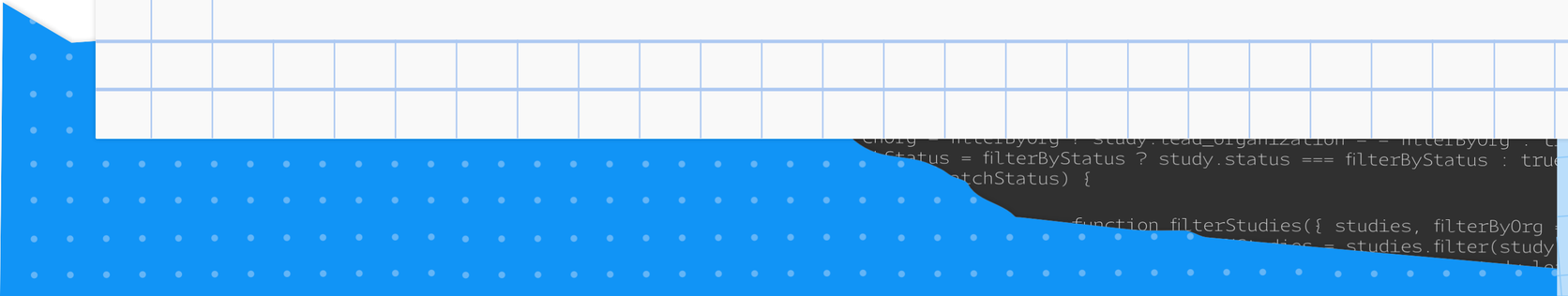
Google Developer Student Clubs

Backend 3th study

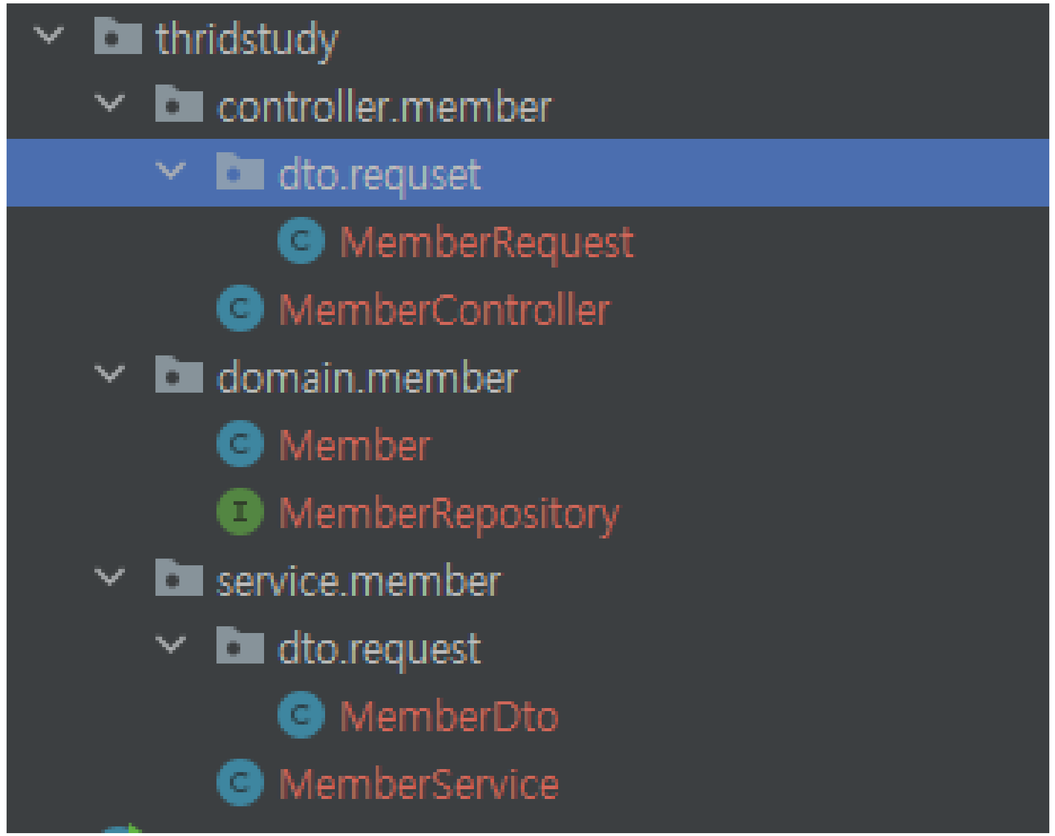
Contents

Google Developer Student Clubs

Goal

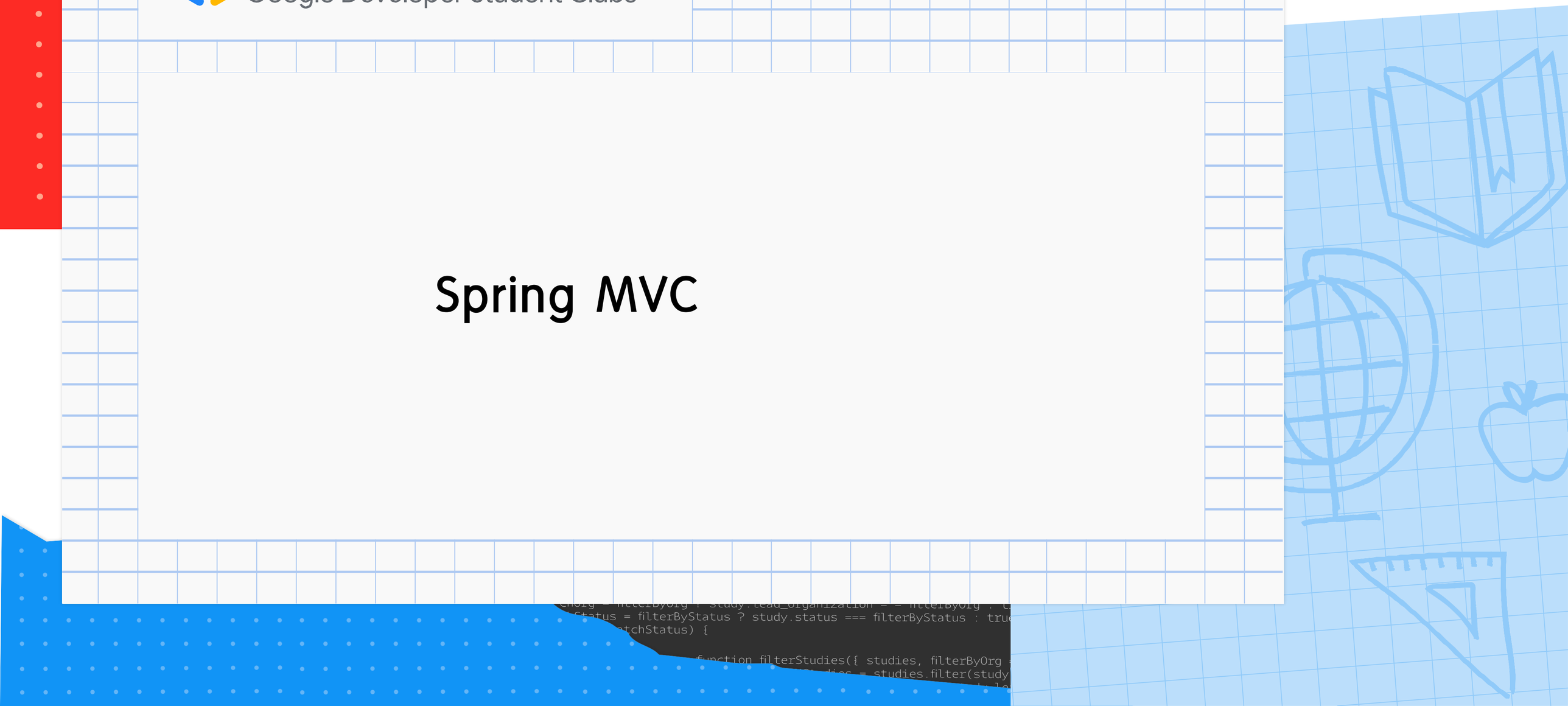


Goal

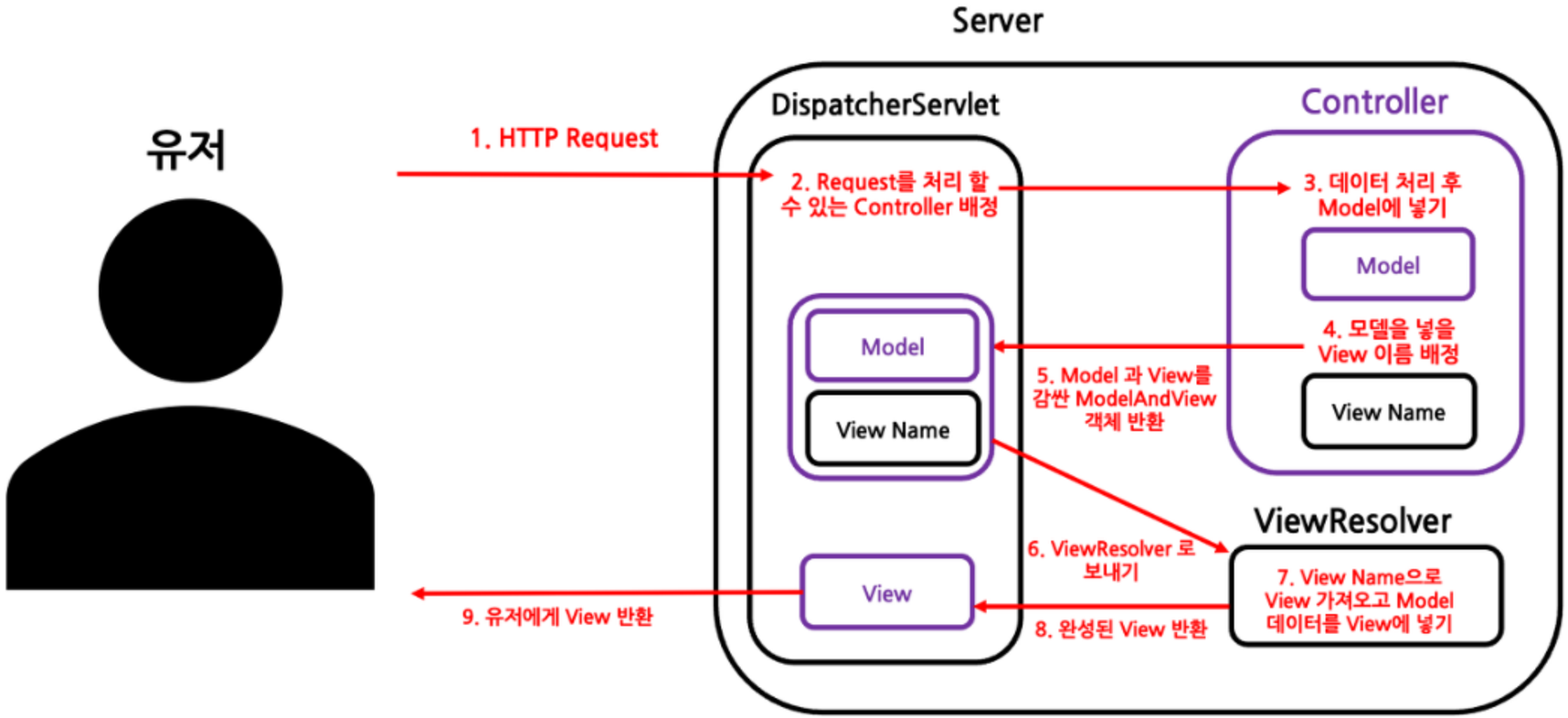

thridstudy
controller.member
dto.requset
MemberRequest
MemberController
domain.member
Member
MemberRepository
service.member
dto.request
MemberDto
MemberService

Google Developer Student Clubs

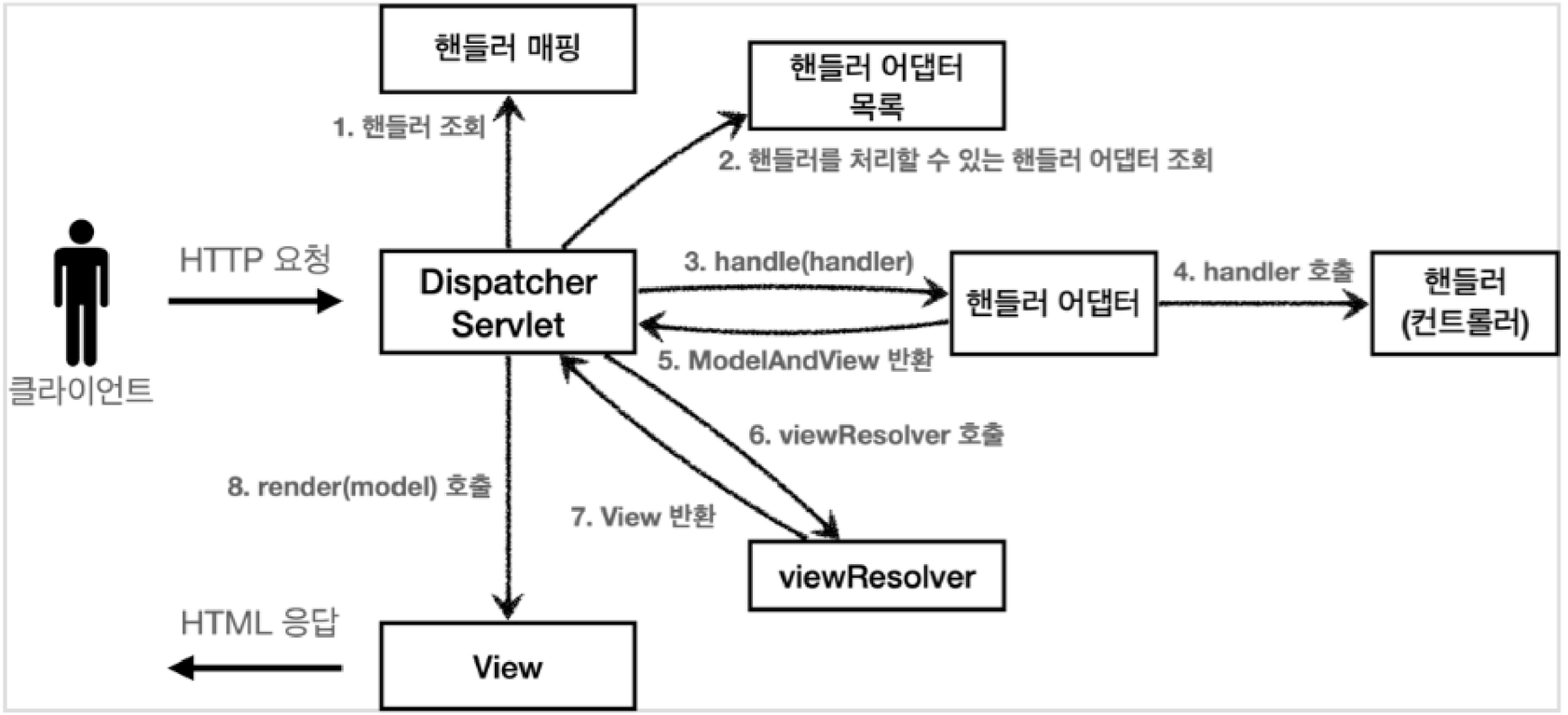
Spring MVC



Spring Web


Server
DispatcherServlet
Controller
유저
1. HTTP Request
2. Request를 처리 할 수 있는 Controller 배정
3. 데이터 처리 후 Model에 넣기
Model
Model
View Name
4. 모델을 넣을 View 이름 배정
5. Model 과 View를 감싼 ModelAndView 객체 반환
View Name
ViewResolver
6. ViewResolver 로 보내기
7. View Name으로 View 가져오고 Model 데이터를 View에 넣기
View
8. 완성된 View 반환
9. 유저에게 View 반환

Spring MVC


핸들러 매핑
핸들러 어댑터 목록
1. 핸들러 조회
2. 핸들러를 처리할 수 있는 핸들러 어댑터 조회
HTTP 요청
클라이언트
Dispatcher Servlet
3. handle(handler)
핸들러 어댑터
4. handler 호출
핸들러 (컨트롤러)
5. ModelAndView 반환
6. viewResolver 호출
8. render(model) 호출
7. View 반환
viewResolver
HTML 응답
View

Spring MVC

1. **핸들러 조회** : 핸들러 매핑을 통해 URL에 매핑된 핸들러(컨트롤러) 조회

2. **핸들러 어댑터 조회**: 핸들러를 실행할 수 있는 핸들러 어댑터 조회

3. **핸들러 어댑터 실행**: 핸들러 어댑터 실행

4. **핸들러 실행**: 핸들러 어댑터가 실제 핸들러를 실행

5. **ModelAndView 반환**: 핸들러 어댑터는 핸들러가 반환하는 정보를 ModelAndView로 변환해 반환.

6. **viewResolver 호출**: 뷰 리솔버를 찾아 실행한다.
⇒ *JSP: InternalResourceViewResolver가 자등 등록되어 사용된다.*

7. **View 반환**: 뷰 리솔버는 렌더링 역할을 담당하는 뷰 객체 반환.
⇒ *JSP: InternalResourceView(JstlView)를 반환하는데, 내부에는 forward() 가 있다.*

8. **뷰 렌더링**: 뷰를 통해서 뷰를 렌더링한다.

Spring MVC

이처럼 springMVC의 구조와 흐름에 대해 몰라도 기본적인 사용법만 익혀서 어노테이션만 사용하면 서버를 구현하는데 어려움이 없고, 대부분의 기능은 수년간 이미 다 구현되고 확장되었기에 스프링에 기능자체가 없어서 직접 구현할일은 거의 없다.

하지만, 이런 내부 로직과 흐름에 대해 이해한다면 내 프로젝트의 문제가 생겼을때 디버깅을 할때 몹시 유리하다.

Spring MVC

@RequestMapping	기본 매핑 – HTTP Method 전체 허용
@GetMapping	GET Mapping – Get Mapping 전용 어노테이션
@PostMapping	POST Mapping – POST Mapping 전용 어노테이션
@PutMapping	Put Mapping – POST Mapping 전용 어노테이션
@DeleteMapping	Delete Mapping – Delete Mapping 전용 어노테이션

Spring MVC

@PathVariable

PathVariable(경로 변수)를 사용한 매핑

```
@GetMapping("member/{userId}")
public MemberResponse getMember(@PathVariable Long userId){
    return memberService.getMember(userId);
}
```

Consumes = application/json

Content-Type 헤더 기반 추가 매핑 Media Type

@RestController

@Controller와 @ResponseBody가 합쳐진 것.

Spring MVC

@PathVariable

PathVariable(경로 변수)를 사용한 매핑

```java
@GetMapping("member/{userId}")
public MemberResponse getMember(@PathVariable Long userId){
    return memberService.getMember(userId);
}
```

@RequestParam

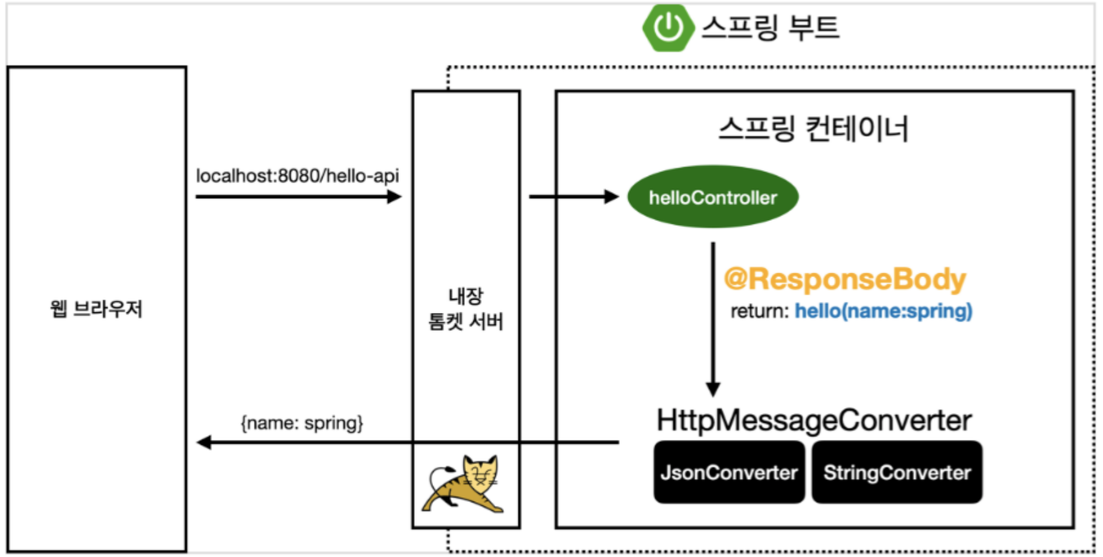
파라미터 이름으로 바인딩

```java
@GetMapping("/member")
public MemberResponse getMemberByName(@RequestParam("name") String name){
    return memberService.getMemberByName(name);
}
```

Spring MVC

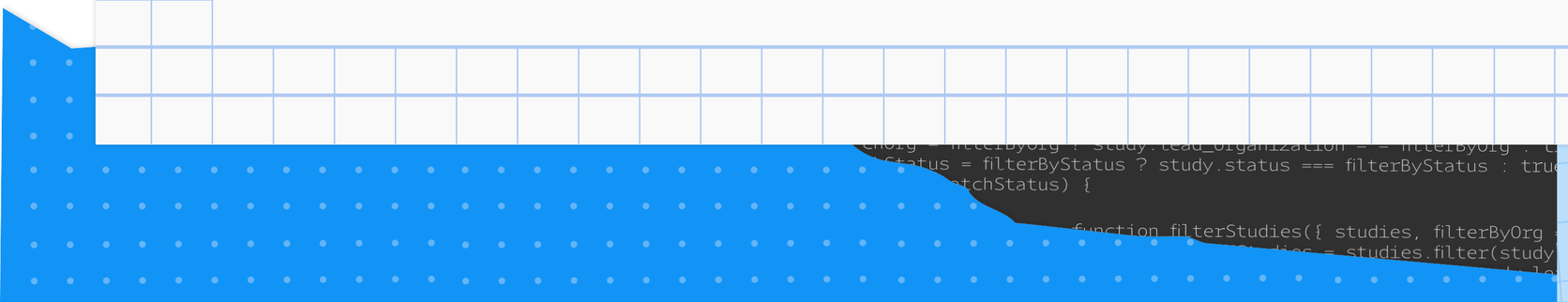
@RequestBody HTTP 메세지 바디 정보를 편리하게 조회하게 해주는 애노테이션

```
@PostMapping(value = "/member/new", consumes = "application/json")
  @RequestMapping(value = "/member/new", method = RequestMethod.POST)
public void saveMember(@Valid @RequestBody final MemberRequest request){
    memberService.signUp(request.toServiceDto());
}
```

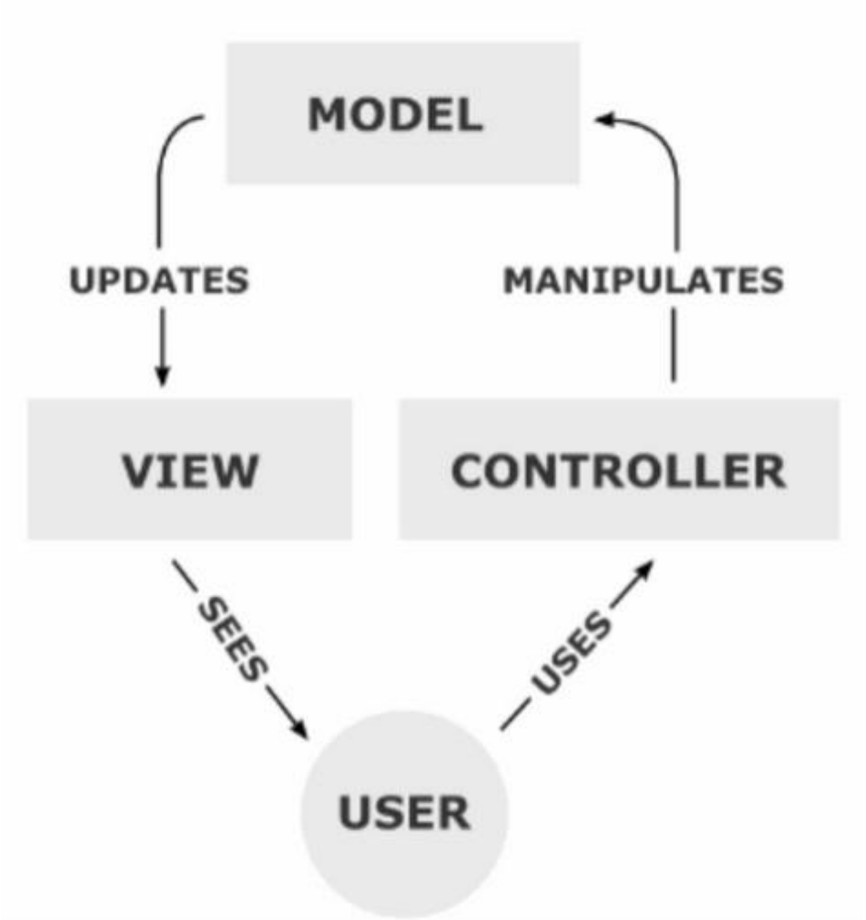


Google Developer Student Clubs

3 tier - Architecture



3 tier - Architecture



3 tier - Architecture

Model

프로그램이 작업하는 세계관의 요소들을 개념적으로 정의한 것

프로그램이 목표하는 작업을 원활하게 수행하기 위해 필요한 물리적 개체, 규칙, 작업 등의 요소들을 구분하는 역할

앱의 데이터와 비즈니스 로직 가지고 있다.

결과적으로 Model을 잘 설계한다라는 것은!
→ 해당 도메인 세계를 얼마만큼 이해하고 있는지와도 밀접한 연관이 있다.

→ 물리적인 요소뿐아니라 추상적인 요소 또한 해당 작업을 수행하는데
특정 책임과 역할로서 구분될 수 있다면 최대한 구체적이고
작은 entitiy를 유지하면서 Model 설계하는 것이 중요합니다.

3 tier - Architecture

View

사용자가 보는 화면에 입출력 과정 및 결과를 보여주기 위한 역할 (UI 담당)

입출력의 순서나 데이터 양식은 Controller에 종속되어 결정된다.

이때 View는 도메인 모델의 상태를 변환하거나, 받아서 렌더링하는 역할을 한다.

객체를 전달받아 상태를 바로 출력하는 역할만을 담당해야 한다.

view에서는 도메인 객체의 상태를 따로 저장하고 관리하는 클래스 변수나 인스턴스 변수가 필요 없다.

Controller

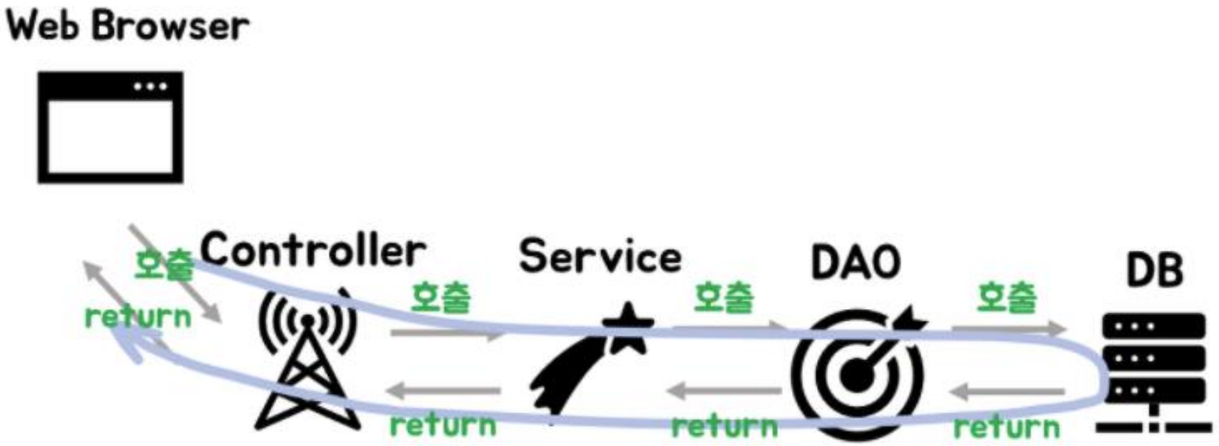
model과 view를 연결시켜주는 다리 역할

도메인 객체들의 조합을 통해 프로그램의 작동 순서나 방식을 제어한다.

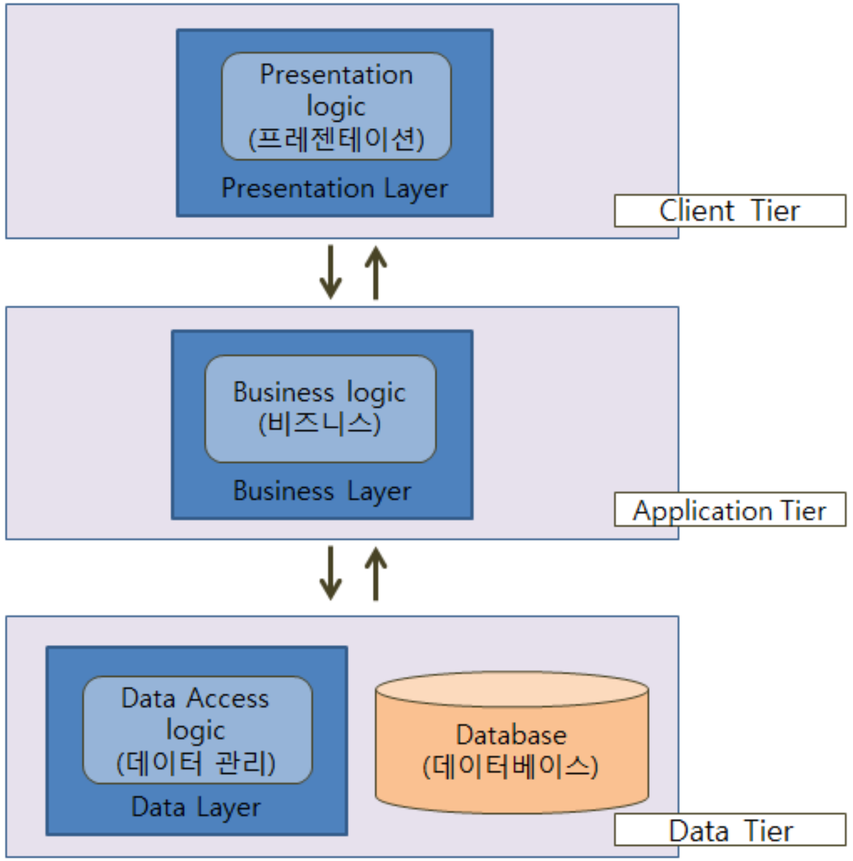
view와 model이 각각 어떤 역할과 책임이 있는지 알고 있어야한다.

View로부터 사용자의 action을 받아
Model에게 어떤 작업을 해야하는지 알려주거나 Model의 데이터 변화를
View에게 전달하여 View를 어떻게 업데이트할지 알려줍니다.
Model, View에게 직접 지시 가능

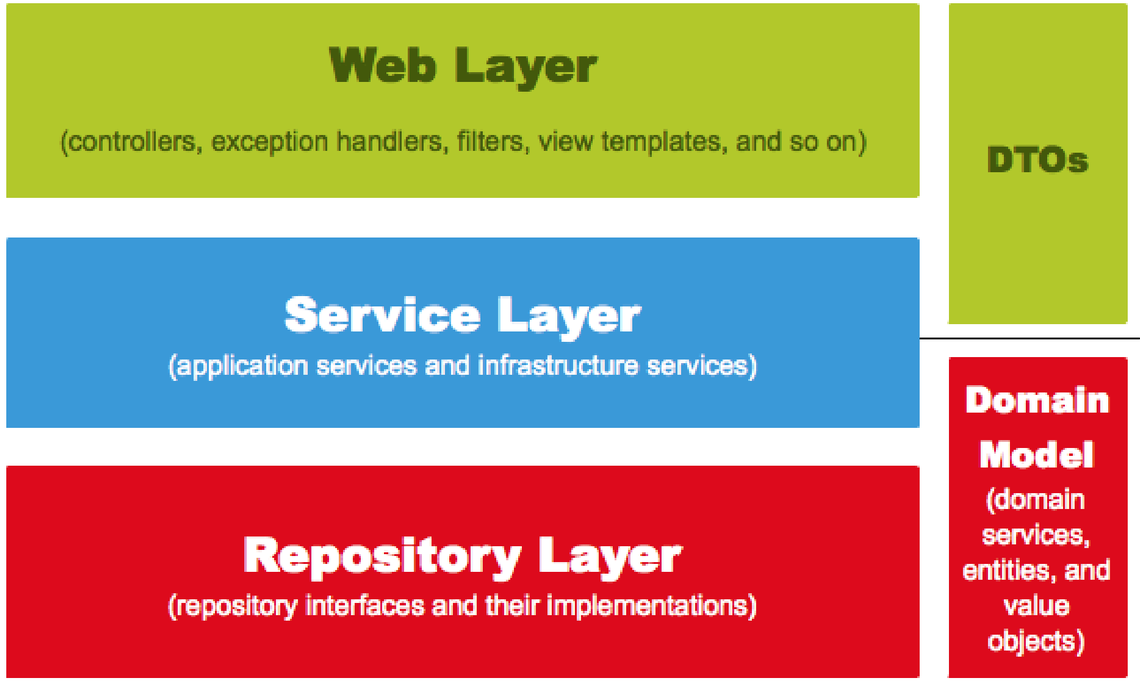
3 tier - Architecture



3 tier - Architecture



3 tier - Architecture In Spring



3 tier - Architecture

In Spring

Web Layer

흔히 사용하는 Controller와 JSP/freemarker 등 View 템플릿 영역이다.

이외에도 Filter, Intercepter, ControllerAdvice 등 **외부 요청과 응답**에 대한 전반적인 영역을 말한다.

3 tier - Architecture

In Spring

Service Layer

일반적으로 Controller와 Dao의 중간 영역에서 사용된다.

@Transactional이 사용되어야 하는 영역이다.

보통 여기서 **비즈니스 로직**을 처리하나 관점에 따라 Domain Model에서 비즈니스 로직을 처리하기도 한다.

3 tier - Architecture In Spring

Repository Layer(Persistence Layer)

DB에 접근하는 영역
Dao(Data Access Object) 영역

3 tier - Architecture In Spring

Dtos

계층 간에 데이터 교환을 위한 객체

3 tier - Architecture In Spring

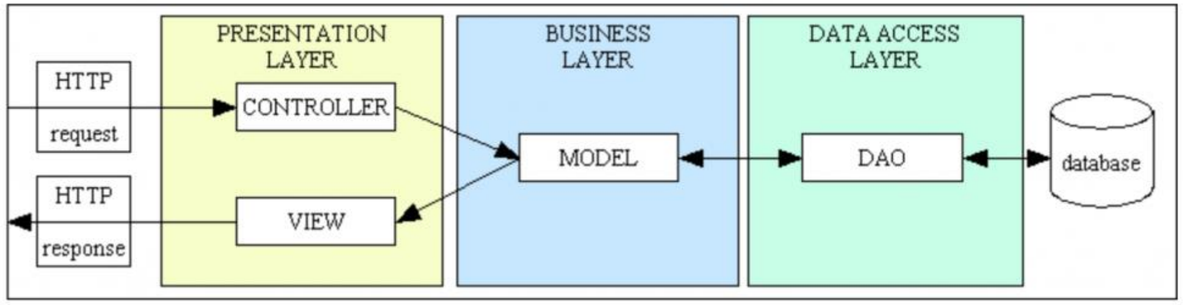
Domain Model(Business Layer)

도메인이라 불리는 개발 대상을 모든 사람이 동일한 관점에서 이해할 수 있고 공유할 수 있도록 단순화 시킨 것을 도메인 모델이라고 한다.

ex) 택시 앱 -> 배차, 탑승, 요금 등이 모두 도메인이 될 수 있다.
@Entity가 사용된 영역이 도메인 모델이다.

3 tier - Architecture In Spring

Figure 8a - MVC plus 3 Tier Architecture



xxxController, xxxService, xxxServiceImpl

Google Developer Student Clubs

DTO

DTO(Data Transfer Object)

계층 간 데이터 교환을 하기 위해 사용하는 객체로,
DTO는 로직을 가지지 않는 순수한 데이터 객체(getter, 생성자)만 가진 클래스)

굉장히 중요) 도메인 모델을 캡슐화 하여 보호할 수 있다.

DTO

그럼 하나의 DTO 클래스를 만들어서 Service 레이어까지 보내주면 되겠구나!

DTO

하나의 DTO로 서비스까지 쭉 전달하는 것은 컨트롤러와 서비스 간 의존을 강화하니까 안된다!

특히 컨트롤러와 서비스의 의존은 매우 위험하다.

DTO

그러면 DTO는 그냥 데이터 전달을 위해 쓰이는 건가요?

DTO

DTO to entity는 사람마다 정말 다양하게 다르다.