Spring Boot Architecture

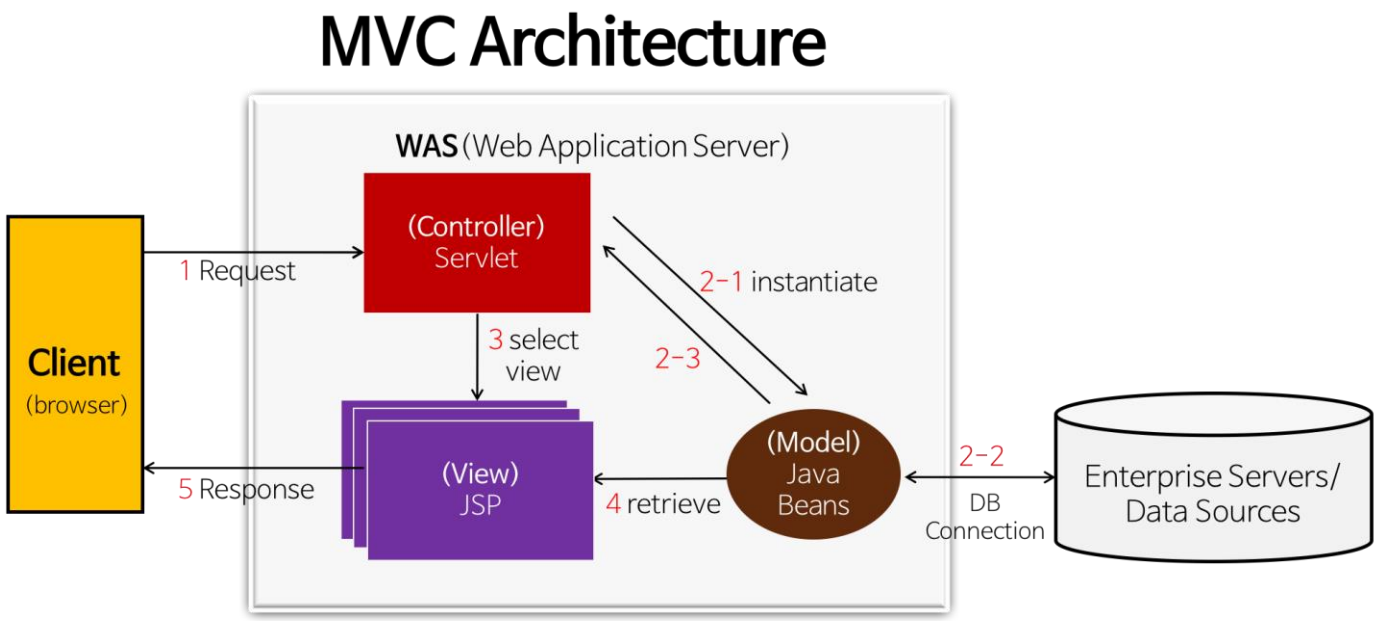
Jae Hyeon Kim

Contents

MVC Architecture

MVC Architecture

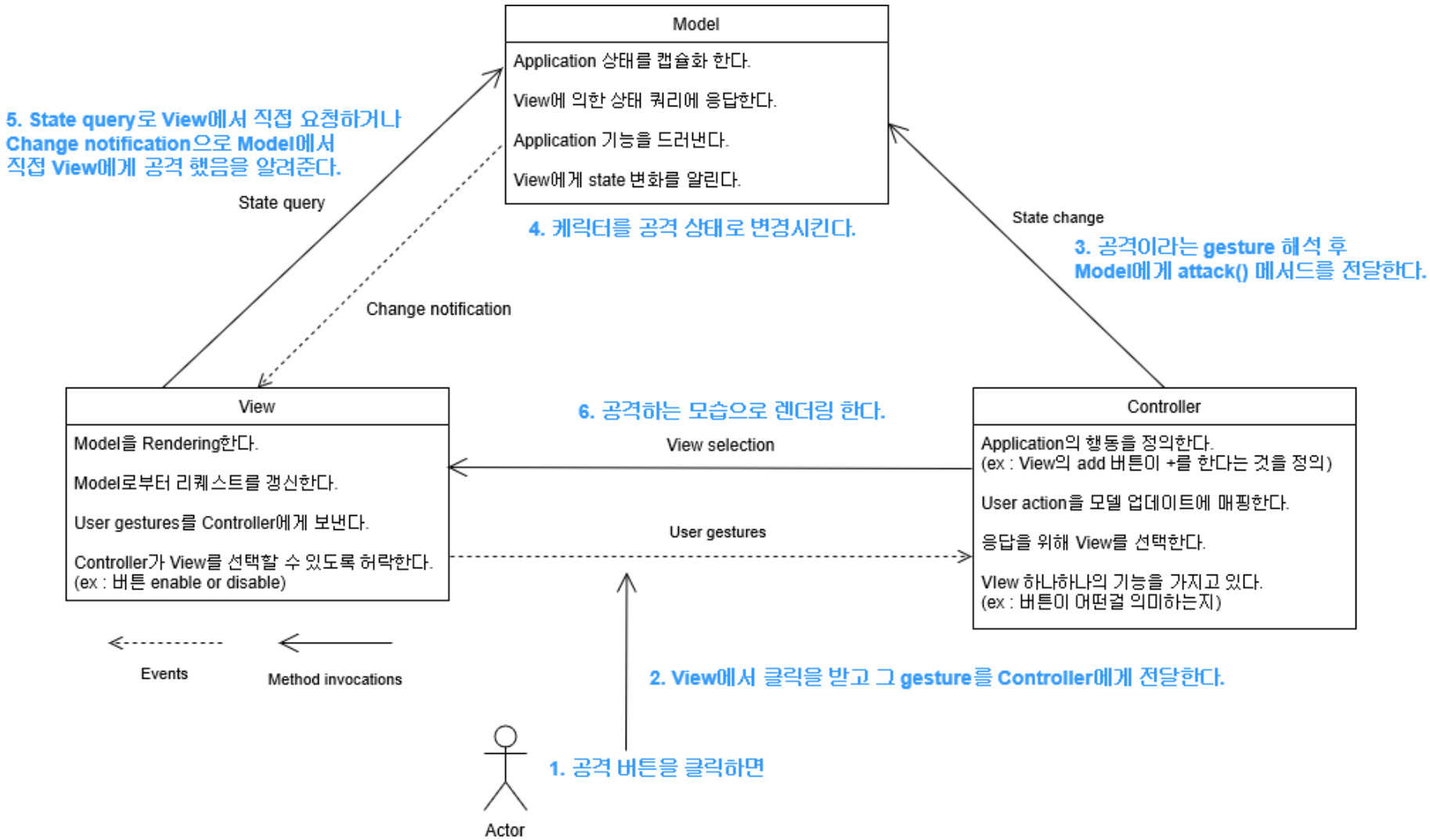
- 사용자의 상호작용과 데이터베이스를 비즈니스 로직과 분리시키기 위해 소프트웨어를 Model, View, Controller로 나눈 소프트웨어 아키텍처



MVC Architecture

- **Model** - 데이터와 비즈니스 로직을 저장 또는 처리
- **View** - 사용자 인터페이스 컴포넌트
- **Controller** - model과 view 사이의 상호작용 관리

MVC Architecture



MVC Architecture

model

```java
package MVCPattern;

public class CharacterModel {
  private String name;
  private int level;
  private int life;

  public CharacterModel(String name, int level, int life) {
    this.name = name;
    this.level = level;
    this.life = life;
  }

  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }

  public int getLevel() {
    return level;
  }
  public void setLevel(int level) {
    this.level = level;
  }

  public int getLife() {
    return life;
  }
  public void setLife(int life) {
    this.life = life;
  }
}
```

view

```java
package MVCPattern;

public class CharacterView {
  // State query
  public void printView(CharacterModel model) {
    System.out.println("Name :: " + model.getName());
    System.out.println("Level :: " + model.getLevel());
    System.out.println("Life :: " + model.getLife());
  }
}
```

controller

```java
package MVCPattern;

public class CharacterController {
  private CharacterModel model;
  private CharacterView view;

  public CharacterController(CharacterModel model, CharacterView view) {
    this.model = model;
    this.view = view;
  }

  // State change
  public void setCharacterName(String name) {
    model.setName(name);
  }
  public String getCharacterName() {
    return model.getName();
  }

  // State change
  public void setCharacterLevel(int level) {
    model.setLevel(level);
  }
  public int getCharacterLevel() {
    return model.getLevel();
  }

  // State change
  public void setCharacterLife(int life) {
    model.setLife(life);
  }
  public int getCharacterLife() {
    return model.getLife();
  }

  // View selection(Rendering)
  public void updateView() {
    view.printView(model);
  }
}
```

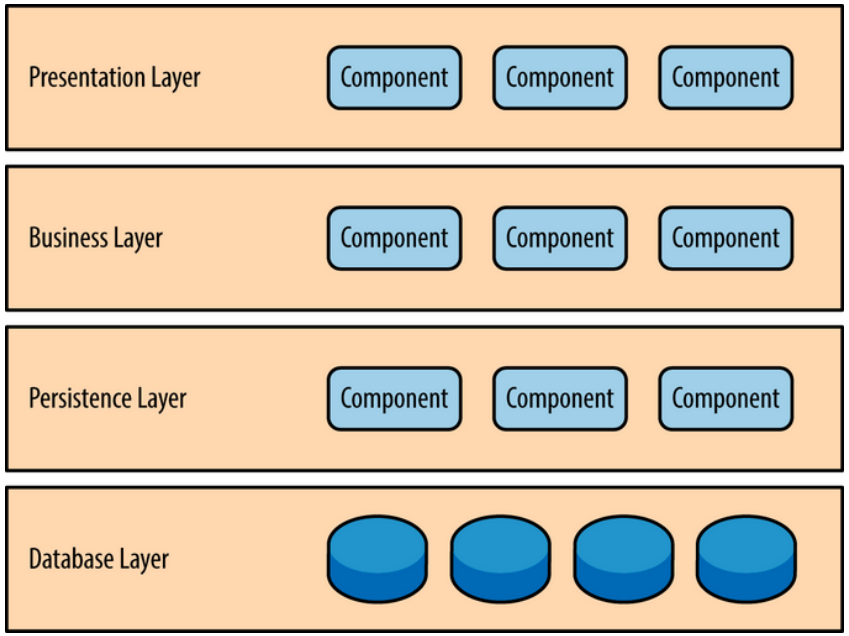
MVC Architecture

```
package MVCPattern;

public class MVCMain {

  public static void main(String[] args) {
    CharacterModel model = new CharacterModel("Crocus", 20, 3);
    CharacterView view = new CharacterView();
    CharacterController controller = new CharacterController(model, view);
    controller.updateView();

    controller.setCharacterLevel(controller.getCharacterLevel() + 1);
    controller.updateView();

    controller.setCharacterLife(controller.getCharacterLife() + 10);
    controller.updateView();
  }

}
```

Layered Architecture

Layered Architecture

- 소프트웨어 어플리케이션 내에서의 특정 역할과 관심사를 분리하여 각 계층별로 나눈 소프트웨어 아키텍처
- Layered Architecture에서 구성 레이어 숫자나 각 레이어의 유형을 명시하고 있지 않지만, 일반적인 경우 4개의 레이어로 구분한다



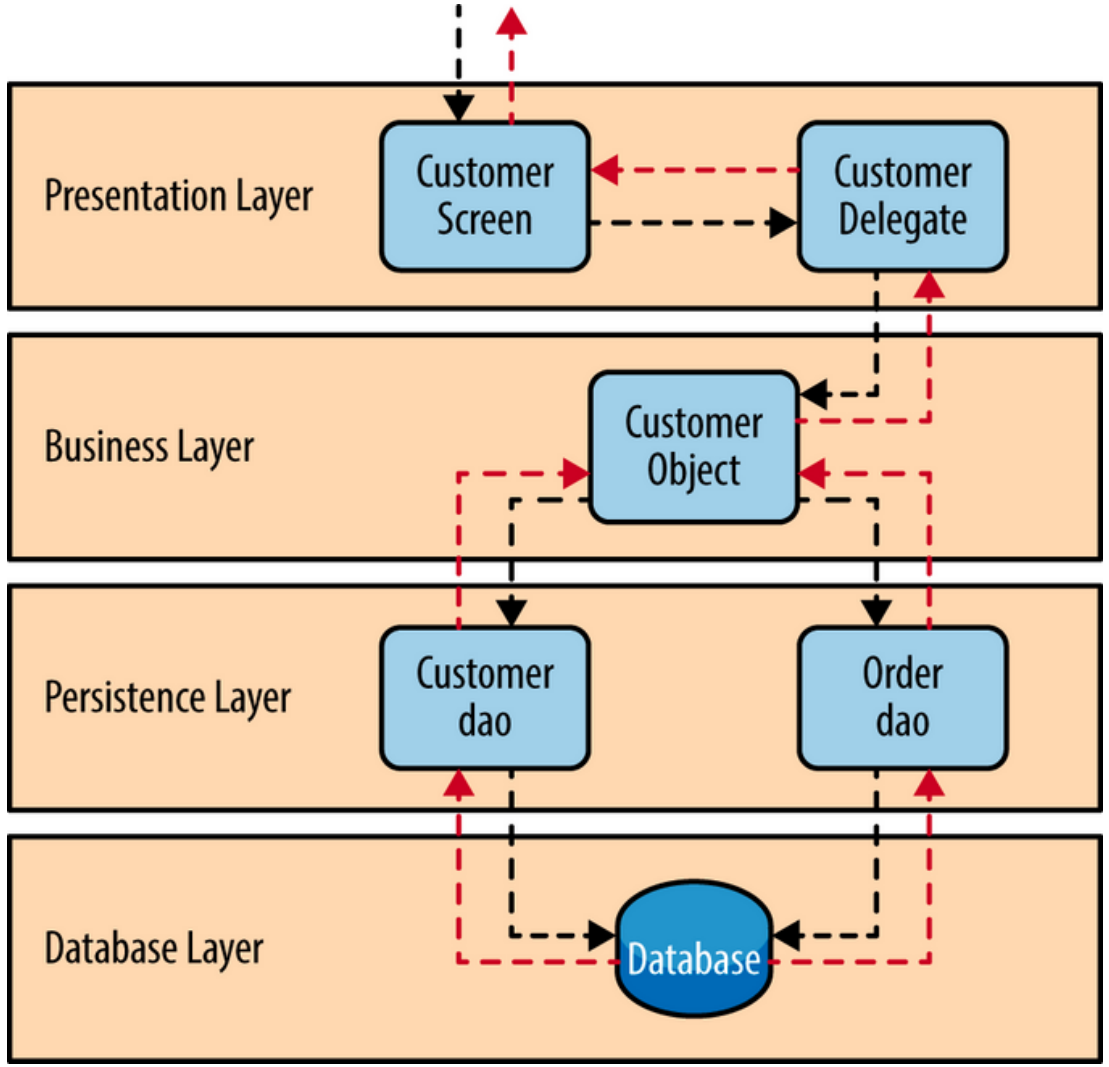
Layered Architecture

- **Presentation Layer** - 사용자가 데이터를 전달하기 위해 화면에 정보를 표시하는 것을 주 관심사로 둔다. Presentation Layer는 비즈니스 로직이 어떻게 수행되는지 알 필요가 없다. 대표적인 구성 요소는 View, Controller
- **Business Layer** – 비즈니스 로직을 수행하는 것을 주 관심사로 둔다. 화면에 데이터를 출력하는 방법이나 혹은 데이터를 어디서, 어떻게 가져오는지에 대한 내용은 알고 있지 않다. 그저 Persistence Layer에서 데이터를 가져와 비즈니스 로직을 수행하고 그 결과를 Presentation Layer로 전달하면 된다. 대표적인 구성 요소는 Service, Domain Model

Layered Architecture

- **Persistence Layer** – 어플리케이션의 영속성을 구현하기 위해, 데이터 출처와 그 데이터를 가져오고 다루는 것을 주 관심사로 둔다. 대표적인 구성요소는 Repository, DAO
- **Database Layer** – MySQL, MariaDB, PostgreSQL, MongoDB 등 데이터베이스가 위치한 계층을 의미

Layered Architecture



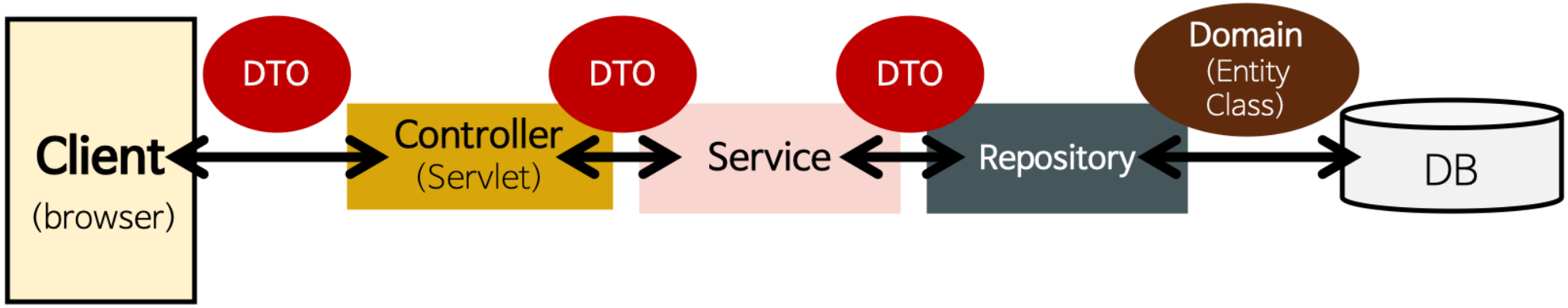
Entity

- Entity 클래스는 DB의 테이블에 존재하는 Column들을 필드로 가지는 객체. Entity는 DB의 테이블과 1대 1로 대응되며, 때문에 테이블이 가지지 않는 컬럼을 필드로 가져서는 안 된다.

```
@Entity
public class Post {
    private String title;
    private String content;
    private String author;
}
```

DTO(Data Transfer Object)

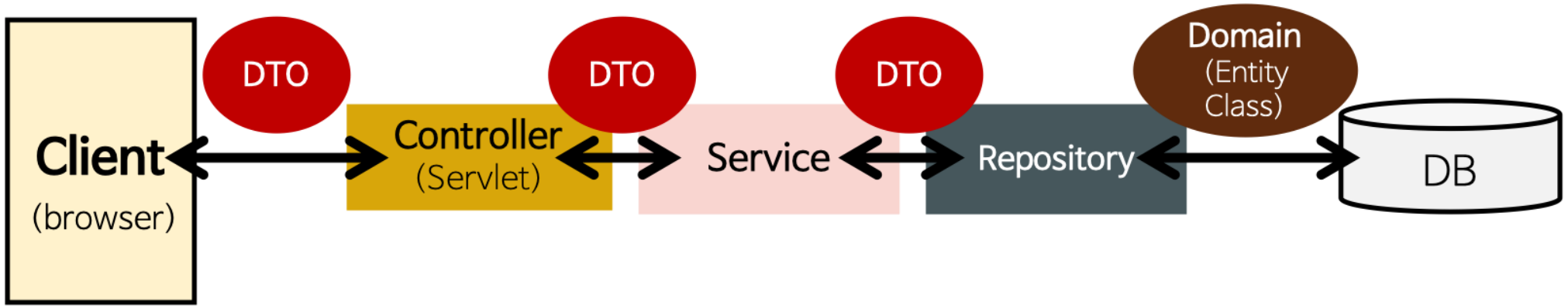
- DTO는 계층 간 데이터 교환 역할을 한다. DB에서 꺼낸 데이터를 저장하는 Entity를 가지고 만드는 일종의 Wrapper라고 볼 수 있는데, Entity를 Controller 같은 클라이언트단과 직접 마주하는 계층에 전달하는 대신 DTO를 사용해 데이터를 교환한다.



Client ⟨-dto—⟩ controller(web) - service - repository(dao) ⟨—domain(entity)—⟩ DB

DAO(Data Access Object)

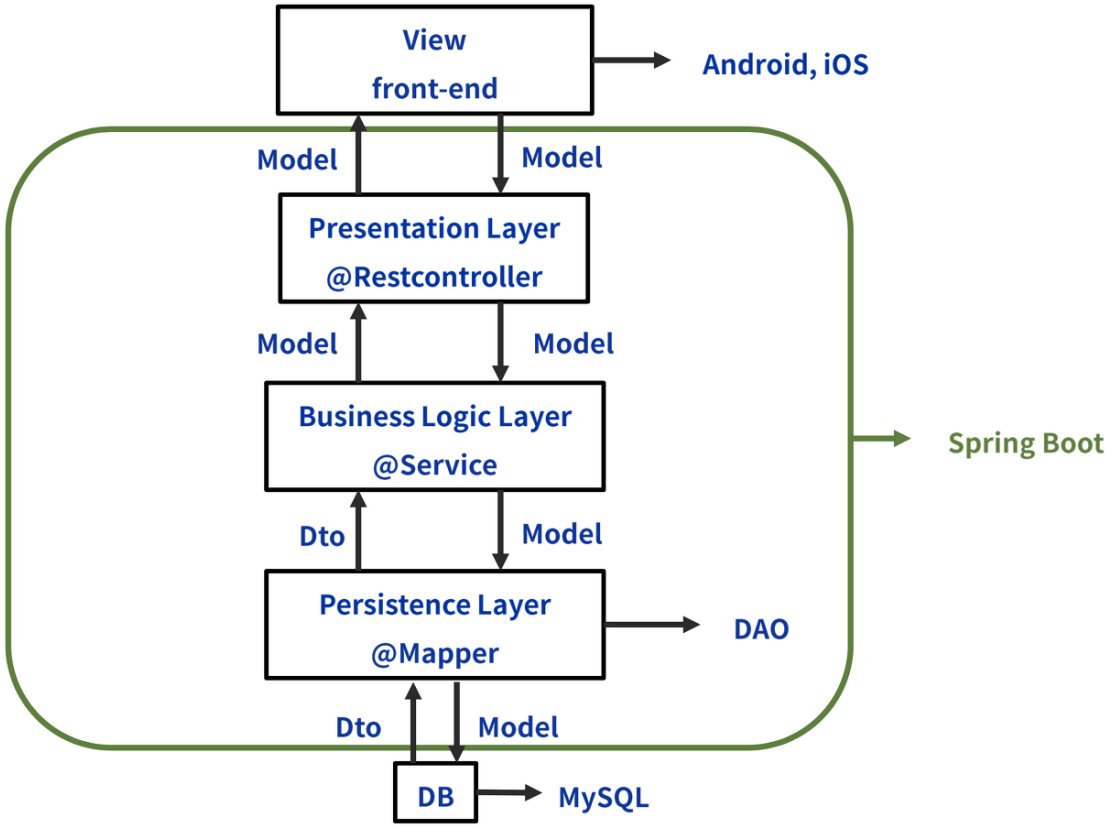
- DAO는 실제로 DB에 접근하는 객체. DAO는 프로젝트의 서비스 모델과 실제 데이터베이스를 연결하는 역할을 하며, JPA에서는 DB에 데이터를 CRUD하는 Repository 객체들이 DAO라고 볼 수 있다.



Client 〈-dto—〉 controller(web) - service - repository(dao) 〈—domain(entity)—〉 DB

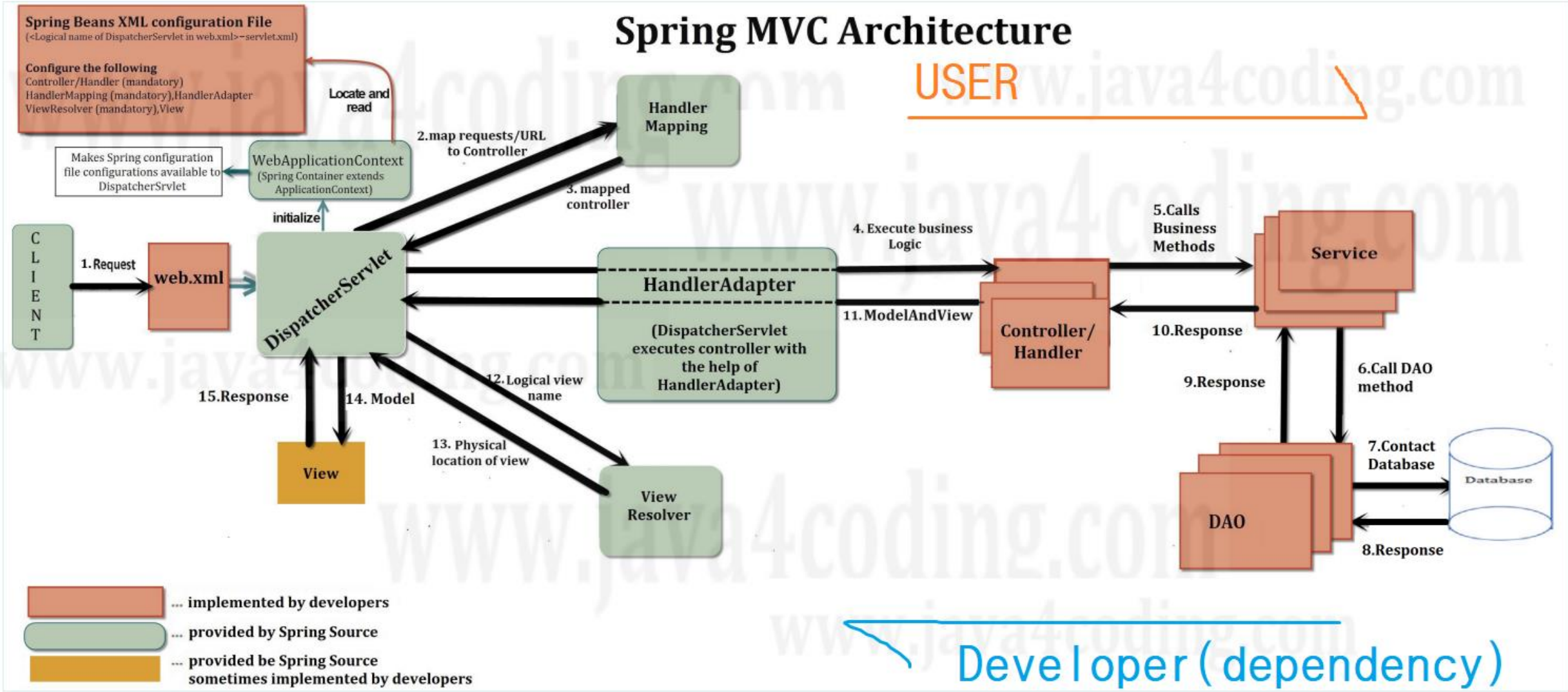
Service

- Service의 역할은 DAO가 DB에서 받아온 데이터를 전달받아 가공하고 Controller에게 정보를 보내주는 역할



Spring Boot Architecture

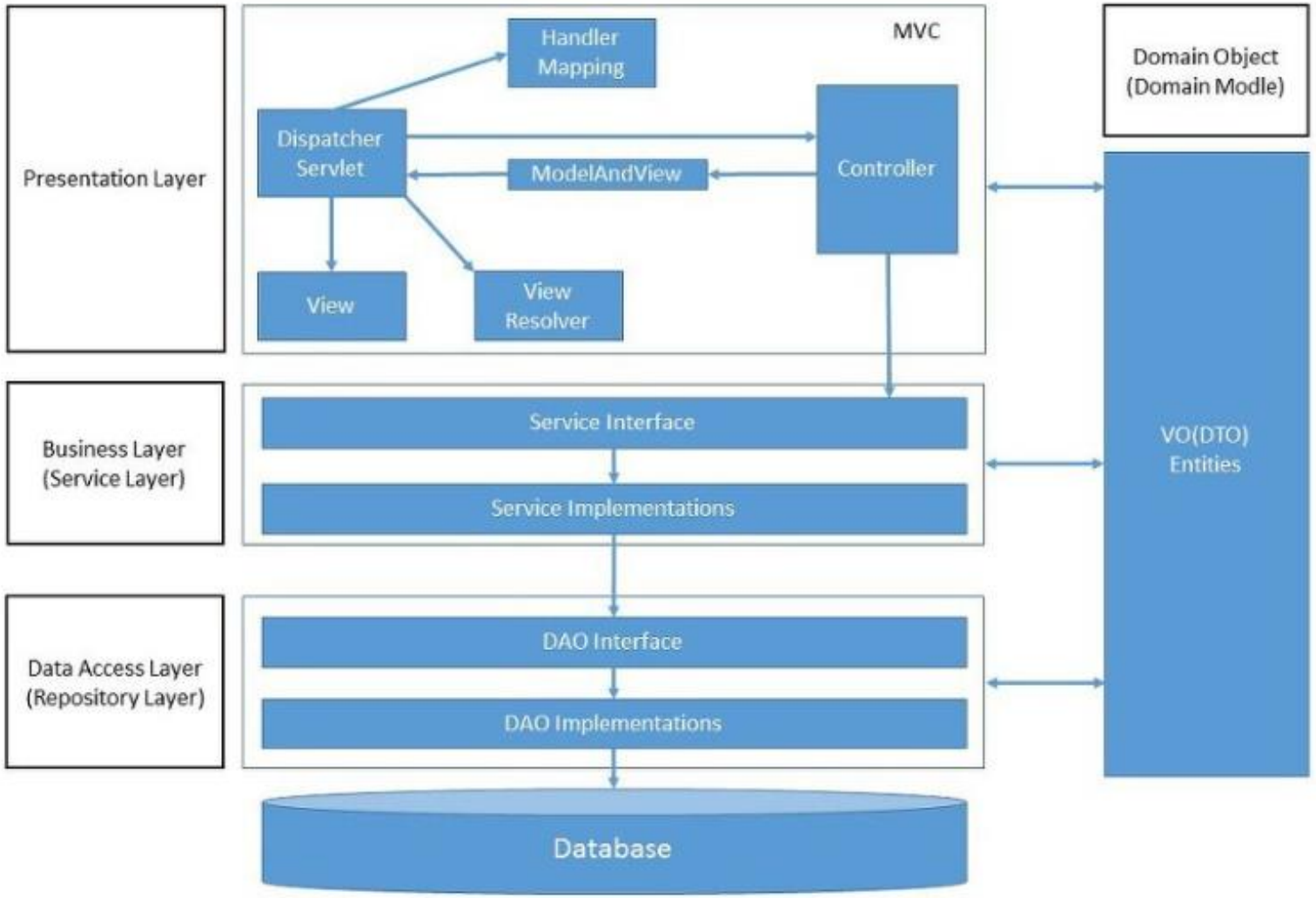
Spring Architecture – MVC View



Dispatcher Servlet

- Dispatcher Servlet은 HTTP protocol로 들어오는 모든 요청을 가장 먼저 받아 적합한 컨트롤러에 위임해주는 프론트 컨트롤러(Front Controller)라고 정의
- 클라이언트로부터 어떠한 요청이 오면 Tomcat과 같은 servlet container가 요청을 받아 Dispatcher Servlet에게 넘긴다
- Dispatcher Servlet은 해당 요청을 처리해야 하는 컨트롤러를 찾아서 작업을 위임한다.

Spring Architecture – Layered View



Spring Architecture - Login Example

Entity

```java
kiku99
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@Getter
@Entity
@Table(name = "tb_user")
public class User {

    1개 사용 위치
    @Id
    @Column(name = "id")
    private String id;

    1개 사용 위치
    @Column(name = "password")
    private String password;

    1개 사용 위치
    @Column(name = "name")
    private String name;

    kiku99
    @Builder
    public User(String id, String password, String name){
        this.id = id;
        this.password = password;
        this.name = name;
    }
}
```

DTO

```java
@Getter
@Setter
@NoArgsConstructor
public class UserDto {

    1개 사용 위치
    private String id;
    1개 사용 위치
    private String password;
    1개 사용 위치
    private String name;

    1개 사용 위치  kiku99
    public User toEntity() { return User.builder().id(id).password(password).name(name).build(); }
}
```

Spring Architecture - Login Example

Controller

```
kiku99
@Controller
@AllArgsConstructor
public class LoginController {

    1개 사용 위치
    private final UserService userService;

    kiku99
    @GetMapping("/login")
    public String login(HttpServletRequest request) { return "login"; }

    kiku99
    @GetMapping("/signUp")
    public String signUp(Model model){
        model.addAttribute( attributeName: "userDto", new UserDto());
        return "signUp";
    }

    kiku99
    @PostMapping("/signUp")
    public String signUp(@ModelAttribute("userDto") UserDto userDto){
        userService.insert(userDto);
        return "redirect:/login";
    }
}
```

Service

```
@Service
@RequiredArgsConstructor
public class LoginService {

    1개 사용 위치
    private final UserMapper userMapper;
    1개 사용 위치
    private final BCryptPasswordEncoder bCryptPasswordEncoder;

    1개 사용 위치   kiku99
    public void insert(UserDto userDto){
        userDto.setPassword(bCryptPasswordEncoder.encode(userDto.getPassword()));
        userMapper.save(userDto.toEntity());
    }
}
```