

S H O U T · O U R · P A S S I O N · T O G E T H E R

ODo IT SOPT O

보충 세미나

SHOUT OUR PASSION TOGETHER
SOPT

모든 설명은 Window 10 설명 기준입니다.

01

Controller

02

Service

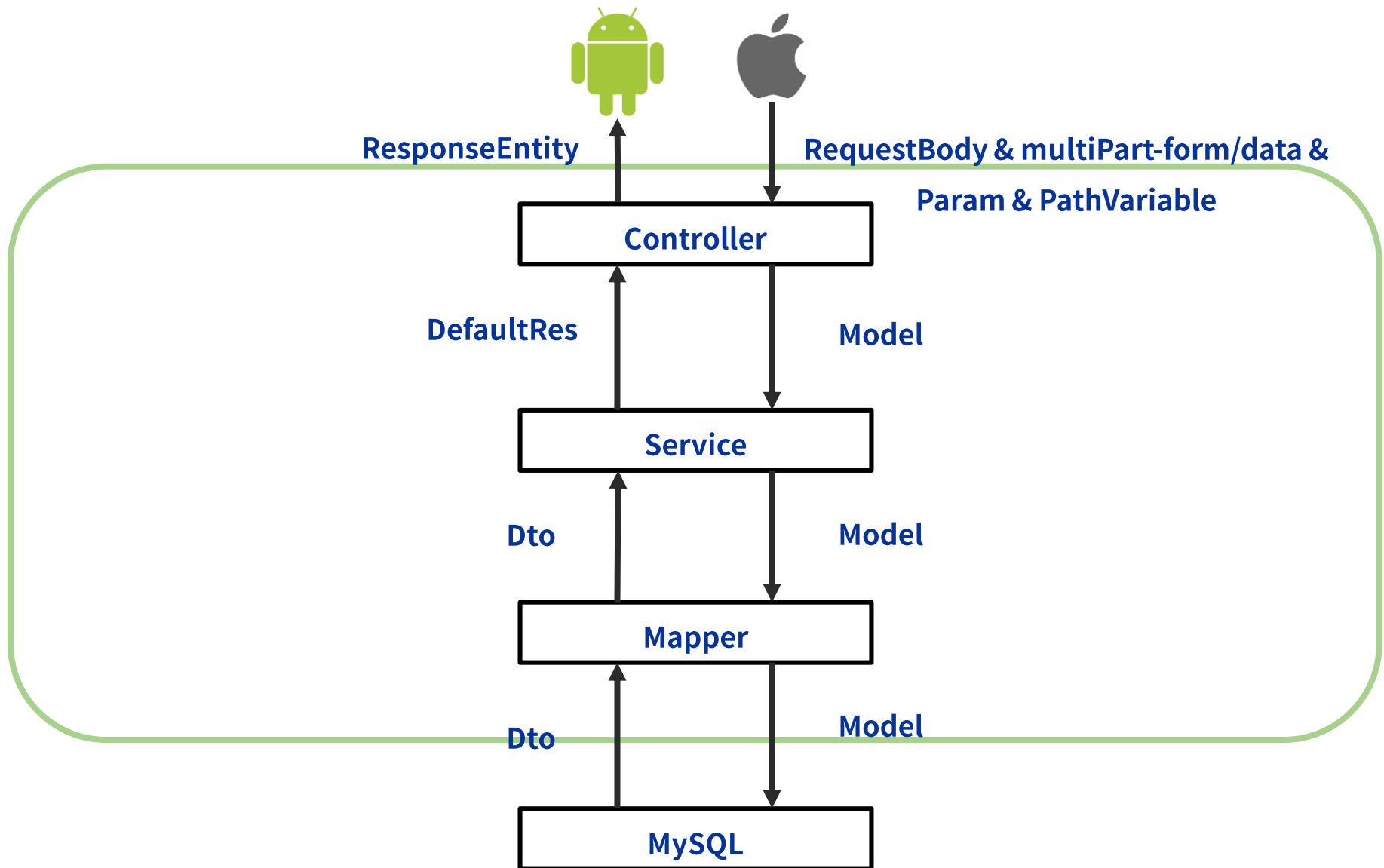
03

Mapper

04

AWS

1. AWS
2. EC2
3. RDS
4. S3

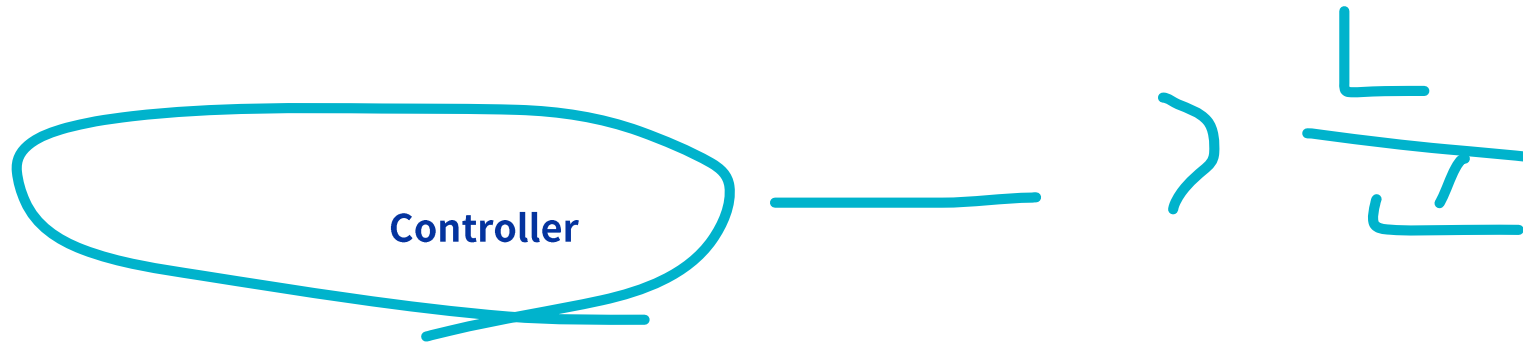




01



Controller



1. 프론트 엔드로 부터 요청을 받기 위한 출입구, 창구 같은 것이다.
2. 요청에 맞는 기능을 수행하고 응답을 프론트 엔드에게 전송한다.
3. URL을 지정해 기능을 만든다. 이것이 API다.
4. `@RestControllerAnnotation`을 지정하면 Controller가 된다.
5. 이곳에선 로그인(Auth), 프론트 엔드가 보낸 파라미터, body 등 확인, 서비스 호출, 자원에 대한 권한 확인 등을 한다.

Controller

```
@RestController  
public class StoreController {  
  
}
```

1. 꼭 @RestController가 붙어 있어야 한다.

Controller

```
@GetMapping("/users/{userId}")  
public ResponseEntity getUsers(@PathVariable(value = "userId") final int userId) {  
    return new ResponseEntity(null, HttpStatus.OK);  
}
```

1. @GetMapping
 1. 데이터를 요청/조회 할 때 쓰는 Annotation이다.

Controller

```
@PostMapping("/users")
public ResponseEntity saveUsers(@RequestBody final User user) {
    return new ResponseEntity(null, HttpStatus.CREATED);
}
```

1. @PostMapping
 1. 데이터를 저장할 때 쓰는 Annotation이다.

Controller

```
@PutMapping("/users/{userIdx}")  
public ResponseEntity updateUser(  
    @PathVariable(value = "userIdx") final int userIdx,  
    @RequestBody final User user) {  
    return new ResponseEntity(null, HttpStatus.NO_CONTENT);  
}
```

1. @PutMapping
 1. 데이터를 수정할 때 쓰는 Annotation이다.

Controller

```
@DeleteMapping("/users/{userId}")
public ResponseEntity deleteUser(
    @PathVariable(value = "userId") final int userId) {
    return new ResponseEntity(null, HttpStatus.NO_CONTENT);
}
```

1. @DeleteMapping
 1. 데이터를 삭제할 때 쓰는 Annotation이다.

@PathVariable

```
@RestController
public class FirstController {

    @GetMapping("/name/{name}")
    public String getName(@PathVariable(value = "name") final String name) {
        return name;
    }
}
```

The diagram illustrates the mapping of the path variable `{name}` in the URL `/name/{name}` to the `@PathVariable` parameter in the method signature. Red boxes highlight the `{name}` in the URL, the `@PathVariable(value = "name")` annotation, and the `String name` parameter. Red arrows show the flow from the `{name}` in the URL to the `@PathVariable` annotation, and then from the `@PathVariable` annotation to the `String name` parameter. A final red arrow points from the `return name;` statement back to the `String name` parameter, indicating the return value.

1. `@GetMapping("/name/{name}")`: URL Mapping에 `{ }` 문법 추가
2. `@PathVariable(value = "name")`
 1. URL에서 각 구분자에 들어는 값을 처리할 때 사용
 2. /뒤에 특정 값 `{ }` 을 `name`이라는 Parameter로 받으라는 의미이다.
 3. `/name/`의 값을 `@PathVariable(value = "name")`으로 받고
 4. 이것을 다시 `String name`으로 변환한다.

@RequestParam

1. @PathVariable
 1. QueryString을 처리할 때 사용
 2. /part?part=서버 에서 ?part=서버 와 같은 문법
2. value : queryString의 key 값
3. defaultValue : queryString 값이 없을 경우의 기본값

```
@GetMapping("/part")
public String getPart(@RequestParam(value = "part", defaultValue = "") final String part) {
    return part;
}
```

```
@RestController
@RequestMapping("post")
public class PostController {

    @PostMapping("")
    public String postUser(@RequestBody final User user) {
        return user.getName();
    }
}
```

1. @RequestBody : Parameter로 객체를 받는 Annotation
 1. HTTP Message 본문을 자바 객체로 변환해 준다.(Mapping)
 2. Spring MVC 내의 HttpMessageConverter가 변환을 처리해 준다.
 3. 전송한 객체와 전송 받을 Controller의 객체 타입이 같아야 한다.
 4. 같지 않으면 값이 자동으로 채워지지 않고 기본값이 들어간다.
 5. User 객체에는 Default 생성자가 있어야 한다.

Request Body 객체 받는 순서

1. Default 생성자를 통해 빈 객체가 생성된다.
2. 전송 받은 Http Message Body에 생성된 객체의 속성값이 있다면 Set Method를 통해 값이 채워진다.

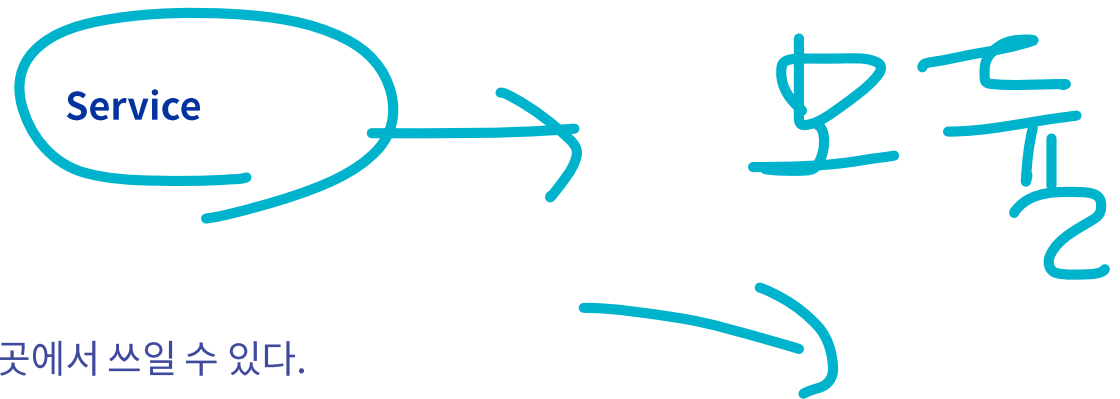
```
public class User {  
    private String name;  
    private String part;  
  
    1 public User() {}  
  
    public User(final String name, final String part) {  
        this.name = name;  
        this.part = part;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    2 public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getPart() {  
        return part;  
    }  
  
    2 public void setPart(String part) {  
        this.part = part;  
    }  
}
```



Service

Service

1. 실제 기능을 구현하는 곳이다.
2. 모듈, 싱글톤 객체, 공통 기능, node의 require하는 모듈, static, 컴포넌트, 모듈화, 부품화 와 비슷한 의미이다.
3. 생성자 의존성 주입을 사용해 Controller나 다른 Service에서 사용한다.
4. @Service Annotation을 지정하면 Service가 Spring IoC에 등록된다.



1. 예를 들어...
2. 회원 데이터 조회 기능은 정말 많은 곳에서 쓰일 수 있다.
3. 게시글을 조회할 때 작성자 정보, 프로필 조회, 나에게 쪽지 보낸 사람 등등...
4. 이럴 때 마다 회원 조회 기능을 구현하기엔 너무 비 효율적이다.
5. 반복되는 코드가 많아 진다.
6. 우리는 코드를 재 사용하고 싶다.
7. 그래서 Service를 만들고 여기에 회원 데이터 조회 기능을 구현한다.
8. 그렇게 되면 회원 데이터를 원하는 기능에 이 Service의 회원 데이터 조회 기능을 사용만 하면 된다.
9. 반복되는 코드를 줄일 수 있고, 유지 보수도 간편해 질 수 있다.

Service

1. @Service Annotation 명시를 통해 서비스임을 알린다.
2. 다른 Service, Controller 에서 사용할 수 있다.
3. 이곳에 실제 기능을 구현하면 된다.

```
@Service
public class ProductService {

    private final ProductMapper productMapper;

    public ProductService(final ProductMapper productMapper) {
        this.productMapper = productMapper;
    }

    public List<Product> getAllProduct() {
        return productMapper.findAll();
    }

    public Product getProduct(final int productId) {
        return productMapper.findById(productId);
    }

    public void saveProduct(final Product product) {
        productMapper.save(product);
    }

    public Product updateProduct(final Product product) {
        productMapper.update(product);
        return productMapper.findById(product.getId());
    }

    public void deleteProduct(final int id) {
        productMapper.deleteById(id);
    }
}
```

Service 사용

```
@RestController
public class StoreController {

    private final ProductService productService;

    public StoreController(final ProductService productService) {
        this.productService = productService;
    }

    @GetMapping("/products")
    public ResponseEntity getAllProduct() {
        return new ResponseEntity(productService.getAllProduct(), HttpStatus.OK);
    }
}
```

1. 원하는 컨트롤러 / 다른 서비스에서 생성자 의존성 주입으로 추가해 주면 사용할 수 있다.
2. 만약 의존성 서비스가 이미 있으면 파라미터로 또 추가하면 된다.



7/07

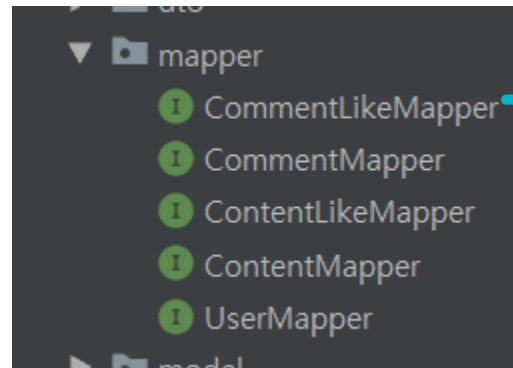
Mapper



Mapper

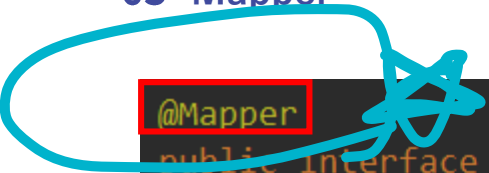
1. DataBase(MySQL)에서 데이터를 조회하고, 삽입하고, 수정하고 삭제하는 역할을 한다.
2. interface로 만들고 @Select, @Update, @Insert, @Delete Annotation을 지정한 다음 SQL을 작성만 하면 된다.
3. 이 Interface를 MyBatis가 실제 구현체로 만들어서 사용한다.
4. @Mapper Annotation을 지정하면 Mapper로 등록된다.
5. Mapper로 등록된 것을 생성자 의존성 주입을 통해 Service에서 사용하면 된다.
6. 각 테이블에 대한 Mapper는 따로 만들어 주는 것이 관리 하기 좋다.
 1. ex) user 테이블에 대한 Mapper – userMapper,
 2. Content 테이블에 대한 Mapper – contentMapper 등등...

Mapper 생성



그 2 개

1. 각 테이블에 맞게 Mapper Interface를 만들어서 사용한다.



```
@Mapper
public interface PeopleMapper {
    //사람 고유 번호로 조회
    @Select("SELECT * FROM people WHERE id = #{id}")
    People findById(@Param("id") final int id);
    //사람 데이터 저장
    @Insert("INSERT INTO people(name, age) VALUES(#{people.name}, #{people.age})")
    @Options(useGeneratedKeys = true, keyProperty = "b_id")
    int save(final People people);
    //사람 데이터 수정
    @Update("UPDATE people SET age = #{age} WHERE name = #{name}")
    void update(@Param("age") final int age, @Param("name") final String name);
    //사람 데이터 삭제
    @Delete("DELETE FROM people WHERE id = #{id}")
    void deleteById(@Param("id") final int id);
}
```

1. @Mapper Annotation을 명시해 Mapper임을 알린다.
2. @Select, @Insert, @Update, @Delete Annotation과 SQL을 이용해 쿼리문을 작성하면 MyBatis가 알아서 SQL을 실행해 주고, 결과값을 반환해 준다.
3. 파라미터는 @Param("키값")을 이용해 명시해 준다.

Mapper 사용

```
@Service
public class FriendService {

    private final PeopleMapper peopleMapper;

    public FriendService(final PeopleMapper peopleMapper) {
        this.peopleMapper = peopleMapper;
    }

    public People getPeople(final int id) {
        return peopleMapper.findById(id);
    }
}
```

생성자 의존성 주입

1. 원하는 서비스 / 다른 컴포넌트에서 생성자 의존성 주입으로 추가해 주면 사용할 수 있다.
2. 만약 의존성 서비스가 이미 있으면 파라미터로 또 추가하면 된다.
3. peopleMapper의 빨간 줄은 무시해도 된다.

Mapper 사용

```
@Service
public class FriendService {

    private final PeopleMapper peopleMapper;

    private final ContentMapper contentMapper;

    public FriendService(final PeopleMapper peopleMapper, final ContentMapper contentMapper) {
        this.peopleMapper = peopleMapper;
        this.contentMapper = contentMapper;
    }

    public People getPeople(final int id) {
        return peopleMapper.findById(id);
    }

    public Content getContent(final int id) {
        return contentMapper.findByContentIdx(id);
    }
}
```

생성자 의존성 주입에 추가

1. 새로운 Mapper 혹은 다른 서비스를 사용해야 한다면 추가하면 된다.



04




AWS

AWS를 사용하는 이유

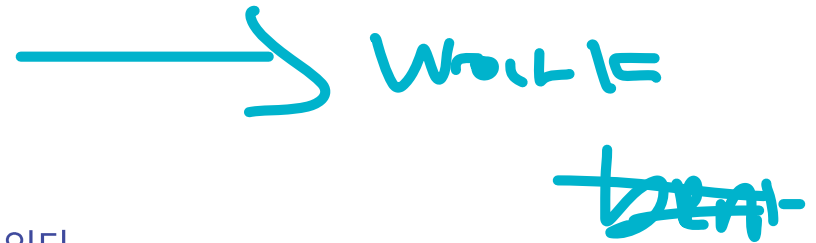
1. 서버는 24시간 365일 내내 돌아가야 한다.
2. 고정IP가 필요하다. 그래야 어디서든 프론트 엔드가 접속할 수 있기 때문이다.
3. 우리가 사용하는 pc를 24시간 365일 켜 두기엔 불가능하다.
4. AWS가 이러한 문제를 해결해 줬다.
5. 우리에게 상황에 맞는 PC를 빌려준다.
 1. EC2는 서버를 구동하기 위한 PC
 2. RDS는 DB만 전문적으로 구동하기 위한 PC
 3. S3는 파일을 저장하기 위한 PC(파일 업로드 및 다운로드, 기타 기능..)



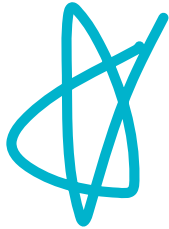
서버 컨트롤러
러
네트워크

1. Spring Boot 애플리케이션(서버)를 EC2로 옮긴다.
2. EC2는 보통 Linux다. (UNIX 기반)
3. EC2에서 명령어를 이용해 Spring Boot 서버를 실행 시킨다.
4. 명령어 : `nohup java -jar jar파일이름 &`
5. 서버만 실행 되어 있다면, 고정 IP(Elastic IP)를 이용해 클라이언트가 언제, 어디서든 접근할 수 있다. 
6. `netstat -tnlp` 명령어를 통해 서버가 구동 중인지 확인 할 수 있다.
7. `tail -f nohup.out` 명령어를 통해 로그를 확인 할 수 있다.
8. Elastic IP는 EC2가 꺼져 있다면 돈이 청구 된다!
9. Elastic IP를 해제하고 EC2를 종료, 정지 해야 한다.

RDS



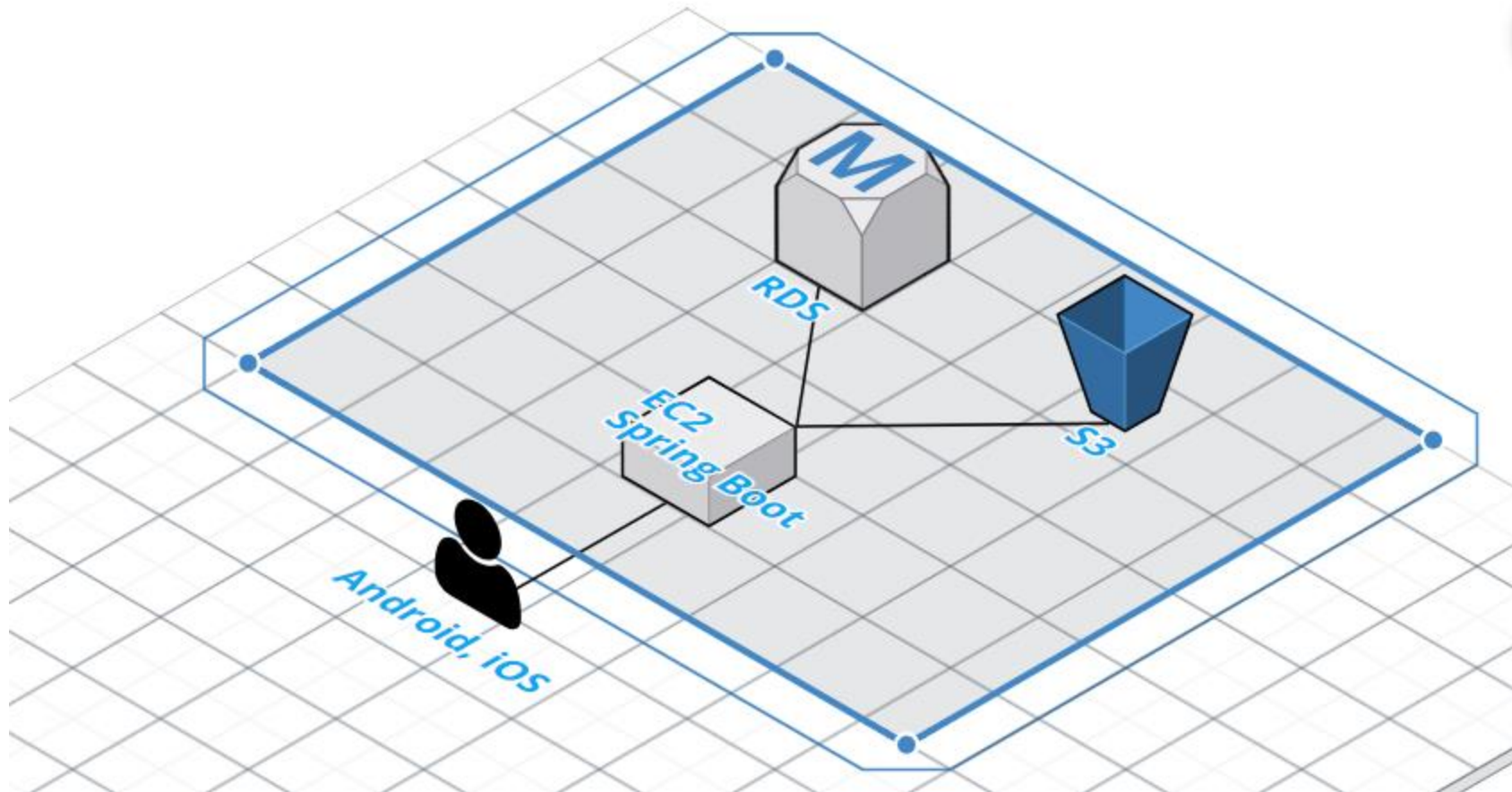
1. 내 PC에 있는 DB는 내 PC가 켜져 있을 때만 접근할 수 있다.
2. 내 PC를 24시간 365일 켜 둘 순 없다.
3. AWS RDS는 DB다. 24시간 365일 켜져 있다.
4. DB 주소를 AWS RDS로 설정하면 언제, 어디서든 DB에 접근할 수 있다.
5. Mapper를 이용해 MySQL을 사용한다.



파일

저장

1. 이미지, 동영상 등 멀티 미디어 파일을 저장하는 공간이다.
2. 24시간, 365일 접근할 수 있다.
3. S3에 파일을 업로드하는 서비스는 가져다 가 쓰면 된다.
4. 링크 : <https://github.com/bghgu/SOPT-23-Server/blob/master/5%EC%B0%A8/seminar5/src/main/java/org/sopt/seminar5/service/S3FileUploadService.java>
5. 서비스는 모듈 같은 개념이다.
6. S3에 HTML, CSS, JS를 저장하면 웹 페이지를 호스팅 할 수도 있다.



SHOUT · OUR · PASSION · TOGETHER

ODo IT SOPTO

THANK U

PPT 디자인 한승미 세미나 자료 배다슬

SHOUT OUR PASSION TOGETHER
SOPT