

RESTful API



Resstful API



Spring Boot

RESTful API

▪ RESTful API

- RESTful API는 웹에서 서버와 클라이언트가 데이터를 주고받는 방식을 설계하는 아키텍처 스타일이다.
- REST (Representational State Transfer) 는 2000년 로이 필딩(Roy Fielding) 이 논문에서 제안한 웹 기반의 설계 원칙으로 REST 원칙을 잘 지킨 API를 RESTful API라 함.
- REST는 "**리소스(Resource)를 URL로 표현하고, 그 리소스에 대한 행위는 HTTP 메서드로 구분한다**" 개념이다.
- 행위는 **HTTP 메서드(GET, POST, PUT, DELETE 등)** 로 구분한다.

예)

<http://api.oursite.com/users/2>



RESTful API

- 주요 HTTP 메서드와 의미

작업	URL	메서드
데이터 조회(Read)	/users – 모든 정보 조회 /users/{id} – 특정 id 조회	GET
데이터 생성(Create)	/users	POST
데이터 수정(Update)	/users/{id}	PUT
데이터 삭제>Delete)	/users/{id}	DELETE



회원 관리 API

▪ 회원(Users) 관리 API 만들기

- 프레임워크 – 스프링부트(Spring Boot) 4.0
 - 테스트 도구 – postman
 - 프로젝트 템플릿 – Spring Initializer(Spring.io)
 - DB 언어 – JPA
 - DBMS - MySQL
- * 깃허브 – 설정(README.md)
- <https://github.com/kiyongee2/fullstack-web.git>



회원 관리 API

▪ Spring.io > projects > Spring Initializer

Project
☐ Gradle - Groovy
☐ Gradle - Kotlin

Language
☒ Java ☐ Kotlin ☐ Groovy

Spring Boot
☐ 4.0.1 (SNAPSHOT) ☒ 4.0.0 ☐ 3.5.9 (SNAPSHOT) ☐ 3.5.8
☐ 3.4.13 (SNAPSHOT) ☐ 3.4.12

Project Metadata

Group

com.springboot

Artifact

user-api

Name

user-api

Description

Demo project for Spring Boot

Package name

com.springboot

Packaging

☒ Jar ☐ War

Configuration

☒ Properties ☐ YAML

Java

☐ 25 ☒ 21 ☐ 17



회원 관리 API

■ 의존성(Dependency)

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

MySQL Driver SQL
MySQL JDBC driver.



회원 관리 API

- **application.property**

MySQL 설정

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/webdb?serverTime=Asia/Seoul  
spring.datasource.username=bootuser  
spring.datasource.hikari.password=pwboot
```

JPA 설정

```
spring.jpa.hibernate.ddl-auto=create  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.format_sql=true  
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
```



회원 관리 API

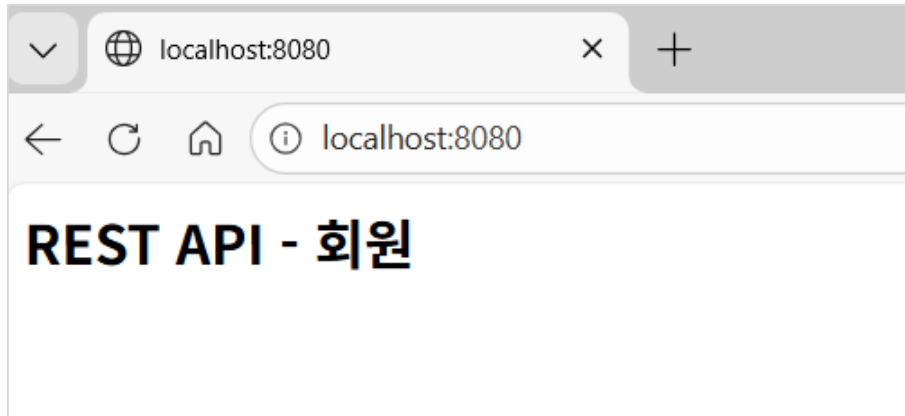
- HomeController

```
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class HomeController {  
  
    @GetMapping("/")  
    public String home() {  
        return "<h2>REST API - 회원</h2>";  
    }  
}
```



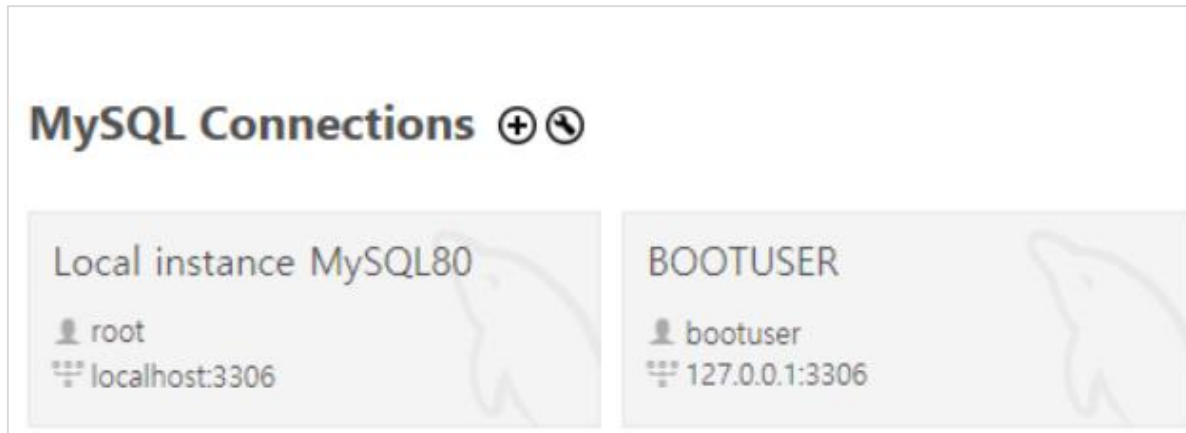
회원 관리 API

- <http://localhost:8080>



회원 관리 API

- MySQL 워크 벤치 – root에서 새 계정 만들기



```
-- 계정 bootuser 생성 및 비밀번호 설정
create user bootuser@localhost identified by 'pwboot';

-- 권한 부여
grant all privileges on *.* to bootuser@localhost;
```



회원 관리 API

- bootuser 접속 인터페이스 만들기

The screenshot shows a database connection configuration window. At the top, the 'Connection Name' is 'BOOTUSER'. Below this are three tabs: 'Connection', 'Remote Management', and 'System Profile'. The 'Connection' tab is active. It contains a 'Connection Method' dropdown set to 'Standard (TCP/IP)' with a description 'Method to use to connect to the RDBMS'. Below this are three sub-tabs: 'Parameters', 'SSL', and 'Advanced'. The 'Parameters' sub-tab is active and contains the following fields:

- Hostname:** 127.0.0.1
- Port:** 3306
- Username:** bootuser
- Password:** A field with a 'Store in Vault ...' button and a 'Clear' button.
- Default Schema:** An empty text field.

Each field has a descriptive text to its right: 'Name or IP address of the server host - and TCP/IP port.', 'Name of the user to connect with.', 'The user's password. Will be requested later if it's not set.', and 'The schema to use as default schema. Leave blank to select it later.'



회원 관리 API

▪ User 엔티티

```
@NoArgsConstructor
@AllArgsConstructor
@Getter @Setter
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;    //번호

    private String name;    //이름

    private String email;    //이메일

    @CreationTimestamp
    @Column(updatable = false)
    private Timestamp regDate;    //가입일

    @UpdateTimestamp
    @Column(insertable = false)
    private Timestamp updateDate;    //수정일
}
```



회원 관리 API

- User Repository

```
import org.springframework.data.jpa.repository.JpaRepository;  
  
import com.springboot.entity.User;  
  
public interface UserRepository extends JpaRepository<User, Long>  
{  
}
```



회원 관리 API

▪ User 컨트롤러

```
@RequiredArgsConstructor
@RequestMapping("/users")
@RestController
public class UserController {

    private final UserService userService;

    //회원 추가
    @PostMapping
    public String createUser(@RequestBody User user) {
        userService.save(user);
        return "회원 가입 완료";
    }

    //회원 목록
    @GetMapping
    public List<User> getUserList() {
        return userService.findAll();
    }
}
```



회원 관리 API

▪ User 컨트롤러

```
//회원 정보(상세보기)
@GetMapping("/{id}")
public User getUser(@PathVariable Long id) {
    return userService.findById(id);
}

//회원 삭제
@DeleteMapping("/{id}")
public String deleteUser(@PathVariable Long id) {
    userService.delete(id);
    return "회원 삭제 완료!";
}

//회원 수정
@PutMapping("/{id}")
public User updateUser(@PathVariable Long id,
    @RequestBody User user) {
    return userService.update(id, user);
}
}
```



회원 관리 API

▪ User 서비스

```
@RequiredArgsConstructor
@Service
public class UserService {

    private final UserRepository userRepository;

    //회원 추가
    public void save(User user) {
        userRepository.save(user);
    }

    //회원 목록
    public List<User> findAll() {
        return userRepository.findAll();
    }

    //회원 정보(상세)
    public User findById(Long id) {
        return userRepository.findById(id)
            .orElseThrow(() -> new RuntimeException("회원을 찾을 수 없습니다."));
    }
}
```



회원 관리 API

▪ User 서비스

```
//회원 삭제
public void delete(Long id) {
    userRepository.deleteById(id);
}

//회원 수정
public User update(Long id, User user) {
    //수정할 user 가져오기
    User updateUser = userRepository.findById(id)
        .orElseThrow(() -> new RuntimeException("회원을 찾을 수 없습니다."));

    //수정 처리
    updateUser.setName(user.getName());
    updateUser.setEmail(user.getEmail());
    return userRepository.save(updateUser);
}
```



회원 관리 API

■ 포스트 맨(Postman) 툴 사용

- Postman은 API(특히 RESTful API)를 테스트하고 디버깅하기 위한 강력한 도구이다.
- 서버에 HTTP 요청을 보내고, 응답(결과)을 바로 확인할 수 있는 API 클라이언트 프로그램이다.
- 보통 웹 브라우저는 **GET** 요청밖에 직접 보낼 수 없지만, API 테스트에서는 **POST, PUT, DELETE** 등 다양한 HTTP 메서드가 필요하다.



회원 관리 API

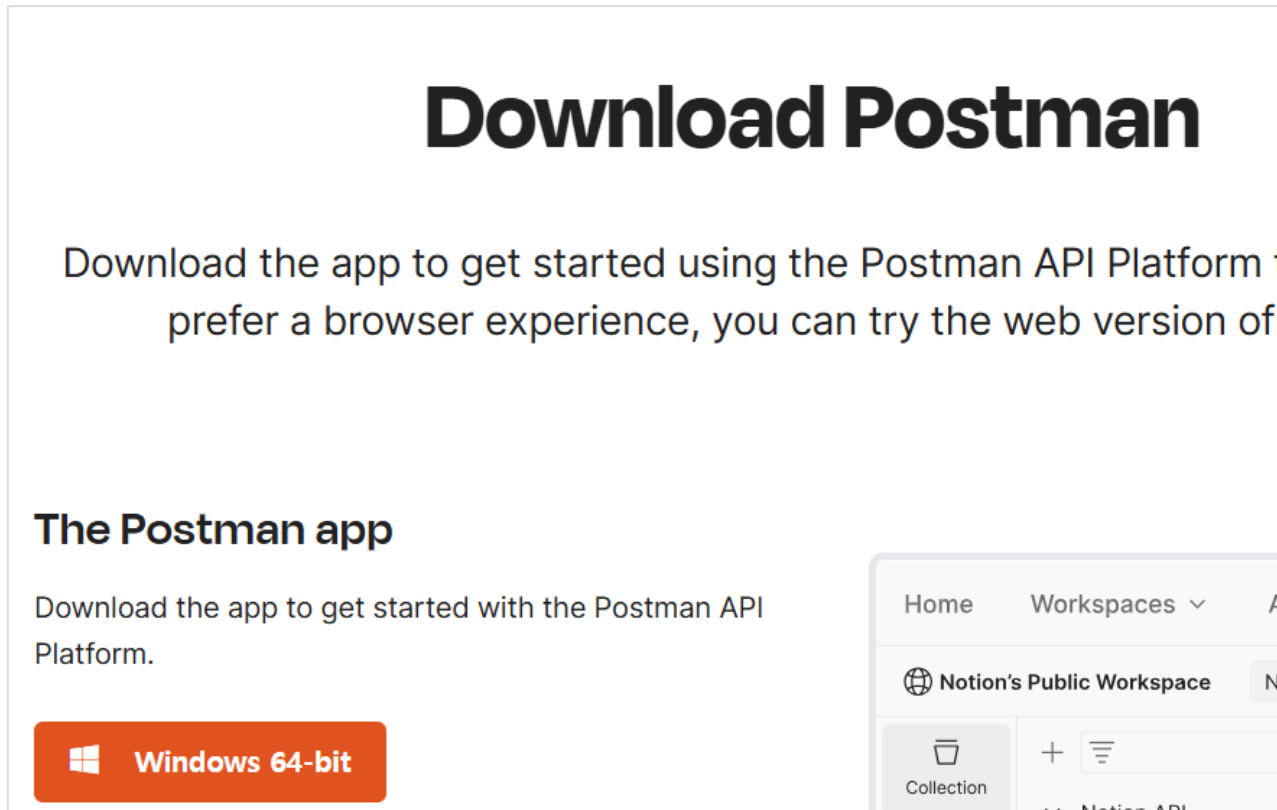
- 포스트 맨(Postman) 기능

- ✓ 모든 HTTP 메서드 테스트 가능
- ✓ 요청 헤더, 파라미터, 바디를 자유롭게 설정 가능
- ✓ 서버의 응답(JSON, XML 등)을 보기 쉽게 출력
- ✓ API 테스트 자동화도 가능



회원 관리 API

- Postman DeskTop Tool 다운로드




Download Postman

Download the app to get started using the Postman API Platform. If you prefer a browser experience, you can try the web version of Postman.

The Postman app

Download the app to get started with the Postman API Platform.

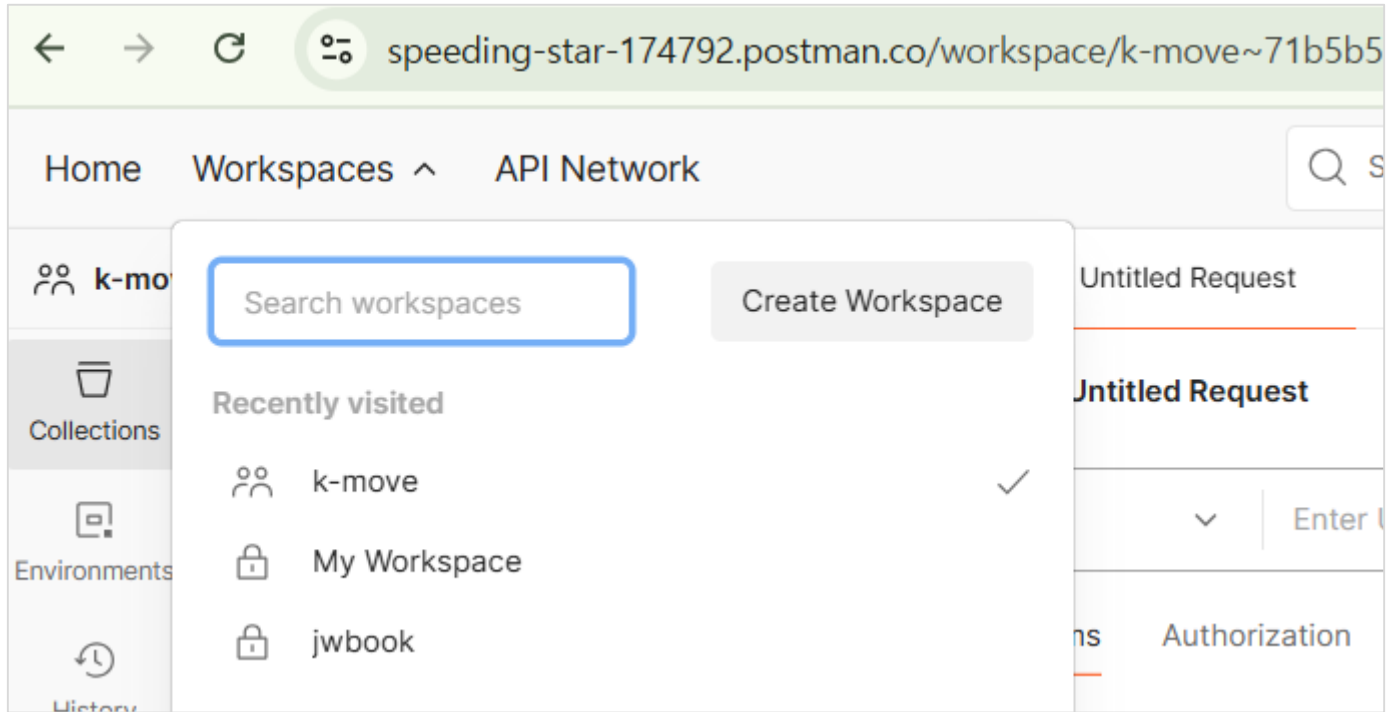
 **Windows 64-bit**

The screenshot shows the Postman desktop application interface. At the top, there are tabs for 'Home', 'Workspaces', and 'APIs'. Below these, there's a section for 'Notion's Public Workspace'. On the left sidebar, there's a 'Collection' button with a trash icon. The main area shows a list of collections, including one for 'Notion API'.



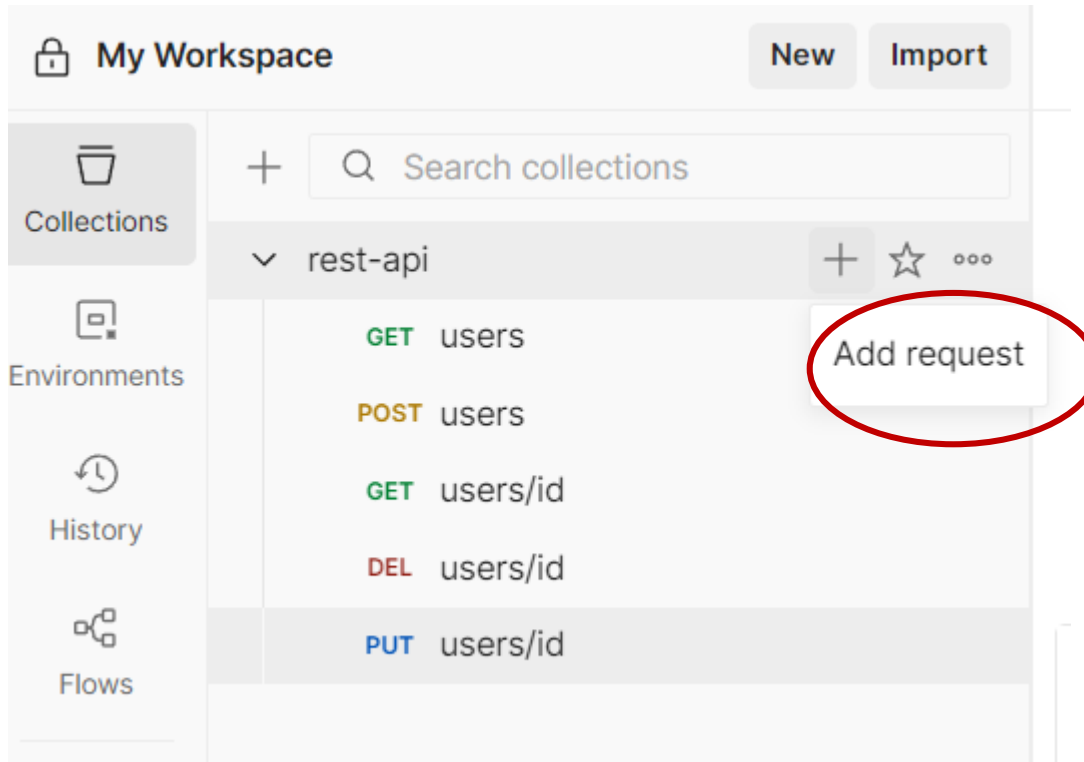
회원 관리 API

- Workspaces 생성



회원 관리 API

- 컬렉션(Collections)



회원 관리 API

■ 회원 가입

Save > POST 저장

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/users`
- Method:** `POST`
- Body:**

```
{
  "name": "클라우드",
  "email": "cloud100@time.com"
}
```
- Response:** `200 OK`, 19 ms, 184 B
- Save:** A red circle highlights the `Save` button in the top right corner.
- Test Results:** A list of test results is shown at the bottom, including `회원 가입 완료`.



회원 관리 API

- MySQL WorkBench

```
4 • use webdb;
5
6 • select * from user;
7
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

	id	reg_date	update_date	email	name
▶	1	2025-12-12 13:40:51.343804	NULL	cloud100@space.com	클라우드
	2	2025-12-12 13:41:28.759079	NULL	today12@time.kr	김기용
*	NULL	NULL	NULL	NULL	NULL



회원 관리 API

■ 회원 목록

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/users`
- Method:** `GET`
- Status:** `200 OK` (16 ms, 389 B)
- Response Type:** `JSON`
- Response Body:** A JSON array of two user objects.

Key	Value	Description
Key	Value	Description

```
1  [
2    {
3      "id": 1,
4      "name": "클라우드",
5      "email": "cloud100@space.com",
6      "regDate": "2025-12-12T04:40:51.343Z",
7      "updateDate": null
8    },
9    {
10     "id": 2,
11     "name": "김기용",
12     "email": "today12@time.kr",
13     "regDate": "2025-12-12T04:41:28.759Z",
14     "updateDate": null
15   }
16 ]
```



회원 관리 API

- 회원 정보(상세 보기)

The screenshot shows a REST client interface with the following details:

- URL Bar:** HTTP rest-api / users/id. Buttons for Save, Share, and a dropdown arrow are present.
- Method and URL:** GET http://localhost:8080/users/2. A Send button is to the right.
- Tabs:** Docs, Params, Auth, Headers (6), Body, Scripts, Tests, Settings. A Cookies link is on the far right.
- Query Params Table:**

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		
- Body Section:**
 - Status: 200 OK, 14 ms, 272 B. Icons for globe, save response, and more options are shown.
 - Format: JSON (selected). Buttons for Preview and Visualize are available.
 - Response Content:

```
1 {
2   "id": 2,
3   "name": "김기용",
4   "email": "today12@time.kr",
5   "regDate": "2025-12-12T04:41:28.759Z",
6   "updateDate": null
7 }
```



회원 관리 API

- 회원 삭제(id - 3)

rest-api / users/id

DELETE http://localhost:8080/users/3

Send

Docs Params Auth Headers (6) Body Scripts Tests Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body 200 OK • 15 ms • 185 B • Save Response

Raw Preview Visualize

1 회원 삭제 완료!



회원 관리 API

■ 회원 수정(id - 1)

The screenshot displays a REST client interface for a PUT request to `http://localhost:8080/users/1`. The request body is a JSON object with the following fields:

```
{
  "id": 1,
  "name": "클로봇",
  "email": "cloud@robot.com"
}
```

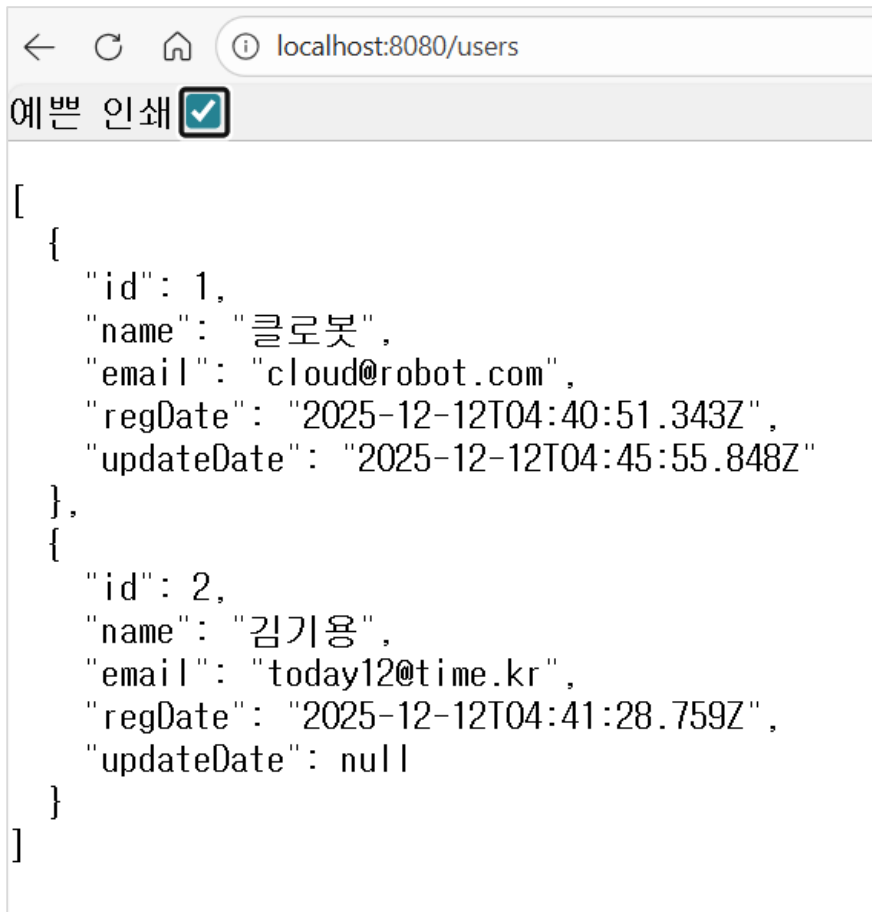
The response is a `200 OK` status with a response time of 393 ms and a body size of 294 B. The response body is a JSON object with the following fields:

```
{
  "id": 1,
  "name": "클로봇",
  "email": "cloud@robot.com",
  "regDate": "2025-12-12T04:40:51.343Z",
  "updateDate": "2025-12-12T04:45:55.848Z"
}
```



회원 관리 API

- 회원 목록 - 웹 브라우저로 보기



```
[
  {
    "id": 1,
    "name": "클로봇",
    "email": "cloud@robot.com",
    "regDate": "2025-12-12T04:40:51.343Z",
    "updateDate": "2025-12-12T04:45:55.848Z"
  },
  {
    "id": 2,
    "name": "김기용",
    "email": "today12@time.kr",
    "regDate": "2025-12-12T04:41:28.759Z",
    "updateDate": null
  }
]
```

