
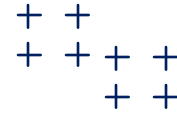
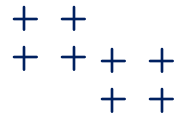


전북대 
소중해유(You)

RESTful API 개발: API 디자인과 기초 구현



전북대학교
SOFTWARE중심대학사업단

SW중심대학



정보통신기획평가원



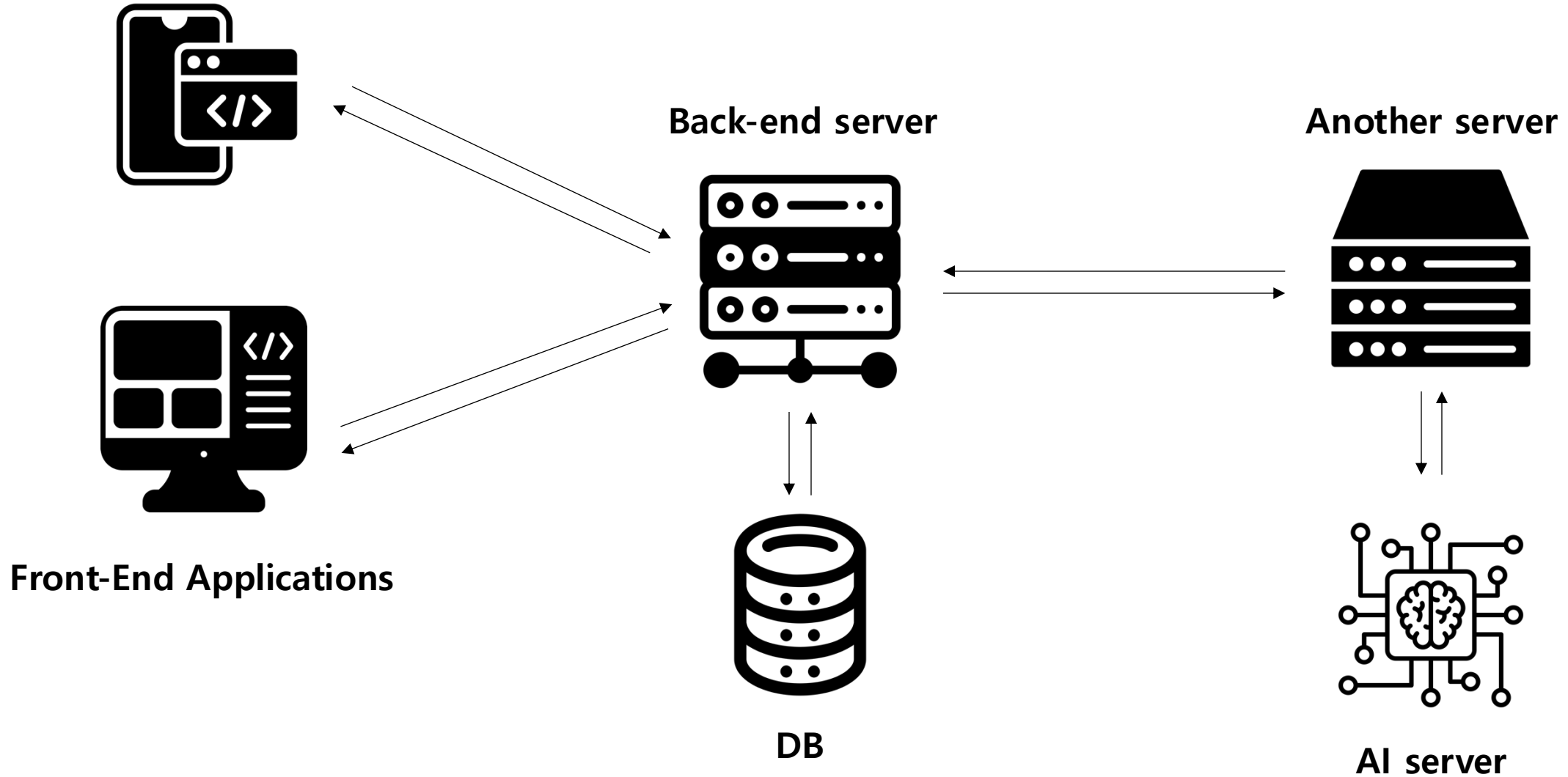
목차

1/
REST API란?

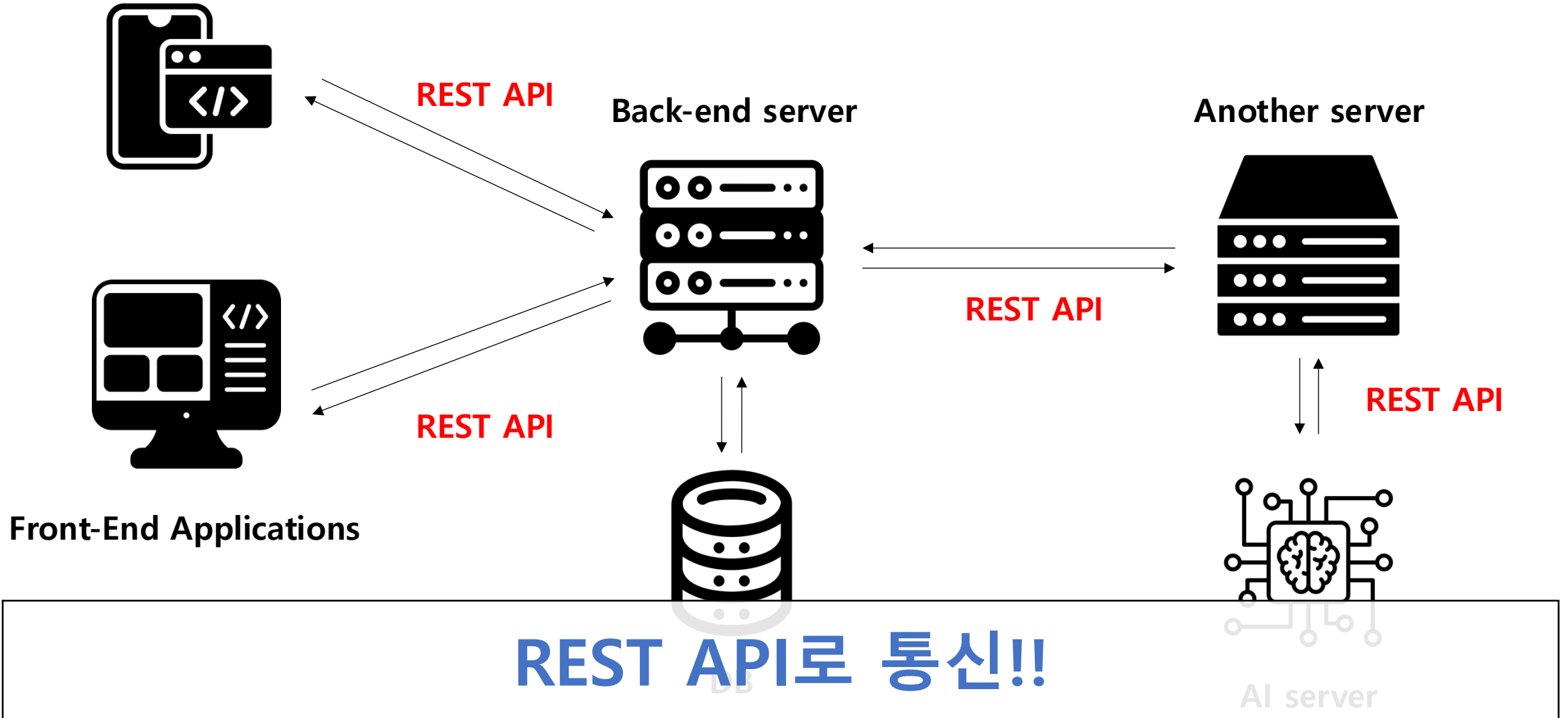
2/
REST API 구현 방식

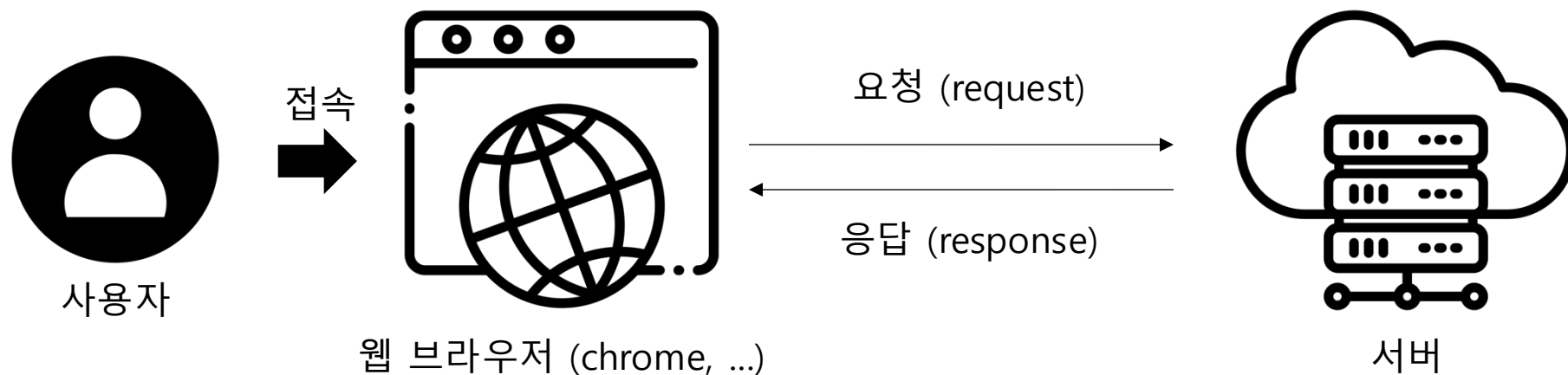
3/
REST API & Design Pattern

REST API란?



REST API란?

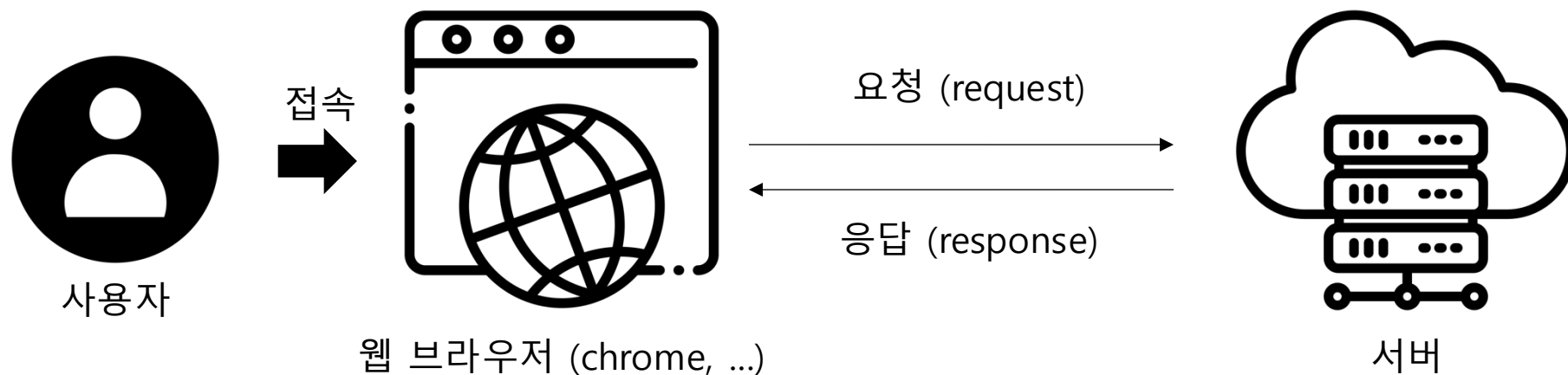




웹 API의 역할

- 서버와 데이터베이스안의 리소스에 접근
 - 데이터베이스의 정보를 누구나 열람하면 곤란하기에, 필요에 의해서만 열람 가능.
 - API는 접근 권한이 인가된 사람에게만 서버와 데이터베이스에 접근하도록 함.
- 모든 요청과 응답을 표준화
 - 아이폰을 쓰던 갤럭시 폰을 사용하든 상관없이 동일한 API를 사용하기 때문에 클라이언트의 요청과 서버의 응답을 하나의 API로 표준화 할 수 있음.

REST API란?



REST API

- REST API(Representational State Transfer)는 웹상에서 사용되는 여러 리소스를 HTTP URI로 표현하고, 해당 리소스에 대한 행위를 HTTP Method로 정의하는 방식
- 즉, 리소스(HTTP URI로 정의됨)를 어떻게 하겠다(HTTP Method + Payload)를 구조적으로 깔끔하게 표현하는 방법

REST API 설계 방식

- 리소스에 대한 행위는 HTTP Method(POST, GET, PUT, DELETE)로 표현
- /(슬래시)는 계층 관계를 나타낼때 사용
- URI 마지막 문자에 /(슬래시)를 사용하지 않음
- URI에 _(underscore)는 사용하지 않도록 함. 또한 영어 대문자보다는 소문자 사용을 권장
- 그리고 가독성을 위해서 긴 단어는 잘 사용하지 않음
- URI에 동사는 GET, POST와 같은 HTTP Method를 표현하기 때문에 동사가 아니라 명사를 사용 권장.
- URI에 파일의 확장자(예를들어 .json , .JPG)를 포함 시키지 않음.
- self-descriptiveness: RESTful API는 그 자체만으로도 API의 목적이 무엇인지 쉽게 알 수 있게 설계.

REST API의 Methods

- **GET**: 리소스 조회 (최근에는 Representation이라는 이름을 많이 사용한다.)
- **POST**: 요청 데이터 처리, 주로 등록에 사용
- **PUT**: 리소스를 대체, 해당 리소스가 없으면 생성
- **PATCH**: 리소스 부분 변경
- **DELETE**: 리소스 삭제

REST API의 Methods

- **GET**: 리소스 조회 (최근에는 Representation이라는 이름을 많이 사용한다.)
- **POST**: 요청 데이터 처리, 주로 등록에 사용
- **PUT**: 리소스를 대체, 해당 리소스가 없으면 생성
- **PATCH**: 리소스 부분 변경
- **DELETE**: 리소스 삭제



server



DB

- **GET**: search (select)
- **POST**: insert (insert)
- **PUT**: replace (modify)
- **PATCH**: modify (modify)
- **DELETE**: delete (delete)

GET Method

리소스 조회1 - 메시지 전달

```
GET /members/100 HTTP/1.1  
Host: localhost:8080
```

/members/100

```
{  
  "username": "young",  
  "age": 20  
}
```



GET Method

리소스 조회2 - 서버도착

```
GET /members/100 HTTP/1.1  
Host: localhost:8080
```

/members/100

```
{  
  "username": "young",  
  "age": 20  
}
```



클라이언트

GET



서버

GET Method

리소스 조회3 - 응답 데이터

응답 데이터

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 34

{
  "username": "young",
  "age": 20
}
```

/members/100

```
{
  "username": "young",
  "age": 20
}
```



클라이언트



Response



서버

GET Method

Q. GET 메서드로 요청할 때는 왜 query(쿼리 파라미터, 쿼리 스트링)를 써야 할까?

- GET 메서드도 POST 메서드와 같이 데이터를 담을 수 있는 메시지 바디가 존재한다. 하지만, GET 메서드의 메시지 바디를 확인하지 않는 서버가 대부분임으로 거의 사용되지 않는다.
- GET은 리소스를 요청하기 위해 만들어 졌다. URL의 URI가 자원이 위치한 곳을 나타낸다. 메시지 바디에 데이터를 실어야 하는 상황은 거의 없다. 따라서, 일반적인 서버에서는 GET 요청의 메시지 바디를 확인하지 않는다. 실제로 메시지 바디에 데이터가 있더라도 처리되지 않을 확률이 높다.
- 이렇게 설계해야 한다!

POST Method

리소스 등록1 - 메시지 전달

```
POST /members HTTP/1.1
Content-Type: application/json

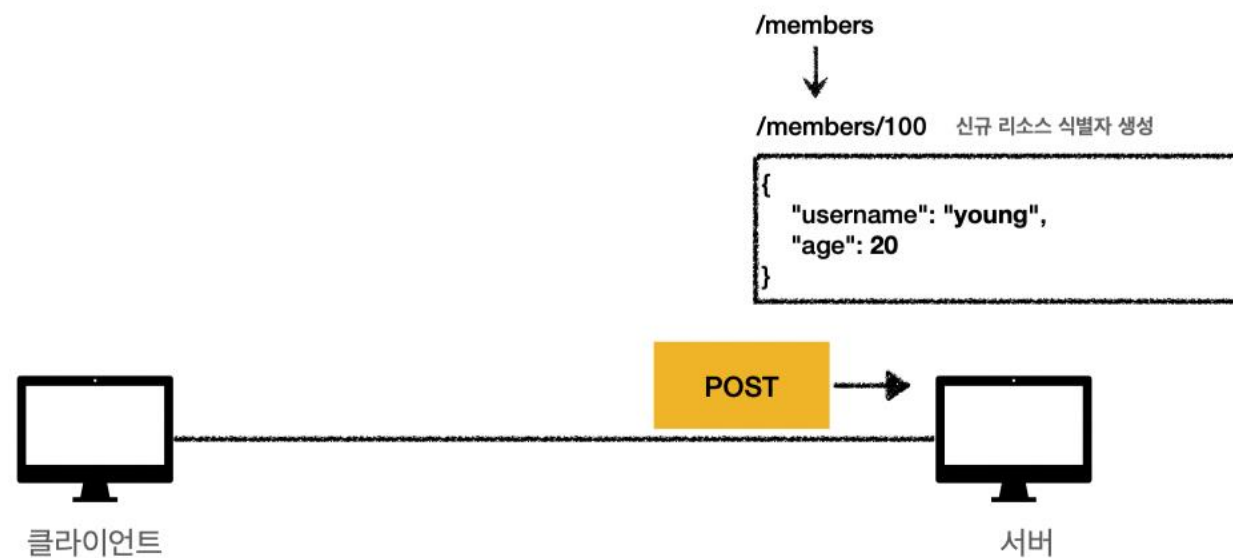
{
  "username": "young",
  "age": 20
}
```

/members



POST Method

리소스 등록2 - 신규 리소스 생성



POST Method

리소스 등록3 - 응답 데이터

응답 데이터

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Length: 34
Location: /members/100

{
  "username": "young",
  "age": 20
}
```

/members/100

```
{
  "username": "young",
  "age": 20
}
```



클라이언트



Response



서버

POST Method는 언제 사용할까?

- HTML 양식으로 입력된 데이터 블록을 리소스 로직에 제공
 - 예) HTML FORM에 입력한 정보로 회원 가입, 주문 등에서 사용
- 서버가 아직 식별하지 않은 새 리소스 생성
 - 예) 신규 주문 생성
- 기존 자원에 데이터 추가
 - 예) 한 문서 끝에 내용 추가하기
- 단순히 데이터를 생성하거나, 변경하는 것을 넘어서 프로세스를 처리해야 하는 경우
 - 예) 주문에서 결제완료 -> 배달시작 -> 배달완료 처럼 단순히 값 변경을 넘어 프로세스의 상태가 변경되는 경우. POST의 결과로 새로운 리소스가 생성되지 않을 수도 있다.
- 기타
 - 애매한 경우 POST를 사용한다. 예) JSON으로 조회 데이터를 넘겨야 하는데, GET 메서드를 사용하기 어려운 경우. POST는 사실상 모든지 할 수 있다.

PUT Method

PUT

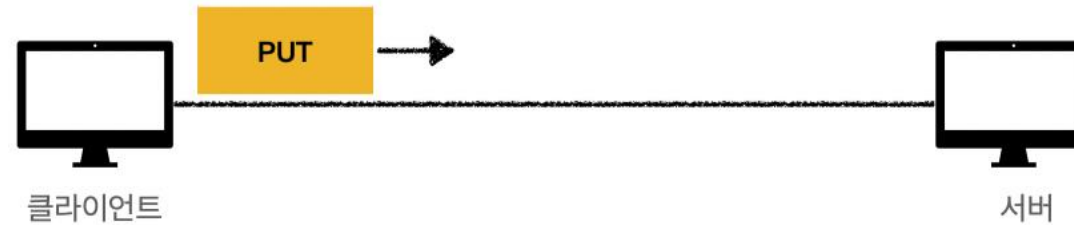
리소스가 있는 경우1

```
PUT /members/100 HTTP/1.1  
Content-Type: application/json
```

```
{  
  "username": "old",  
  "age": 50  
}
```

/members/100

```
{  
  "username": "young",  
  "age": 20  
}
```



리소스가 이미 존재하는 경우, 기존 데이터를 대체

PUT Method

PUT

리소스가 있는 경우2

리소스 대체

/members/100

```
{  
  "username": "old",  
  "age": 50  
}
```



리소스가 이미 존재하는 경우, 기존 데이터를 대체

PUT Method

PUT

리소스가 없는 경우1

```
PUT /members/100 HTTP/1.1  
Content-Type: application/json
```

```
{  
  "username": "old",  
  "age": 50  
}
```

이런 리소스는 없음
/members/100



리소스가 없는 경우, 새로운 데이터를 생성

PUT Method

PUT

리소스가 없는 경우2

신규 리소스 생성

/members/100

```
{  
  "username": "old",  
  "age": 50  
}
```

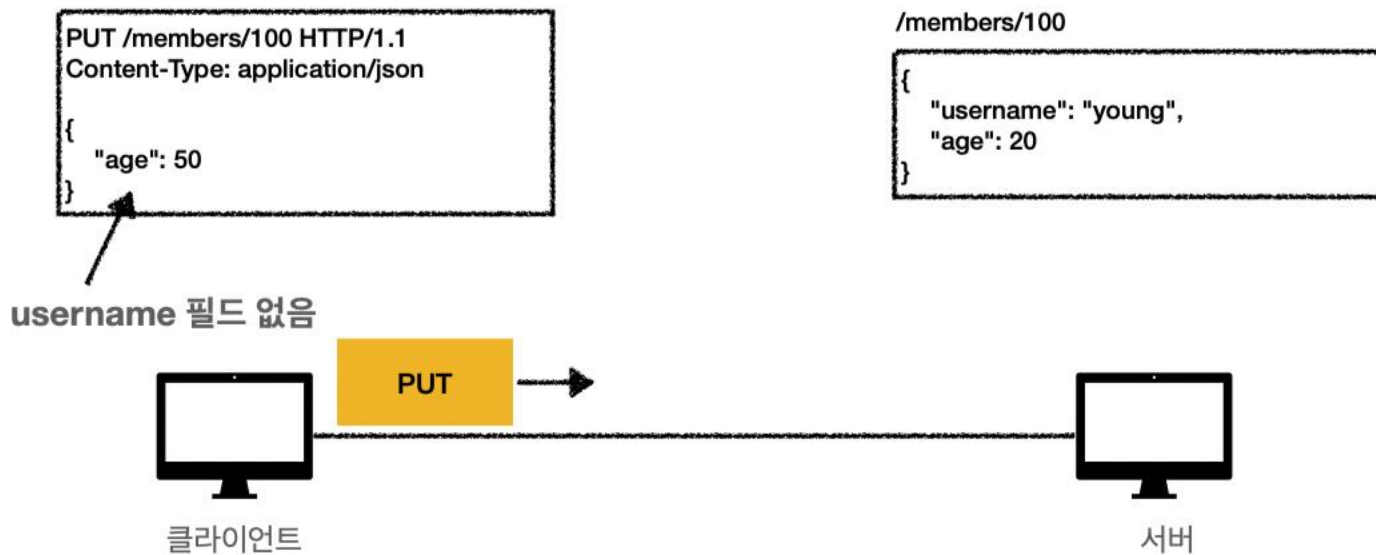


리소스가 없는 경우, 새로운 데이터를 생성

PUT Method

PUT

주의! - 리소스를 완전히 대체한다1

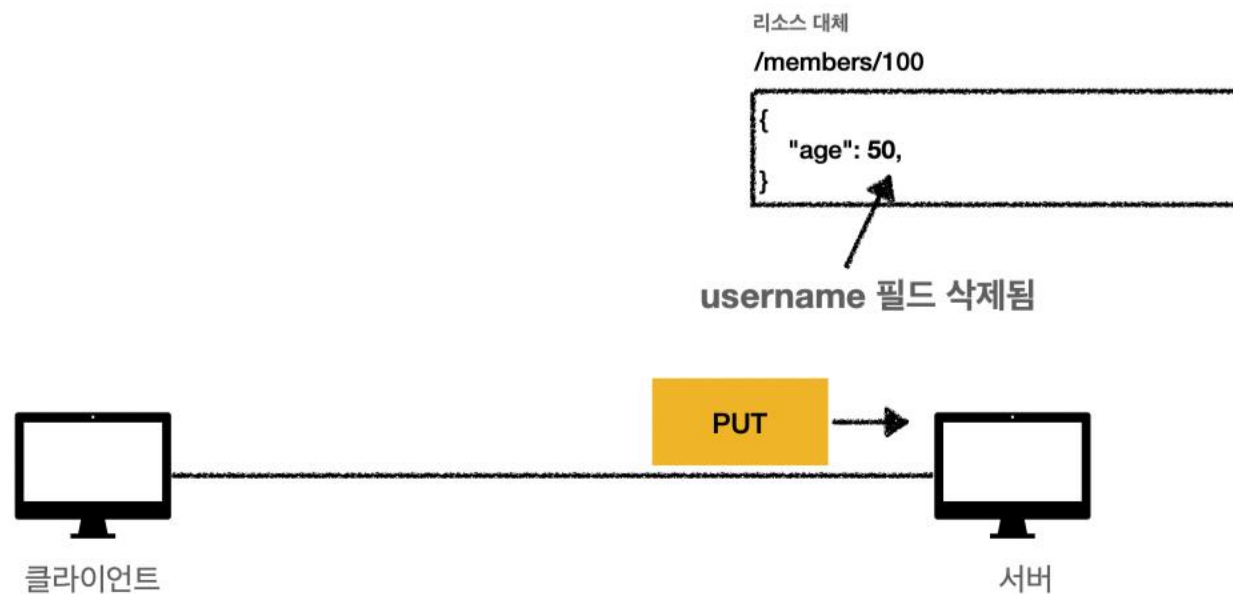


주의할 점은 리소스를 완전히 대체한다는 것이다. 부분적인 수정이 불가능하다.

PUT Method

PUT

주의! - 리소스를 완전히 대체한다2



주의할 점은 리소스를 완전히 대체한다는 것이다. 부분적인 수정이 불가능하다.

PATCH Method

PATCH

리소스 부분 변경1

```
PATCH /members/100 HTTP/1.1  
Content-Type: application/json
```

```
{  
  "age": 50  
}
```

username 필드 없음

/members/100

```
{  
  "username": "young",  
  "age": 20  
}
```



클라이언트

PATCH



서버

PATCH Method

PATCH 리소스 부분 변경2

리소스 부분 변경
/members/100

```
{  
  "username": "young",  
  "age": 50  
}
```

age만 50으로 변경



DELETE Method

DELETE

리소스 제거1

```
DELETE /members/100 HTTP/1.1  
Host: localhost:8080
```

/members/100

```
{  
  "username": "young",  
  "age": 20  
}
```



DELETE Method

DELETE 리소스 제거2

리소스 제거 ~~X~~
/members/100



Resource & Routing

Request URL:	https://cologger.shopping.naver.com/api/v1/validexpose/biz/MALL_NAME_LINK/expstrtr/001179?adCntsSeqs=8732273&inflowCd=sbml&cntsTypeCd=K54001&expsPage=1
Request Method:	GET
Status Code:	● 200 OK
Remote Address:	210.89.168.79:443
Referrer Policy:	unsafe-url

URL: https://cologger.shopping.naver.com/

Routing: api/v1/validexpose/biz/MALL_NAME_LINK/expstrtr/001179

data: adCntsSeqs=8732273 & inflowCd=sbml & cntsTypeCd=K54001 & expsPage=1

Resource & Routing

Request URL:	https://cologger.shopping.naver.com/api/v1/validexpose/biz/MALL_NAME_LINK/expstrtr/001179?adCntsSeqs=8732273 &inflowCd=sbml&cntsTypeCd=K54001&expsPage=1
Request Method:	GET
Status Code:	● 200 OK
Remote Address:	210.89.168.79:443
Referrer Policy:	unsafe-url

Data part는 항상 **Key=Value** 형식! 즉, 이는 **JSON과 호환** 가능하다!

data: adCntsSeqs=8732273 & inflowCd=sbml & cntsTypeCd=K54001 & expsPage=1

```
{  
  "adCntsSeqs": 8732273,  
  "inflowCd": "sbml",  
  "cntsTypeCd": "K54001",  
  "expsPage": 1  
}
```

REST API: Data Part

request 객체는 API를 컨트롤 하기 위한 메소드로 3가지를 포함

- param
- query
- body

REST API: Data Part

request 객체는 API를 컨트롤 하기 위한 메소드로 3가지를 포함

- param
 - 주소에서 포함된 변수를 담는다.
 - path parameter는 엔드포인트의 일부
 - 아래 예시의 엔드포인트에서 {user}와 {bicycleId}에 각각 그 값이 들어간다고 보면 됨!
 - /service/myresource/user/{user}/bicycles/{bicycleId}
 - 원하는 조건의 데이터들 혹은 하나의 데이터에 대한 정보를 받아올때 적절
 - 서버에서 Path Variable 로 부름.

REST API: Data Part

request 객체

- para

```
// @route    GET api/posts/:id
// @desc     'get' 메소드를 써서 파라미터 프로퍼티인 id값에 맞는 포스트를 가져오는 요청

router.get("/:id", auth, async (req, res) => { // 'id'라는 프로퍼티
  try {
    const post = await Post.findById(req.params.id);
    res.json(post);
  } catch (err) {
    res.status(500).send("Server Error");
  }
});
```

- 서버에서 Path Variable 로 부름.

보면 됨!

REST API: Data Part

request 객체는 API를 컨트롤 하기 위한 메소드로 3가지를 포함

- query
 - 엔드포인트에서 물음표(?) 뒤에 등장하며, 변수를 담는다.
 - /surfreport?days=3&units=metric&time=1400
 - key=value로 이루어져 있고 &으로 연결할 수 있다.
 - 조건을 줘서 정제된 결과물을 얻을 수 있다.
 - filtering, sorting, searching에 적절.
 - 서버에서 Query parameter로 칭한다.

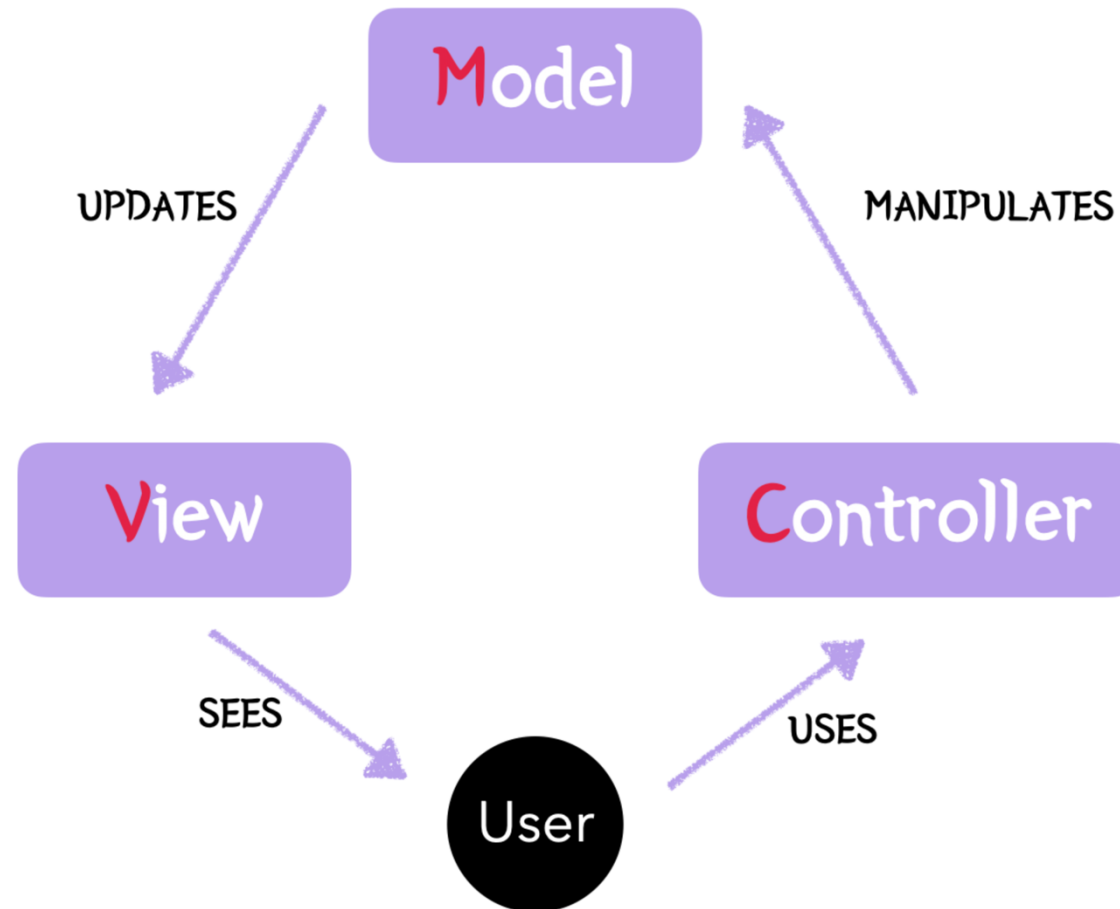
REST API: Data Part

request 객체는 API를 컨트롤 하기 위한 메소드로 3가지를 포함

- **body**
 - URL에는 보이지 않는 오브젝트 데이터(XML, JSON, MultiForm)를 담는다.
 - 보통 POST를 사용하여 JSON 오브젝트를 request body 안에 넣어 보낸다.
 - request body에 key-value 데이터 쌍을 포함한다.
 - 기본적으로 정의되어 있지 않아 `express.json()`, `express.urlencoded()` 등의 미들웨어를 사용해야 한다.

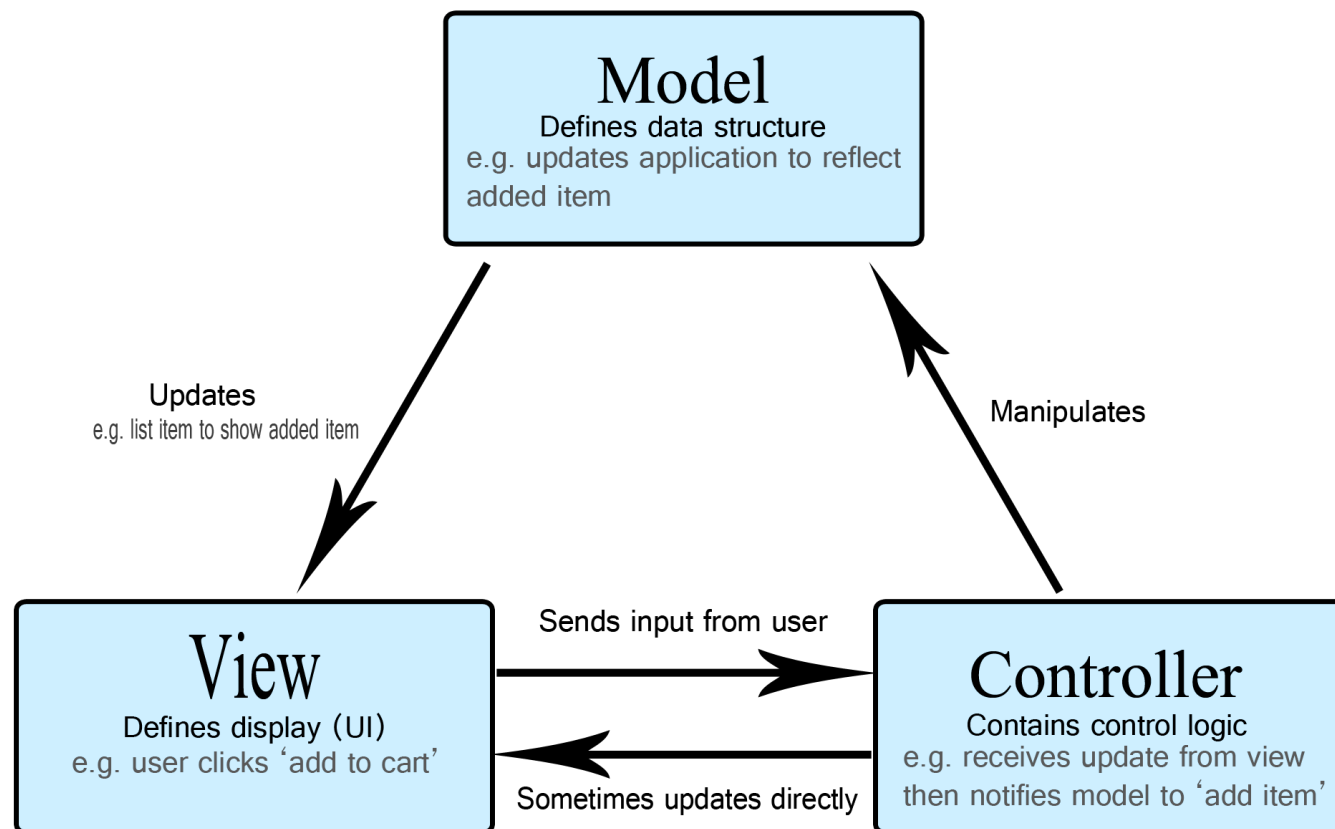
MVC Pattern?

- Model
- View
- Controller



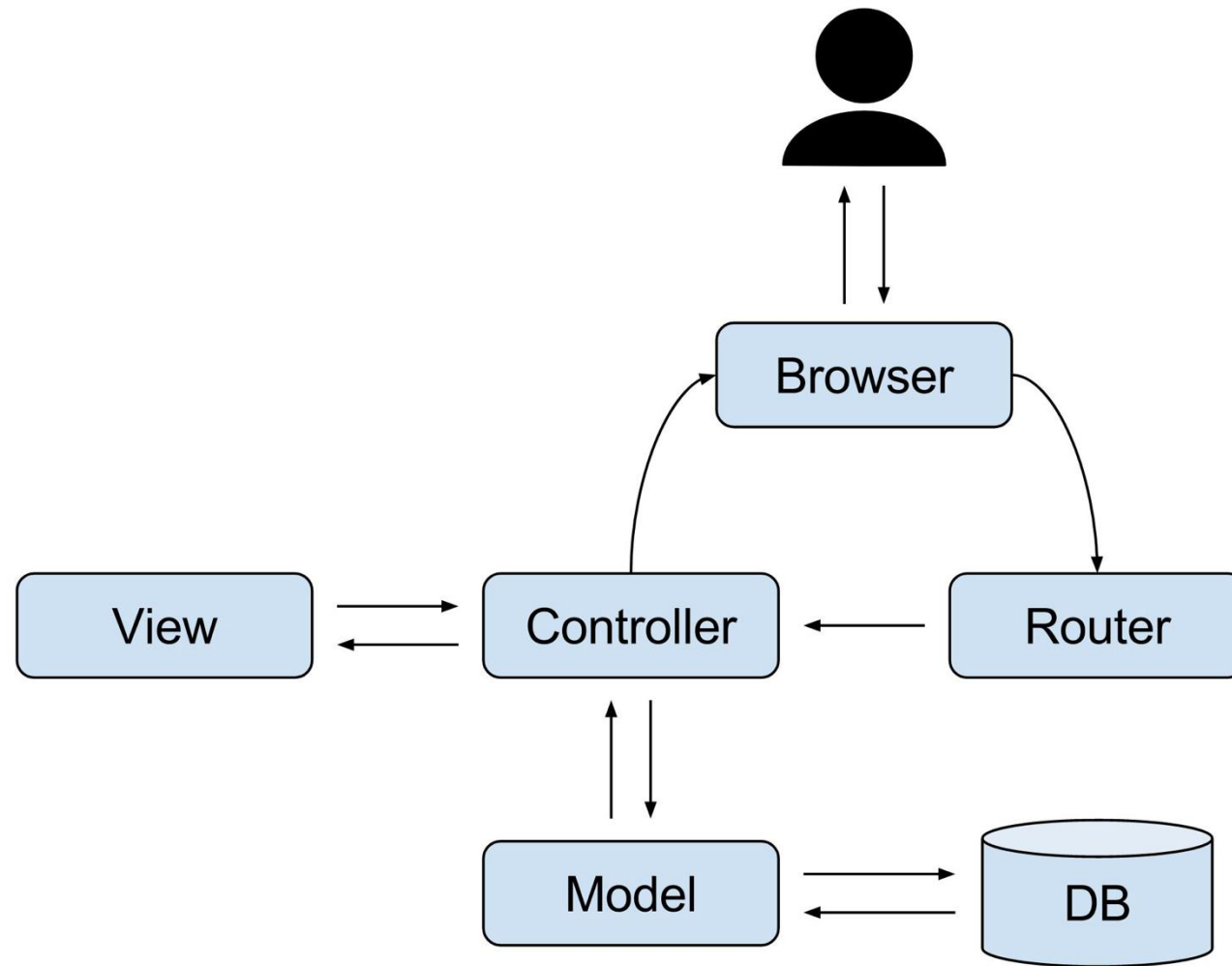
MVC Pattern?

- Model
- View
- Controller



MVC Pattern?

- Model
- View
- Controller



MVC Pattern?

- MVC Pattern은 Model/ View/ Controller로 구성된 Design Pattern
- MVC Pattern을 이용하게 되면 웹페이지를 구성하는 요소들끼리 간섭 없이 독립적으로 개발 가능!
- 효율적인 개발이 가능하다!

Model?

- 어플리케이션의 정보, 데이터를 나타내며, 즉, 초기화 된 상수나 값, 변수 (DB) 등을 의미
- 비즈니스 로직을 처리한 후 모델의 변경 사항을 컨트롤러와 뷰에 전달
- **앱이 포함해야할 데이터가 무엇인지를 정의**
 - 데이터를 처리하는 역할을 담당
 - Controller에서 명령을 받고 Database에서 데이터를 저장하거나 삭제/업데이트/변환 등의 작업을 수행!
- **모델은 3가지 규칙을 갖고 설계!**
 - 사용자가 편집하길 원하는 모든 데이터를 갖고 있어야 함!
 - View나 Controller에 대해서 어떤 정보도 알지 말아야 함!
 - 변경이 발생하면, 변경 통지에 대한 처리 방법을 구현해야 함!

View?

- **사용자에게 보여지는 부분, 즉 유저 인터페이스(User interface)를 의미**
- MVC 패턴은 여러 개의 뷰가 존재할 수 있으며, 모델에게 질의하여 데이터를 전달
- 뷰는 받은 데이터를 화면에 표시해주는 역할을 가지고 있으며, 모델에게 전달받은 데이터를 별도로 저장하지 않아야 함.
- 사용자가 화면에 표시된 내용을 변경하게 되면 모델에게 전달하여 모델을 변경해야 함.
- **View는 3가지 규칙을 갖고 설계!**
 - 모델이 가지고 있는 정보를 따로 저장해서는 안됨
 - 모델이나 컨트롤러와 같이 다른 구성 요소들을 몰라야 함.
 - 변경이 일어나면 변경통지에 대한 처리방법을 구현해야만 함.

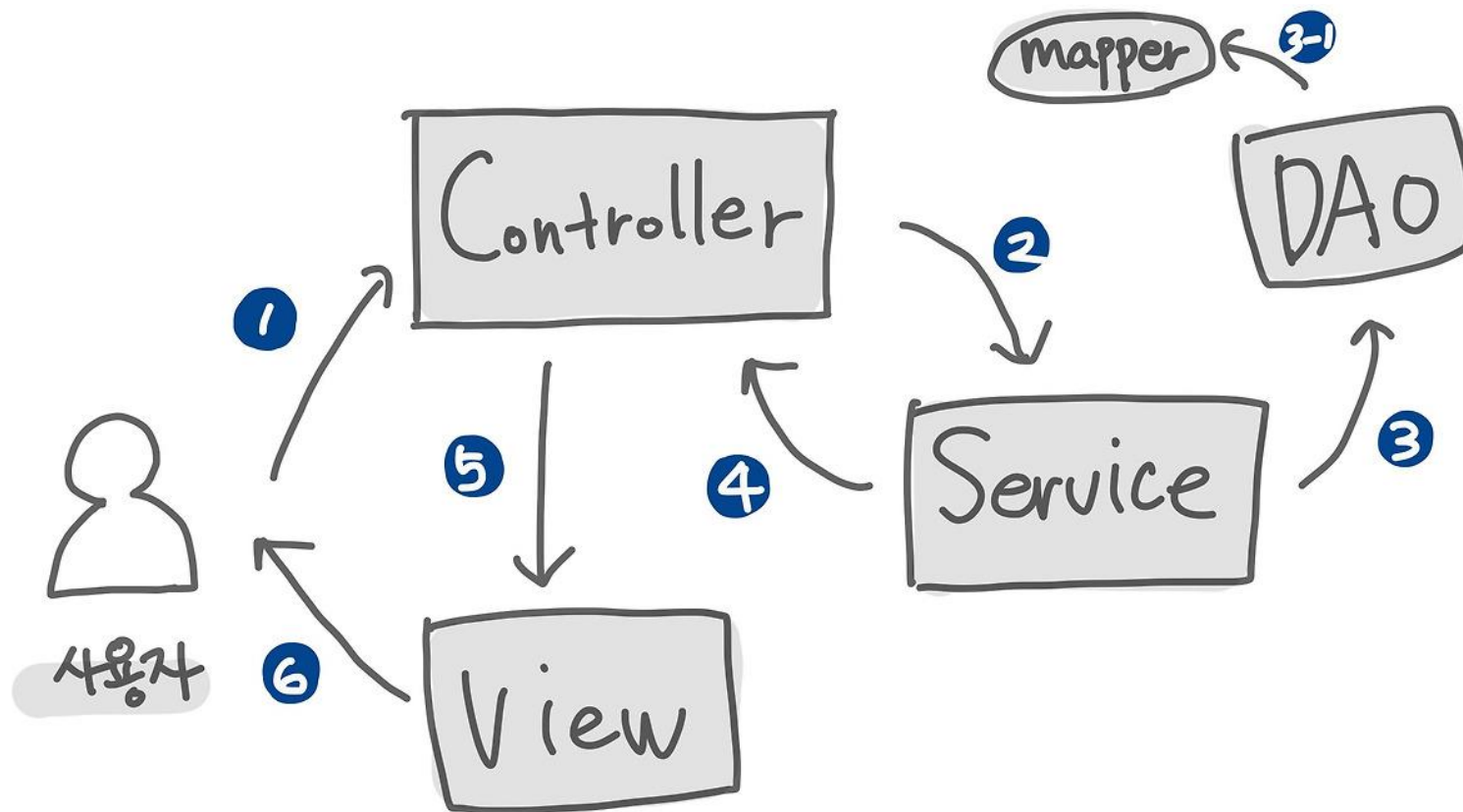
Controller?

- 앱의 사용자로부터의 입력에 대한 응답으로 모델 및/또는 뷰를 업데이트하는 로직
- 모델과 뷰 사이를 이어주는 브릿지(bridge) 역할을 수행
- 사용자가 접근하려는 URL에 따라 요청을 파악한 후, 그 요청에 맞는 **Model**을 의뢰하고, 데이터를 **View**에 반영하여 사용자에게 알려 줌.
- 예를 들어, 쇼핑 리스트는 항목을 추가하거나 제거할 수 있게 해주는 입력 폼과 버튼을 갖고 있음. 이러한 액션들은 모델이 업데이트되는 것이므로, 입력이 컨트롤러에게 전송되고, 모델을 적당하게 처리한 다음, 업데이트된 데이터를 뷰로 전송하도록 함.
- **컨트롤러는 2가지 규칙을 갖고 설계!**
 - 모델이나 뷰에 대해서 알고 있어야 합니다.
 - 모델이나 뷰의 변경을 모니터링 해야 합니다.

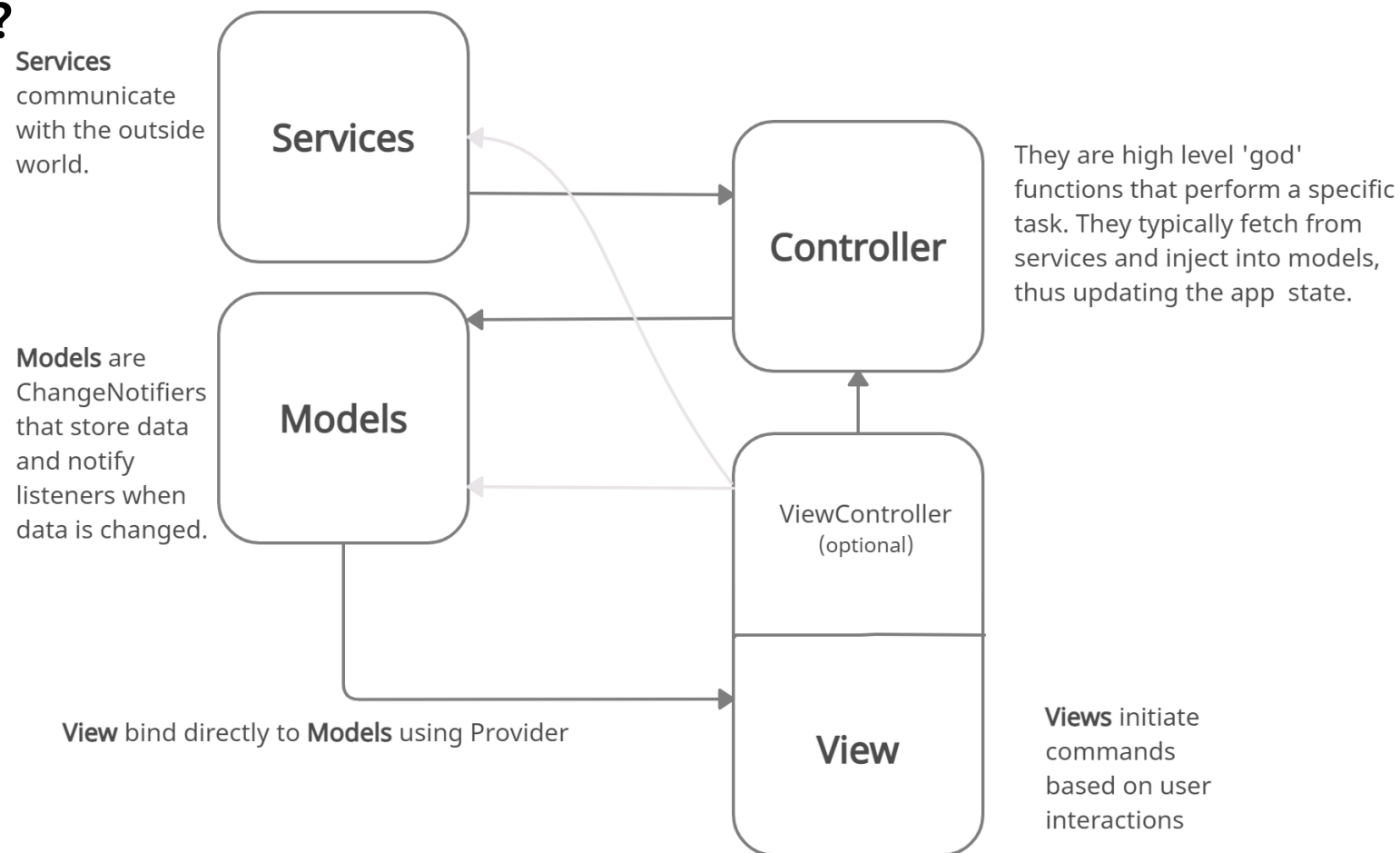
MVC Pattern?



MVCS Pattern!



MVCS Pattern?

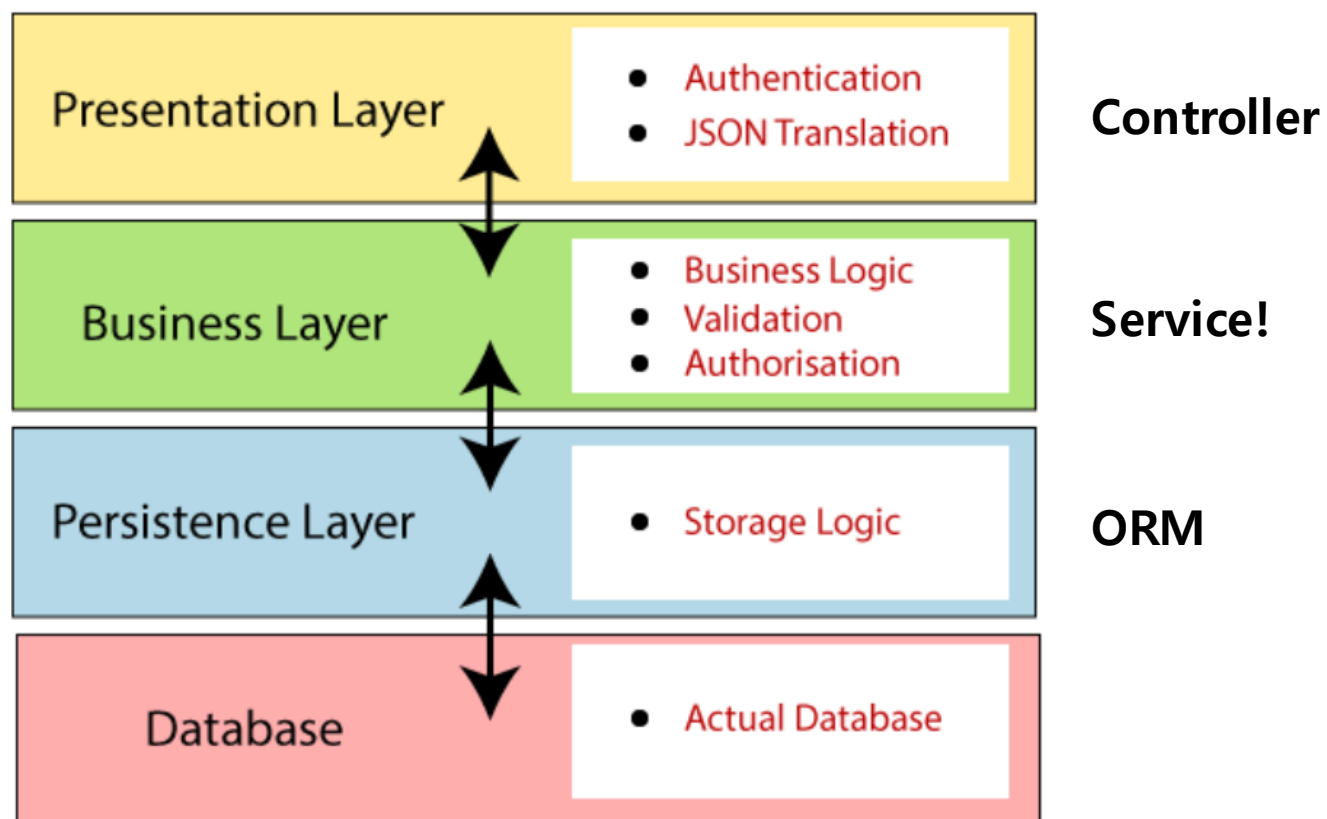


Service?

- 1) 웹사이트 URL로 접근하여 정보를 요청 (버튼클릭, 글 작성 등)
- 2) Controller에서 요청 정보를 받고, Service를 호출
- 3) Service에서 DAO를 호출하여 DB에 접근
- 4) Service의 작업이 완료된 후 Service를 호출했던 Controller로 돌아 옴
- 5) Controller는 데이터를 View에 전달
- 6) View에서 사용자에게 최종적으로 보여줄 화면을 생성

Service?

- Presentation Layer
- Business Layer
- Persistence Layer
- Database Layer



DAO?

- 데이터베이스에 데이터를 저장, 수정, 검색, 삭제를 하기 위해서 만들어지는 모델
- 데이터를 하나로 묶어야 될 경우 **데이터를 하나로 수집하는 역할**을 하는 바구니가 필요..!
- CRUD 동작을 가지고있는 클래스
- **비즈니스 로직**을 처리하는 클래스
 - 홈페이지 회원가입 예시
 1. 회원이 작성한 아이디 값을 저장
 2. 회원정보가 있는 데이터베이스 연결
 3. 데이터베이스에 회원이 작성한 아이디 값이 있는지 중복검사
 4. 회원의 아이디가 이미 있는지 없는지 여부를 데이터화 하여 저장
 5. 데이터베이스 연결 끊기
 6. View 영역에게 가공된 데이터 전달

DAO?

- 데이터베이스에 데이터를 저장, 수정, 검색, 삭제를 하기 위해서 만들어지는 모델
- 데이터를 하나로 묶어야 될 때 **DAO**는 역할을 하는 바구니가 필요..!
- CRUD 동작을 가지고있는 클래스
- **비즈니스 로직**을 처리하는 클래스
 - 홈페이지 회원가입 예시
 1. 회원이 작성한 아이디
 2. 회원정보가 있는 데이터
 3. 데이터베이스에 회원이 작성한 아이디 값이 있는지 중복검사
 4. 회원의 아이디가 이미 있는지 없는지 여부를 데이터화 하여 저장
 5. 데이터베이스 연결 끊기
 6. View 영역에게 가공된 데이터 전달

```
public class CarDTO {  
    public int carSn;  
    public String carName;  
    public int carPrice;  
    public String carOwner;  
    public int carYear;  
    public String carType;  
}
```

DAO?

- 데이터
- 데이터
- CRUD
- 비즈니스

```
car.carSn=carSn;
car.carName=carName;
car.carPrice=carPrice;
car.carOwner=carOwner;
car.carYear=carYear;
car.carType=carType;
carInfoPrint(car);
}

// Q. 매개변수로 자동차의 정보를 받아서 / 출력하는 / 메서드를 정의하세요.
1 usage
public static void carInfoPrint(CarDTO car){
    System.out.println(car.carSn+"\t"+car.carName+"\t"+car.carPr
}
```

4. 회원의 아이디가 이미 있는지 없는지 여부를 데이터와 하여 저장
5. 데이터베이스 연결 끊기
6. View 영역에게 가공된 데이터 전달

하는 모델

바구니가 필요..!

Examples) Controller

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
@Controller
```

```
public class KurienController {
```

```
    @Autowired
```

```
    private 은행원인터페이스 은행원;
```

```
    public String 저축(Model model) {
```

```
        String 계좌번호 = "0000-1111-2222";
```

```
        int 내돈 = 10000;
```

```
        입출금전표 전표 = new 입출금전표(계좌번호, 5000);
```

```
        try {
```

```
            int 거스름돈 = 은행원.입금(전표, 내돈);
```

```
            내돈 = 거스름돈;
```

```
            return "success";
```

```
        } catch(은행강도Exception e) {
```

```
            //도망친다.
```

```
            return "run";
```

```
        } catch(Exception e) { //예상하지 못한 오류
```

```
            return "그때 가서 생각한다";
```

```
        }
```

```
    }
```

```
}
```

Examples) DTO1

```
public class 입출금전표 {  
    private String 계좌;  
    private int 금액;  
  
    입출금전표(String 계좌, int 금액) {  
        this.계좌 = 계좌;  
        this.금액 = 금액;  
    }  
  
    public String get계좌() {  
        return 계좌;  
    }  
  
    public int get금액() {  
        return 금액;  
    }  
}
```

Examples) Service1

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

import java.time.LocalDate;

@Service
public class 국민은행원 implements 은행원인터페이스 {

    @Autowired
    private 입출금관리시스템인터페이스 입출금관리시스템;

    @Override
    public int 입금(입출금전표 전표, int 입금할돈) {
        if(전표 == null) {
            throw new 전표미입력Exception("저기 있는 입출금전표 작성해주세요.");
        }

        if(전표.get계좌() == null || 전표.get계좌().equals("") || 전표.get금액() <= 0) {
            throw new 전표오작성Exception("여기 여기 잘못 적으셨어요. 다시 적어주세요.");
        }

        if(전표.get금액() > 입금할돈) {
            throw new 입금할돈부족Exception("입금할 돈이 " + 전표.get금액() - 입금할돈 + "원 부족한데요?");
        }

        // 이 부분은 은행원이 입력할 내용
        String 입금지점 = "국민은행한국지점";
        LocalDate 입금시간 = LocalDate.now();

        입출금관리시스템입력양식 입력양식 = new 입출금관리시스템입력양식(전표.get계좌(), 전표.get금액(), 입금시간);

        try {
            입출금관리시스템.입금(입력양식);
        } catch(은행전산마비Exception e) {
            throw new 은행원당황Exception("죄송한데 은행 전산이 마비돼서요;; 지금은 입금이 안될 것 같습니다;;");
        }

        return 입금할돈 - 전표.get금액();
    }
}
```

Examples) DTO2

```
import java.time.LocalDateTime;
```

```
public class 입출금관리시스템입력양식 {  
    private String 계좌;  
    private int 금액;  
    private String 지점;  
    private LocalDateTime 시간;
```

```
    입출금관리시스템입력양식(String 계좌, int 금액, String 지점, LocalDateTime 시간) {  
        this.계좌 = 계좌;  
        this.금액 = 금액;  
        this.지점 = 지점;  
        this.시간 = 시간;  
    }
```

```
    public String get계좌() {  
        return 계좌;  
    }
```

```
    public int get금액() {  
        return 금액;  
    }
```

```
    public String get지점() {  
        return 지점;  
    }
```

```
    public LocalDateTime get시간() {  
        return 시간;  
    }
```

```
}
```

Examples) Service2

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

import java.sql.SQLException;

@Service
public class 국민은행입출금관리시스템 implements 입출금관리시스템인터페이스 {
    @Autowired
    private 입출금관리DAO인터페이스 입출금관리DAO;

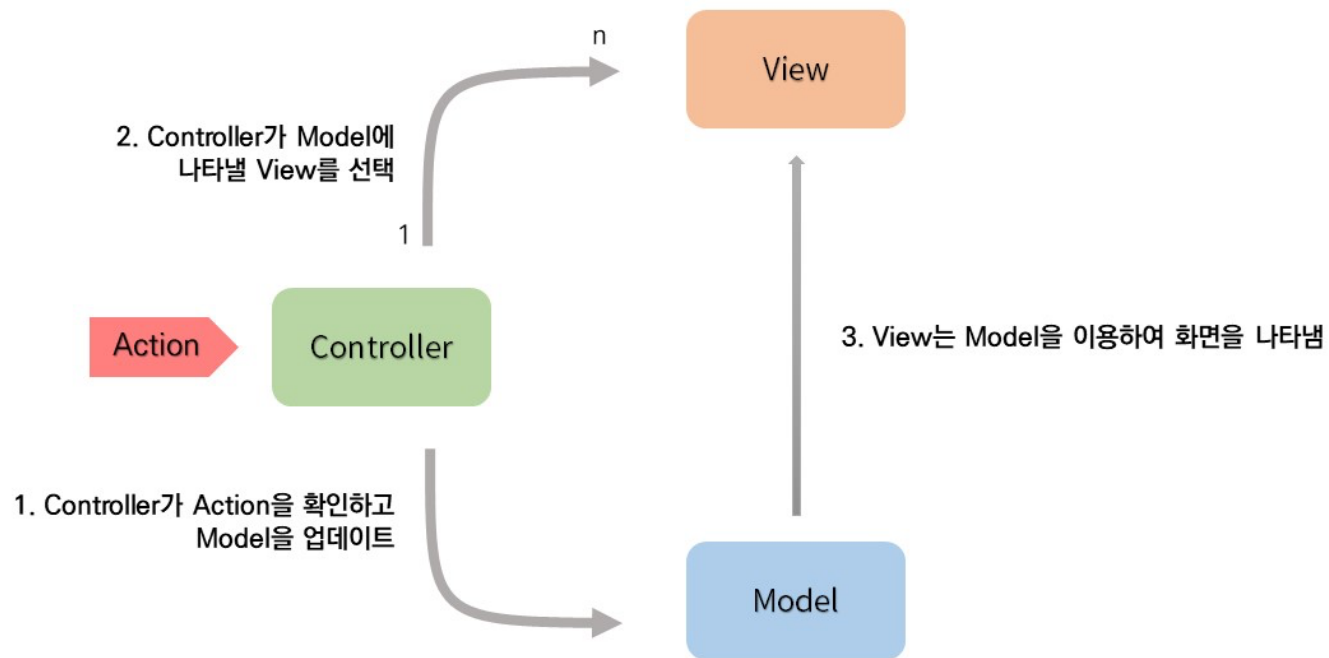
    @Autowired
    private 입출금안내서비스인터페이스 입출금안내서비스;

    @Override
    public boolean 입금(입출금관리시스템입력양식 입력양식) {
        try {
            입출금관리DAO.insert(입력양식);
        } catch (SQLException e) {
            throw 은행전산마비Exception("DB가 평하고 터져서 전산이 마비되었습니다.");
        }
        try {
            // 입출금안내서비스 클래스는 추가 작성하기가 귀찮아서 생략했습니다...
            입출금안내서비스.입금알림전송(입력양식.get시간() + " 입금 " + 입력양식.get금액() + " (" + 입력양식.get지점명());
        } catch (Exception e) {
            //입금알림이 나가지 않았다는 로그 발생
            //입금 알림이 나가진 않았으나, 치명적이지 않으므로 로그만 남김.
        }

        return true;
    }
}
```

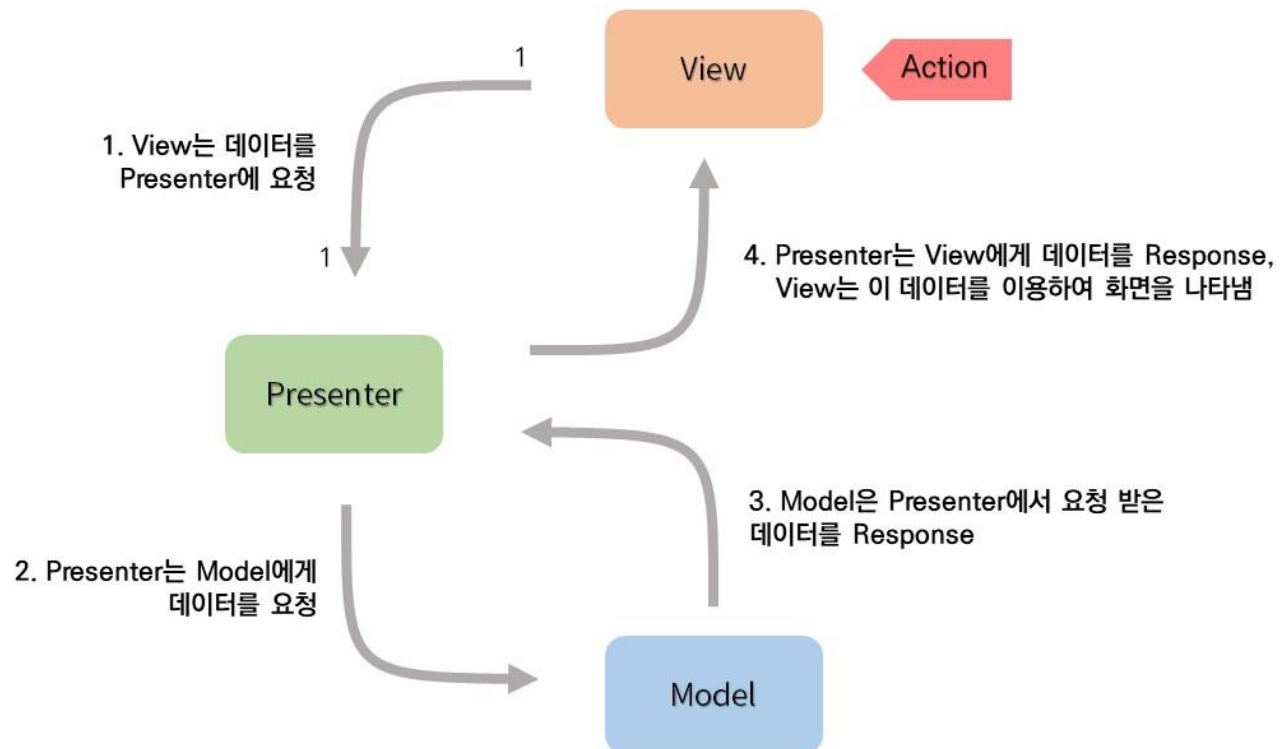
Other Design Patterns

- MVC
- MVP
- MVVM



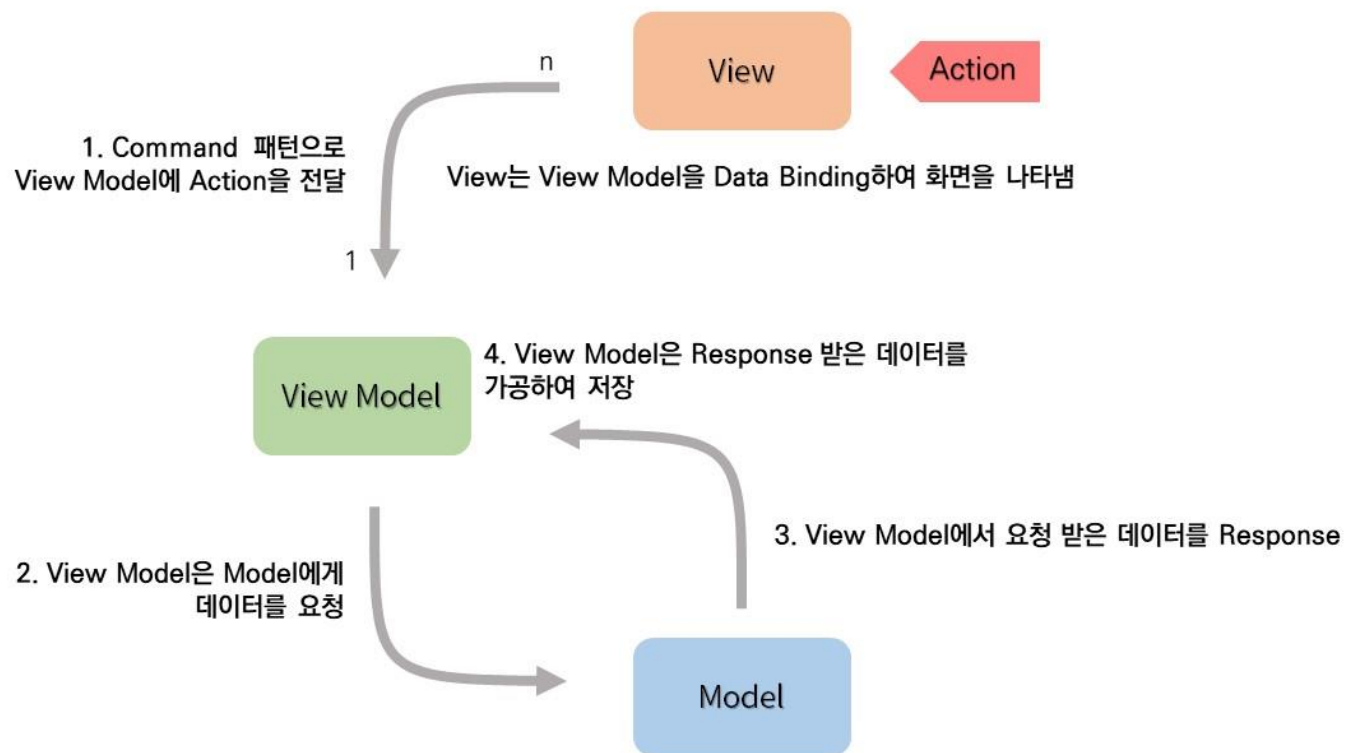
Other Design Patterns

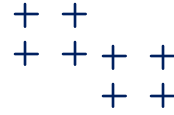
- MVC
- **MVP**
- MVVM



Other Design Patterns

- MVC
- MVP
- **MVVM**





감사합니다.

- 본 온라인 콘텐츠는 2024년도 과학기술 정보통신부 및 정보통신기획평가원의 'SW중심대학사업' 지원을 받아 제작되었습니다.
- 본 결과물의 내용을 전재할 수 없으며, 인용(재사용)할 때에는 반드시 과학기술정보통신부와 정보통신기획평가원이 지원한 'SW중심대학'의 결과물이라는 출처를 밝혀야 합니다.

