

MongoDB CRUD REST API

소프트웨어공학과 / 201332001 강인성 / hogu9401@gmail.com

0 초기 설정	2~3
1 Domain, Repository 생성	4~5
A. Domain 클래스 생성	4
B. Repository 인터페이스 생성	5
C. Repository 종류	5
2 Service 클래스 생성	6~7
A. Service 클래스 정리	6
B. Repository 함수의 차이점	7
C. Optional<T>	7
3 MongoConfig 생성	8~10
A. AbstractMongoConfiguration 정의	8
B. MongoConfig 생성하기	8~9
C. Example01Application	9~10
4 Controller 생성	11~12
A. MusicController	11~12
B. RequestMethod의 종류 파악하기	12
5 서버 실행 결과	13~17
6 GitHub 주소 참조	17

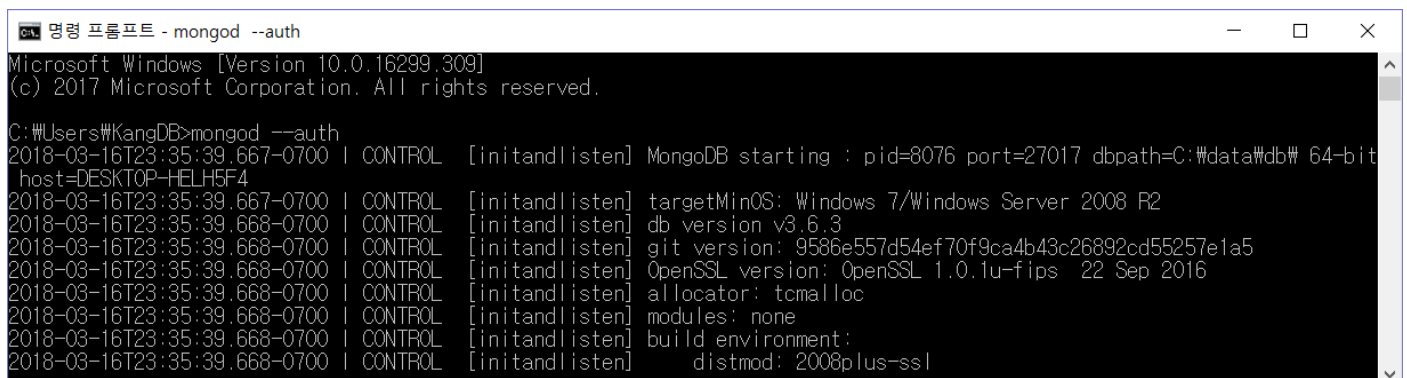
0. 초기 설정

우선 MongoDB와 Spring WEB MVC를 접목하기 위한 기본적인 대안으로는 아직까지는 cmd 창에 mongod 명령어를 이용해서 실행을 해 줘야 한다. 이를 진행하지 않고 현재 GitHub에 있는 파일들을 Import한다면 아래와 같은 오류가 걸린다. 이에 대해서는 어떻게 보면 MySQL에서는 JDBC 서버를 연동해서 사용을 하면 손쉽게 실행을 할 수 있었지만 MongoDB는 SQL JDBC 드라이버와는 달리 추가적으로 Configuration 클래스를 생성을 해 줘야 되기도 한다.

```
com.mongodb.MongoSocketOpenException: Exception opening socket
    at com.mongodb.connection.SocketStream.open(SocketStream.java:62) ~[mongo-java-driver-3.6.3.jar:na]
    at com.mongodb.connection.InternalStreamConnection.open(InternalStreamConnection.java:126) ~[mongo-java-driver-3.6.3.jar:na]
    at com.mongodb.connection.DefaultServerMonitor$ServerMonitorRunnable.run(DefaultServerMonitor.java:114) ~[mongo-java-driver-3.6.3.jar:na]
    at java.lang.Thread.run(Unknown Source) [na:1.8.0_151]
Caused by: java.net.ConnectException: Connection refused: connect
    at java.net.DualStackPlainSocketImpl.waitForConnect(Native Method) ~[na:1.8.0_151]
    at java.net.DualStackPlainSocketImpl.socketConnect(Unknown Source) ~[na:1.8.0_151]
    at java.net.AbstractPlainSocketImpl.doConnect(Unknown Source) ~[na:1.8.0_151]
    at java.net.AbstractPlainSocketImpl.connectToAddress(Unknown Source) ~[na:1.8.0_151]
    at java.net.AbstractPlainSocketImpl.connect(Unknown Source) ~[na:1.8.0_151]
    at java.net.PlainSocketImpl.connect(Unknown Source) ~[na:1.8.0_151]
    at java.net.SocksSocketImpl.connect(Unknown Source) ~[na:1.8.0_151]
    at java.net.Socket.connect(Unknown Source) ~[na:1.8.0_151]
    at com.mongodb.connection.SocketStreamHelper.initialize(SocketStreamHelper.java:59) ~[mongo-java-driver-3.6.3.jar:na]
    at com.mongodb.connection.SocketStream.open(SocketStream.java:57) ~[mongo-java-driver-3.6.3.jar:na]
    ... 3 common frames omitted
```

이 에러 메시지가 출력된 이후에는 30초가 지나면 MongoDB 소켓 Exception으로 인하여 또 하나의 에러 메시지가 실행이 되고 종료 된다. 초반부에 공부를 하는 경우에는 MongoDB 서버가 일단 실행을 하고 난 후에 작성을 하도록 하고 이에 대한 추후 해결 방안은 빠른 시일 이내로 알아와서 작성을 하겠다.

MongoDB 서버는 mongod --auth 명령어를 이용해서 실행을 하도록 한다. --auth를 써주면 타 사용자가 다른 데이터베이스를 접근하는데 있어서 보안성을 확실하게 심어줄 수 있다.



```
명령 프롬프트 - mongod --auth
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\KangDB>mongod --auth
2018-03-16T23:35:39.667-0700 | CONTROL | [initandlisten] MongoDB starting : pid=8076 port=27017 dbpath=C:\data\db\ 64-bit
host=DESKTOP-HELH5F4
2018-03-16T23:35:39.667-0700 | CONTROL | [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2018-03-16T23:35:39.668-0700 | CONTROL | [initandlisten] db version v3.6.3
2018-03-16T23:35:39.668-0700 | CONTROL | [initandlisten] git version: 9586e557d54ef70f9ca4b43c26892cd55257e1a5
2018-03-16T23:35:39.668-0700 | CONTROL | [initandlisten] OpenSSL version: OpenSSL 1.0.1u-fips 22 Sep 2016
2018-03-16T23:35:39.668-0700 | CONTROL | [initandlisten] allocator: tcmalloc
2018-03-16T23:35:39.668-0700 | CONTROL | [initandlisten] modules: none
2018-03-16T23:35:39.668-0700 | CONTROL | [initandlisten] build environment:
2018-03-16T23:35:39.668-0700 | CONTROL | [initandlisten] distmod: 2008plus-ssl
```

추가로 이 문장을 작성한 본인은 이번 스터디 노트에서 진행하는 music_test_01 데이터베이스에 대해서 kang 이라는 User를 생성을 해서 이 User에게 readWrite와 dbAdmin 권한을 추가하였다. 각 데이터베이스에 User 생성과 권한 부여를 하는 방법에 대해서는 MongoDB 기초 04번 노트를 구독해서 해결하길 바란다.

> Spring Starter Project 생성

Main Package : net.kang / Dependency : Lombok, MongoDB, Web, JPA로 설정하도록 한다.

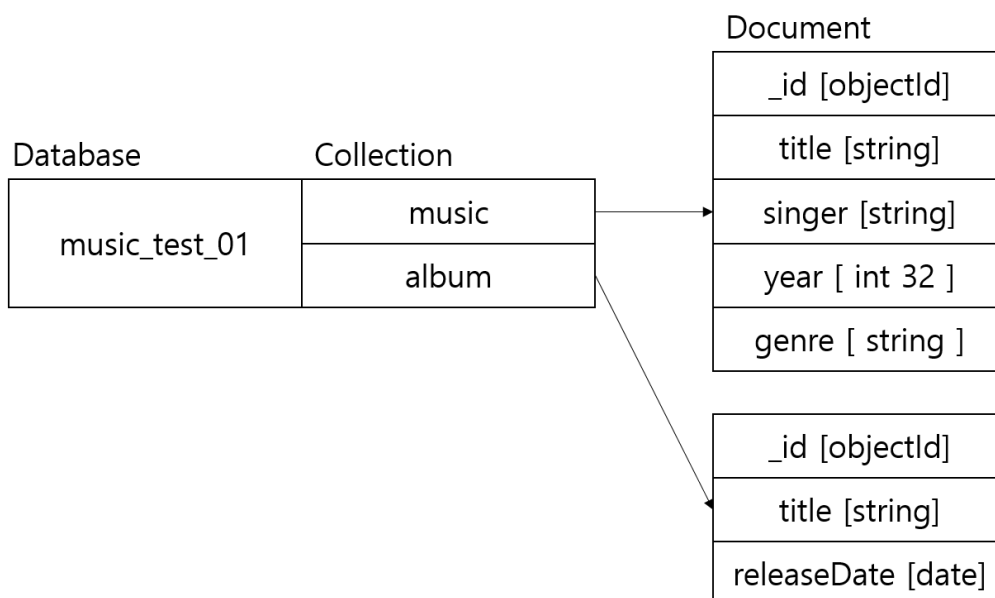
> application.properties 설정하기

Spring Web MVC에서 어느 데이터베이스를 접목을 할 때 우선적으로 하는 작업이 바로 application.properties 파일을 설정하는 것이다. 이 설정 파일은 과거에 MySQL를 접목했을 때와는 다르게 변수가 설정이 되어 있어서 알고 넘어가도록 하자.

```
spring.data.mongodb.host=127.0.0.1
spring.data.mongodb.port=27017
spring.data.mongodb.database=music_test_01
spring.data.mongodb.username=kang
spring.data.mongodb.password=-----
```

[mongodb_JPA_Start01 > src > main > resources > application.properties]

host, port는 MongoDB에서 현재 실행되고 있는 서버 번호를 입력하면 되는데 대부분 Host는 127.0.0.1로 작동을 하는 경우도 있고, localhost로 작성을 해도 무관하다. 그리고 Port는 27017로 쓰면 된다. MySQL는 3306이었는데 MongoDB에서 클러스터링 작업을 한다면 27017, 27018, ... 순으로 클러스터링 Port가 지정이 되기 때문에 주로 쓰는 27017로 설정하면 된다. 그리고 database는 이번에 써 볼 music_test_01를 쓰면 된다. music_test_01 데이터베이스의 구조는 아래 그림에 기재하겠다. 그리고 music_test_01에 대한 권한을 보호하기 위해서 사용자를 생성하여 username은 kang, password는 비밀번호 대로 작성을 하였다. 참고로 여기서는 쓰지 않았지만 현재 kang이라는 User는 한 데이터베이스에 대한 권한을 가진 일반 User로서 admin 데이터베이스를 통해 모든 Database를 총괄하는 관리자 User를 이용해서 접속을 하려면 spring.data.mongodb.authentication-database=admin이란 문장을 첨가하면 된다.



[그림] music_test_01 Database의 구조

1. Domain, Repository 생성

A. Domain 클래스 생성

Domain 클래스는 Spring Data MySQL를 기반으로 했을 때에는 테이블을 통하여 Entity 어노테이션을 사용했지만, Spring Data MongoDB를 기반으로 하는 경우에는 Document 어노테이션을 이용해야 한다. Music Class에서 작성한 Domain 클래스의 사례를 살펴보도록 하겠다.

```
package net.kang.domain;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import lombok.Data;

@Data
@Document(collection="music")
public class Music {
    @Id
    String id;
    String title;
    String singer;
    int year;
    String genre;
}
```

[mongoDB_JPA_Start01 > src > main > java > net > kang > domain > Music.java]

여기서 새로 추가된 개념은 바로 @Id 어노테이션과 @Document 어노테이션이다. 이에 대해서 어떻게 구성되는 지에 대해서 알아보도록 하겠다.

@Id 어노테이션 : 우리가 Spring Data MySQL에서는 javax.persistence.Id 어노테이션을 이용했다. 그렇지만 여기서는 Spring Data에서 제공하는 @Id 어노테이션을 이용한다. 이의 차이는 RDBMS(MySQL, Oracle, MS-SQL, MariaDB etc.)에서 쓰는 Primary Key와 NoSQL에서 쓰는 Primary Key의 차이로 보면 된다. Spring JPA에서 제공하는 기능들은 실제로 RDBMS의 입맛에 맞춰졌기 때문에 NoSQL(MongoDB, Redis etc.)에서는 Spring Data에서 MongoDB를 위해 이용할 수 있는 @Id 어노테이션으로 작성해야 한다. 그리고 MongoDB에서 쓰는 Primary Key는 String 형으로 작성을 해도 무관한데 MongoDB에서 관계형 Mapping을 할 때에는 BSON Type인 ObjectId를 사용하는 방안 에 대해서는 추후에 작성하도록 하겠다.

@Document 어노테이션 : Document들은 Collection 내부에 있다. 그래서 Collection의 이름을 작성해서 Domain 클래스를 연동하여 MongoDB와의 DTO(Data Transfer Object)의 역할을 해줄 수 있도록 하기 위해 이 어노테이션을 작성한다. Spring Data MySQL에서는 @Entity(table="")와 같은 개념으로 생각하면 되겠다.

이외에 Album 클래스에 대해서는 마지막에 기재된 GitHub에서 참고를 하길 바란다.

B. Repository 인터페이스 생성

```
package net.kang.repository;

import java.util.List;

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

import net.kang.domain.Music;

public interface MusicRepository extends MongoRepository<Music, String>{
    List<Music> findByYearBetween(int year1, int year2);
    List<Music> findBySinger(String singer);
}
```

[mongodb_JPA_Start01 > src > main > java > net > kang > repository > MusicRepository.java]

Spring Data MySQL에서와 같은 방법으로 Repository는 인터페이스로 생성이 가능하다. 그렇지만 몇 가지 차이가 있다면 바로 JpaRepository를 이용한 것이 아니라 MongoRepository를 이용한 점과 제네릭이 <Music, String>을 이용했다는 점이다. 우리는 각각 단일 Collection에 대한 Primary Key를 Domain 클래스 내부에서 String으로 생성을 하였기 때문에 이에 맞춰서 작성을 했다. 물론 ObjectId로 작성하는 경우에도 이로 작성을 하면 된다. 그리고 MongoRepository에서도 천만다행히 Query Creation 기능과 Query Language(JPQL) 기능을 제공할 한다. 이 예제에서는 연도 사이에 발간 된 음악들에 대한 목록 출력(where year Between ? and ?), 가수 이름으로 찾는 음악 목록들(where singer=?)에 대하여 Query Creation을 접목해서 작성을 해 뒀다. Query Language에 대해서는 MongoDB에서 쓰는 질의를 작성해서 이용하면 된다.

C. Repository 종류

Repository의 종류는 Spring Data MySQL와 Spring Data MongoDB를 토대로 이용할 수 있는 Repository를 기반으로 나뉘어서 알아보도록 하겠다. 여태 동안은 Spring Data MySQL을 이용했다면 JpaRepository에 대해서는 이미 인지하고 있을 것이다. 그렇지만 실제로는 CrudRepository, PagingAndSortingRepository 등등이 존재한다. 이에 대해서 간략하게 어떤 역할을 하는지에 대해서 인지를 하고 넘어가도록 하자.

- CrudRepository : 말 그대로 CRUD(Create, Read(Select), Update, Delete)를 하기 위한 목적으로 만든 Repository. 여기서 CRUD를 하기 위한 기능을 제공하는 아주 기본적인 기능만 제공한다.
- PagingAndSortingRepository : 여기서는 현재 데이터 목록들을 Pagination을 이용해서 정리하기 위한 목적으로 만든 Repository. CrudRepository에 존재하는 기능들을 포함해서 제공한다.
- JpaRepository : Spring Data MySQL에서 이미 다뤄서 이해할 수 있지만, RDBMS에 대해서 JPA를 기반으로 구축된 Repository로 인지할 수 있다. 위에 있는 Repository들의 기능들을 모두 제공할 한다.
- MongoRepository : Spring Data MongoDB에서 주로 다룰 Repository로 볼 수 있다. 물론 MongoDB에서도 CrudRepository, PagingAndSortingRepository를 사용해도 상관은 없다.

이 4가지를 잠깐 살펴봤는데 제공할 하는 패키지들이 다르다. CrudRepository, PagingAndSortingRepository는 Spring Data에서 제공할 하고, JpaRepository는 RDBMS에 대해 JPA를 이용하기 위해 제공할 하고, MongoRepository는 Spring Data MongoDB에서 제공할 한다. 물론 RDBMS이든 NoSQL이든 CrudRepository, PagingAndSortingRepository를 사용해도 상관은 없다. 또한 Redis에서는 데이터들에 대해서 정렬을 하는 경우는 크게 없기 때문에 CrudRepository를 대표적으로 활용한다고 한다.

2. Service 클래스 생성

Service 클래스는 Repository에서 제공하는 함수들에 대해서 안전하게 접근을 하기 위한 오작교(烏鵲橋)와 같은 역할을 한다. 본래는 DAO 클래스를 생성해서 각각 접근을 하는 것이 옳은 방법이지만 여기서는 간단하게 REST API를 만들어 보는 과정에 대해서만 다뤄볼 예정이기 때문에 이에 대해서는 JUnit에서 확실하게 반영을 하겠다. 실제로 작성한 예제들 중에서 MusicService 클래스에 대해서만 다루고 넘어가겠다. AlbumService는 GitHub를 참고하면 된다.

A. Service 클래스 생성

```
package net.kang.service;

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

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

import net.kang.domain.Music;
import net.kang.repository.MusicRepository;

@Service
public class MusicService {
    @Autowired MusicRepository musicRepository;
    public List<Music> findAll(){
        return musicRepository.findAll();
    }
    public Optional<Music> findById(String id){
        return musicRepository.findById(id);
    }
    public List<Music> findByYearBetween(int year1, int year2){
        return musicRepository.findByYearBetween(year1, year2);
    }
    public List<Music> findBySinger(String singer){
        return musicRepository.findBySinger(singer);
    }
    public void insert(Music music) {
        musicRepository.insert(music);
    }
    public void update(Music music) {
        musicRepository.save(music);
    }
    public void delete(String id) {
        musicRepository.deleteById(id);
    }
}
```

[mongoDB_JPA_Start01 > src > main > java > net > kang > service > MusicService.java]

단일 Collection에 대한 CRUD를 간략하게 작성을 하였기 때문에 여기서는 복잡한 논리를 작성할 필요 없이 간략하게 MusicRepository에 있는 함수들로 정리를 하였다. 이름 그대로 모든 음악 찾기, 하나의 음악 찾기, 연도 값에 해당되는 음악 목록, 가수 이름으로 음악 목록 출력, 음악 추가, 음악 갱신, 음악 삭제 7개의 메소드로 구성된다.

B. Repository 함수의 차이점

JpaRepository에서 기본적으로 제공한 함수와 MongoRepository에서 기본적으로 제공한 함수에 대해서는 차이가 존재한다. JpaRepository에서는 findAll(), findOne(id 값), save(객체), delete(id 값) 이렇게 있는데 MongoRepository에서는 아래와 같이 비교를 해서 이용을 해야 한다.

Spring Data MySQL + JPA T는 Entity 객체			Spring Data MongoDB T는 Document 객체	
반환형	함수 이름	목적	함수 이름	반환형
List<T>	findAll()	모든 데이터 조회	findAll()	List<T>
T	findOne(id 값)	하나의 데이터 조회	findById(_id 값)	Optional<T>
void	save(T)	데이터 추가	insert(T)	void
void	id가 0이면 추가, 이외에는 갱신	데이터 갱신	save(T)	void
void	delete(id 값)	데이터 삭제	deleteById(_id 값)	void
void	deleteAll()	모든 데이터 삭제	deleteAll()	void
long	count()	총 데이터의 수	count()	long

C. Optional<T>

Optional<T> 클래스는 findById(_id 값) 함수를 통해서 데이터가 존재하는 여부에 대해서 출력을 할 때 쓰는 자료형이다. 하지만 Java 8 버전의 성향을 인지하기 위해서는 이에 대한 개념도 약간 알아 두고 가면 약이 되기 때문에 우선은 이 개념에 대해서는 내용이 너무 길어져서 짧게 설명을 하고 넘어가도록 하겠다.

Java에서는 NullPointerException 에러가 언제 발생할지 모른다. 그래서 Java 언어를 항상 쓰는 사람들은 이 NullPointerException에 대해서 어떻게 구상을 할지에 대해 의문을 가지게 되는데, 예를 들어 간단한 Integer 객체와 String 객체에 대해서는 간단하게 null 여부를 == null 문장을 이용해서 따지면 됐었는데 객체 속에 객체의 패턴(즉, Wrapper 객체인 경우)을 가지는 경우에는 어떻게 null 여부를 감당해야 할까?

바로 Optional<T>를 이용해서 반복적인 null 체크를 없애기 위해서 이를 대체하였다. 물론 기술적으로는 이를 그대로 적용하면 그만이지만 null 체크를 여러 번 하는 함수를 눈에는 보이지 않게 Optional<T> 클래스가 알아서 비교를 해 주도록 하여 NullPointerException에 대해서 그나마 해결이 가능하도록 만들어준다. MongoDB에서 _id의 값을 통해서 하나의 객체를 조회하는 경우에 데이터가 존재하는 여부(즉 null이 아닌 경우)를 따질 때 이를 이용해서 반환을 한다면 Null에 대한 문제를 원활하게 해결이 가능하기 때문에 좋은 사례로 볼 수 있다.

우리가 살펴보는 예제에서는 orElse(), get() 함수의 차이점만 다룰 듯 하지만 더욱 자세한 내용은 차주에 공부할 수 있는 시간을 내서 GitHub에 예시를 작성을 하여 자세하게 다뤄볼 예정이다. 추후에 스터디 노트에 기타 내용에 첨부를 해서 어떠한 개념인지에 대해서 간단한 사례를 토대로 적어서 실제로 프로젝트 진행할 때 도움이 될 수 있으면 좋겠다.

3. MongoConfig 클래스 생성

A. AbstractMongoConfiguration 정의

AbstractMongoConfiguration은 MongoDB와 Spring을 연동하기 위한 Configuration을 접근하기 쉽도록 하기 위한 기본적인 설정을 파악할 수 있도록 유도하는 Configuration으로 추상 메소드로 인식할 수 있다. 우리가 작성한 MongoConfig에서는 다음과 같은 함수들을 작성하였다.

- String getDatabaseName() : application.properties를 통해서 일반 User가 접근할 Database를 반환하도록 만들었다. 이 함수는 AbstractMongoConfiguration의 추상 메소드 중 일부로 접근 Database의 이름을 반환한다.
- MongoClient mongoClient() : 과거까지는 Mongo mongo() 함수이지만, 버전이 올라가면서 mongoClient() 함수로 대체하게 되었다. 이는 클라이언트에서 Mongo 서버를 접근하는 기록을 properties를 통해 정리를 하여 MongoClientFactory를 통해서 재정리를 하기 위하여 쓰는 추상 메소드로 보면 된다. 여기서 Mongo DB 사용자 계정을 정리를 해 주고 넘어가야 하는데 여기서 최종적으로 이용을 하는 것이 아니라 MongoTemplate를 통해서 쓰이게 된다.
- MongoClientFactory mongoDbFactory() : MongoTemplate에서 필요로 한 DB Factory를 생성하기 위하여 작성함. 이도 마찬가지로 추상 클래스로 볼 수 있다.
- MongoMappingContext mongoMappingContext() : 여기서 쓰인 MongoDB와의 Collection Domain 클래스들이 MongoDB에 현존하는 Collection과의 매핑을 할 수 있도록 이를 반환하여 차질이 생기지 않게 만든 함수로 이도 마찬가지로 추상 클래스이다.
- MongoTemplate mongoTemplate() : 위에서 순서대로 처리된 MongoClientFactory와 MongoMappingContext를 최종적으로 확인을 해서 이를 이용해서 현재 클라이언트와 MongoDB 서버와의 연계를 도와주는 함수로 보면 된다. 여기서는 MappingMongoConverter를 이용을 해서 추가적인 작업을 또한 해 준다.

B. MongoConfig 생성

```
package net.kang.config;

import java.util.Arrays;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.mongodb.MongoDbFactory;
import org.springframework.data.mongodb.config.AbstractMongoConfiguration;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.data.mongodb.core.SimpleMongoDbFactory;
import org.springframework.data.mongodb.core.convert.DefaultMongoTypeMapper;
import org.springframework.data.mongodb.core.convert.MappingMongoConverter;
import org.springframework.data.mongodb.core.mapping.MongoMappingContext;

import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;
import com.mongodb.ServerAddress;

@Configuration
public class MongoConfig extends AbstractMongoConfiguration{

    @Value("${spring.data.mongodb.host}")
    private String mongoHost;
```



```

@Value("${spring.data.mongodb.port}")
private Integer mongoPort;

@Value("${spring.data.mongodb.database}")
private String mongoDatabase;

@Value("${spring.data.mongodb.username}")
private String userName;

@Value("${spring.data.mongodb.password}")
private String password;

@Override
protected String getDatabaseName() {
    return mongoDatabase;
}

@Override
public MongoClient mongoClient(){
    MongoCredential credential = MongoCredential.createCredential(userName, mongoDatabase,
password.toCharArray());
    return new MongoClient(new ServerAddress(mongoHost, mongoPort),
Arrays.asList(credential));
}

@Override
public MongoDBFactory mongoDbFactory() {
    return new SimpleMongoDbFactory(mongoClient(), getDatabaseName());
}

@Override
public MongoMappingContext mongoMappingContext() throws ClassNotFoundException {
    return super.mongoMappingContext();
}

@Override
public MongoTemplate mongoTemplate() throws Exception{
    MappingMongoConverter converter=new MappingMongoConverter(mongoDbFactory(),
mongoMappingContext());
    converter.setTypeMapper(new DefaultMongoTypeMapper(null));
    MongoTemplate mongoTemplate=new MongoTemplate(mongoDbFactory(), converter);
    return mongoTemplate;
}
}

```

[mongoDB_JPA_Start01 > src > main > java > net > kang > config > MongoConfig.java]

생각보다 소스 코드가 길어지는데 여기서는 간단하게 주의를 해야 할 사항들에 대해서 작성을 하겠다.

@Value 어노테이션 : application.properties에 현존하는 요소들에 대해서 Config 멤버 변수에 기재를 하도록 도와 주는 어노테이션으로 볼 수 있다.

MappingMongoConverter converter : 여기서 converter.setTypeMapper(new DefaultMongoTypeMapper(null)); 문장을 쓴 이유가 이를 빼놓고 REST API에 Insert, Update 작업을 한다면 MongoDB 서버 자체에서 _class Field를 통해서 우리가 작성한 Domain 클래스의 패키지 주소를 저장할 하기 때문에 이를 방지하고자 추가를 하였다.

C. Example01Application

```
package net.kang;

import javax.sql.DataSource;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.data.jpa.JpaRepositoriesAutoConfiguration;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.boot.jdbc.DataSourceBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.data.mongodb.repository.config.EnableMongoRepositories;

@EnableAutoConfiguration(exclude = {JpaRepositoriesAutoConfiguration.class})
@EnableMongoRepositories(basePackages = "net.kang.repository")
@EntityScan(basePackages = "net.kang.domain")
@ComponentScan(basePackages = "net.kang")
@SpringBootApplication
public class Example01Application {
    public static void main(String[] args) {
        SpringApplication.run(Example01Application.class, args);
    }

    @Bean
    @ConfigurationProperties(prefix = "spring.data.db-main")
    public DataSource mainDataSource() {
        return DataSourceBuilder.create().build();
    }

    @Bean
    @ConfigurationProperties(prefix = "spring.data.db-log")
    public DataSource contractDataSource() {
        return DataSourceBuilder.create().build();
    }
}
```

[mongoDB_JPA_Start01 > src > main > java > net > kang > Example01Application.java]

MongoConfig를 통해 설정을 완료했다면 이를 기반으로 작동을 하는데 Example01Application에서 해결을 해야 하는 문제가 바로 AutoConfiguration 문제이다. 우리가 작성한 properties는 spring.datasource를 이용한 것이 아니라 spring.data.mongo를 이용해서 작성을 하였기 때문에 DataSource가 없다는 예러가 뜨기 때문에 이를 해결하기 위해 @EnableAutoConfiguration에서 JpaRepositoriesAutoConfiguration을 빼고(이는 MongoRepository에 문제가 없도록 하기 위해서이다.) 추가로 작성을 하였고 @Bean을 통해 추가한 DataSource를 이용해서 이에 대한 문제를 해결 하도록 추가로 작성을 하였다. 그리고 @EnableMongoRepositories, @EntityScan, @ComponentScan 어노테이션은 각각 Repository, Entity(Document Domain 클래스), Component들에 대해서 패키지 주소를 입력해서 지정해두기 위해 추가한 문장이다.

4. Controller 생성

여기서는 어떻게 동작을 하는지에 대해서 간략하게 정리하는 목적 차에 MusicController에 대해서만 설명을 하겠다. AlbumController도 같은 원리이기 때문에 이에 대한 자세한 소스 코드는 GitHub에서 참고하길 바란다. 최종적으로 Configuration 작업과 Service 객체 생성 작업까지 모두 완료를 하였으니 이제는 Controller에서 REST API를 연동할 수 있는 기능을 추가하도록 하겠다.

A. MusicController

```
package net.kang.controller;

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

import org.springframework.beans.factory.annotation.Autowired;
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.Music;
import net.kang.service.MusicService;

@RestController
@CrossOrigin
@RequestMapping("music")
public class MusicController {
    @Autowired MusicService musicService;
    @RequestMapping("findAll")
    public List<Music> findAll(){
        return musicService.findAll();
    }
    @RequestMapping("findOne/{id}")
    public Music findOne(@PathVariable("id") String id) {
        Optional<Music> result=musicService.findById(id);
        return result.orElse(new Music());
    }
    @RequestMapping("findBySinger/{singer}")
    public List<Music> findBySinger(@PathVariable("singer") String singer){
        return musicService.findBySinger(singer);
    }
    @RequestMapping("findByYearBetween/{before}/{after}")
    public List<Music> findByYearBetween(@PathVariable("before") int before,
    @PathVariable("after") int after){
        return musicService.findByYearBetween(before, after);
    }
    @RequestMapping(value="insert", method=RequestMethod.POST)
    public void insert(@RequestBody Music music) {
        musicService.insert(music);
    }
    @RequestMapping(value="update", method=RequestMethod.PUT)
    public void update(@RequestBody Music music) {
        musicService.update(music);
    }
    @RequestMapping(value="delete/{id}", method=RequestMethod.DELETE)
```

```

    public void delete(@PathVariable("id") String id) {
        musicService.delete(id);
    }
}

```

[mongoDB_JPA_Start01 > src > main > java > net > kang > controller > MusicController.java]

여기서는 각각 MusicService 클래스에서 생성했던 함수들에 대해서 적용을 할 수 있도록 RequestMapping을 통해 이 URL로 접근을 하도록 설정을 하였고, 간략하게 실행을 할 수 있기 때문에 큰 설명 없이 넘어가도록 하는 대신에 Optional 함수에 대해서 잠깐 언급을 하고 넘어가겠다.

- Optional<T>에서 쓰이는 간단한 함수 사례

```

Optional<Music> result=musicService.findById(id);
return result.orElse(new Music());

```

이처럼 작성을 한 이유가 _id를 통하여 데이터를 조회하는데 결과가 null인 경우에는 아무 내용도 없는 Music 객체를 생성하도록 작성을 한 함수가 바로 orElse(T) 함수이다. 이 함수에 대해서는 만일 result에 있는 객체가 존재한다면 이를 반환하고, null인 경우에는 새 Music 객체를 생성함으로써 NullPointerException을 예방하기 위한 문장으로 볼 수 있다. Optional<T>에서 쓰이는 함수는 일단 3가지만 알아 두고 넘어가자.

- boolean isPresent() : 내부 객체가 null인지 확인함.
- T get() : 내부 객체를 반환하되 null도 나올 수 있음.
- T orElse(T) : 내부 객체를 반환하지만 null인 경우에는 기본 객체로 생성을 한다.

B. RequestMethod의 종류 파악하기

과거에는 RequestMethod를 GET, POST 2가지를 이용했지만 CRUD 별로 권장을 하는 RequestMethod가 존재한다. REST API를 형성하는데 있어서 이에 대해서는 이러한 방식을 이용해서 작성을 한다는 사실을 인지하고 넘어가도록 하자.

작업 종류	RequestMethod
데이터 목록을 불러오는 경우	RequestMethod.GET(생략 가능)
데이터를 새로 추가하는 경우	RequestMethod.POST
데이터를 갱신하는 경우	RequestMethod.PUT
데이터를 삭제하는 경우	RequestMethod.DELETE

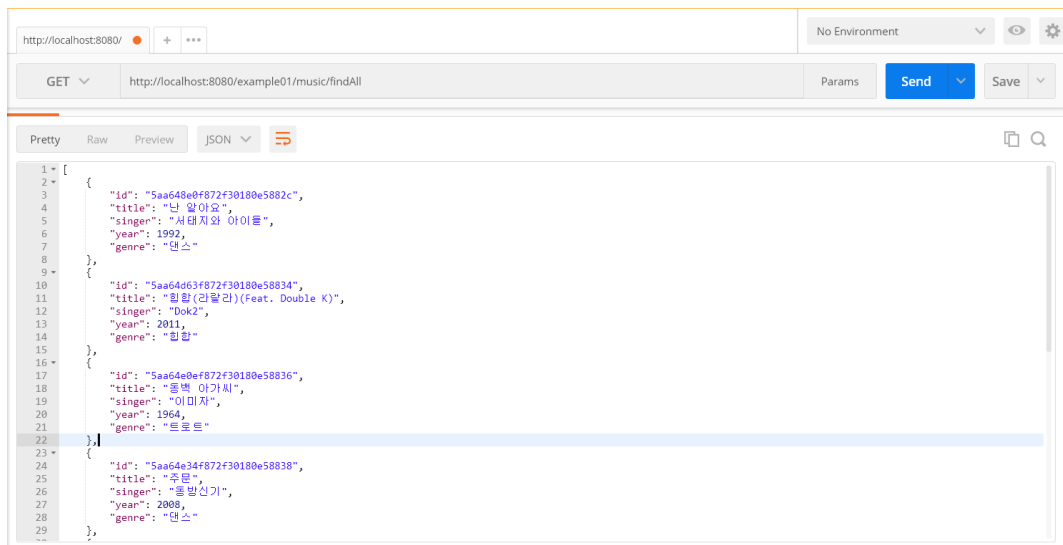
5. 서버 실행 결과

서버를 실행하기 위해서는 Postman Application을 다운로드 받아서 REST API 통신이 잘 이뤄지고 있는지 확인을 할 수가 있다. 물론 GET 방식은 Chrome 창 하나만 있어도 무관하지만, POST, PUT, DELETE에 대해서는 보장을 하지 못한다. Postman 이용 방법은 의외로 간단하고 REST API를 개발하는데 큰 도움을 주니 참고해서 이용을 하길 권한다. 그리고 실행 결과는 MusicController를 기반으로 작동 결과를 선보이겠다. AlbumController에 대해서는 Date 객체에 대해서 문제가 약간 있어서 해결을 해야 하기 때문이다.

-> Postman 설치 방법 및 사용 안내서

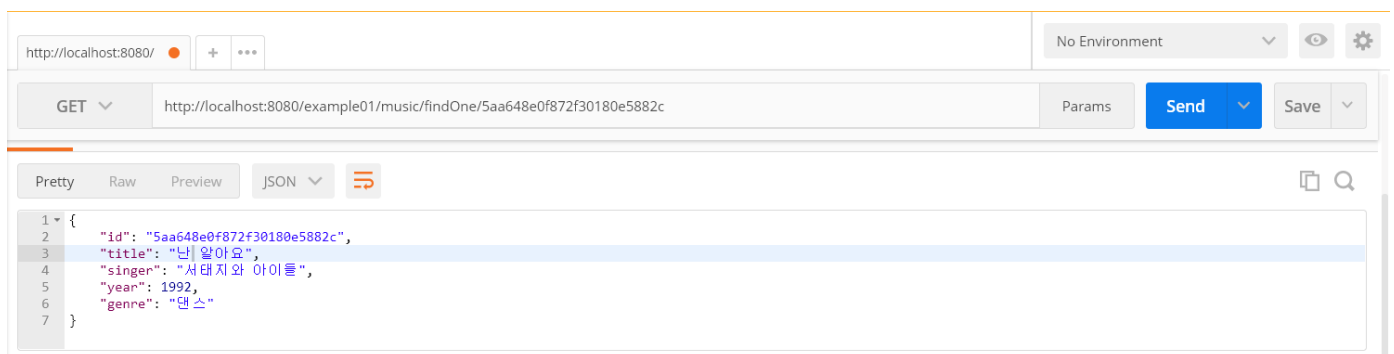
<http://devkyeol.tistory.com/entry/Postman-%EA%B0%9C%EC%9A%94-%EC%84%A4%EC%B9%98-%EC%82%AC%EC%9A%A9%EB%B2%95-%ED%99%9C%EC%9A%A9-%EB%B0%A9%EB%B2%95>

(1) 모든 음악들을 조회한 결과



정상적으로 모든 음악들이 나온다. URL은 example01/music/findAll이다.

(2) 음악 제목이 난 알아요 인 음악을 조회하기(_id를 이용한 조회 결과)



난 알아요 의 _id 값은 5aa648e0f872f30180e5882c이다. _id 값이 Document들 중에 없다면 빈 Music이 나온다.

URL은 example01/music/findOne/5aa648e0f872f30180e5882c 이다.

(3) 가수 이름이 Dok2인 음악 목록 출력하기

REST client interface showing a GET request to `http://localhost:8080/example01/music/findBySinger/Dok2`. The response is a JSON array of two music items:

```
[
  {
    "id": "5aa64d63f872f30180e58834",
    "title": "힙합(라달라)(Feat. Double K)",
    "singer": "Dok2",
    "year": 2011,
    "genre": "힙합"
  },
  {
    "id": "5aac9f33c99ff90450e7a5cd",
    "title": "홀쳐 (Feat. Double K)",
    "singer": "Dok2",
    "year": 2009,
    "genre": "힙합"
  }
]
```

List에 대해서 `NullPointerException`에 대해서는 안심을 해도 된다. 물론 Dok2 뿐만 아니라 리쌍, 이미자, 동방신기 등 한글 이름으로 작성을 하여도 정상적으로 작동된다.

URL은 `example01/music/findBySinger/Dok2` 이다.

(4) 2000년대(2000년~2009년) 음악 목록 출력하기

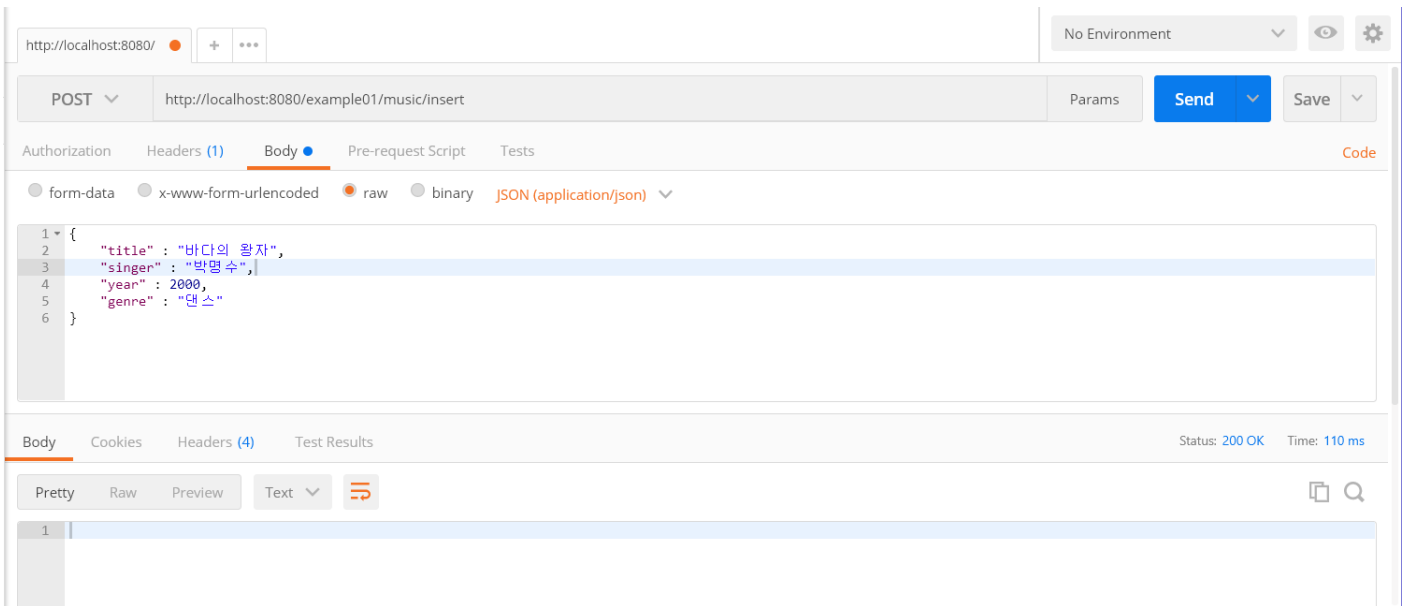
REST client interface showing a GET request to `http://localhost:8080/example01/music/findByYearBetween/2000/2009`. The response is a JSON array of three music items:

```
[
  {
    "id": "5aa64e34f872f30180e58838",
    "title": "주문",
    "singer": "동방신기",
    "year": 2008,
    "genre": "댄스"
  },
  {
    "id": "5aa64e87f872f30180e5883b",
    "title": "잊을게",
    "singer": "VB",
    "year": 2003,
    "genre": "락"
  },
  {
    "id": "5aac9ef5c99ff90450e7a5cc",
    "title": "광대 (Feat. BMK)",
    "singer": "리쌍",
    "year": 2005,
    "genre": "발라드"
  }
]
```

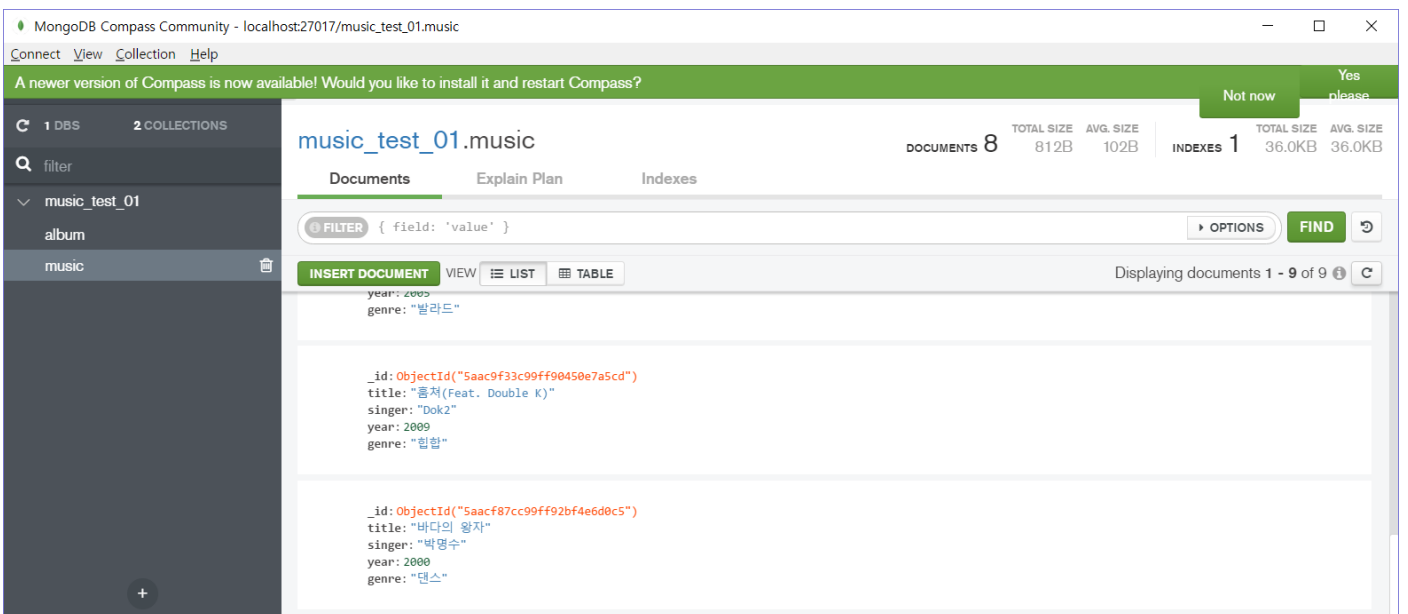
Between을 이용해서 나온 결과는 위와 같다.

URL은 `example01/music/findByYearBetween/2000/2009` 이다.

(5) 박명수의 바다의 왕자 노래를 추가하기



Insert는 당연히 POST로 작성해야 한다. 추가할 Music에 대해서는 굳이 id의 값을 안 써 줘도 상관은 없지만 각 Field 별로 큰 따옴표로 구분을 해 주도록 하자. 이처럼 추가를 하면 MongoDB Compass를 이용해서 실행을 한다면 이처럼 새로 추가가 된다. URL은 `example01/music/insert` 이다. Body를 JSON 형으로 하는 건 기본으로 잊지 말고 진행하도록 한다.

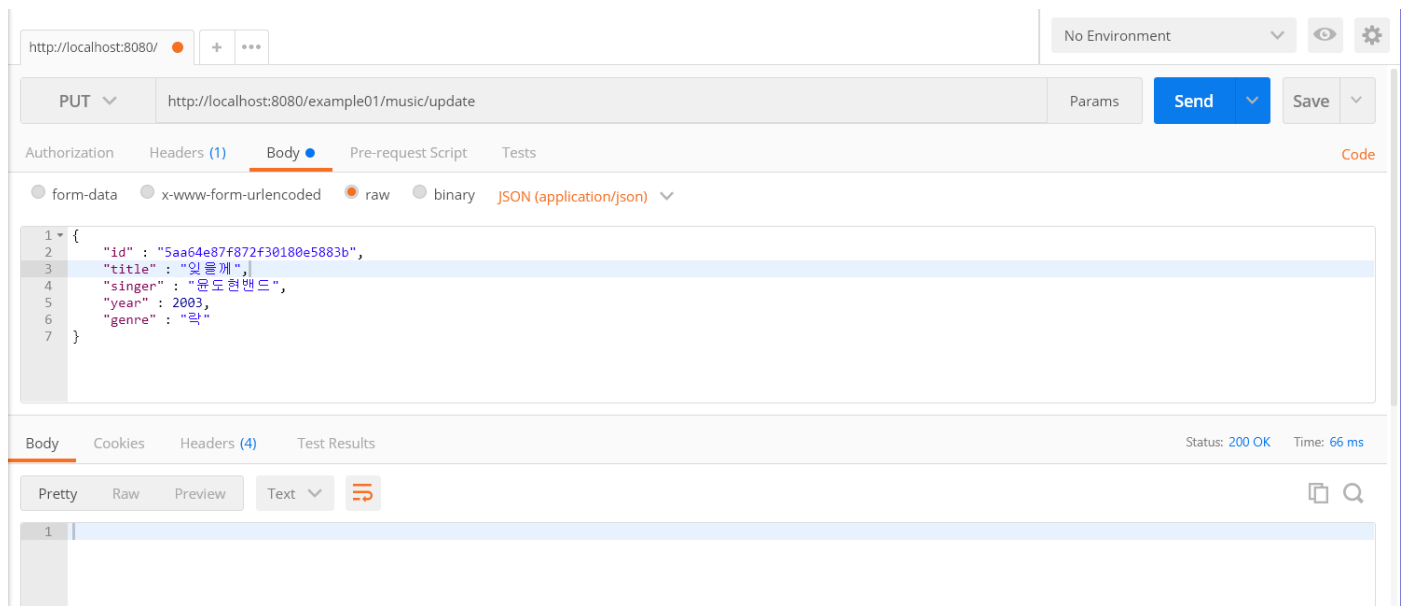


(6) YB의 잊을게 노래 정보 수정하기

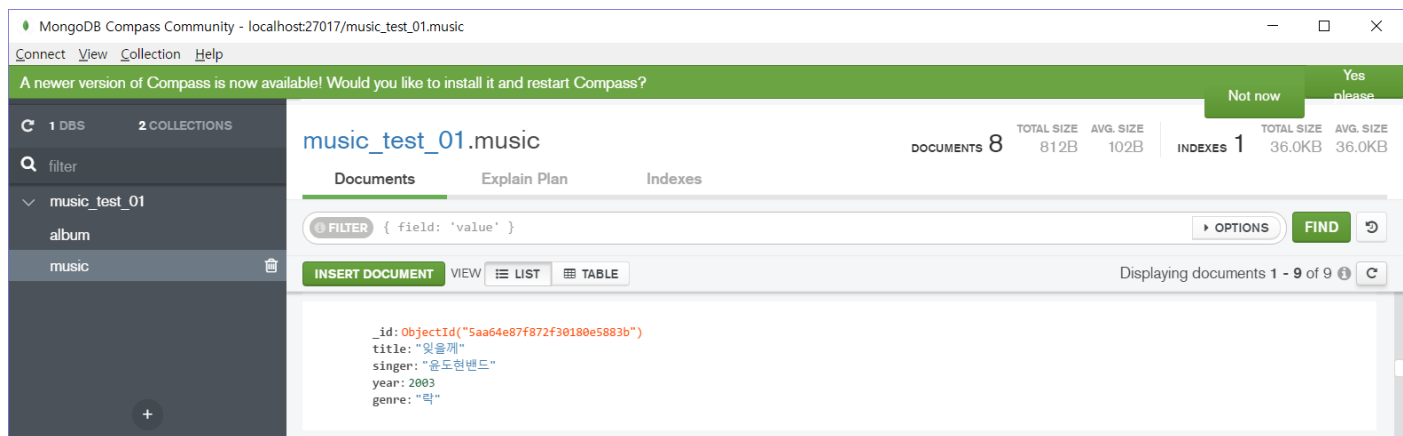
Music Collection 내부에 있는 잊을게의 Document는 다음과 같다.

```
{
  "_id": ObjectId("5aa64e87f872f30180e5883b"),
  "title": "잊을게",
  "singer": "YB",
  "year": 2003,
  "genre": "락"
}
```

YB를 윤도현밴드로 고치도록 해 보겠다.

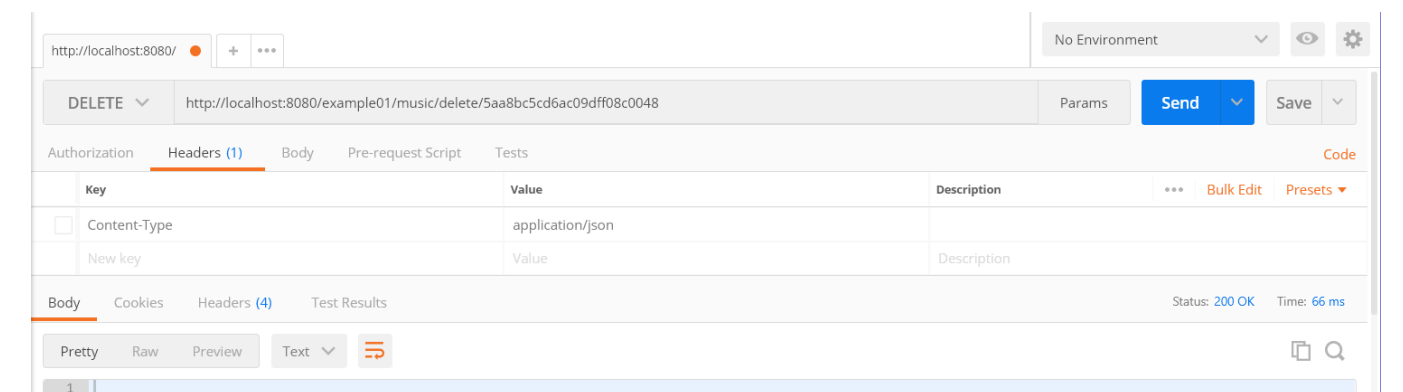


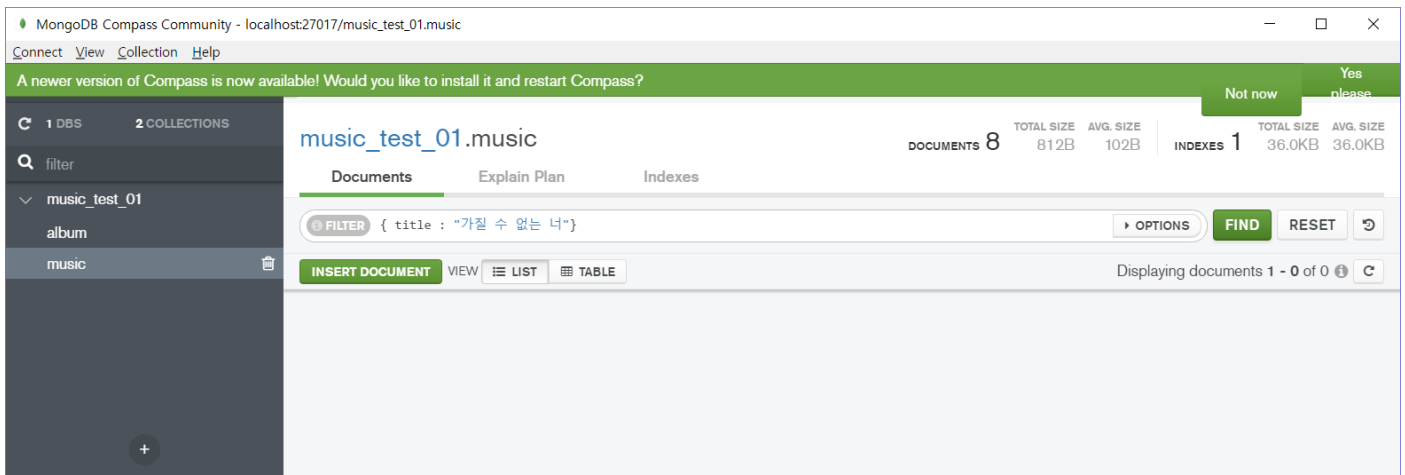
이처럼 작성을 하고 난 후에 MongoDB Compass에서 확인하면 정상적으로 수정이 되었음을 인지할 수 있다. URL은 `example01/music/update` 이다.



(7) बैं크의 가질 수 없는 너 음악을 삭제하기

뱅크의 가질 수 없는 너의 `_id` Field 값은 `5aa8bc5cd6ac09dff08c0048` 이다. 이를 기반으로 음악을 삭제하는 URL을 `example01/music/delete/5aa8bc5cd6ac09dff08c0048` 로 맞추고 진행 결과를 MongoDB Compass에서 뜨게끔 한다면 가질 수 없는 너 란 노래가 없어지게 된다.





실행 결과는 아무 Document도 나오지 않는 것으로 종결 된다.

6. GitHub 주소 참조

자세한 소스 코드는 아래 GitHub 주소를 통해서 소스코드를 참고하면 된다.

https://github.com/tails5555/mongoDB_JPA_Start01

[Spring JPA + MongoDB REST API 기본]

[출처]

소스코드 참고 페이지

- <http://scrumbucket.org/tutorials/mongodb-spring-data-queries/part-1-setting-up-mongodb-spring-data-and-junit/>
- <http://scrumbucket.org/tutorials/mongodb-spring-data-queries/part-2-setting-up-some-spring-queries/>
- <https://stackoverflow.com/questions/16846763/spring-mvc-controller-with-date>

0. 초기 설정

<https://springframework.guru/configuring-spring-boot-for-mongo/>

1. Domain, Repository 클래스 생성

A. Domain 클래스 생성

- <https://stackoverflow.com/questions/39643960/whats-the-difference-between-javax-persistence-id-and-org-springframework-data-id-annotation> / @Id Annotation의 차이

C. Repository 인터페이스 종류

- <https://stackoverflow.com/questions/14014086/what-is-difference-between-crudrepository-and-jparepository-interfaces-in-spring>

2. Service 클래스 생성

B. Repository 함수의 차이점

- <https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html>

C. Optional<T>

- <http://multifrontgarden.tistory.com/131>

3. MongoConfig 클래스 생성

A. AbstractMongoConfiguration 정의

- <https://docs.spring.io/spring-data/mongodb/docs/current/api/org/springframework/data/mongodb/config/AbstractMongoConfiguration.html>
- <https://lishman.io/spring-data-mongodb-configuration>

B. MongoConfig 생성

- https://www.mkyoung.com/mongodb/spring-data-mongodb-remove-_class-column/

C. Example01Application

- <https://stackoverflow.com/questions/25108389/using-multiple-datasources-with-spring>

5. 서버 실행

<http://devkyeol.tistory.com/entry/Postman-%EA%B0%9C%EC%9A%94-%EC%84%A4%EC%B9%98-%EC%82%AC%EC%9A%A9%EB%B2%95-%ED%99%9C%EC%9A%A9-%EB%B0%A9%EB%B2%95>