Spring REST API

558280-1
2026년 봄학기
5/7/2026
박경신

Why REST?

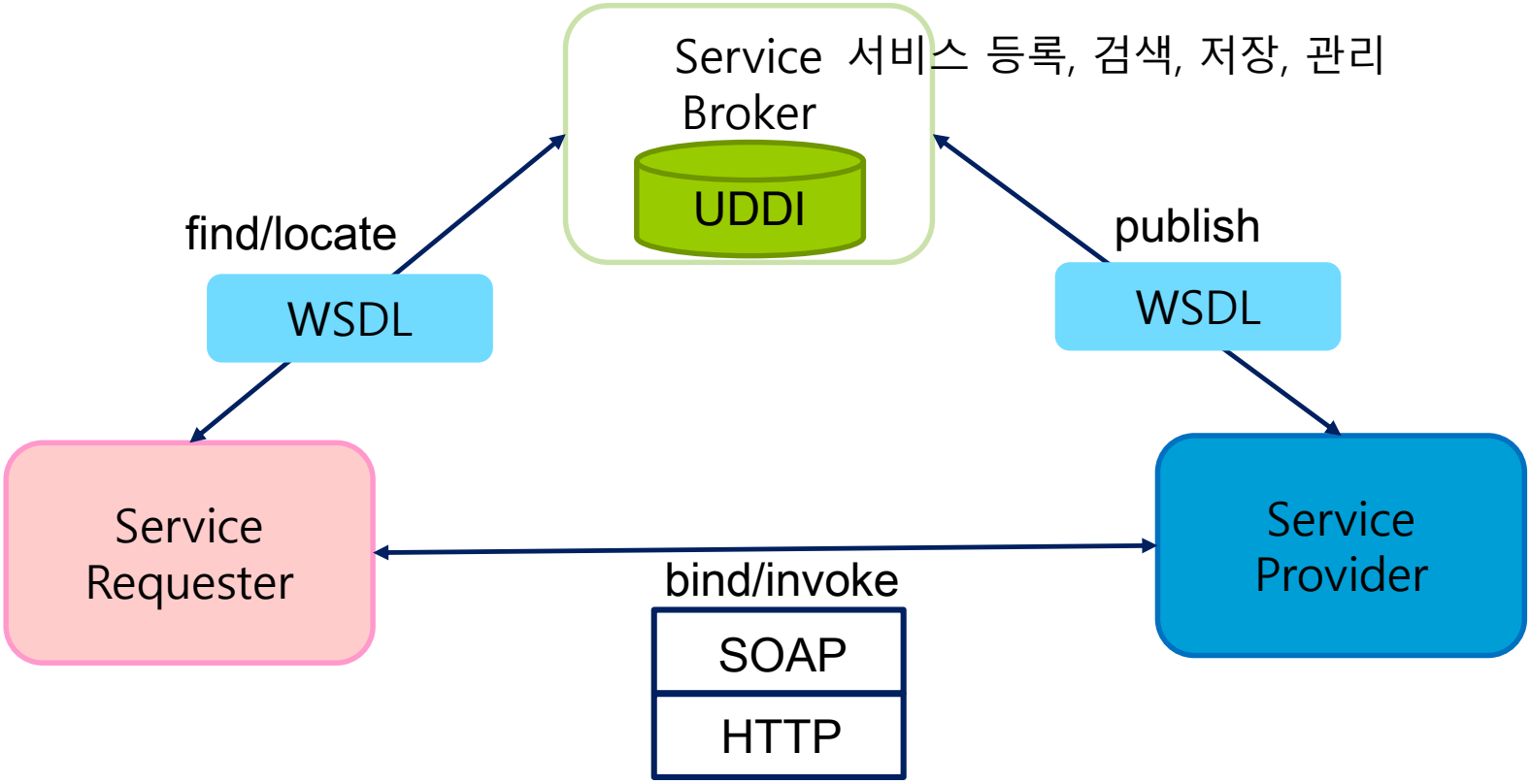
- 기존 방식의 문제
  - SOAP/WSDL 기반 Tight coupling – **클라이언트-서버 강하게 결합**
  - Overhead – XML 기반 메시지 (크고 느림) 모바일/웹 환경에 비효율적
  - 확장성 문제 – 상태 유지 (stateful) 서버 확장 어려움 (load balancing 제한)
  - 다양한 클라이언트 요구 – Web/ Mobile/ IOT에 동일 API 필요
- REST는 "경량 + 확장성 + 범용성"을 위한 설계 방식

SOAP와 REST

- SOAP와 REST는 네크워크 데이터 교환 메커니즘
  - 응용프로그램 개발에 다양한 Programming Language, Architecture 및 Platform을 사용할 수 있는데, 데이터 형식이 서로 다르기 때문에 이러한 다양한 기술 간에 데이터 공유가 어려움. SOAP와 REST는 이 문제를 해결하기 위해 개발됨.
- SOAP와 REST 유사점
  - 둘 다 서로 다른 응용프로그램들 간의 데이터 요청을 작성, 처리, 응답하는 방식에 대한 규칙과 표준을 설명함
  - 둘 다 표준화된 인터넷 프로토콜인 HTTP를 사용하여 정보를 교환함
  - 둘 다 안전하고 암호화된 통신을 위해 SSL(Secure Sockets Layer)/TLS(Transport Layer Security)을 지원함

SOAP

- SOAP (Simple Object Access Protocol)
  - SOA (Service Oriented Architecture)를 구현한 프로토콜

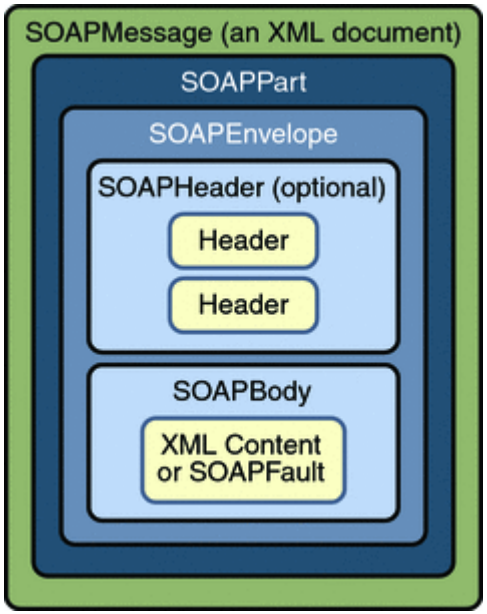


SOAP

- SOAP (Simple Object Access Protocol)
  - **HTTP, HTTPS, SMTP 등을 통해 XML 기반의 메시지를 네트워크 상에서 교환하는 프로토콜**
  - XML 기반의 헤더(메타 정보)와 바디(실제 정보)를 조합한 SOAP 메시지 패턴으로 설계
  - 서로 다른 **서비스**들 간의 연동을 목적으로 상호 이해 가능한 포멧의 메시지를 송수신함으로써 원격지에 있는 서비스 객체나 API를 자유롭게 사용하고자 함
- WSDL(Web Service Description Language)
  - **웹 서비스**에 대한 상세 정보가 기술된 XML 형식 언어
  - SOAP와 XML 스키마와 결합하여 웹 서비스 제공
- UDDI(Universal Description, Discovery and Integration)
  - WSDL 등록, 탐색, 바인딩을 위한 Registry **서비스 저장소**
  - **웹 서비스**에 대한 정보를 공개하고 발견하는 방법을 정의

SOAP

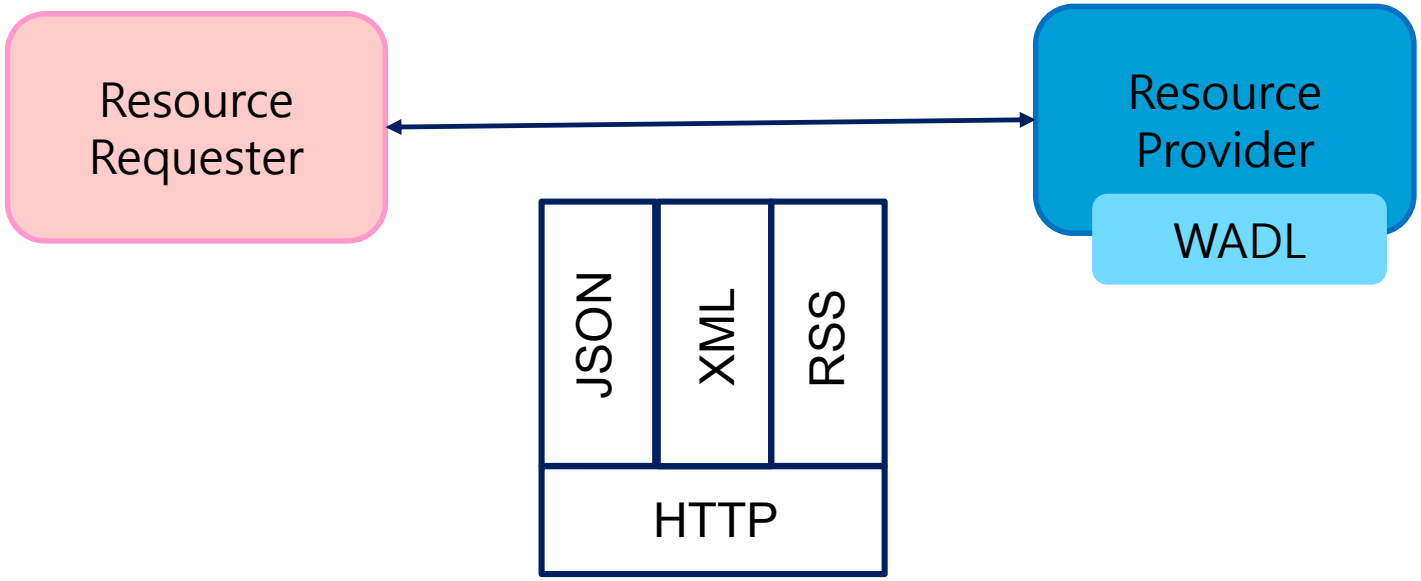
- SOAP 메시지
  - HTTP Headers
  - Root Part (SOAP Envelope)
  - Attachment Parts
- SOAP 문제
  - 복잡한 구조 - WSDL을 HTTP에 전달되기 무거움
  - 개발 환경 지원이 필요 – 웹 서비스 개발 난이도가 높음
  - UDDI(일종의 중계소)를 거쳐야만 서비스를 이용할 수 있음



https://docs.oracle.com/cd/E19575-01/819-3669/bnbhf/index.html

REST

- **REST (Representational State Transfer)**
  - ROA (Resource Oriented Architecture)를 따르는 웹 서비스 디자인 표준, Software Architecture 한 형식
  - 자원(이미지, 오디오, 텍스트 등)의 이름으로 구분하여, 해당 자원의 상태를 교환하는 것을 의미
  - HTTP **URI(Uniform Resource Identifier)**를 통해 자원을 명시하고 **HTTP Method(CRUD)**를 통해 자원을 교환하는 것



REST

- WADL (Web Application Description Language)
  - WADL (Web Application Description Language)은 REST 웹 서비스를 기술하기 위한 XML 기반의 메타데이터 언어
  - WADL은 REST API의 구조, 리소스, 메서드, 입력/출력 형식 등을 기술하는 데 사용됨
  - 현재는 OpenAPI (Swagger) (JSON/YAML)가 WADL을 대체하여 더 널리 사용되고 있음

REST

- REST 장점
  - HTTP 표준 프로토콜을 사용하는 모든 플랫폼에서 호환 가능
  - 서버와 클라이언트의 역할을 명확하게 분리
  - 여러 서비스 설계에서 생길 수 있는 문제를 최소화
  - SOAP보다 웹 서비스 개발이 더 쉬움
- REST 한계
  - 표준이 없음 (관례적으로 사용하는 것임)
  - 사용할 수 있는 HTTP Method가 4개임
  - RESTful 하게 만든 Method를 사용하여 속도가 느려질 수 있음

REST

- REST 구성 요소
  - **자원 (Resource)** – 모든 자원은 고유한 **URI(Uniform Resource Identifier)** 즉, URI로 식별되는 엔티티가 존재하고, 서버에 저장됨
  - **행위 (Action)** – **HTTP Method** (POST, GET, PUT, DELETE, PATCH ..)
  - **표현 (Representational)** – 클라이언트의 데이터 요청에 서버가 JSON, XML 등의 형태로 응답함

SOAP vs REST

항목	SOAP	REST
설계 철학	서비스 중심(SOA)	자원 중심(ROA)
메시지 형식	XML (WSDL 포함)	JSON, XML 등 다양
프로토콜	HTTP, SMTP 등 다양	HTTP 표준 Method 사용
복잡도	구조 복잡, 무거움	간단, 경량
서비스 설명	WSDL 필요	선택적으로 WADL 사용 가능
보안	WS-Security 기반 보안 지원	HTTPS 기반 보안 사용
유연성	프로토콜 독립적이지만, 무겁고 제한적	플랫폼 독립적, 언어와 플랫폼 자유로움
상태	상태 유지 가능	Stateless 기반

REST API

- REST API 정의
  - REST API는 **REST architecture 스타일**을 준수하여 설계된 웹 기반 API(Application Programming Interface)
  - REST 원칙을 충실히 구현한 웹 서비스를 흔히 **RESTful API**라고 부름
  - 최근 많은 API가 REST API로 제공되고 있음

REST API

- REST API 특징
  1. Server-Client (클라이언트-서버 구조)
     - 자원이 있는 쪽이 Server, 요청하는 쪽이 Client
     - 클라이언트와 서버 역할 분리로, 독립적인 개발과 배포 가능
  2. Stateless (무상태성)
     - 각 요청은 독립적이며, 서버는 이전 요청의 상태나 세션 유지 없음
     - 모든 요청에 필요한 정보(예: 인증 토큰)가 포함되어야 함
  3. Cacheable (캐시 처리)
     - HTTP 기반 캐시 매커니즘(예: Last-Modified, ETag) 사용 가능
     - 클라이언트는 서버의 응답을 캐싱하여 네트워크 효율성과 응답 속도를 개선할 수 있음
     - 응답에는 반드시 캐시 가능 여부를 명시해야 함

REST API

- REST API 특징
  4. Uniform Interface (일관된 인터페이스)
     - REST는 **표준화된 인터페이스(URI + HTTP Method + MIME Type)**을 사용
     - 리소스는 URI로 식별되고, 조작은 HTTP Method 로 수행
     - HTTP 프로토콜을 따르는 모든 플랫폼에서 사용 가능하게끔 설계
  5. Layered System (계층화된 시스템)
     - 클라이언트는 REST 서버의 내부 구조 (계층 수, 로드 밸런싱, 보안, 캐시 등)을 인식하지 않음
     - 서버는 보안, 성능, 확장을 위해 중간 계층 (Proxy, Gateway, Load Balancer 등) 지원 가능
  6. Code on Demand (Optional) (요청 시 코드 전송)
     - 필요 시 서버에서 클라이언트로 코드 (JavaScript) 전송
     - 클라이언트는 해당 코드를 동적으로 실행하여 기능을 확장할 수 있음

RESTful API Design Rule

- RESTful URI 설계 원칙
  - **URI(Uniform Resource Identifier)는 자원(Resource)을 표현함**
    - **https://example.com/pets/3**
    - **Resource type: pets**
    - **Resource id: 3**
  - **자원에 대한 조작은 HTTP Method(CRUD)**를 통해 표현함
    - 자원에 대한 CRUD는 URI가 아니라 HTTP Method로 표현
    - **GET** /pets/3 - ID 3번 반려동물 조회
    - **POST** /pets - 새로운 반려동물 등록
    - **PUT** /pets/3 - ID 3번 반려동물 정보 수정
    - **DELETE** /pets/3 - ID 3번 반려동물 삭제
  - **리소스 조작을 위한 메시지 표현**
    - 요청/응답 메시지 HEADER를 통해 콘텐츠 형식(content-type)을 명시함
    - 대표적인 MIME 형식으로는 **HTML, XML, JSON, TEXT**가 있음

RESTful API Design Rule

- RESTful URI 설계 원칙
  - 자원은 명사로 표현 (동사 X)
    - /books, /orders 복수형은 컬렉션 표현
  - 소문자만 사용 (대문자 X)
    - 대소문자에 따라 다른 리소스로 인식되므로 대문자 사용을 피해야 함
    - /members/postComments (X)
    - /members/post-comments
  - 계층 관계는 슬래시(/) 사용
    - /members/dooly/hobby
  - 마지막 문자로 slash(/)를 포함하지 않음
    - /members/dooly/ (X)
  - Underscore(_) 대신 dash(-) 사용
    - /members/post_comments (X)
    - /members/post-comments
  - 파일 확장자 사용 금지
    - /members/soccer/345/photo.jpg (X)

REST API Design Rule

- RESTful URI 설계 원칙
  - URI는 정보의 자원을 표현함 (자원 이름은 동사보다 명사 사용)
  - 자원에 대한 행위는 HTTP Method로 표현해야 함
  - 자원 삭제 예시
    - GET http://example.com/members/delete/3 (잘못된 표현)
    - DELETE http://example.com/members/3 (id=3 멤버 자원을 삭제)
  - 자원 추가 예시
    - GET http://example.com/members/insert/3 (잘못된 표현)
    - POST http://localhost:8080/members/3 (id=3 멤버 자원을 추가)
  - 자원 정보 조회 예시
    - GET http://example.com/members/show/3 (잘못된 표현)
    - GET http://example.com/members/3 (id=3 멤버 자원을 조회)

REST API Design Rule

- HTTP Method 매핑 예시
  - POST – 요청 데이터 처리, 자원 등록에 사용
  - GET – 자원 조회
  - PUT – 자원 전체 수정
  - DELETE – 자원 삭제

CRUD	SQL
Create	insert
Read	select
Update	update
Delete	delete

행위	HTTP Method	URI 예시	설명
조회	GET	/members	사용자 목록 조회
조회	GET	/members/{id}	특정 사용자 조회
생성	POST	/members	사용자 생성
수정	PUT	/members/{id}	사용자 정보 전체 수정
삭제	DELETE	/members/{id}	사용자 삭제

REST API Design Rule

- HTML <form>은 기본적으로 GET과 POST만 지원
  - 다른 메서드 사용 시 폼 내부에 숨겨진 필드 또는 _method로 우회
  - REST 설계 원칙을 따르기 위해 URI 구조화와 메서드 의미 부여 필요
    1. Step1 사용자를 위한 Form 제공 (GET)
    2. Step2 Form 제출을 통한 실제 상태 변경 요청
  - 자원 추가(Create) 예시
    1. GET /pets/3/reviews/**new**
    2. POST /pets/3/reviews (pet #3 새로운 리뷰 자원 추가)
  - 자원 수정(Update) 예시
    1. GET /pets/3/reviews/4/**edit**
    2. POST /pets/3/reviews/4?_method=PUT (pet #3의 리뷰#4 자원 수정)
  - 자원 삭제(Delete) 예시
    1. GET /pets/3/reviews/4/**delete**
    2. POST /pets/3/reviews/4?_method=DELETE (pet #3의 리뷰 #4 자원 삭제)

Set HTTP Headers

- Content-Location
  - Content-Location은 응답 본문이 표현하는 **자원 URI**를 명시
  - 주로 POST 요청으로 새 자원 생성시, **해당 자원 위치**를 응답에 명시
    - POST /members 요청 -> Content-Location: /members/1 응답
    - GET 요청은 멱등(idempotent) f(f(x)) = f(x)
      - GET /members/1 요청을 여러 번 수행해도 동일한 결과로 응답함
    - POST 요청은 비멱등(non-idempotent)
      - POST /members { "name": "K" } 요청은 매번 새로운 자원을 생성하며 , 각각의 요청은 고유한 식별자 /members/1, /members/2 ... 자원 반환
  - 클라이언트가 요청한 자원과 다른 표현이 반환될 경우에도 사용
    - GET /documents/foo 요청 헤더에 따라 Content-Location에 **다른 표현**을 명시

Request Header	Response Header
Accept: application/json, text/json	Content-Location: /documents/foo.json
Accept: application/xml, text/xml	Content-Location: /documents/foo.xml
Accept: text/plain, text/*	Content-Location: /documents/foo.txt

Set HTTP Headers

- Content-Type
  - 클라이언트와 서버 간 HTTP 메시지에서 **본문(body)의 미디어타입 (MIME type)**을 명시
  - 데이터를 받는 측(서버/클라이언트)이 어떻게 데이터를 처리해야 할지 결정하는 기준

Content-Type 설정	설명
클라이언트->서버 Request Header	데이터 타입 지정 (예: 이미지 업로드 등) 바이너리/텍스 타입 식별
서버->클라이언트 Response Header	반환 데이터 타입 지정 (예: HTML, JSON, image 등)

  - 주요 Content-Type 유형
    - Content-Type: text/html 텍스트 파일
    - Content-Type: application/json JSON 사용
    - Content-Type: application/x-www-form-urlencode HTML<form> 사용
    - Content-Type: multipart/form-data 파일업로드 등 다중 파트 전송

Use HTTP Status Code

- HTTP 상태 코드 반환 원칙

  HTTP/1.1 400 Bad Request {

  "msg" : "'name'(body) must be Number, input 'name': test123"

  }

  - HTTP 응답은 상태 코드(Status Code)로 처리 결과를 표현해야 함
  - 상태 코드만으로 요청 성공/실패 여부와 원인 파악이 가능해야 함
    - 성공은 2xx로 응답할 것
    - 실패는 4xx로 응답할 것
    - 서버오류(5xx)는 내부 로깅은 하되, 사용자에게 나타내지 말 것
  - 상세 오류 메시지는 본문에서 제공 (즉, 오류에 대한 설명 메시지), 그리고 필요 시 해당 에러를 확인할 수 있는 참조 link 를 포함할 것
  - 상태 코드는 응답 바디(body)에 중복 표시하지 않음

Use HTTP Status Code

Code	Description
200	**OK 클라이언트의 요청을 정상적으로 수행함**
201	**Created** 클라이언트가 어떠한 리소스 생성을 요청, **해당 리소스가 성공적으로 생성됨** (POST를 통한 리소스 생성 작업 시)
400	**Bad Request** 클라이언트의 요청이 부적절할 경우 사용하는 응답 코드
401	**Unauthorized** 클라이언트가 인증되지 않은 상태에서 보호된 리소스를 요청했을 때 사용하는 응답 코드 (로그인하지 않은 유저가 로그인 했을 때, 요청 가능한 리소스를 요청했을 때)
403	**Forbidden** 유저 인증상태와 관계없이 응답하고 싶지 않은 리소스를 클라이언트가 요청했을 때 사용하는 응답 코드 (400이나 404 사용을 권고)
404	**Not Found**
405	**Method Not Allowed** 클라이언트가 요청한 리소스에서는 사용 불가능한 Method를 이용했을 경우 사용하는 응답 코드
301	**Moved Permanently** 클라이언트가 요청한 리소스에 대한 URI가 변경되었을 때 응답 코드 (Location Header에 변경된 URI를 적어줘야 함)
500	**Internal Server Error** 서버 내부 오류 응답 코드

Use HATEOAS

- 잘 설계된 REST API 란 Level 3를 만족시켜야 함
  - 이를 위해서 **HATEOAS(Hypermedia As The Engine Of Application State)** 개념을 도입해야 함

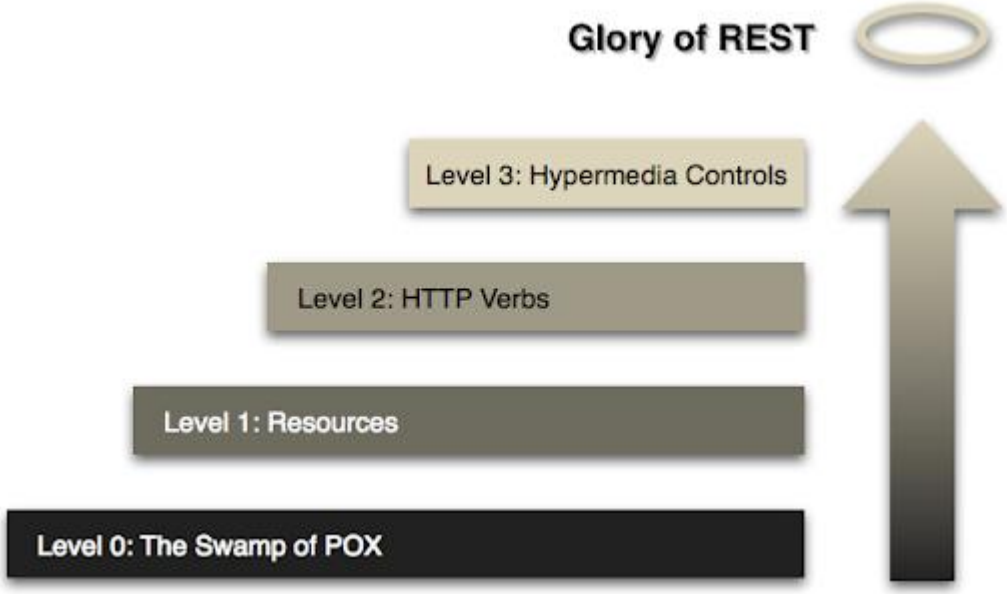


Image from https://martinfowler.com/articles/richardsonMaturityModel.html

Use HATEOAS

- REST API 구현 단계별 예시

Richardson maturity model with API JSON response type

Lvl	Name	Example request	Response
0	Plain of XML (POX)	method: POST URI: /movie/	JSON
1	Resources with specific URI for action	method: POST URI: /movie/1/delete	JSON
2	Resources + HHTP methods	method: DELETE URI: /movie/1	JSON
3	HATEOAS (level 2 + extra links to navigate through API)	method: DELETE URI: /movie/1	JSON with HAL (extra links)

Image from https://grapeup.com/blog/how-to-build-hypermedia-api-with-spring-hateoas/

Use HATEOAS

- Richardson Maturity Model (REST 아키텍처 성숙도 모델)
  - Level 0 - The Swap of POX
    - API 구현은 HTTP 프로토콜을 사용하지만 전체 기능을 활용하지는 않음. 또한 리소스에 대한 고유 주소가 제공되지 않음.
  - Level 1 - Resources
    - 리소스에 대한 고유 식별자가 있지만, 리소스에 대한 각 작업(action)에는 고유한 URL이 있음
  - Level 2 - HTTP Verbs
    - 동작(action) 표현을 위해, 동사(verbs) 대신 HTTP 메서드(GET, POST, PUT, DELETE)를 사용함 (예시: URL .../delete 대신 DELETE 메서드)
  - Level 3 - Hypermedia Controls
    - **HATEOAS(Hypermedia As The Engine Of Application State)**. 간단히 말해서 리소스에 하이퍼미디어를 도입한 것임. 이를 통해 알리는 응답에 가능한 작업(action)에 대해 링크를 배치함으로써 API를 통해 탐색이 가능함.

Use HATEOAS

- HATEOAS(Hypermedia As The Engine Of Application State)
  - REST 아키텍처에서 응답 메시지 안에 **관련된 리소스의 URI 링크들**을 포함
  - 클라이언트가 서버에서 제공한 링크를 따라 **다음 작업을 동적으로 탐색 가능**
  - API 사용자는 별도 문서 없이도 **자원의 상태 전이(state transitions)** 흐름을 이해할 수 있음
  - 리소스 상태에 따라 가능한 동작 (예, 수정, 삭제 등)을 동적으로 안내하는 것이 핵심

Use HATEOAS

- 전형적인 REST API 응답 예시 (HATEOAS 적용 전)
  - POST /members {"name": "K"}이라는 요청을 보내면, 응답으로

```
{
  "id": 1,
  "name": "K"
}
```

기존 전형적인
REST API
JSON 응답

Use HATEOAS

- HATEOAS 적용 응답 예시
  - POST /members {"name": "K"}이라는 요청을 보내면, 응답으로
  - 링크를 통해 클라이언트가 가능한 동작들을 직관적으로 파악할 수 있음

```
{
  "id": 1,
  "name": "K",
  "links": [
    { "rel": "self", "href": "http://example.com/members/1", "method": "GET" },
    { "rel": "delete", "href": "http://example.com/members/1", "method": "DELETE" },
    { "rel": "update", "href": "http://example.com/members/1", "method": "PUT",
      "more_info": "http://example.com/docs/member-update",
      "body": { "name": "{The value to be modified}" }
    },
    { "rel": "member.posts", "href": "http://example.com/members/1/posts", "method": "GET" }
  ]
}
```

HAL(Hypertext Application Language) 추가

Use HATEOAS

- HATEOAS의 장점
  - API 탐색 가능성 - 클라이언트가 서버 제공 링크를 따라 자율적으로 상태 전이를 수행
  - 계층 독립성 향상 - 클라이언트는 서버의 내부 구조를 몰라도 됨
  - 문서 의존도 감소 - 응답 자체가 API 사용법을 내포

Pagination and Sorting

- Pagination
  - Collection 리소스에 대해, 한 번에 모든 결과를 응답하지 않고 적당한 크기로 데이터 셋을 나눠서 응답 받을 수 있음
- Sorting
  - Collection 리소스에 대해, 리스트를 클라이언트의 요청에 맞게 정렬해서 응답 받을 수 있음
- Pagination과 Sorting은 Spring JPA와 Pageable로 구현
  - /api/people – sort by id descending (default) & pagination page=0, size=3 (default)
  - /api/people?sort=age,asc – sort by age ascending & pagination page=0, size=3 (default)
  - /api/people?sort=age,asc&sort=weight,desc – sort by age ascending, weight descending & pagination page=0, size=3 (default)
  - /api/people?page=1&size=2&sort=age,asc – sort by age ascending & pagination page=1, size=2

Pagination and Sorting

http://localhost:8080/api/people?page=1&size=2&sort=age,asc


```
{
    "content": [
        {
            "id": 4,
            "name": "Heedong",
            "age": 3,
            "weight": 36.0,
            "height": 100.6,
            "gender": "MALE"
        },
        {
            "id": 7,
            "name": "Younghi",
            "age": 10,
            "weight": 47.2,
            "height": 152.4,
            "gender": "FEMALE"
        }
    ],
    "pageable": {
        "pageNumber": 0,
        "pageSize": 2,
        "sort": {
            "empty": false,
            "unsorted": false,
            "sorted": true
        },
        "offset": 0,
        "paged": true,
        "unpaged": false
    },
    "last": false,
    "totalPages": 5,
    "totalElements": 9,
    "size": 2,
    "number": 0,
    "sort": {
        "empty": false,
        "unsorted": false,
        "sorted": true
    },
    "numberOfElements": 2,
    "first": true,
    "empty": false
}
```

Projections and Excerpts

- Projections
  - Collection 리소스에 대해 @Projection 사용하여 원하는 필드만 제한할 수 있음
    - @Projection(name="personNameAndAgeOnly", types={Person.class})
    - 프로젝션 인터페이스 생성시 반드시 해당 도메인 클래스와 같은 패키지 경로 또는 하위 패키지 경로에 생성해야 함
    - @RespositoryRestResource 어노테이션의 excerptProjection 프로퍼티로 관리되는 리소스 참조는 단일 참조 시 적용되지 않음
- Excerpts
  - Collection 리소스에 대해 기본 보기로 적용됨
  - excerptProjection 옵션으로 원하는 projection을 자동으로 사용하도록 지정할 수 있음
    - @RepositoryRestResource(path = "people", excerptProjection=PersonNameAndAgeOnly.class)

Spring Data REST

- Spring Data REST
  - Spring Data REST는 Spring Data JPA로 정의한 Entity와 Repository를 분석해서 자동으로 RESTful API 제공
  - 즉, 별도의 Controller를 작성하지 않아도 (Entity 클래스와 Repository 인터페이스만으로도 RestController 없이) CRUD 기반 REST API 구현 가능
    - 예를 들어, 기존의 Entity 클래스 (Pet.java)와 Repository 인터페이스 (PetRepository.java)를 그대로 사용하고, Service 클래스 (PetService.java)와 Controller 클래스(PetController.java)는 삭제하고 REST 테스트
  - Spring HATEOAS와의 통합
    - 응답 메시지에 HATEOAS 기반 링크 정보를 자동으로 포함

Spring Boot Data Rest

- pom.xml에 Spring Data REST와 HATEOAS 추가

```
<!-- Spring Data Rest -->
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
<!-- HATEOAS -->
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-hateoas</artifactId>
</dependency>
<!-- Spring Data Rest Hal Explorer -->
<dependency>
        <groupId>org.springframework.data</groupId>
        <artifactId>spring-data-rest-hal-explorer</artifactId>
</dependency>
```

Spring Boot Data Rest

- application.properties에 Spring Data REST 관련정보 추가

```
# spring data rest
spring.data.rest.base-path=/api
# page-size (default=20)
spring.data.rest.default-page-size=10
# max-page-size (default=2000)
spring.data.rest.max-page-size=10
```

@RepositoryRestResource

- @RepositoryRestResource
  - Spring Data REST dependency가 추가되면 기본 설정으로 REST API가 생성됨 (@RepositoryRestResource 없어도)
  - 기본 설정 외 추가 설정이 필요하다면 @RepositoryRestResource 사용
  - path() - URI 매핑 설정 (기본 URI는 Entity의 복수)
  - excerptProjection() - 기본 Projection 설정

```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
@RepositoryRestResource(path = "people",
excerptProjection=PersonNameAndAgeOnly.class)
public interface PeopleRepository extends JpaRepository<Person, Long> {
    Page<Person> findAll(Pageable pageable);
}
```

@RepositoryRestController

- @RepositoryRestController
  - @RestController를 대체하며 매핑하는 URL 형식이 Spring Boot Data REST에서 정의하는 REST API 형식에 맞아야 함
  - 또한 기존에 기본으로 제공하는 URL 형식과 같게 제공해야 해당 컨트롤러의 메서드가 기존의 기본 API를 오버라이드 함

```
@RepositoryRestController
@RequiredArgsConstructor
public class PeopleController {
    private final PeopleRepository repository;
    private final ProjectionFactory factory;
    @GetMapping("/people/name-age")
    public @ResponseBody Page<PersonNameAndAgeOnly>
getAllWithNameAndAge(@PageableDefault Pageable pageable) {
        Page<Person> people = repository.findAll(pageable);
        return people.map(p ->
factory.createProjection(PersonNameAndAgeOnly.class, p));
    } ... }
```

@Projection

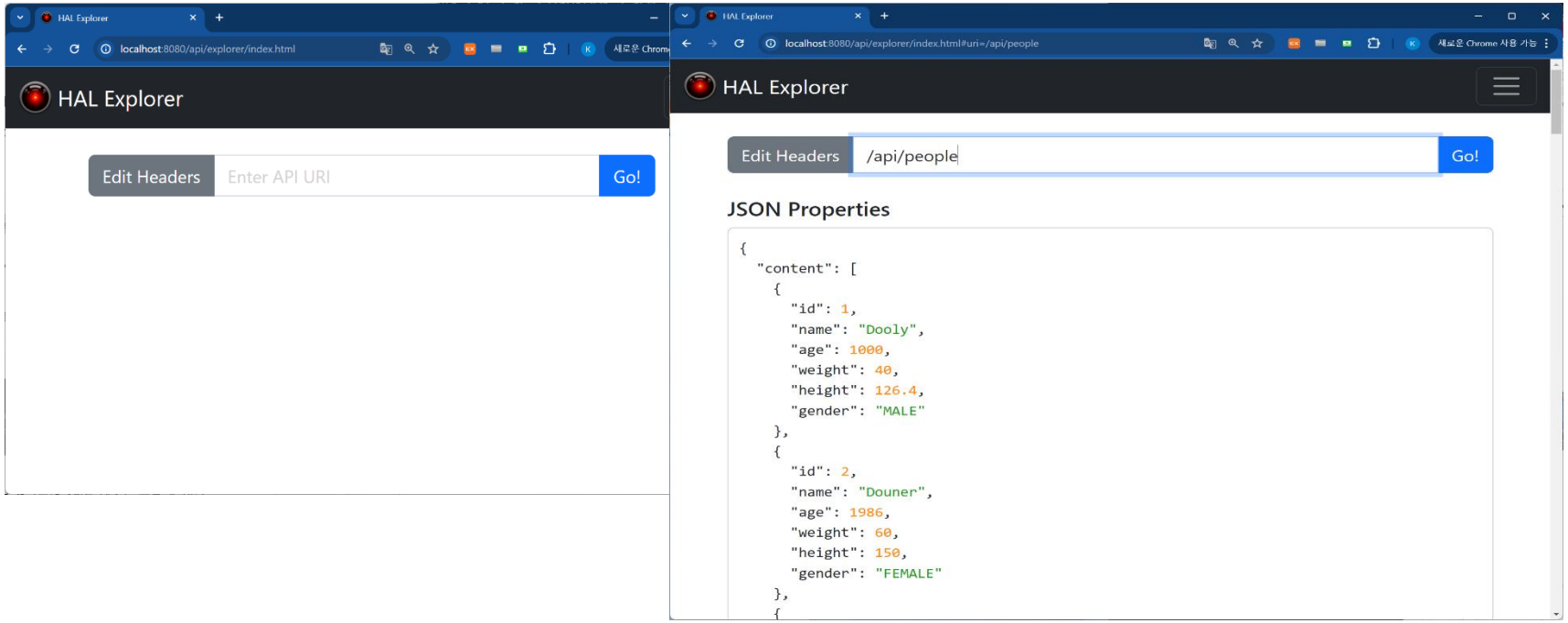
- @Projection

```
@Projection(name="PersonNameAndAgeOnly", types={ Person.class })
public interface PersonNameAndAgeOnly {
    String getName(); // person's name
    int getAge(); // person's age
}

@Configuration
public class FactoryConfig {
    @Bean
    public SpelAwareProxyProjectionFactory projectFactory() {
        return new SpelAwareProxyProjectionFactory();
    }
}
```

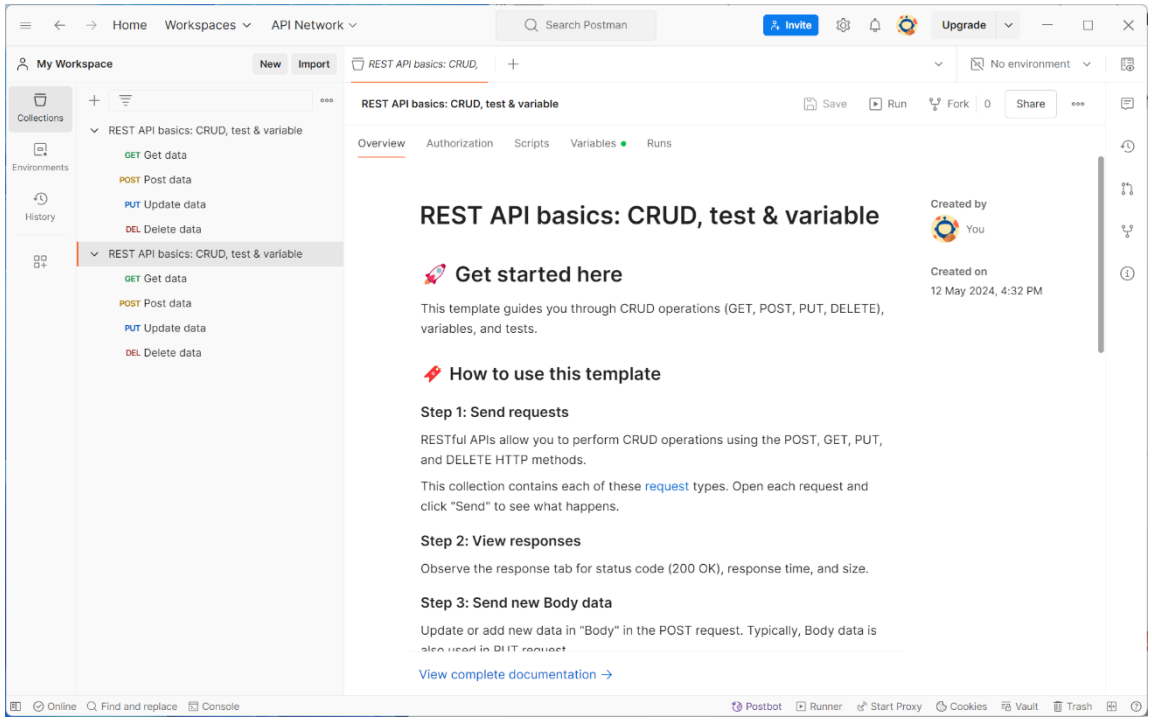
HAL Explorer

- HAL Explorer 실행 및 테스트
  - HAL은 리소스와 API 사이를 하이퍼링크를 쉽게 하고 일관성 있게 하기 위한 간단한 포맷
  - HAL을 사용하는 RESTful API 라면 HAL Explorer 를 통하여 API 를 테스트하거나 실행해볼 수 있음
  - http://localhost:8080/api/explorer/index.html



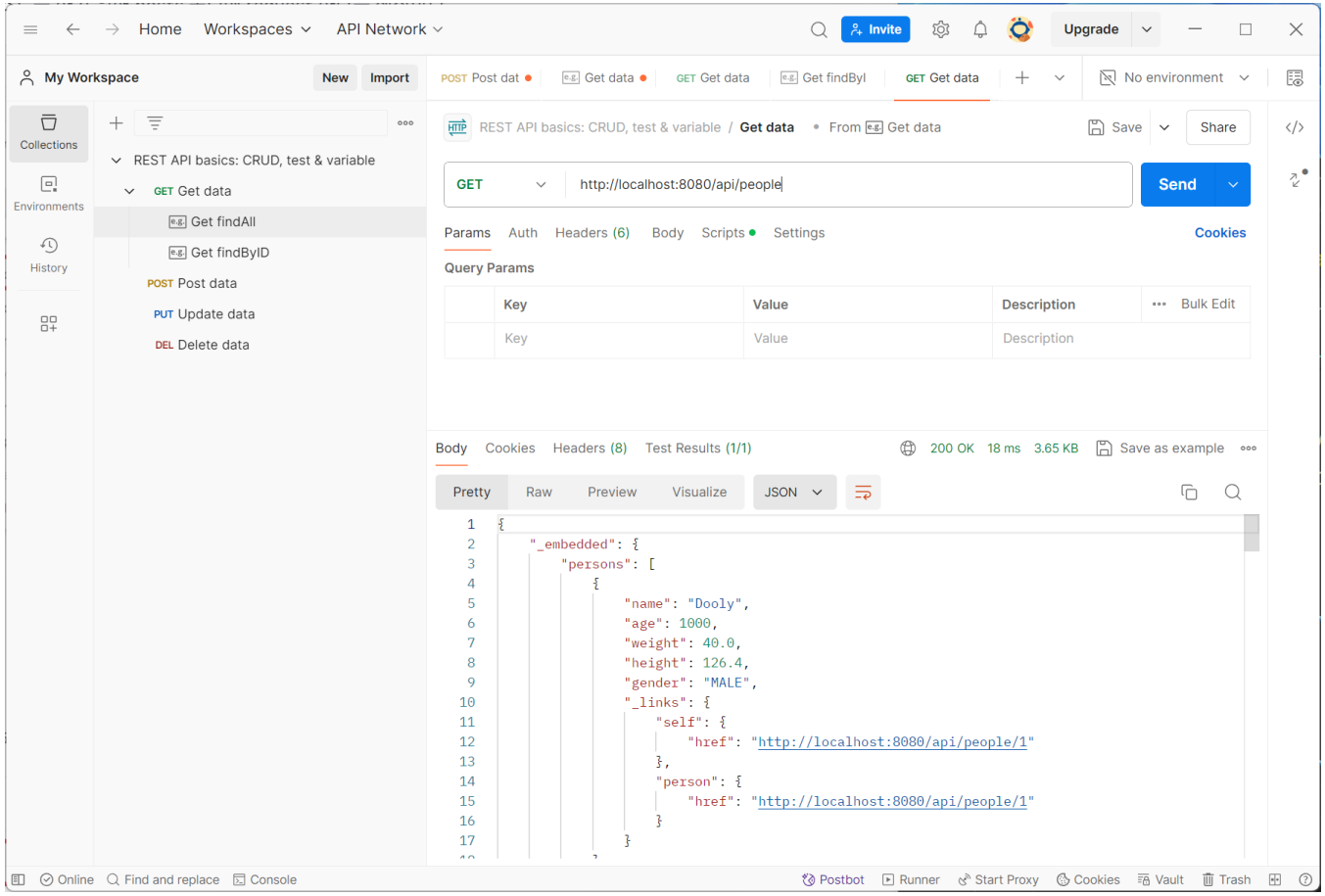
Postman

- Postman 다운로드 및 설치
  - Postman은 API 개발, 테스트, 공유 및 문서화 도구
  - https://www.postman.com/downloads/
- My Workspace의 Collection에서 +New
  - REST API Basics: CRUD, test & variable 테스트



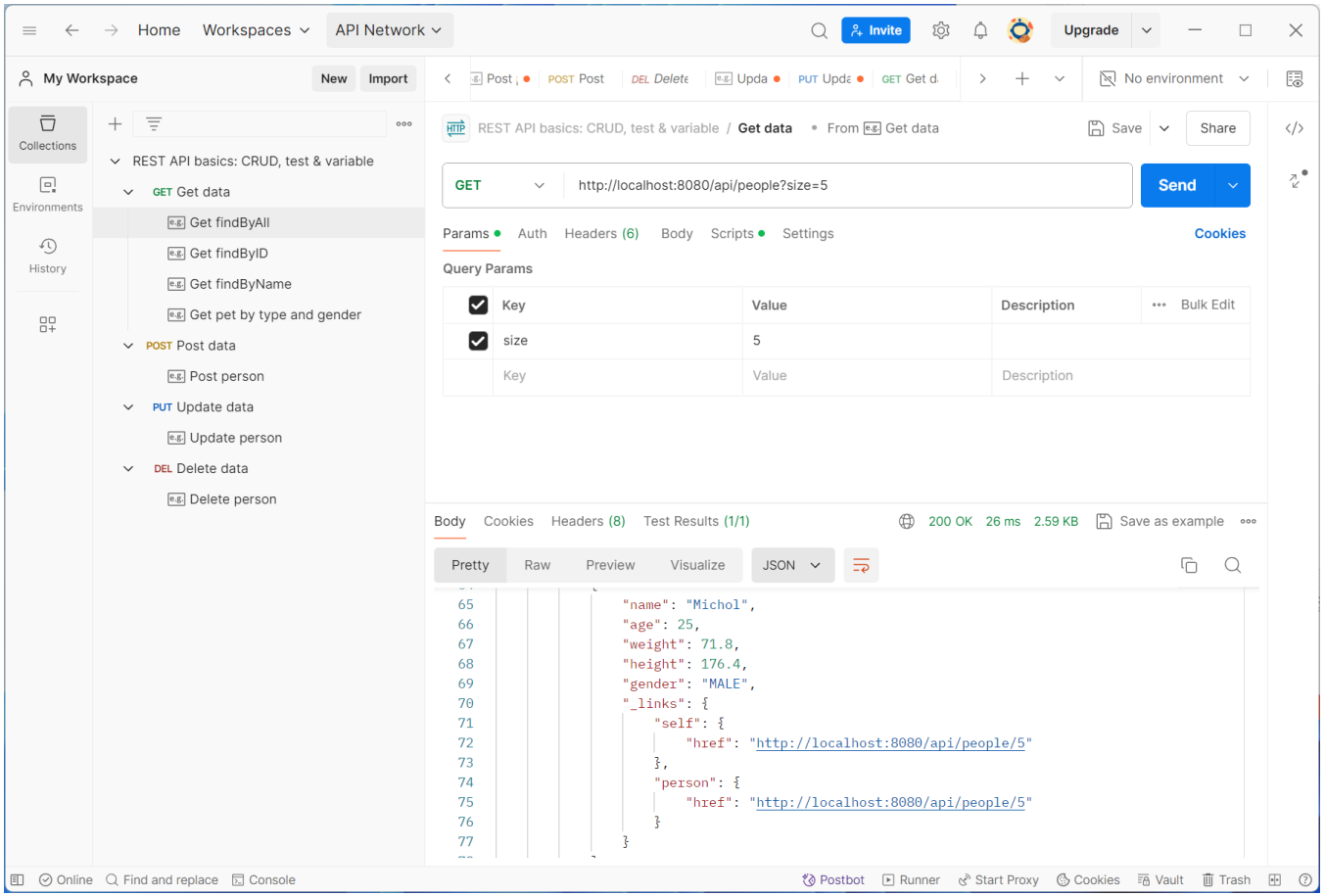
Postman REST API Basics

- GET http://localhost:8080/api/people



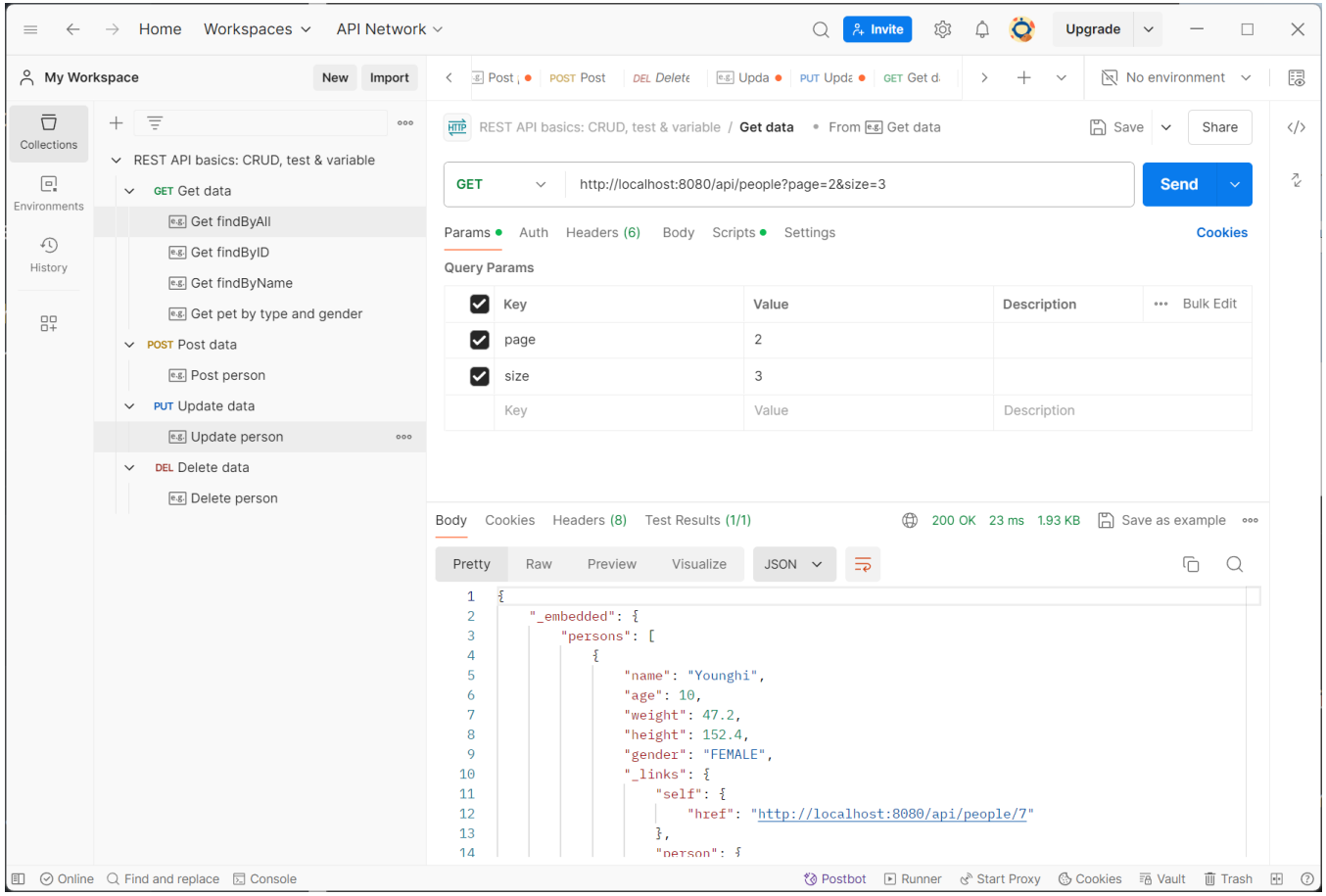
Postman REST API Basics

- GET http://localhost:8080/api/people?size=5



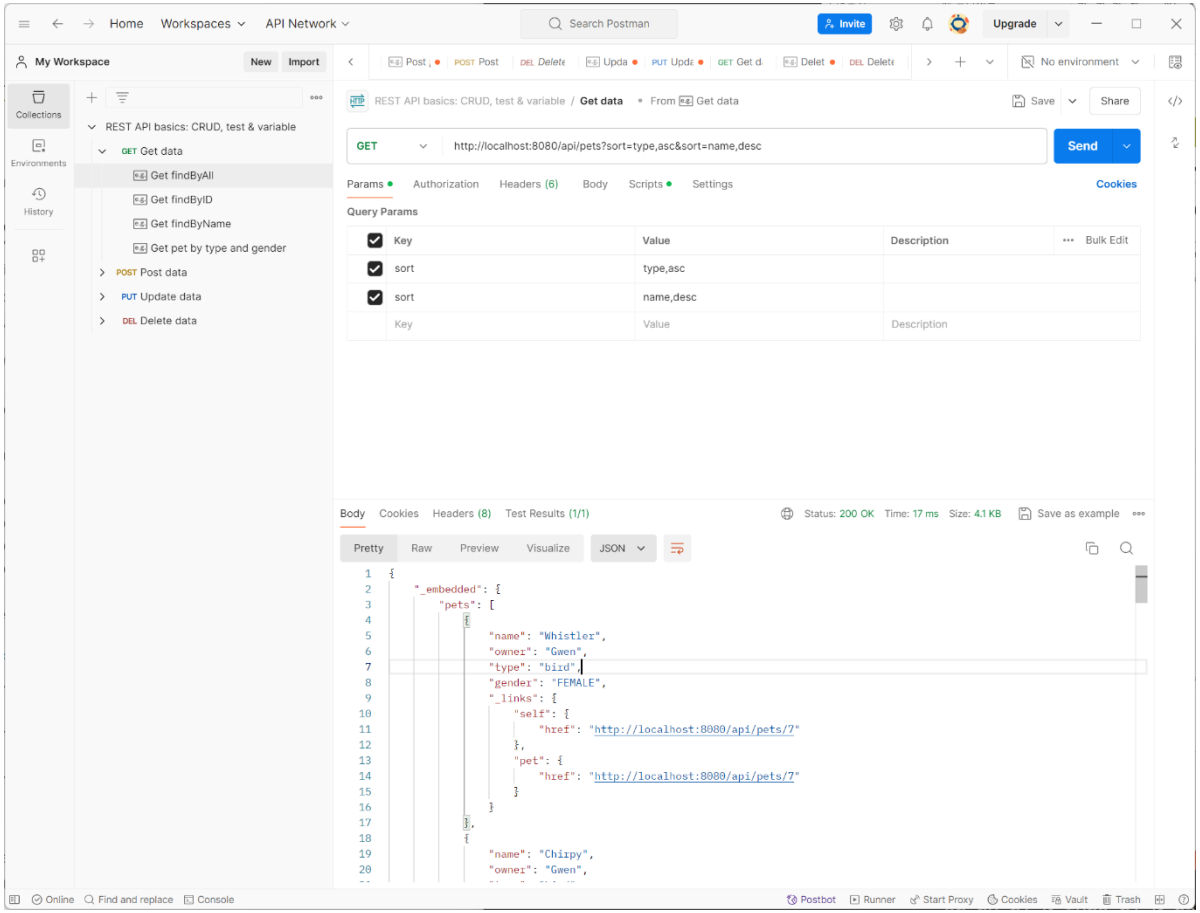
Postman REST API Basics

- GET http://localhost:8080/api/people?page=2&size=3



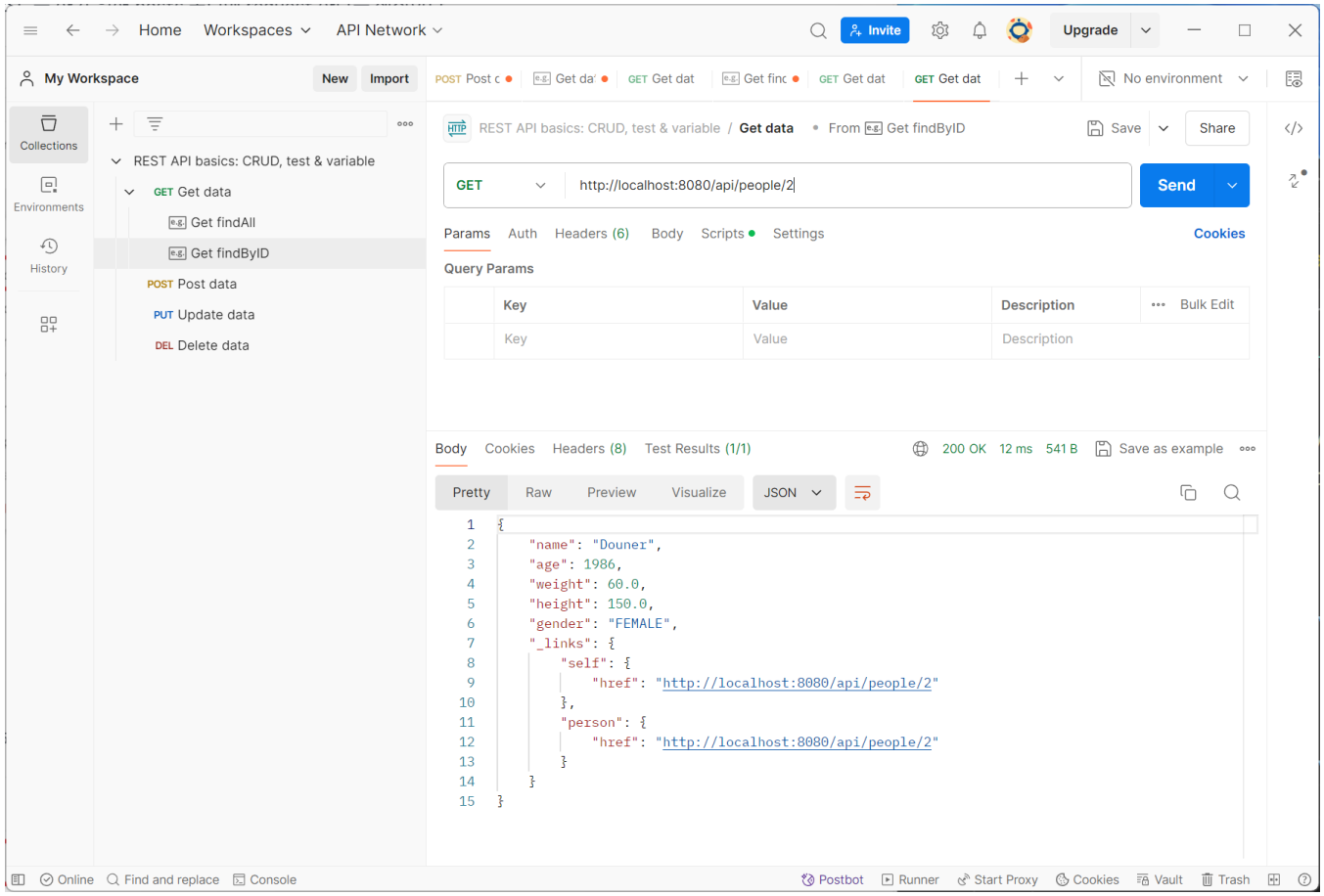
Postman REST API Basics

- GET http://localhost:8080/api/pets?sort=type,asc&sort=name,desc



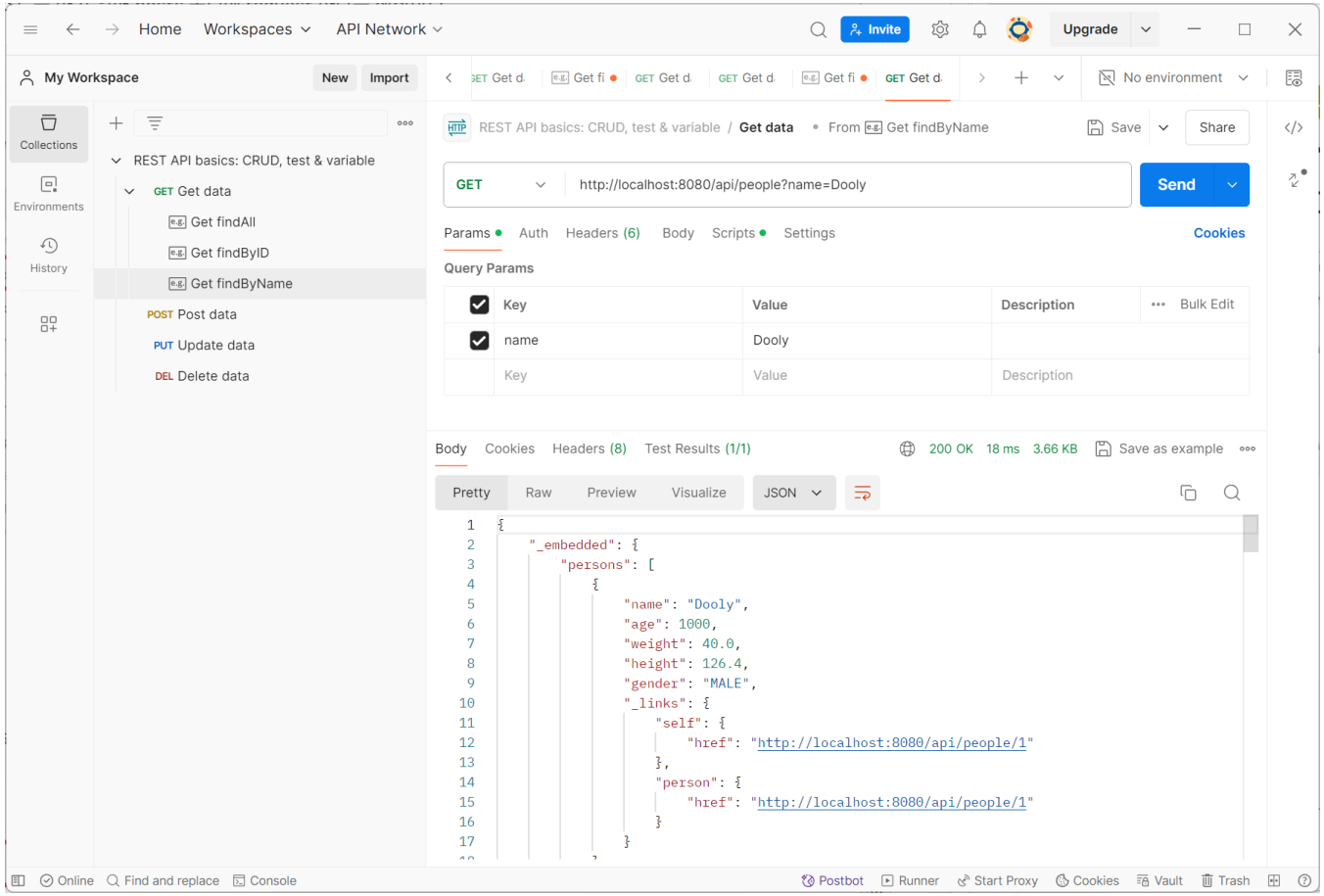
Postman REST API Basics

- GET http://localhost:8080/api/people/2



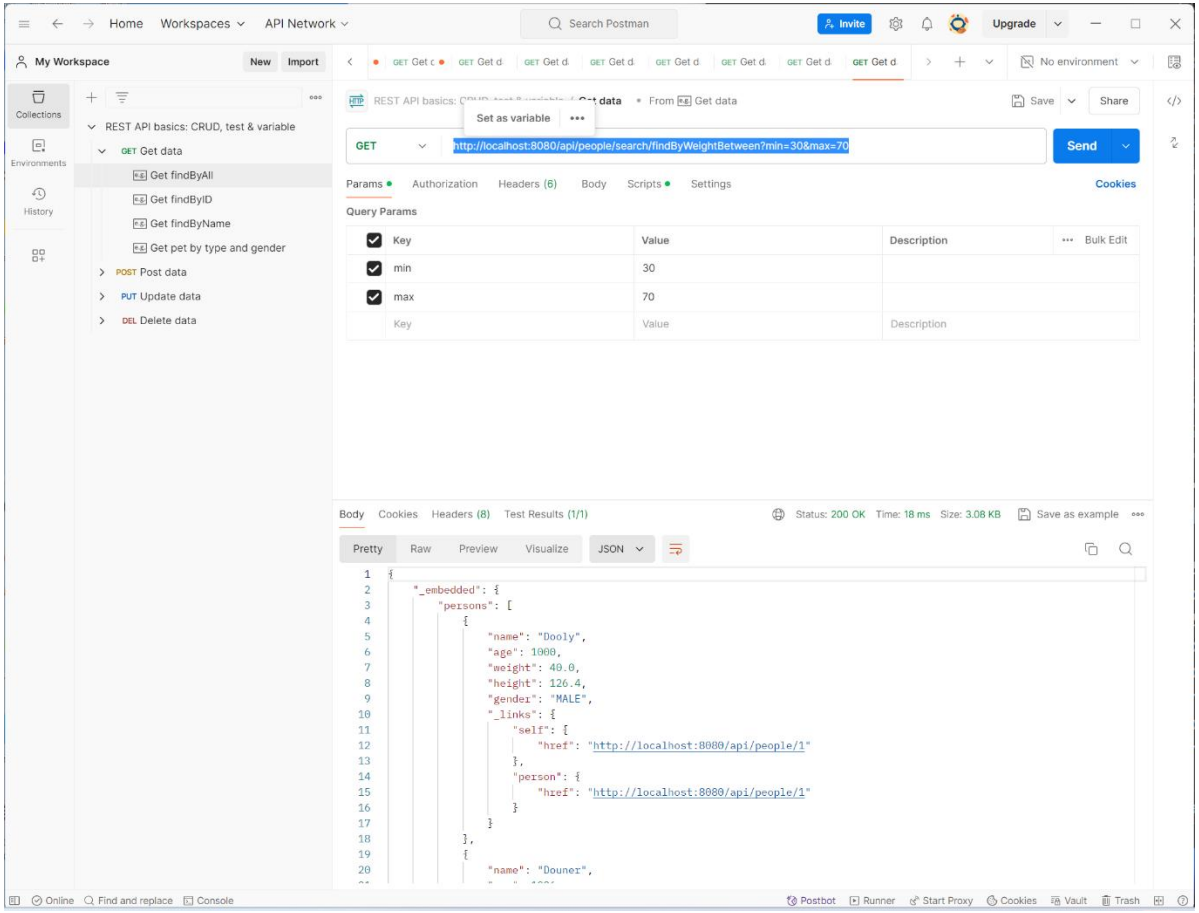
Postman REST API Basics

- GET http://localhost:8080/api/people?name=Dooly



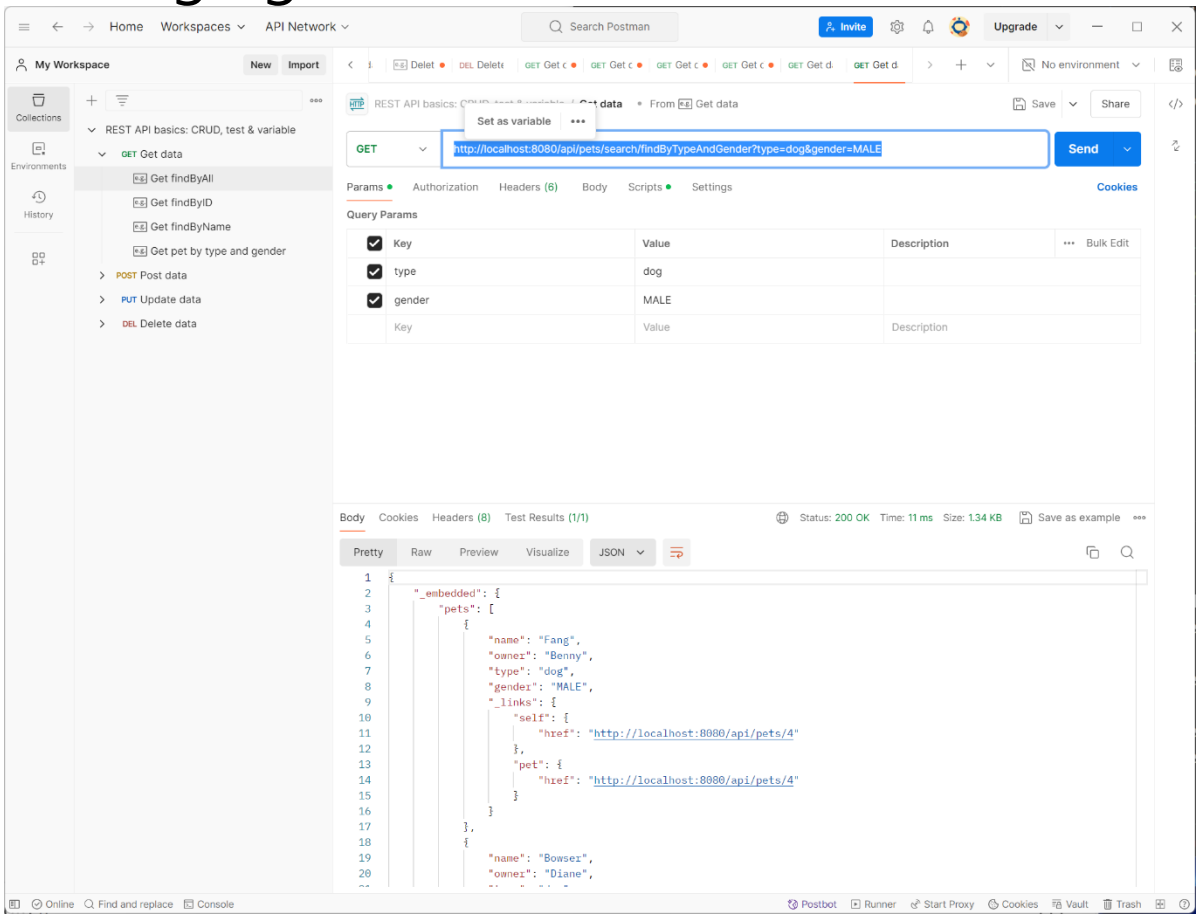
Postman REST API Basics

- GET http://localhost:8080/api/people/search/findByWeightBetween?min=30&max=70



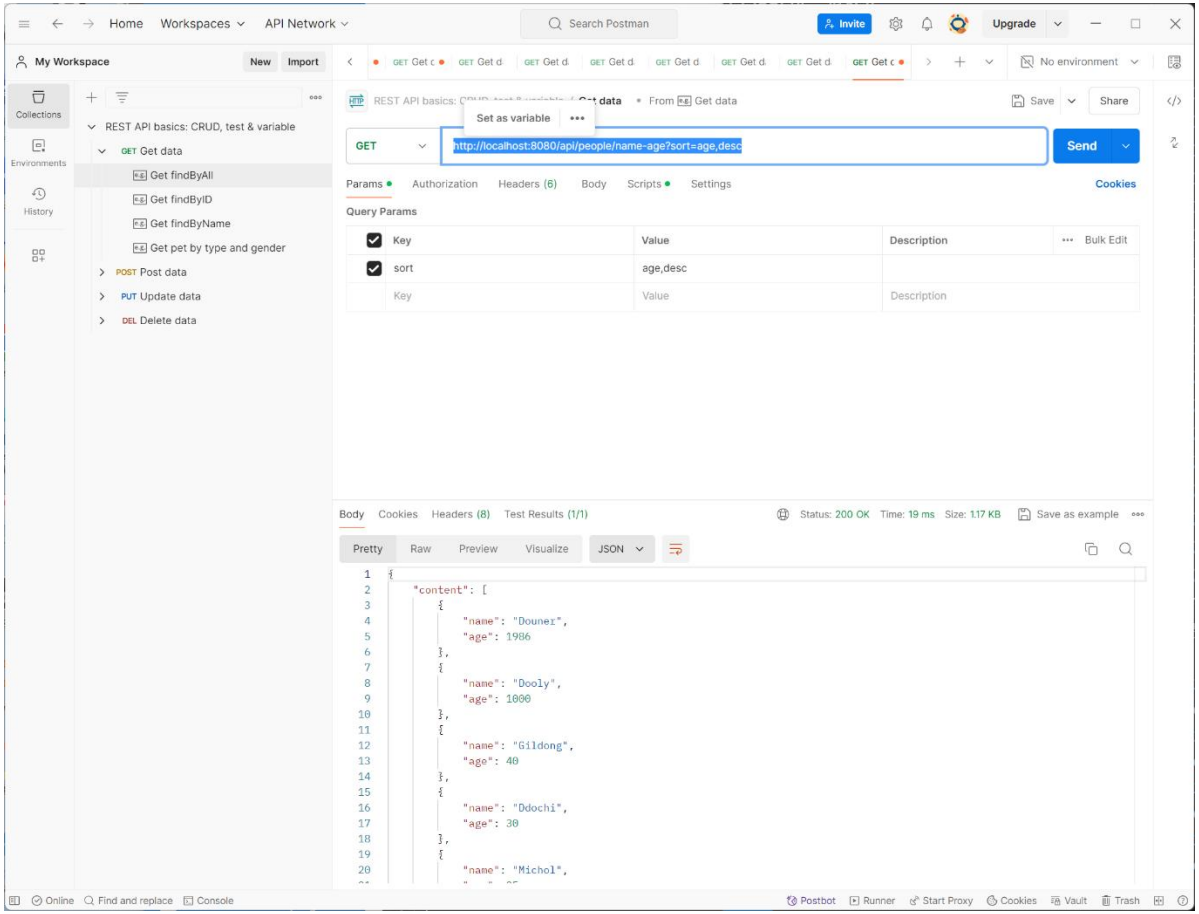
Postman REST API Basics

- GET http://localhost:8080/api/pets/search/findByTypeAndGender?type=dog&gender=MALE



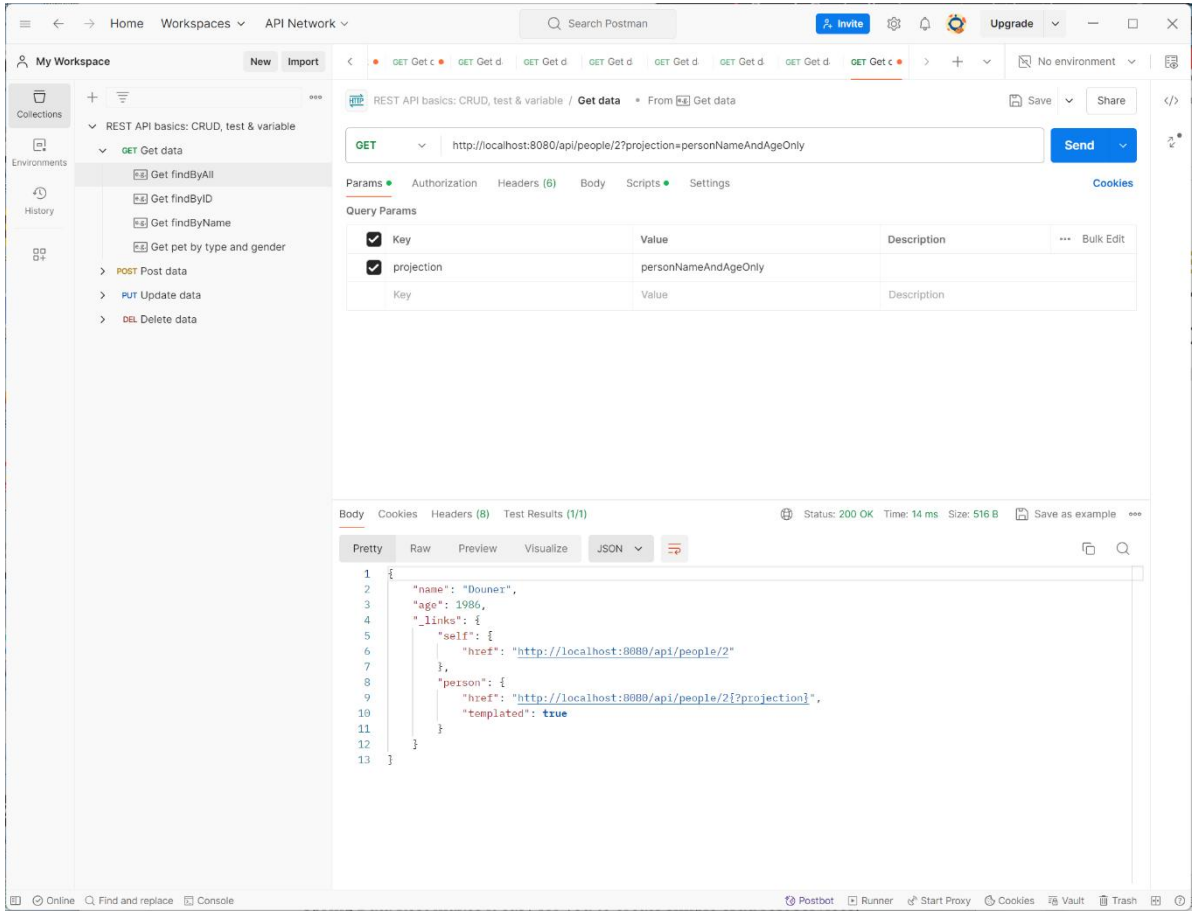
Postman REST API Basics

- http://localhost:8080/api/people/name-age?sort=age,desc (projection 예시)



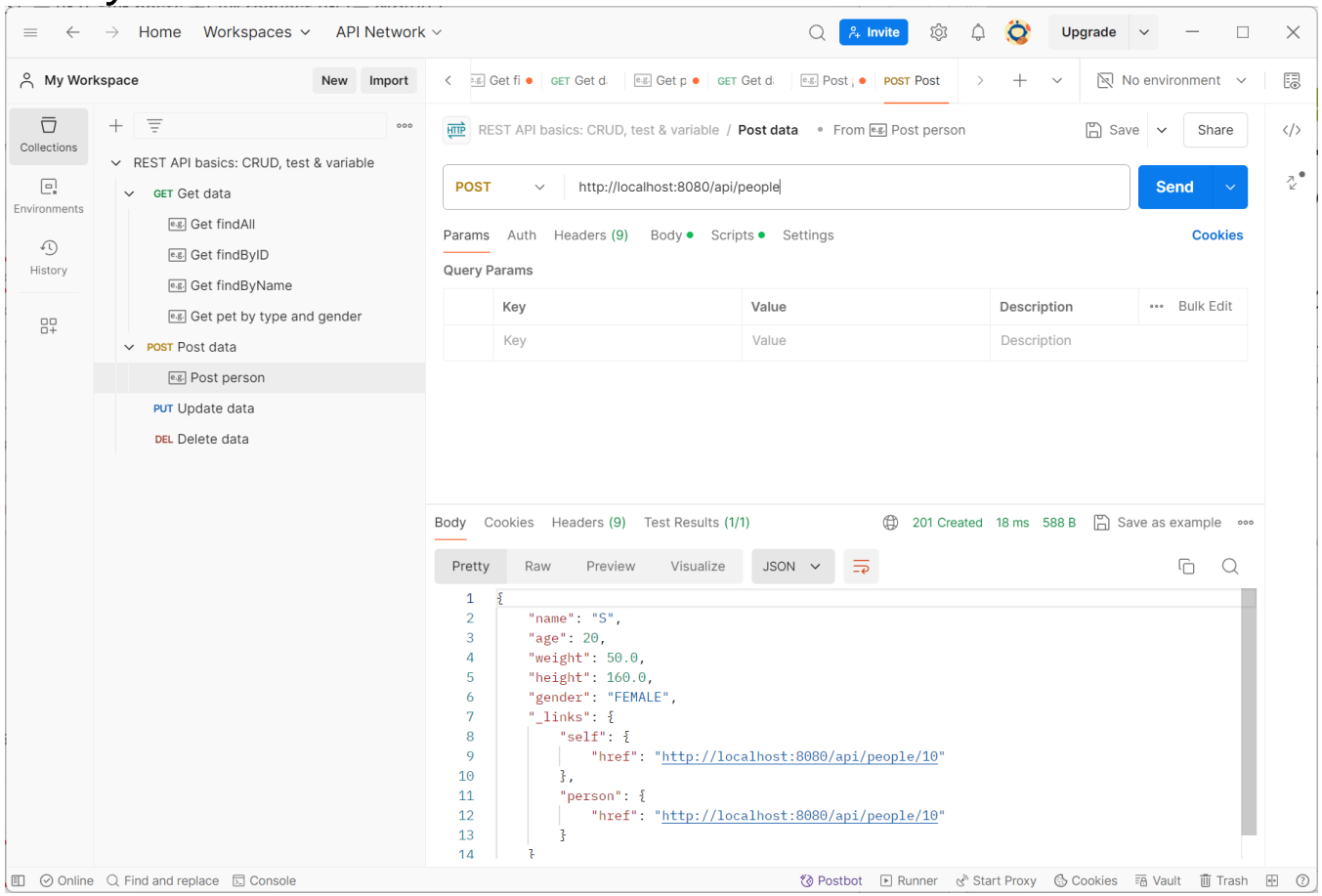
Postman REST API Basics

- http://localhost:8080/api/people/2?projection=personNameAndAgeOnly (projection 예시)



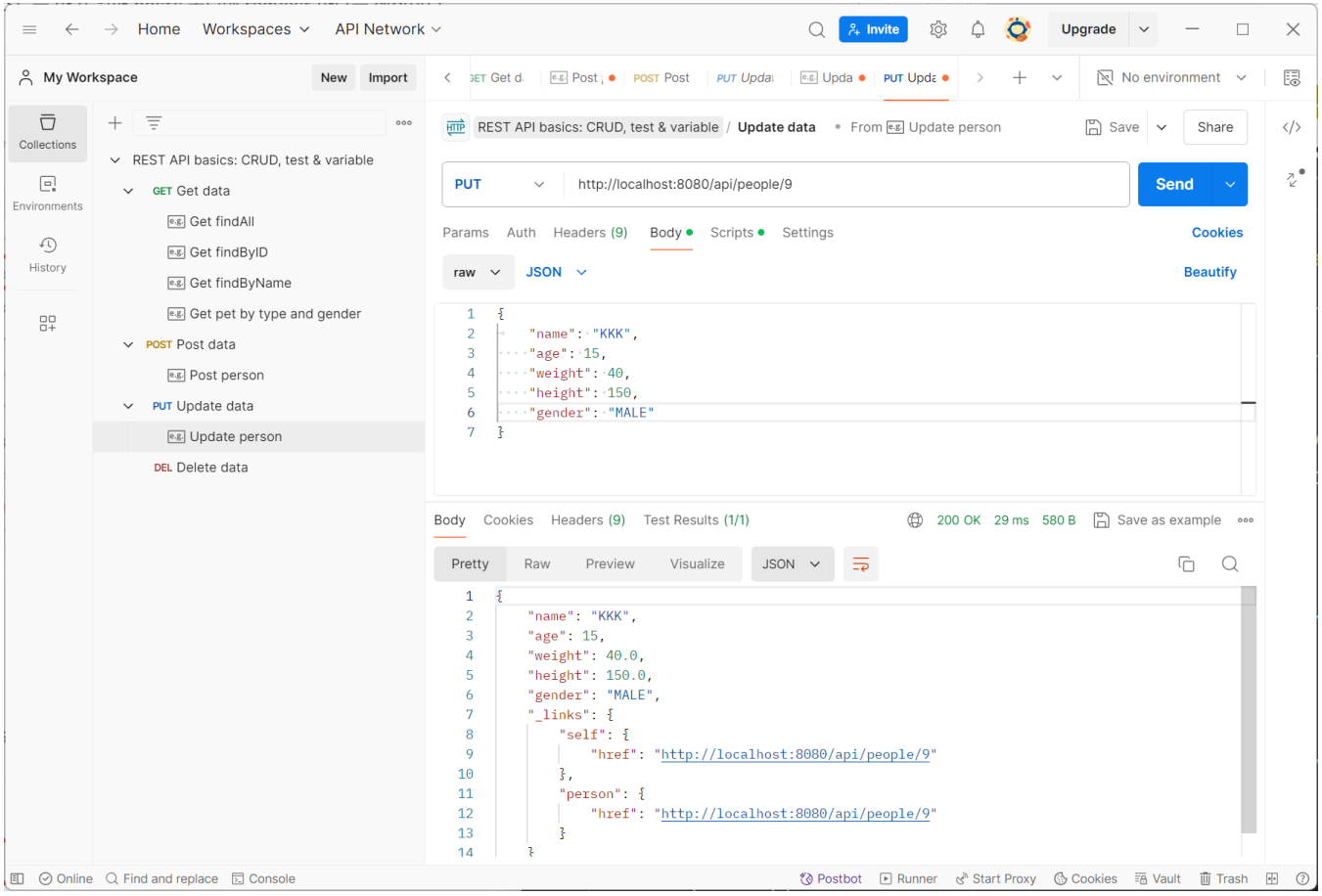
Postman REST API Basics

- POST http://localhost:8080/api/people
  - Headers에 Content-Type을 application/json 적용
  - Body에 테스트 JSON 데이터 적용



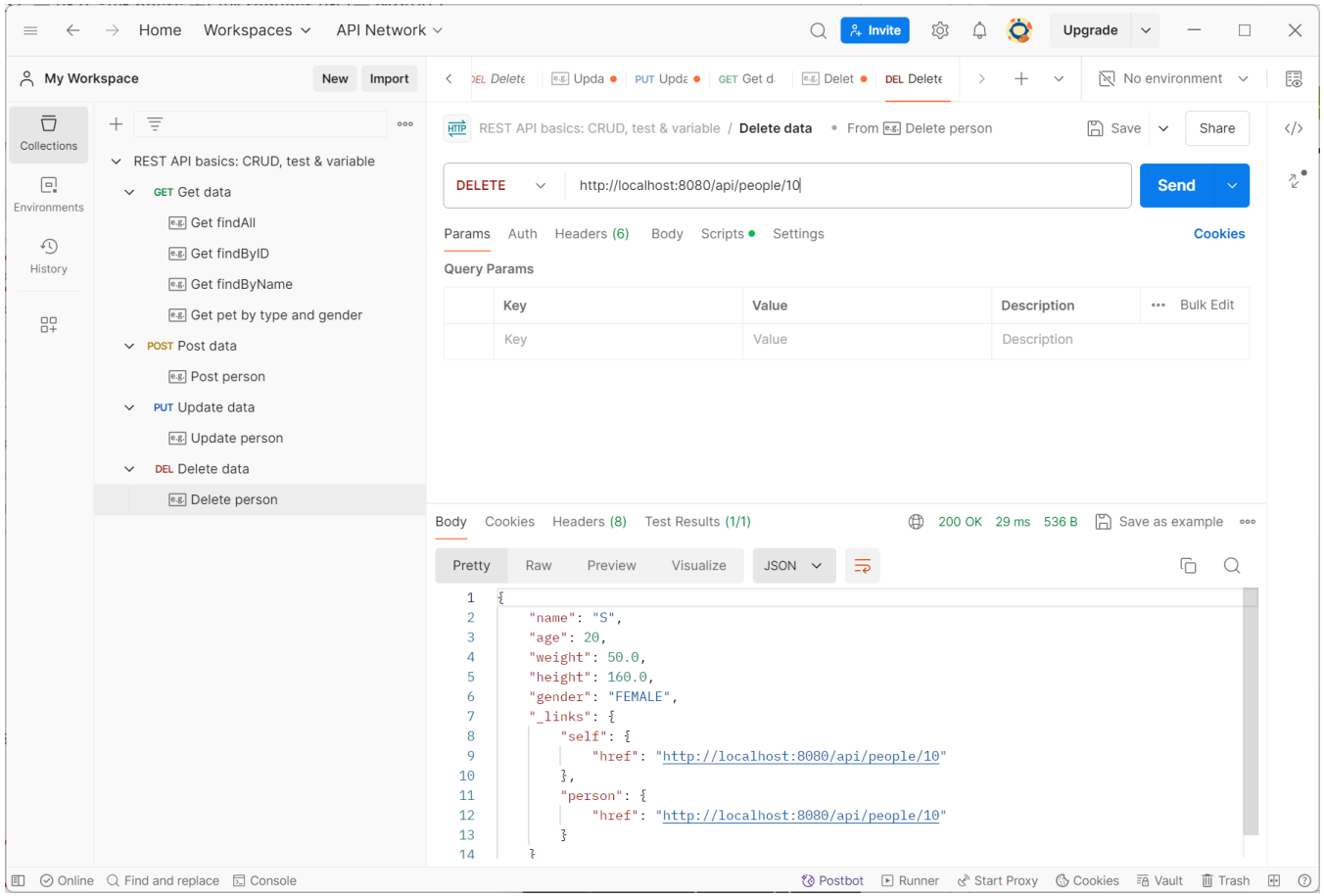
Postman REST API Basics

- PUT http://localhost:8080/api/people/9
  - Headers에 Content-Type을 application/json 적용
  - Body에 수정된 테스트 JSON 데이터 적용



Postman REST API Basics

- DELETE http://localhost:8080/api/people/10



외부 API 호출

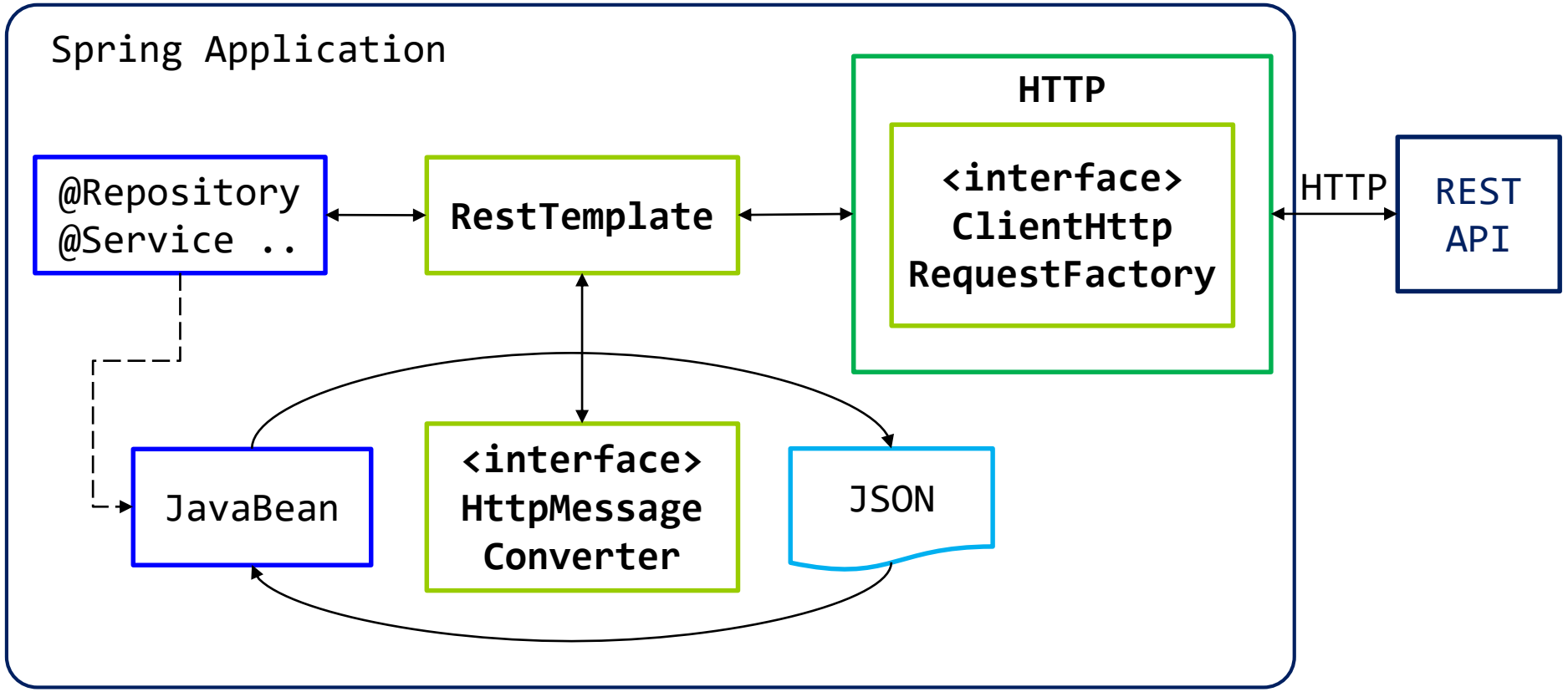
- 스프링에서 외부 API 호출
  - HTTPURLConnection/URLConnection - 순수 자바 HTTP 동기적 통신
  - Apache HttpClient - 아파치 HTTP 컴포넌트
  - Java.net.http.HttpClient - Java11 이후, Apache HttpClient와 비슷, 비동기적 통신 가능
  - RestTemplate - Spring 3.0 이후, Blocking 기반의 동기적 통신, RESTful 원칙 고수
  - WebClient – Spring 5.0 이후, Non-Blocking 비동기적 통신 가능, WebFlux 의존성 필요
  - HttpInterface - Spring 6.0 이후, WebClient 또는 RestClient를 통해 동적 Proxy 생성
  - RestClient - Spring 6.1.2 이후, WebClient와 비슷하나 WebFlux 의존성 없이 사용 가능

RestTemplate

- RestTemplate
  - Spring에서 제공하는 HTTP 통신 기능을 쉽게 사용할 수 있게 설계되어 있는 템플릿
  - HTTP 서버와의 통신을 단순화하고 RESTful 원칙을 지킴
  - 동기 방식으로 처리, 비동기방식으로는 AsyncRestTemplate이 있음
  - RestTemplate 클래스는 REST 서비스를 호출하도록 설계되어 HTTP 프로토콜의 메소드에 맞게 여러 메소드를 제공함
    - getForEntity()
    - postForObject()
    - put()
    - delete()

RestTemplate

- RestTemplate 동작원리



RestTemplate

- RestTemplate 주요 메소드

Method	HTTP	Description
getForObject	GET	GET 요청하여 객체로 결과를 반환 받음
getForEntity	GET	GET 요청하여 ResponseEntity로 결과를 반환 받음
postForObject	POST	POST 요청하여 객체로 결과를 반환 받음
postForEntity	POST	POST 요청하여 ResponseEntity로 결과를 반환 받음
delete	DELETE	DELETE 요청
put	PUT	PUT 요청
patchForObject	PATCH	PATCH 요청
exchange	Any	HTTP 헤더를 생성하여 추가할 수 있고 어떤 HTTP Method로 사용 가능