

Booting Spring Data REST

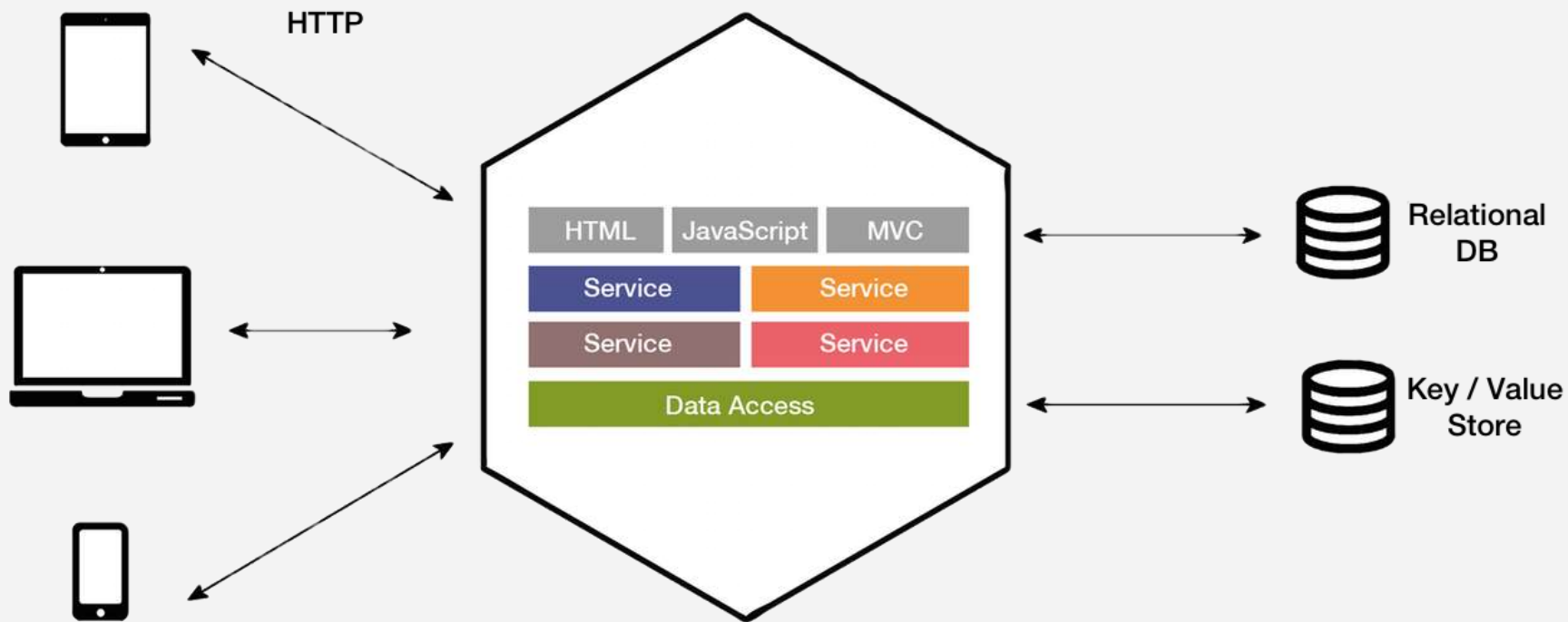
이경원(woniper)

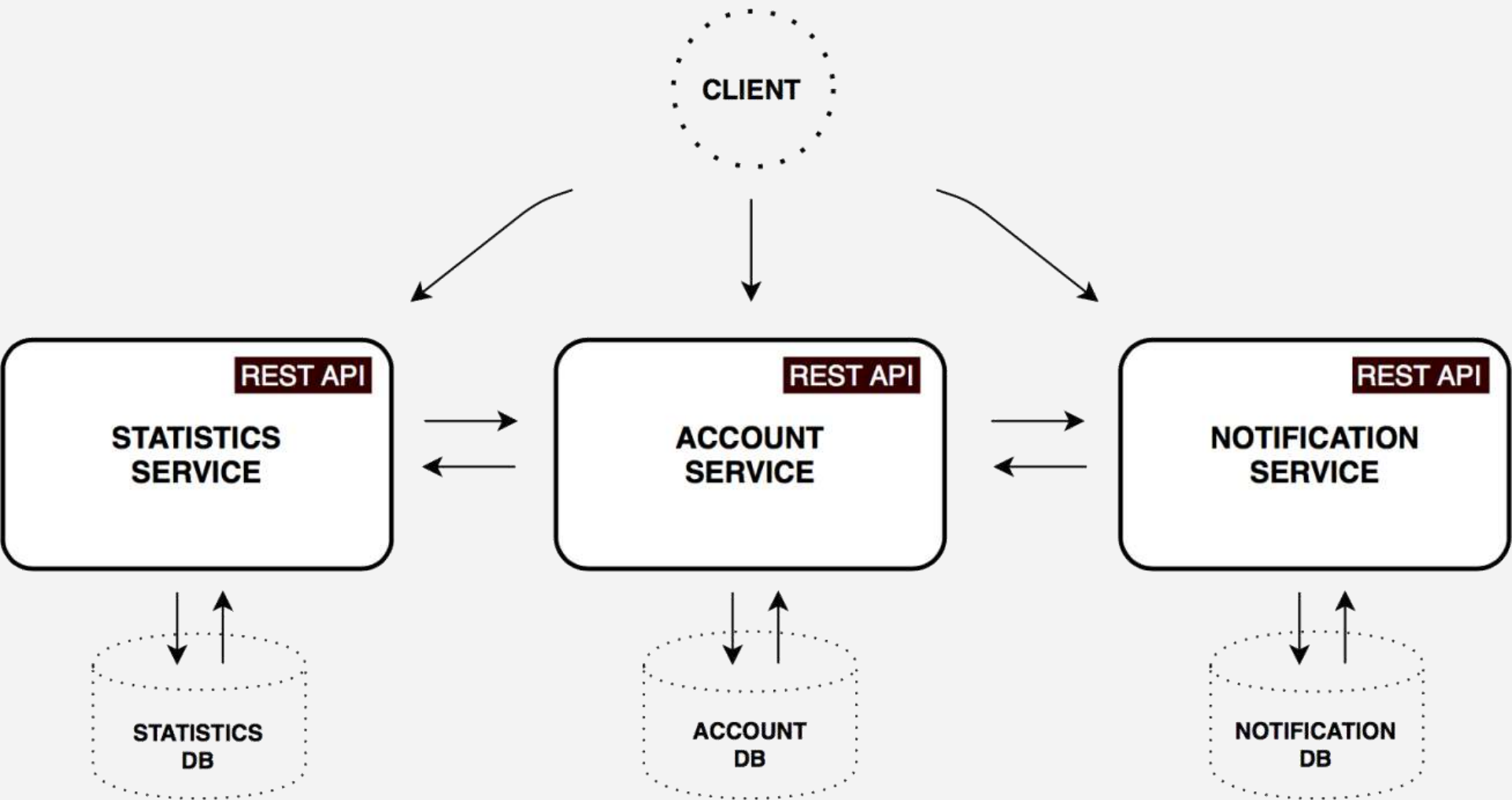
leekw3747@gmail.com

<http://blog.woniper.net>

- **REST API**
- **Spring Web MVC**
- **Spring Data JPA 또는 Java ORM**
- **를 사용해 봤거나, 무엇인지 알고있어야합니다.**

Modern Application





Spring Data REST

REST API를 쉽게 만들기 위한 기술

특징

- **Repository 인터페이스 정의만으로 REST API 제공**

특징

- **Repository 인터페이스** 정의만으로 REST API 제공
- **Query Method** : 메소드 선언으로 검색 API 지원

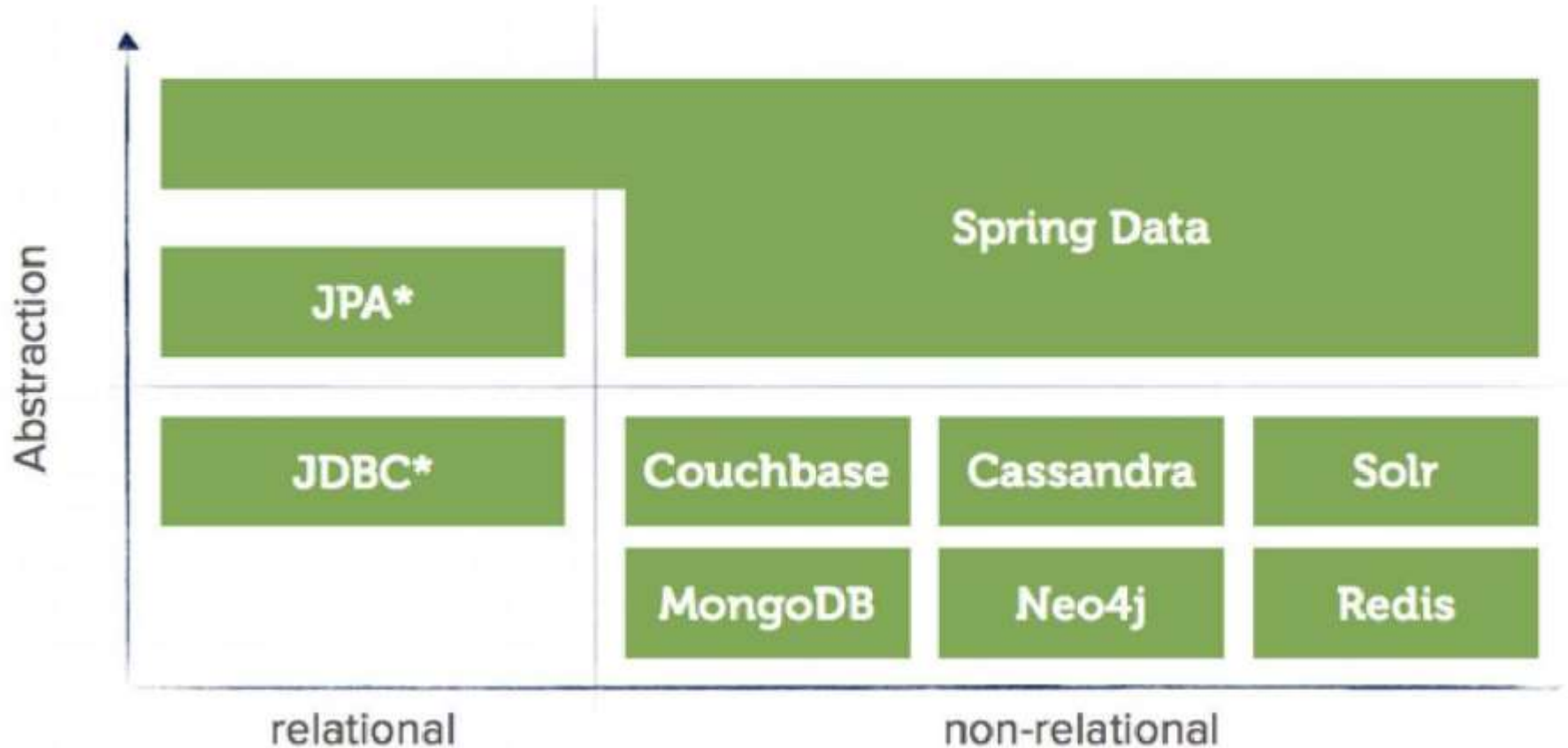
특징

- **Repository 인터페이스** 정의만으로 REST API 제공
- **Query Method** : 메소드 선언으로 검색 API 지원
- **Projection** : 데이터 표현 방식을 다양하게 정의/표현 가능

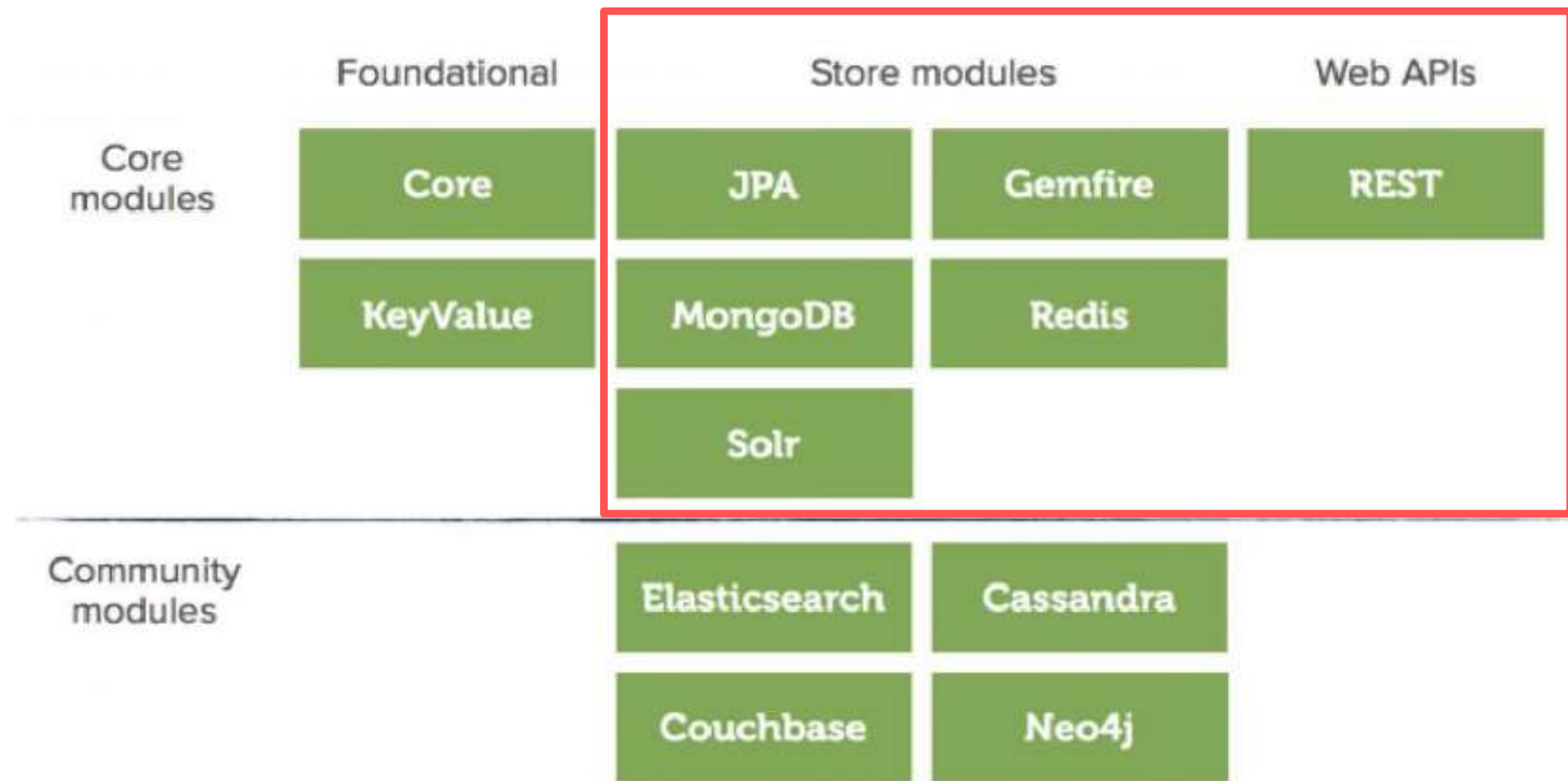
특징

- **Repository 인터페이스** 정의만으로 REST API 제공
- **Query Method** : 메소드 선언으로 검색 API 지원
- **Projection** : 데이터 표현 방식을 다양하게 정의/표현 가능
- **HATEOAS** : MetaData 표현 (Model, Link, Resource)

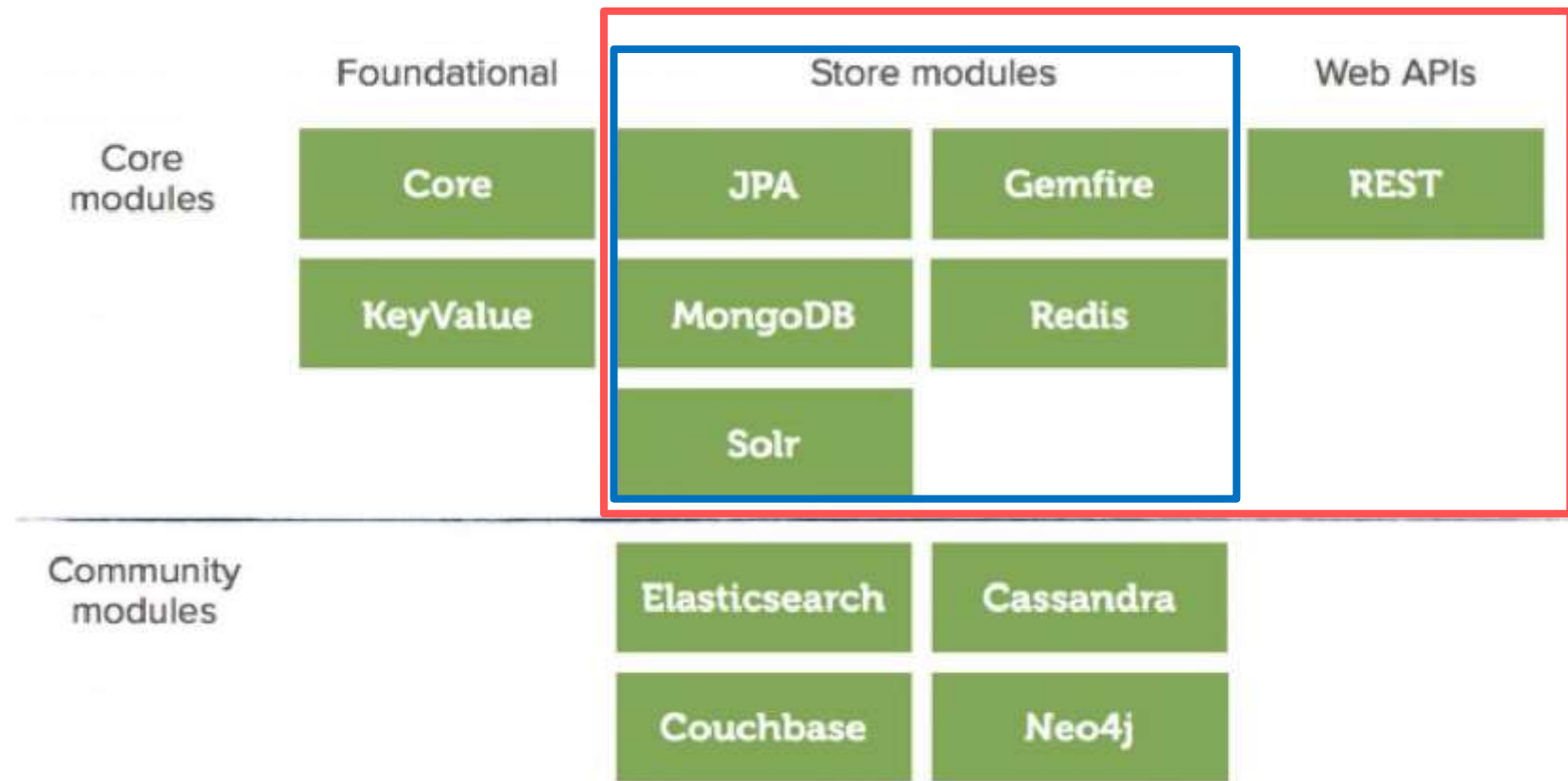
Spring Data



Spring Data REST



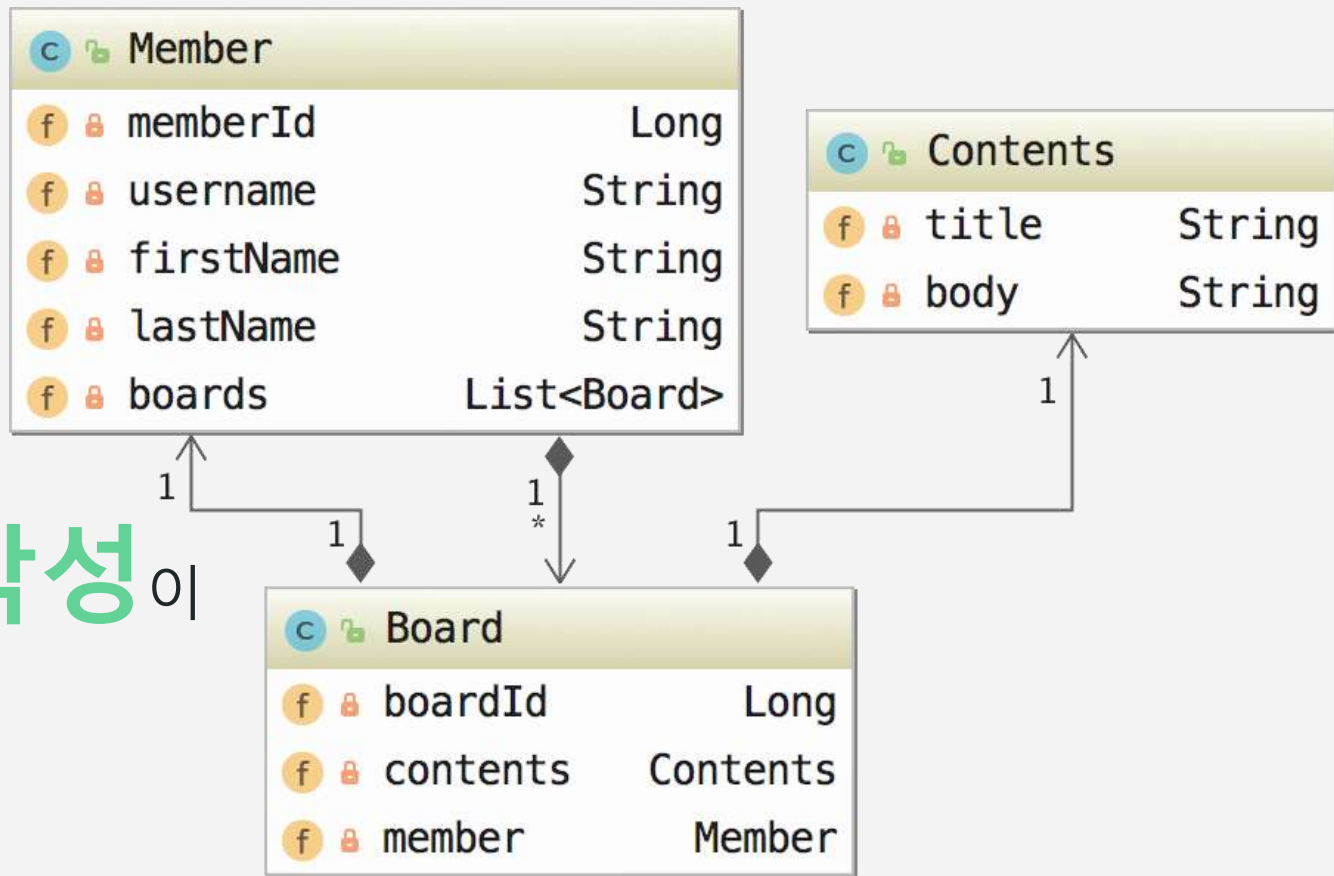
Spring Data REST



게시판을 만들어 볼까요?

회원만
게시글
가능합니다.

작성이



@Entity

public class Board {

@Id @GeneratedValue

private Long **boardId**;

@Embedded

private Contents **contents**;

@ManyToOne(optional = **false**) // 매핑 필수

private Member **member**;

}

@Embeddable

public class Contents {

@Column(name = "title", nullable = false)

private String **title**; // 제목

@Lob

@Column(name = "body", nullable = false)

private String **body**; // 본문 내용

}

@Entity

public class Member {

@Id @GeneratedValue

private Long **memberId**;

@Column(name = "**username**", unique = **true**, nullable = **false**)

private String **username**;

@Column(name = "**first_name**", nullable = **false**)

private String **firstName**;

@Column(name = "**last_name**", nullable = **false**)

private String **lastName**;

@OneToMany(cascade = CascadeType.**ALL**)

private List<Board> **boards** = **new** ArrayList<>();

}

Spring MVC

Controller

@Controller

@RequestMapping("/boards")

public class BoardController {

@ResponseBody

// 생성

@RequestMapping(method = RequestMethod.**POST**)

public Board post(@RequestBody Board board) {};

@ResponseBody

// 조회

@RequestMapping(value =("/{id}", method = RequestMethod.**GET**)

public Board get(@PathVariable("id") Long boardId) {};

@ResponseBody

// 수정

@RequestMapping(value =("/{id}", method = RequestMethod.**PUT**)

public Board put(@PathVariable("id") Long boardId,
@RequestBody Contents contents) {};

@ResponseBody

// 삭제

@RequestMapping(value =("/{id}", method = RequestMethod.**DELETE**)

public void post(@PathVariable("id") Long boardId) {};

}

Service

@Service

@Transactional

```
public class BoardService {  
    public Board new(Board board){};           // 생성  
  
    public Board get(Long boardId){};          // 조회  
  
    public Board update(Board board){};        // 수정  
  
    public void remove(Long boardId){};        // 삭제  
}
```

Repository

@Repository

interface BoardRepository

extends CrudRepository<Board, Long> {

Board save(Board board); **// insert, update**

Board findOne(Long boardId); **// select**

void delete(Long boardId); **// delete**

}



반복되고

단순한 코드

이제 우리가 할 것.

@RepositoryRestResource

interface BoardRepository

extends CrudRepository<Board, Long> {}

나는 REST API 입니다.

@RepositoryRestResource

interface BoardRepository

extends CrudRepository<Board, Long> {}

```
@RepositoryRestResource  
interface BoardRepository  
    extends CrudRepository<Board, Long> {}
```



/{**entity**}s/{**id**}

```
@RepositoryRestResource  
interface BoardRepository  
    extends CrudRepository<Board, Long> {}
```



/boards/1

```
@RepositoryRestResource  
interface BoardRepository  
    extends CrudRepository<Board, Long> {}
```

```
I CrudRepository
m save(S) S
m save(Iterable<S>) Iterable<S>
m findOne(ID) T
m exists(ID) boolean
m findAll() Iterable<T>
m findAll(Iterable<ID>) Iterable<T>
m count() long
m delete(ID) void
m delete(T) void
m delete(Iterable<? extends T>) void
m deleteAll() void
```

메소드가 곧 API

CrudRepository

m	save(S)	S
m	save(Iterable<S>)	Iterable<S>
m	findOne(ID)	T
m	exists(ID)	boolean
m	findAll()	Iterable<T>
m	findAll(Iterable<ID>)	Iterable<T>
m	count()	long
m	delete(ID)	void
m	delete(T)	void
m	delete(Iterable<? extends T>)	void
m	deleteAll()	void

POST : /boards

PUT | PATCH : /boards/{id}

I	CrudRepository	
m	save(S)	S
m	save(Iterable<S>)	Iterable<S>
m	findOne(ID)	T
m	exists(ID)	boolean
m	findAll()	Iterable<T>
m	findAll(Iterable<ID>)	Iterable<T>
m	count()	long
m	delete(ID)	void
m	delete(T)	void
m	delete(Iterable<? extends T>)	void
m	deleteAll()	void

POST : /boards

PUT | PATCH : /boards/{id}

GET : /boards/{id}
























I	CrudRepository	
m	save(S)	S
m	save(Iterable<S>)	Iterable<S>
m	findOne(ID)	T
m	exists(ID)	boolean
m	findAll()	Iterable<T>
m	findAll(Iterable<ID>)	Iterable<T>
m	count()	long
m	delete(ID)	void
m	delete(T)	void
m	delete(Iterable<? extends T>)	void
m	deleteAll()	void

POST : /boards

PUT | PATCH : /boards/{id}

GET : /boards/{id}

GET : /boards

 CrudRepository	
  save(S)	S
  save(Iterable<S>)	Iterable<S>
  findOne(ID)	T
  exists(ID)	boolean
  findAll()	Iterable<T>
  findAll(Iterable<ID>)	Iterable<T>
  count()	long
  delete(ID)	void
  delete(T)	void
  delete(Iterable<? extends T>)	void
  deleteAll()	void

POST : /boards

PUT | PATCH : /boards/{id}

GET : /boards/{id}

GET : /boards

DELETE : /boards/{id}

검색 API도 만들고 싶어요.

@RepositoryRestResource

interface BoardRepository

extends CrudRepository<Board, Long> {

@RestResource(path = "starts-title")

**List<Board> findByContentsTitleStartsWith(
@Param("title") String title);**

}

```
@RepositoryRestResource
```

```
interface BoardRepository
```

```
    ext /boards/search/starts-title Long> {
```

```
    @RestResource(path = "starts-title")
```

```
    List<Board> findByContentsTitleStartsWith(  
        @Param("title") String title);
```

```
}
```

```
@RepositoryRestResource
```

```
interface BoardRepository
```

```
    ext /boards/search/starts-title, Long> {
```

```
    @RestResource(path = "starts-title")
```

```
    List<Board> findByContentsTitleStartsWith(  
        @Param("title") String title);
```

```
?title={}
```

```
}
```


@P
int

제목이 **h로 시작**하는
게시글 리스트를 조회하고 싶어요.

extends CrudRepo<Board, Long> {

@RestResource(path = "starts-title")
List<Board> findByContentsTitleStartsWith(
@param("title") String title);

}

@P
int

제목이 **h로 시작**하는
게시글 리스트를 조회하고 싶어요.

extends CrudRepo<Board, Long> {

@RestResource(path = "**starts-title**")
List<Board> findByContentsTitleStartsWith(
@Param("title") String title);

}

/boards/search/starts-title?**title=h**

```
@RepositoryRestResource  
interface BoardRepository  
    extends CrudRepository
```



```
SELECT *  
FROM board  
WHERE title LIKE h%
```

```
@RestResource(path = "starts-title")  
List<Board> findByContentsTitleStartsWith(  
    @Param("title") String title);
```

```
}
```

LiveCoding!?

장점

- 빠르게 REST API 개발 가능
- 확장이 쉽다.
- 기존 프로젝트에 설정 가능. (Spring Data 사용하는 경우)



보이는게 다가 아니다.

REST API

DDD

(ORM + Database)

Spring

WEB

DATA

DATA REST



적용!!!



적용!!!

의존성이 낮은 도메인



적용!!!

의존성이 낮은 도메인

사내 시스템



적용!!!

의존성 이 낮은 도메인

사내 시스템

조회 만 적용



끝.