 있고, lazy는 Fetch에 대해서 true이면 LAZY 정책(One-To-Many), false이면 EAGER 정책(Many-To-One)으로 로딩을 할 수 있다. 이는 Many-To-One 관계를 가진 데이터베이스를 이용을 하는 것으로 볼 수 있는데 여기서 주의할 점은 Office 클래스에서는 List<Agency> agencies 변수를 이용할 수 없다는 것이다. MongoDB에서는 RDBMS와는 달리 관계성을 한 쪽으로만 주면 이용을 할 수 있는 점이 있기 때문에 office Document 내부에서 agency Document와의 One-To-Many 관계를 주지 않은 이상 자동으로 Mapping을 안 해주기 때문에 주의를 요한다.

@CompoundIndexes({

@CompoundIndex(name="인덱스_이름", def={'필드' : 1 / -1}, unique=true/false), ...

})

MongoDB 내부에서 미리 Index를 생성해서 이용할 수 있지만, Spring Data MongoDB에서는 각 도메인 클래스 내부에서 Index를 생성을 해서 이용을 해도 무관하게 된다. 또한 이 REST API 서버를 작성 완료한 이후에 아래와 같이 Index가 생성을 하게 되는데 여기서 주의를 해야 할 점이 이미 MongoDB Shell에서 인덱스를 생성하였을 경우에 같은 이름으로 인한 충돌이 있기 때문에 이를 유의해야 한다.



[그림] agency Collection에 생성된 Index. _id 인덱스는 agency Collection을 만들 때 자기가 생성이 된다.

```java
package net.kang.domain;

import java.util.Date;
import java.util.List;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.format.annotation.DateTimeFormat.ISO;

import lombok.Data;
import net.kang.model.Position;

@Data
@Document(collection="park")
public class Park { // 공원 데이터
	@Id
	String id;

	String manageNo; // 관리 번호
	String name; // 공원 이름

	@DBRef(db="seongnam_city_park", lazy=false)
	Kind kind; // 공원 종류

	String oldAddress; // 지번주소
	String newAddress; // 도로명주소

	Position position; // 위치(위도, 경도)
	double area; // 면적

	List<String> jymFacility; // 운동시설
	List<String> playFacility; // 유희시설
	List<String> convFacility; // 편의시설
	List<String> cultFacility; // 교양시설
	List<String> anotFacility; // 기타시설

	@DateTimeFormat(iso = ISO.DATE_TIME)
	Date designateDate; // 지정고시일

	@DBRef(db="seongnam_city_park", lazy=false)
	Agency agency; // 기관

	String callPhone; // 연락처
}
```

[mongoDB_JPA_Shared_Example_01 > src > main > java > net > kang > domain > Park.java]

Position position, List<String> ~Facility

이는 Embedded Document에 대해서 넣을 때 쓰는 변수로 볼 수 있다. Embedded Document를 이용하는 이유는 예를 들어 데이터의 관계가 1:Few(아니면 1:1)인 경우에 RDBMS에서 꼭 관계성을 주어서 JSON에서 재귀적으로 출력하여 Stack Overflow 에러를 낼 수 있기 때문에 굳이 관계성이 적은 데이터들에 대해서 이를 쓸 필요가 있는가를 구상할 때에 차라리 Embedded Document를 통해서 데이터들을 내부에서 쓰는 것이 올바를 수도 있다. 그래서 예를 들어 공원과 공원 위치에 대한 관계가 1:1인 경우를 볼 수 있는데 어느 공원은 위치가 공개되지 않은 경우를 고려해서 이에 대해서 굳이 담을 필요가 없다면 Embedded Document를 통해서 작성을 하는 방법도 올바른 방법이다. Position 클래스에 대해서는 아래와 같다.

```
package net.kang.model;

import lombok.Data;

@Data
public class Position {
	double posX;
	double posY;
	public Position(double posX, double posY) {
		this.posX=posX;
		this.posY=posY;
	}
}
```

[mongoDB_JPA_Shared_Example_01 > src > main > java > net > kang > model > Position.java]

@DateTimeFormat(iso=ISO.DATE_TIME)

Java에서 쓰이는 Date 객체에 대해서 이를 그대로 이용하게 된다면 MongoDB에서는 9시간이 지연되어 저장이 되는 문제가 발생하게 된다. 이는 MongoDB에서 쓰이는 시간 체계와 Java에서 쓰이는 시간 체계가 다르기 때문에 이를 보장하는 방법은 위와 같은 어노테이션을 이용해서 사전에 방지를 할 수 있다.

B. Repository 인터페이스 생성

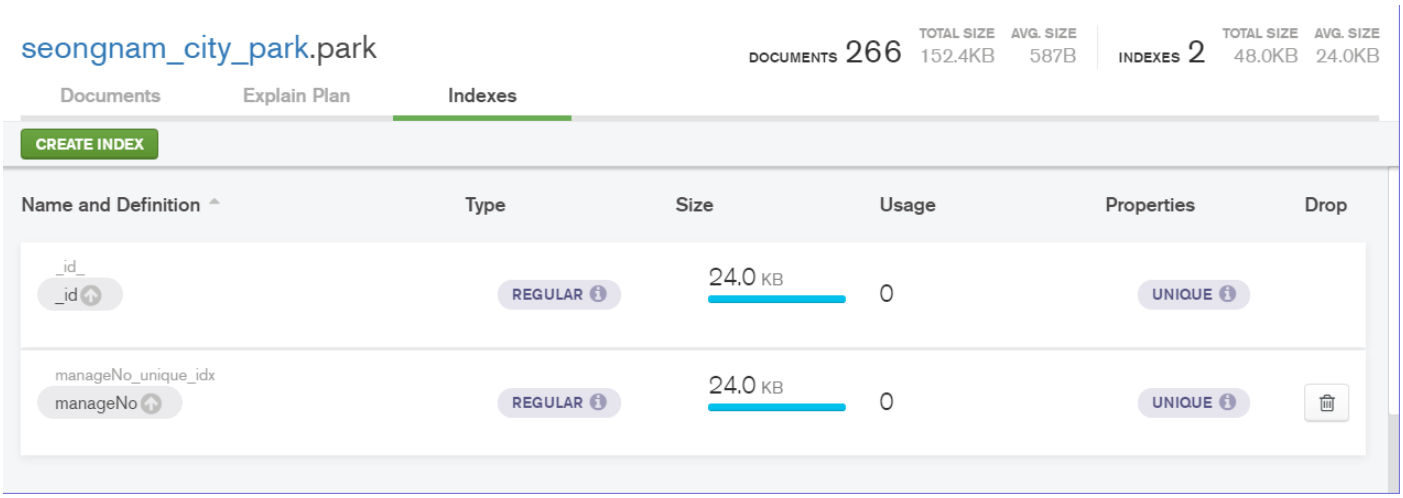
```
package net.kang.repository;

import java.util.List;
import java.util.Optional;

import org.springframework.data.mongodb.repository.MongoRepository;

import net.kang.domain.Agency;
import net.kang.domain.Kind;
import net.kang.domain.Park;

public interface ParkRepository extends MongoRepository<Park, String>{
	Optional<Park> findByManageNo(String manageNo); // 관리번호로 검색. 관리번호는 UNIQUE
설정.
```

```
        long countByKind(Kind kind); // 종류로 카운팅을 한 결과
        long countByAgency(Agency agency); // 기관으로 카운팅을 한 결과
        List<Park> findByKind(Kind kind); // 종류로 찾기
        List<Park> findByAgency(Agency agency); // 기관으로 찾기
        List<Park> findByNameContaining(String name); // 이름 포함 검색
        List<Park> findByAreaBetween(double area1, double area2); // 넓이 범위 이내 검색.
RDBMS의 Between과 같은 개념.
        List<Park> findByCultFacilityContains(String[] cultFacilities); // 문화 시설 포함 여부
검색
        List<Park> findByConvFacilityContains(String[] convFacilities); // 편의 시설 포함 여부
검색
        void deleteByManageNo(String manageNo); // 관리번호로 삭제
}
```

[mongoDB_JPA_Shared_Example_01 > src > main > java > net > kang > repository > ParkRepository.java]

Optional<Park> findByManageNo(String manageNo);

이전에 findOne에 대해서 JPA에서는 Park 변수를 이용해서 작성을 해도 무관했지만 최근 JPA 버전에서는 Optional<T>를 이용해서 작성을 해 줘야 한다. 또한 findOne을 할 수 있는 변수들에 대해서는 Unique 조건이 걸려 있는 Index를 생성을 해야 만들 수 있다는 점을 기억하자.



[그림] manageNo_unique_idx를 이용을 해야 findOne 함수를 만드는데 지장이 없다.

List<Park> findByKind(Kind kind), List<Park> findByAgency(Agency agency)

JPA에서는 데이터베이스와의 Object Mapping이 완료된 클래스에 대해서도 탐색을 할 수 있다. 방금 전에는 Many-To-One에 대한 관계에 대해서만 추가를 하였기 때문에 각 종류와 기관의 객체에 대하여 공원 목록을 검색을 하는 방법에 대해서는 접근이 가능하다는 뜻이다. JPA에서 객체로 검색을 하기 위해서는 Many-To-One 관계에 대해서 Object Mapping을 해 주고 난 후에 가능하다.

C. Service 클래스 생성

```
package net.kang.service;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import net.kang.domain.Agency;
import net.kang.domain.Office;
import net.kang.repository.AgencyRepository;
import net.kang.repository.OfficeRepository;

@Service
public class OfficeService { // 시구청 서비스 클래스 생성
	@Autowired OfficeRepository officeRepository;
	@Autowired AgencyRepository agencyRepository;

	public List<Office> findAll(){ // 시구청 목록 반환
		return officeRepository.findAll();
	}

	public Optional<Office> findOne(String id){ // 시구청 _id를 검색하여 한 시구청 반환
		return officeRepository.findById(id);
	}

	public List<Agency> findOneAndAgencyFindAll(String id){ // 시구청을 통해 해당 기관 검색.
기관이 존재하면 목록 반환.
		Optional<Office> office=officeRepository.findById(id);
		if(!office.orElse(new Office()).equals(new Office())) {
			return agencyRepository.findByOffice(office.get());
		}
		return new ArrayList<Agency>(); // 시구청 검색이 잘 못된다면 빈 기관 목록 반환
	}

	public List<Office> findByNameContaining(String name){ // 시구청 이름 포함 검색
		return officeRepository.findByNameContaining(name);
	}

	public boolean insert(Office office) { // 시구청 추가. 시구청 데이터 존재 여부 확인 이후
추가할 수 있다면 true 반환, 아닌 경우에는 false 반환.
		if(office.getId()==null) {
			officeRepository.insert(office);
			return true;
		}
		else if(!officeRepository.existsById(office.getId())) {
			officeRepository.insert(office);
			return true;
		}else return false;
	}

	public boolean update(Office office) { // 시구청 수정. 시구청 데이터 존재 여부 확인 이후
수정이 가능하면 true 반환, 아닌 경우에는 false를 반환.
```

```
                if(officeRepository.existsById(office.getId())) {
                        officeRepository.save(office);
                        return true;
                }else return false;
        }

        public boolean delete(String id) { // 시구청 삭제. 시구청 데이터 존재 여부 확인 이후
삭제가 가능하면 true를 반환, 아닌 경우에는 false를 반환.
                if(officeRepository.existsById(id)) {
                        officeRepository.deleteById(id);
                        return true;
                }else return false;
        }

        public boolean deleteByNameContaining(String name) { // 시구청 이름 포함 삭제. 시구청
이름 포함 데이터 목록 확인 결과에 따라서 존재하면 true, 아닌 경우 false를 반환.
                if(!officeRepository.findByNameContaining(name).isEmpty()) {
                        officeRepository.deleteByNameContaining(name);
                        return true;
                }else return false;
        }
}
```

[mongoDB_JPA_Shared_Example_01 > src > main > java > net > kang > service > OfficeService.java]

Service 객체를 생성하였지만 여기서는 간단하게 데이터를 CRUD를 하는 함수들에 대해서만 이용을 하기 때문에 대부분 쓰이는 함수가 Repository와 유사하게 쓰이는 경우가 많을 것이다. 이는 지난 시간에 Service 테스팅에 대해서 Mock MVC를 이용하지 않고 JUnit를 이용해서 했기 때문에 이번에는 오로지 Service 객체에 대해서만 테스팅을 하기 위해서 이에 대해 간략하게 진행하는 사례를 소개하기 위해 복잡하지 않은 로직으로 구상을 하였고, 또한 Create, Update, Delete 작업들에 대해서는 각각 Controller에서 요청을 제대로 인지할 수 있도록 하는 연습을 진행하는 목적을 가짐으로서 사전에 REST API에서 사고가 발생하지 않도록 하기 위해 지난 시간에는 void로 작성했던 것을 반대로 boolean을 이용해 각 작업들에 대한 성공과 실패 여부를 나뉘어서 반환을 하게 만들었다.

각 Service 클래스에 쓰인 함수 목록들에 대해서는 실제로 작성한 소스코드의 주석을 참고하면 쉽게 이해 할 수 있도록 작성을 해 뒀으니 이를 참고하면 도움이 된다.

D. Controller 클래스 생성

```
package net.kang.controller;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import net.kang.domain.Agency;
import net.kang.domain.Office;
import net.kang.service.OfficeService;

@RestController
@CrossOrigin
@RequestMapping("office")
public class OfficeController {
	@Autowired OfficeService officeService;

	@RequestMapping("findAll") // 성남시에 존재하는 모든 시-구청 검색
	public ResponseEntity<List<Office>> findAll(){
		List<Office> officeList=officeService.findAll();
		if(officeList.isEmpty()) {
			return new ResponseEntity<List<Office>>(officeList, HttpStatus.NO_CONTENT);
		}
		return new ResponseEntity<List<Office>>(officeList, HttpStatus.OK);
	}

	@RequestMapping("findOne/{id}") // ID를 통하여 시-구청 검색
	public ResponseEntity<Office> findOne(@PathVariable("id") String id) {
		Optional<Office> office=officeService.findOne(id);
		Office result=office.orElse(new Office());
		if(result.equals(new Office())) {
			return new ResponseEntity<Office>(result, HttpStatus.NO_CONTENT);
		}
		return new ResponseEntity<Office>(result, HttpStatus.OK);
	}

	@RequestMapping("findOne/agencyList/{id}") // 시-구청 내에 속하는 기관들을 검색
	public ResponseEntity<List<Agency>> findOneAndAgencyList(@PathVariable("id") String id){
		List<Agency> agencyList=officeService.findOneAndAgencyFindAll(id);
		if(agencyList.isEmpty()) {
			return new ResponseEntity<List<Agency>>(agencyList, HttpStatus.NO_CONTENT);
		}
		return new ResponseEntity<List<Agency>>(agencyList, HttpStatus.OK);
	}

	@RequestMapping("findByNameContaining/{name}") // 이름 포함 검색. Like %문자열%과 같다
	public ResponseEntity<List<Office>> findByNameContaining(@PathVariable("name") String name){
		List<Office> officeList=officeService.findByNameContaining(name);
		if(officeList.isEmpty()) {
			return new ResponseEntity<List<Office>>(officeList, HttpStatus.NO_CONTENT);
		}
		return new ResponseEntity<List<Office>>(officeList, HttpStatus.OK);
	}

	@RequestMapping(value="insert", method=RequestMethod.POST) // 성남에 있는 시-구청, 동주민센터
```

```
Document 추가
        public ResponseEntity<String> insert(@RequestBody Office office){
                if(officeService.insert(office)) {
                        return new ResponseEntity<String>("Office Inserting is Success.",
HttpStatus.CREATED);
                } else {
                        return new ResponseEntity<String>("Office Inserting is Failure. It is
Existed.", HttpStatus.CONFLICT);
                }
        }

        @RequestMapping(value="update", method=RequestMethod.PUT) // 성남에 있는 시-구청, 동주민센터
Document 수정
        public ResponseEntity<String> update(@RequestBody Office office){
                if(officeService.update(office)) {
                        return new ResponseEntity<String>("Office Updating is Success.",
HttpStatus.OK);
                }else {
                        return new ResponseEntity<String>("Office Updating is Failure. It is Not
Existed.", HttpStatus.NOT_MODIFIED);
                }
        }

        @RequestMapping(value="delete/{id}", method=RequestMethod.DELETE) // 성남에 있는 시-구청,
동주민센터 Document 삭제
        public ResponseEntity<String> delete(@PathVariable("id") String id){
                if(officeService.delete(id)) {
                        return new ResponseEntity<String>("Office Deleting is Success.",
HttpStatus.OK);
                }else {
                        return new ResponseEntity<String>("Office Deleting is Failure. It is Not
Existed.", HttpStatus.NOT_FOUND);
                }
        }

        @RequestMapping(value="deleteByNameContaining/{name}") // Office의 name에 포함된 Document 삭제
        public ResponseEntity<String> deleteByNameContaining(@PathVariable("name") String name){
                if(officeService.deleteByNameContaining(name)) {
                        return new ResponseEntity<String>(String.format("Name Containin' %s Deleting is
Success.", name), HttpStatus.OK);
                }else {
                        return new ResponseEntity<String>(String.format("Name Containin' %s Deleting is
Failure.", name), HttpStatus.NOT_FOUND);
                }
        }
}
```

[mongoDB_JPA_Shared_Example_01 > src > main > java > net > kang > controller > OfficeController.java]

ResponseEntity< T > (T t , HttpStatus.?)

우리가 REST API를 작성을 한 경우에는 그냥 데이터에 대해서 반환을 했지만 실제로 Controller 클래스에 대해서 테스팅을 할 때에 URL 요청 결과에 대한 상태(status)를 확인하지 못 하기 때문에 그냥 데이터를 반환하는 방법에 대한 습관을 고쳐야 한다. 실제로 ResponseEntity를 이용해서 반환을 한다 쳐도 Controller 단위 클래스 이외에도 데이터는 정상적으로 반환이 되는 대신에 나중에 서버에서 클라이언트로 Request에 대한 결과를 받아 들일 때 상태를 구분을 해서 클라이언트 내에서 문제 없이 작동을 하기 위해 이를 작성을 해 줌으로서 만연의 사고를 방지 할 필요가 있다. 예를 들어 데이터 목록으로 검색을 하는 경우에 데이터가 존재한다면 200(OK) 상태를 반환하고, 없는 경우에는 Not Found(404)가 아닌 No Content(204)를 반환해야 Request에서 받아 들이는 상태에 대

해 정리를 하면서 올바른 논리로 돌아 갈 수 있기 때문이다. 대표적으로 쓰이는 Status에 대해서는 아래 페이지에서 어떠한 목록들이 있는지에 대해 확인을 해서 올바르게 이용을 하면 좋겠다.

https://ko.wikipedia.org/wiki/HTTP_%EC%83%81%ED%83%9C_%EC%BD%94%EB%93%9C

2. 단위 테스팅 클래스 생성

REST API에 대해서 이미 작성을 완료한 경우에는 단위 테스팅을 진행하고 난 후에 통합 테스팅 진행까지 완료를 해야 한다. 여기서 단위 테스팅을 하는 뜻은 지난 시간에도 이미 공부를 했듯이 Repository, Service, Controller 단위 별로 테스팅을 할 필요가 있다. 그렇지만 Service 객체 내부에 있는 일부 Repository, Controller 객체 내부에 있는 Service 객체에 대해서는 가짜로 움직여서 데이터 결과를 가져와서 추측을 할 수 있도록 하기 위해 Mockito Mock MVC 프레임워크를 이용해서 각 단위 테스팅이 보장이 되도록 해야 한다. 이번에는 Park 클래스를 이용한 각 단위 ParkRepository, ParkService, ParkController 들에 대해서 테스팅을 하기 위해 각각 알아보도록 하고 여기서는 중요한 개념들에 대해서만 짚고 넘어가도록 하겠다.

A. Repository 단위 테스팅 클래스 생성

```
package net.kang.unit.repository;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Optional;
import java.util.Random;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.data.mongodb.repository.config.EnableMongoRepositories;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import net.kang.config.JUnitConfig;
import net.kang.config.MongoConfig;
import net.kang.domain.Agency;
import net.kang.domain.Kind;
import net.kang.domain.Park;
import net.kang.repository.AgencyRepository;
import net.kang.repository.KindRepository;
import net.kang.repository.ParkRepository;
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = {JUnitConfig.class, MongoConfig.class})
@EnableMongoRepositories(basePackageClasses = {net.kang.repository.ParkRepository.class,
net.kang.repository.AgencyRepository.class, net.kang.repository.KindRepository.class})
@EntityScan(basePackageClasses = {net.kang.domain.Park.class, net.kang.domain.Agency.class,
net.kang.domain.Kind.class})
public class ParkRepositoryTest { // 공원 Repository Unit Testing 클래스 생성.

        @Autowired ParkRepository parkRepository;
        @Autowired AgencyRepository agencyRepository;
        @Autowired KindRepository kindRepository;
        private final String[] tmpManageNoList= {"00000-00000", "00000-00001", "00000-00002", "00000-
00003", "00000-00004"}; // 임시로 쓸 관리 번호
```

```
	static final int QTY=5; // 공원의 수는 임시로 5개로 한다.
	static Random random=new Random();

	public List<String> makeFacility(String s, int length){ // 임시로 시설 목록을 만들어준다.
		List<String> tmpFacility=new ArrayList<String>();
		for(int k=0;k<length;k++) {
			tmpFacility.add(String.format("%s%02d", s, k));
		}
		return tmpFacility;
	}

	@Before
	public void initialize() {
		parkRepository.deleteAll();
		List<Kind> kindList=kindRepository.findAll();
		List<Agency> agencyList=agencyRepository.findAll(); // 각각 Kind와 Agency를 가져와서
Park에 설정을 할 수 있도록 한다.
		for(int k=0;k<QTY;k++) {
			Park park=new Park();
			park.setManageNo(tmpManageNoList[k]);
			park.setName(String.format("공원%02d", k));
			park.setKind(kindList.get(random.nextInt(kindList.size())));
			park.setOldAddress(String.format("지번주소%02d", k));
			park.setNewAddress(String.format("도로명주소%02d", k));
			park.setArea(100.0+random.nextDouble()*100);
			park.setConvFacility(makeFacility("편의", 2+random.nextInt(2)));
			park.setCultFacility(makeFacility("문화", 2+random.nextInt(2)));
			park.setDesignateDate(new Date());
			park.setAgency(agencyList.get(random.nextInt(agencyList.size())));
			park.setCallPhone("031-000-0000");
			parkRepository.insert(park);
		} // Mock Data에 대해 각각 추가시킨다.
	}

	@Test
	public void findAllTest() { // findAll 테스팅. findAll를 하고 Mock Data들이 정확히 들어갔는지
확인한다.
		List<Park> parkList=parkRepository.findAll();
		assertEquals(QTY, parkList.size());
	}

	@Test
	public void findByManageNoTest() { // findByManageNo 테스팅. manageNo를 이용해서 찾은 결과에 대해
현존하는지 확인시킨다.
		int getIndex=random.nextInt(QTY);
		Optional<Park> park=parkRepository.findByManageNo(tmpManageNoList[getIndex]);
		assertTrue(parkRepository.existsById(park.get().getId()));
	}

	// 일부 조회 함수들에 대해서는 생략

	@Test
	public void insertTest() { // insert 테스팅. 데이터를 추가하고 난 후 findAll를 한 결과를 통해
확인한다.
		List<Kind> kindList=kindRepository.findAll();
		List<Agency> agencyList=agencyRepository.findAll();
		Park park=new Park();
		park.setManageNo("00000-00005");
		park.setName(String.format("공원%02d", 5));
		park.setKind(kindList.get(random.nextInt(kindList.size())));
```

```
                park.setOldAddress(String.format("지번주소%02d", 5));
                park.setNewAddress(String.format("도로명주소%02d", 5));
                park.setArea(100.0+random.nextDouble()*100);
                park.setConvFacility(makeFacility("편의", 2+random.nextInt(2)));
                park.setCultFacility(makeFacility("문화", 2+random.nextInt(2)));
                park.setDesignateDate(new Date());
                park.setAgency(agencyList.get(random.nextInt(agencyList.size())));
                park.setCallPhone("031-000-0000");
                parkRepository.insert(park);
                List<Park> findAll=parkRepository.findAll();
                assertEquals(findAll.size(), QTY+1);
        }

        @Test
        public void updateTest() { // update 테스팅. 데이터를 갱신하고 난 후에 그 현존하는 데이터가 올바르게
수정됐는지 확인을 한다.
                List<Kind> kindList=kindRepository.findAll();
                List<Agency> agencyList=agencyRepository.findAll();
                List<Park> parkList=parkRepository.findAll();
                Park park=parkList.get(random.nextInt(parkList.size()));
                park.setName("공원Temp");
                park.setKind(kindList.get(random.nextInt(kindList.size())));
                park.setOldAddress("지번주소Temp");
                park.setNewAddress("도로명주소Temp");
                park.setArea(100.0+random.nextDouble()*100);
                park.setConvFacility(makeFacility("편의", 1+random.nextInt(3)));
                park.setCultFacility(makeFacility("문화", 1+random.nextInt(3)));
                park.setDesignateDate(new Date());
                park.setAgency(agencyList.get(random.nextInt(agencyList.size())));
                park.setCallPhone("031-000-0000");
                parkRepository.save(park);

                Park updatePark=parkRepository.findByManageNo(park.getManageNo()).get();
                assertEquals(updatePark, park);
        }

        @Test
        public void deleteTest() { // delete 테스팅. 데이터를 삭제하고 findAll를 한 결과로 확인을 한다.
                List<Park> parkList=parkRepository.findAll();
                Park park=parkList.get(random.nextInt(parkList.size()));
                parkRepository.deleteById(park.getId());
                assertEquals(parkRepository.findAll().size(), QTY-1);
        }

        @After
        public void afterTest() { // 테스팅이 완료되는 시점에서 Mock 데이터 목록들을 삭제한다.
                parkRepository.deleteAll();
        }
}
```

[mongoDB_JPA_Shared_Example_01 > src > main > test > net > kang > unit > repository > ParkRepositoryTest.java]

예를 들어 공원 Repository로 테스팅을 할 수 있는 상황을 볼 수 있다면 각각 시나리오 별로 나뉘어서 이해를 한다면 아래와 같이 정리할 수 있다.

@Before

공원 데이터에서 쓸 Mock 데이터들에 대해서 추가를 미리 해 두고 차후에 각 함수 별로 테스팅을 돌아서 실제로 얻은 결과에 대해 직접 비교를 하면서 Repository 함수를 통해 데이터의 유효성을 한 번 더 확인을 한다.

@Test

각 테스팅 함수들에 대해서 실행되기 전에 Before 함수에 대해 실행을 하고 난 후에 작동을 한다. 테스팅 함수는 아래와 같이 구성되어 있다.

findAllTest() : 모든 공원 목록을 불러와서 데이터를 확인한다.

findByMangeNoTest() : 한 공원에 대한 데이터를 manageNo(관리 번호)를 통해 가져와서 데이터를 확인한다.

insertTest(), updateTest(), deleteTest() : 각각 삽입, 갱신, 삭제에 대하여 작업을 하고 난 이후에 확인을 하도록 한다.

@After

이전에 추가했던 Mock 데이터들에 대해서 삭제를 해 주도록 한다.

B. Service 단위 테스팅 클래스 생성

```
package net.kang.unit.service;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
import static org.mockito.Mockito.doNothing;
import static org.mockito.Mockito.when;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.Random;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.mockito.junit.MockitoJUnitRunner;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import net.kang.config.JUnitConfig;
import net.kang.config.MongoConfig;
import net.kang.domain.Agency;
import net.kang.domain.Kind;
import net.kang.domain.Office;
import net.kang.domain.Park;
import net.kang.model.ParkForm;
import net.kang.model.Position;
import net.kang.repository.AgencyRepository;
import net.kang.repository.KindRepository;
import net.kang.repository.ParkRepository;
import net.kang.service.ParkService;
@RunWith(MockitoJUnitRunner.class)
@ContextConfiguration(classes = {JUnitConfig.class, MongoConfig.class})
@WebAppConfiguration
public class ParkServiceTest {
        static final int PARK_QTY=10;
        static final int AGENCY_QTY=2;
```

```
	static final int KIND_QTY=2;
	static final int FACILITY_QTY=2;
	static Random random=new Random();
	MockMvc mockMvc;
	@Mock ParkRepository parkRepository;
	@Mock KindRepository kindRepository;
	@Mock AgencyRepository agencyRepository;
	@InjectMocks ParkService parkService;

	@Before
	public void initialize() {
		MockitoAnnotations.initMocks(this);
		mockMvc=MockMvcBuilders.standaloneSetup(parkService).build();
	}

	// 중략. 여기서 Mock 데이터들에 대해서 가져와서 테스팅을 할 때 이용이 된다.

	@Test
	public void findAllTest() {
		List<Park> tmpResult=parkList();
		when(parkRepository.findAll()).thenReturn(tmpResult);
		List<Park> findAllResult=parkService.findAll();
		assertEquals(tmpResult, findAllResult);
	}

	// 중략. 여기에 findOnePark를 통해 하나의 Park를 반환한다.

	@Test
	public void findOneTest() {
		Park tmpResult=findOnePark();
		when(parkRepository.findById("1")).thenReturn(Optional.of(tmpResult));
		Optional<Park> findOneResult=parkService.findOne("1");
		assertEquals(tmpResult, findOneResult.get());
	}

	// … 중략 …

	@Test
	public void insertSuccessTest() {
		Park park=findOnePark();
		ParkForm parkForm=parkToForm(park);
		Kind kind=new Kind();
		kind.setId("1");
		kind.setName("종류01");
		Agency agency=new Agency();
		agency.setId("1");
		agency.setName("기관01");
		when(kindRepository.findById("1")).thenReturn(Optional.of(kind));
		when(agencyRepository.findById("1")).thenReturn(Optional.of(agency));
		Park insertAfterPark=formToPark(parkForm, agency, kind);
		when(parkRepository.insert(insertAfterPark)).thenReturn(insertAfterPark);
		assertTrue(parkService.insert(parkForm));
	}

	@Test
	public void insertFailureTest() {
		Park park=findOnePark();
		ParkForm parkForm=parkToForm(park);
		when(kindRepository.findById("1")).thenReturn(Optional.of(new Kind()));
		when(agencyRepository.findById("1")).thenReturn(Optional.of(new Agency()));
		assertFalse(parkService.insert(parkForm));
	}

	@Test
	public void updateSuccessTest() {
		Park park=findOnePark();
```

```
                ParkForm parkForm=parkToForm(park);
                double randArea=100.0+random.nextDouble()*100.0;
                Kind kind=new Kind();
                kind.setId("1");
                kind.setName("종류01");
                Agency agency=new Agency();
                agency.setId("1");
                agency.setName("기관01");
                when(kindRepository.findById("1")).thenReturn(Optional.of(kind));
                when(agencyRepository.findById("1")).thenReturn(Optional.of(agency));
                when(parkRepository.existsById(parkForm.getParkId())).thenReturn(true);
                parkForm.setName("공원TEMP");
                parkForm.setOldAddress("지번주소TEMP");
                parkForm.setNewAddress("도로명주소TEMP");
                parkForm.setArea(randArea);
                Park saveAfterPark=formToPark(parkForm, agency, kind);
                when(parkRepository.save(saveAfterPark)).thenReturn(saveAfterPark);
                assertTrue(parkService.update(parkForm));
        }

        @Test
        public void updateFailureTest() {
                Park park=findOnePark();
                ParkForm parkForm=parkToForm(park);
                when(parkRepository.existsById(parkForm.getParkId())).thenReturn(false);
                when(kindRepository.findById("1")).thenReturn(Optional.of(new Kind()));
                when(agencyRepository.findById("1")).thenReturn(Optional.of(new Agency()));
                assertFalse(parkService.update(parkForm));
        }

        @Test
        public void deleteSuccessTest() {
                Park park=findOnePark();
                when(parkRepository.existsById(park.getId())).thenReturn(true);
                doNothing().when(parkRepository).deleteById(park.getId());
                assertTrue(parkService.delete(park.getId()));
        }

        @Test
        public void deleteFailureTest() {
                Park park=findOnePark();
                when(parkRepository.existsById(park.getId())).thenReturn(false);
                assertFalse(parkService.delete(park.getId()));
        }

        // … 중략 …
}
```

[mongoDB_JPA_Shared_Example_01 > src > main > test > net > kang > unit > service > ParkServiceTest.java]

Service 객체 단위 테스팅에서 새로 추가한 것은 바로 삽입, 삭제에 대해서 성공과 실패를 나뉘어서 테스팅을 하였다. 그리고 이전 Service 객체 단위 테스팅에서는 Repository를 직접 이용을 해서 작성을 하였지만 이에 대해서 적용을 하는 것은 Service 객체를 테스팅하는 방법을 생각한다면 올바른 방법이 아니다. 그래서 Service에서는 Mock을 이용해서 Repository에서만 이를 적용하고 나머지는 Service 객체에게 맡겨서 추측을 하면 결과가 나오게끔 하는 방안으로 작성을 하였으니 참고하길 바란다. 실제로 소스코드는 위에서 작성한 것보다 더 길어지는데 여기서는 간략하게 어떤 원리로 돌아가는지에 대해 이해를 돕기 위해 일부분을 생략하였다.

@Before

Mock 객체들에 대해서 이용을 할 수 있도록 설정을 한다. 그러면 여기서 Mock 객체는 Repository가 되고, 실제로 작동을 하면서 확인을 하는 객체는 Service가 된다.

@Test

원래 테스팅 함수는 많지만 대표적으로 쓰는 함수들에 대해서 테스팅을 할 때 나오는 결과를 토대로 작성을 하겠다.

findAllTest : Mock 데이터들에 대해서 parkRepository의 findAll를 이용해 이 값으로 추측을 하고 난 후에 Service 객체를 통해 얻은 결과와 비교를 한다.

findOneTest : Mock 데이터들에 대해서 parkRepository의 findById를 이용해서 추측을 하고 Service 객체를 통해 얻은 결과를 비교한다.

insertSuccessTest, insertFailureTest, updateSuccessTest, updateFailureTest : Mock 데이터들에 대해서 삽입, 갱신을 할 때 나오는 데이터들(최근 JPA에서는 insert, save를 하면 저장된 객체에 대해 나오는 방향으로 개선되어 있어서 doNothing()으로 처리하면 안 된다.)에 대해 추측을 하고 난 후에 이에 맞춰서 Service 객체에 나오는 boolean 형 데이터를 통해 확인을 한다.

deleteSuccessTest, deleteFailureTest : Mock 데이터가 아니라 오로지 데이터의 ID를 통해 삭제 작업을 이뤄지게끔 하고 난 후에 나오는 결과에 따라서 Service 객체가 올바르게 작동하는지에 대해 확인을 한다.

C. Controller 단위 클래스 생성

```
package net.kang.unit.controller;

import static org.hamcrest.CoreMatchers.equalTo;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.verifyNoMoreInteractions;
import static org.mockito.Mockito.when;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.delete;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.put;
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.Random;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
```

```
import org.mockito.MockitoAnnotations;
import org.mockito.junit.MockitoJUnitRunner;
import org.springframework.http.MediaType;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;

import net.kang.config.JUnitConfig;
import net.kang.config.MongoConfig;
import net.kang.controller.ParkController;
import net.kang.domain.Agency;
import net.kang.domain.Kind;
import net.kang.domain.Office;
import net.kang.domain.Park;
import net.kang.model.ParkForm;
import net.kang.model.Position;
import net.kang.service.ParkService;

@RunWith(MockitoJUnitRunner.class)
@ContextConfiguration(classes = {JUnitConfig.class, MongoConfig.class})
@WebAppConfiguration
public class ParkControllerTest {
	static final int PARK_QTY=10;
	static final int AGENCY_QTY=2;
	static final int KIND_QTY=2;
	static final int FACILITY_QTY=2;
	static Random random=new Random();
	MockMvc mockMvc;
	@Mock ParkService parkService;
	@InjectMocks ParkController parkController;

	private String jsonStringFromObject(Object object) throws JsonProcessingException {
		ObjectMapper mapper = new ObjectMapper();
		return mapper.writeValueAsString(object);
	}

	// Mock 데이터 생성 함수들은 중략.

	@Before
	public void initialize() {
		MockitoAnnotations.initMocks(this);
		mockMvc=MockMvcBuilders.standaloneSetup(parkController).build();
	}

	@Test
	public void findAllSuccessTest() throws Exception{
		List<Park> tmpList=parkList();
		when(parkService.findAll()).thenReturn(tmpList);
		String toJSON=this.jsonStringFromObject(tmpList);
		mockMvc.perform(get("/park/findAll"))
		.andExpect(status().isOk())
		.andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON_UTF8))
		.andExpect(content().string(equalTo(toJSON)))
		.andDo(print());

		verify(parkService, times(1)).findAll();
		verifyNoMoreInteractions(parkService);
	}

	@Test
	public void findAllFailureTest() throws Exception{
		List<Park> tmpList=new ArrayList<Park>();
		when(parkService.findAll()).thenReturn(tmpList);
		String toJSON=this.jsonStringFromObject(tmpList);
```

```
                mockMvc.perform(get("/park/findAll"))
                .andExpect(status().isNoContent())
                .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON_UTF8))
                .andExpect(content().string(equalTo(toJSON)))
                .andDo(print());

                verify(parkService, times(1)).findAll();
                verifyNoMoreInteractions(parkService);
        }

        @Test
        public void findByManageNoSuccessTest() throws Exception{
                Optional<Park> tmpResult=Optional.of(findOnePark());
                when(parkService.findByManageNo("00000-00001")).thenReturn(tmpResult);
                String toJSON=this.jsonStringFromObject(tmpResult.get());
                mockMvc.perform(get("/park/findByManageNo/{manageNo}", "00000-00001"))
                .andExpect(status().isOk())
                .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON_UTF8))
                .andExpect(content().string(equalTo(toJSON)))
                .andDo(print());

                verify(parkService, times(1)).findByManageNo("00000-00001");
                verifyNoMoreInteractions(parkService);
        }

        // 일부 조회 함수들은 내용이 길어져서 중략.

        @Test
        public void insertSuccessTest() throws Exception {
                Park park=findOnePark();
                ParkForm tmpForm=parkToForm(park);
                when(parkService.insert(tmpForm)).thenReturn(true);
                String requestPark=this.jsonStringFromObject(tmpForm);

                mockMvc.perform(
                                post("/park/insert")
                                .contentType(MediaType.APPLICATION_JSON_UTF8)
                                .content(requestPark))
                .andExpect(status().isCreated())
                .andDo(print()).andReturn();

                verify(parkService, times(1)).insert(tmpForm);
                verifyNoMoreInteractions(parkService);
        }

        @Test
        public void insertFailureTest() throws Exception{
                Park park=findOnePark();
                ParkForm tmpForm=parkToForm(park);
                when(parkService.insert(tmpForm)).thenReturn(false);
                String requestPark=this.jsonStringFromObject(tmpForm);

                mockMvc.perform(
                                post("/park/insert")
                                .contentType(MediaType.APPLICATION_JSON_UTF8)
                                .content(requestPark))
                .andExpect(status().isInternalServerError())
                .andDo(print()).andReturn();

                verify(parkService, times(1)).insert(tmpForm);
                verifyNoMoreInteractions(parkService);
        }

        @Test
        public void updateSuccessTest() throws Exception {
                Park park=findOnePark();
                ParkForm tmpForm=parkToForm(park);
                when(parkService.update(tmpForm)).thenReturn(true);
```

```
                String requestPark=this.jsonStringFromObject(tmpForm);

                mockMvc.perform(
                                put("/park/update")
                                .contentType(MediaType.APPLICATION_JSON_UTF8)
                                .content(requestPark))
                .andExpect(status().isOk())
                .andDo(print()).andReturn();

                verify(parkService, times(1)).update(tmpForm);
                verifyNoMoreInteractions(parkService);
        }

        @Test
        public void updateFailureTest() throws Exception{
                Park park=findOnePark();
                ParkForm tmpForm=parkToForm(park);
                when(parkService.update(tmpForm)).thenReturn(false);
                String requestPark=this.jsonStringFromObject(tmpForm);

                mockMvc.perform(
                                put("/park/update")
                                .contentType(MediaType.APPLICATION_JSON_UTF8)
                                .content(requestPark))
                .andExpect(status().isInternalServerError())
                .andDo(print()).andReturn();

                verify(parkService, times(1)).update(tmpForm);
                verifyNoMoreInteractions(parkService);
        }

        @Test
        public void deleteSuccessTest() throws Exception {
                when(parkService.delete("1")).thenReturn(true);
                mockMvc.perform(delete("/park/delete/{id}", 1))
                .andExpect(status().isOk())
                .andDo(print()).andReturn();

                verify(parkService, times(1)).delete("1");
                verifyNoMoreInteractions(parkService);
        }

        @Test
        public void deleteFailureTest() throws Exception{
                when(parkService.delete("1")).thenReturn(false);
                mockMvc.perform(delete("/park/delete/{id}", 1))
                .andExpect(status().isNotFound())
                .andDo(print()).andReturn();

                verify(parkService, times(1)).delete("1");
                verifyNoMoreInteractions(parkService);
        }

        // 일부 삭제 함수들에 대해서 생략.
}
```

[mongoDB_JPA_Shared_Example_01 > src > main > test > net > kang > unit > controller > ParkControllerTest.java]

Controller 테스팅은 지난 시간에 작성했던 방법과 같기 때문에 간략하게 설명을 하고 넘어가겠다. 그렇지만 방금 전에 ResponseEntity로 반환을 하면서 각각 status를 비교를 해 주는 것을 보장을 하면서 이에 대한 실행 결과에 따라서 구분을 할 수 있어서 더욱 효과적인 REST API를 작성할 수 있는 점을 살렸다. Service 객체에는 어느 데이터를 반환을 하지만, Controller에서는 status와 객체를 반환을 하기 때문에 이에 대해서 비교를 하면서 어떻게 돌아가는지에 대해서 소스 코드를 통해 구상을 할 필요가 있다.

@Before

Mock 객체들에 대해서 이용을 할 수 있도록 설정을 한다. 그러면 여기서 Mock 객체는 Service가 되고, 실제로 작동을 하면서 확인을 하는 객체는 Controller가 된다.

@Test

원래 테스팅 함수는 많지만 대표적으로 쓰는 함수들에 대해서 테스팅을 할 때 나오는 결과를 토대로 작성을 하겠다.

findAllSuccessTest / findAllFailureTest / findByManageNoSuccessTest / findByManageNoFailureTest : Mock 데이터들에 대해서는 Service에서 반환된 공원 목록과 공원 데이터를 토대로 Controller에서 작동을 하면 MVC를 토대로 정상적으로 움직이는지에 대해 확인을 하도록 작성을 한다. 이 때 데이터의 존재 여부에 따라서 status는 OK, NoContent 2가지로 나오게 된다.

insertSuccessTest, insertFailureTest, updateSuccessTest, updateFailureTest : Mock 데이터들에 대해서 실제로 Form을 입력을 받아서 삽입과 삭제 실행 결과를 반환하고 난 후에 이에 맞춰서 status와 안내문을 반환을 하여 성공 / 실패 여부를 비교하게끔 만들었으며 데이터 추가 성공시에는 Created, 실패 시에는 Internal Server Error가 나오게끔 작성하였고, 데이터 갱신에서만 성공 시에 OK로 설정하였다.

deleteSuccessTest, deleteFailureTest : URL를 받아서 데이터를 삭제하는 경우에 나오는 안내문과 status를 통해 비교를 하여 삭제 성공 시에는 OK, 실패 시에는 Not Found가 나오게끔 하였다.

3. GitHub 주소 참조

이번 스터디에서 작성을 한 소스코드가 생각보다 길어져서 많이 다루지 못 하였다. 그렇지만 대부분 소스코드에 대해서 반영이 되기 때문에 이 사례에 대해서 GitHub를 통해 작성을 하겠다.ㄴ

https://github.com/tails5555/mongoDB_JPA_Shared_Example01