

A faint, light blue world map is visible in the background of the slide, centered behind the text.

# INTERNET ACADEMY

Institute of Web Design & Software Services

## Spring Boot 7

インターネット・アカデミー

# Spring Boot 7 目次

- REST APIとは
- REST APIでのCRUD操作

A faint, light blue world map is visible in the background of the slide, centered behind the title text.

# REST APIとは

# RESTとは

REST(REpresentational State Transfer)とは、APIの設計ルール(アーキテクチャスタイル)。2000年に、ロイ・フィールドینگが考案。

【通常(RESTではない形式)】

クライアント



GET、POST



サーバー



HTML

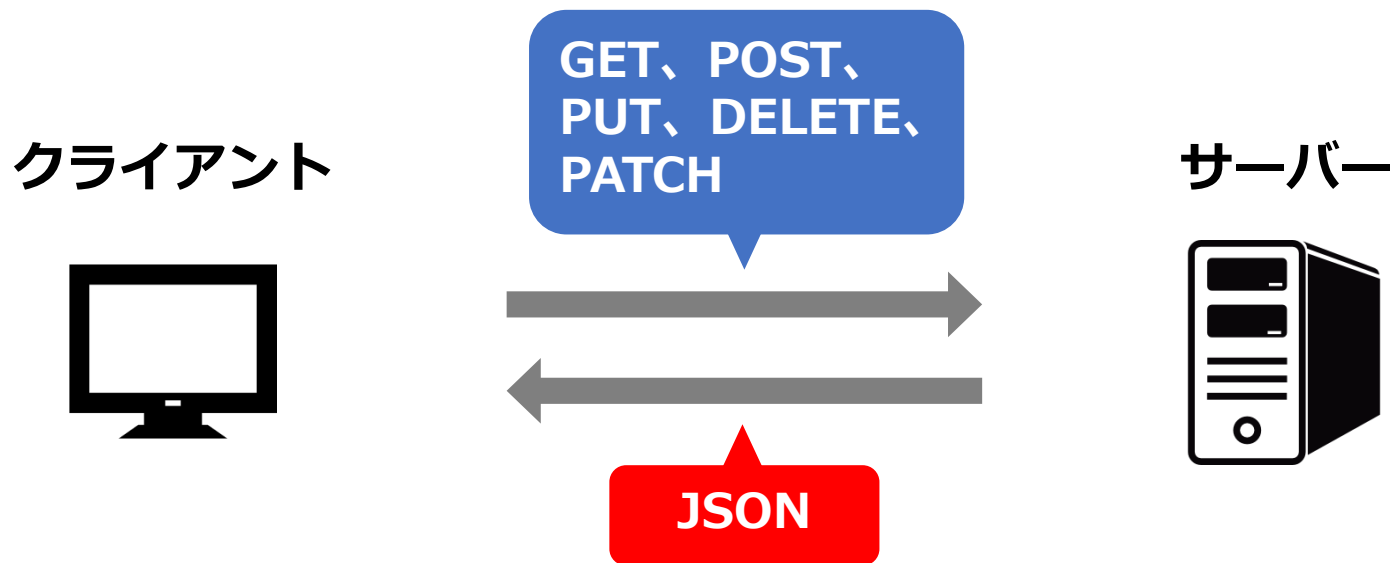
！ポイント

HTML形式で返ってくるため、次のページの内容に表示しなおす必要がある

# RESTとは

REST(REpresentational State Transfer)とは、APIの設計ルール(アーキテクチャスタイル)。2000年に、ロイ・フィールドینگが考案。

## 【REST形式】



## ！ポイント

JSON形式で返ってくるため、次のページに飛ばずにページの内容を書き換えで済む

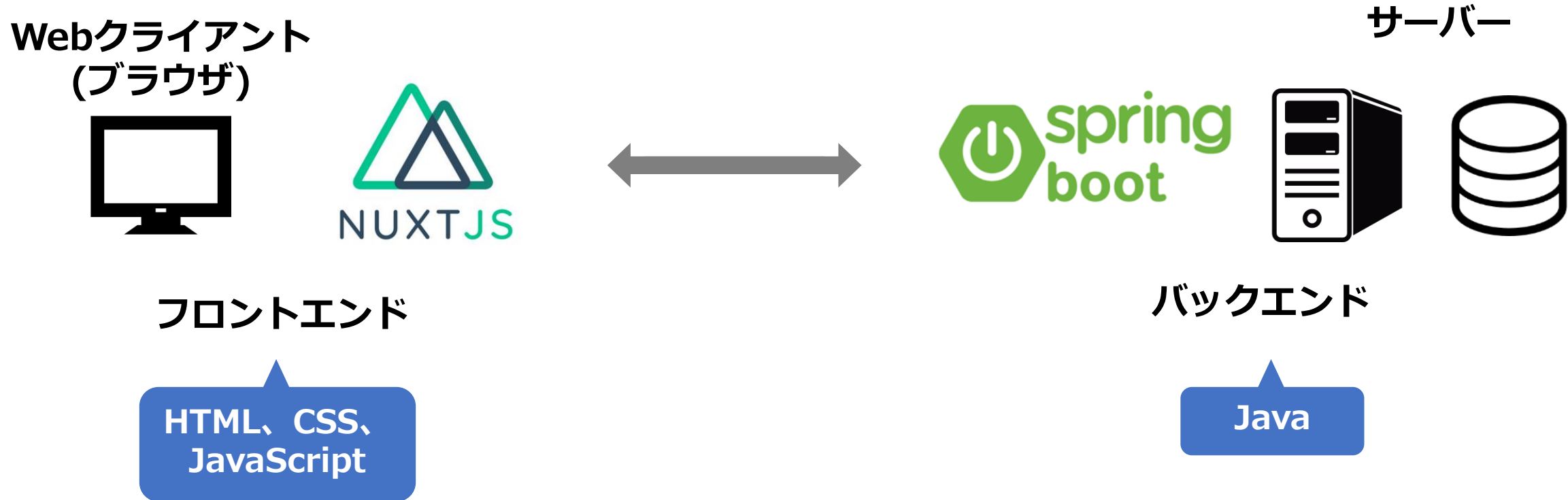
# RESTとCRUD

メソッド	CRUD	URL	処理内容
GET	Read	/tasks	全件を取得
GET	Read	/tasks/{id}	1件を取得
POST	Create	/tasks	登録
PUT	Update	/tasks/{id}	更新
DELETE	Delete	/tasks/{id}	削除
PATCH	Update	/tasks/{id}	部分更新

## ！ポイント

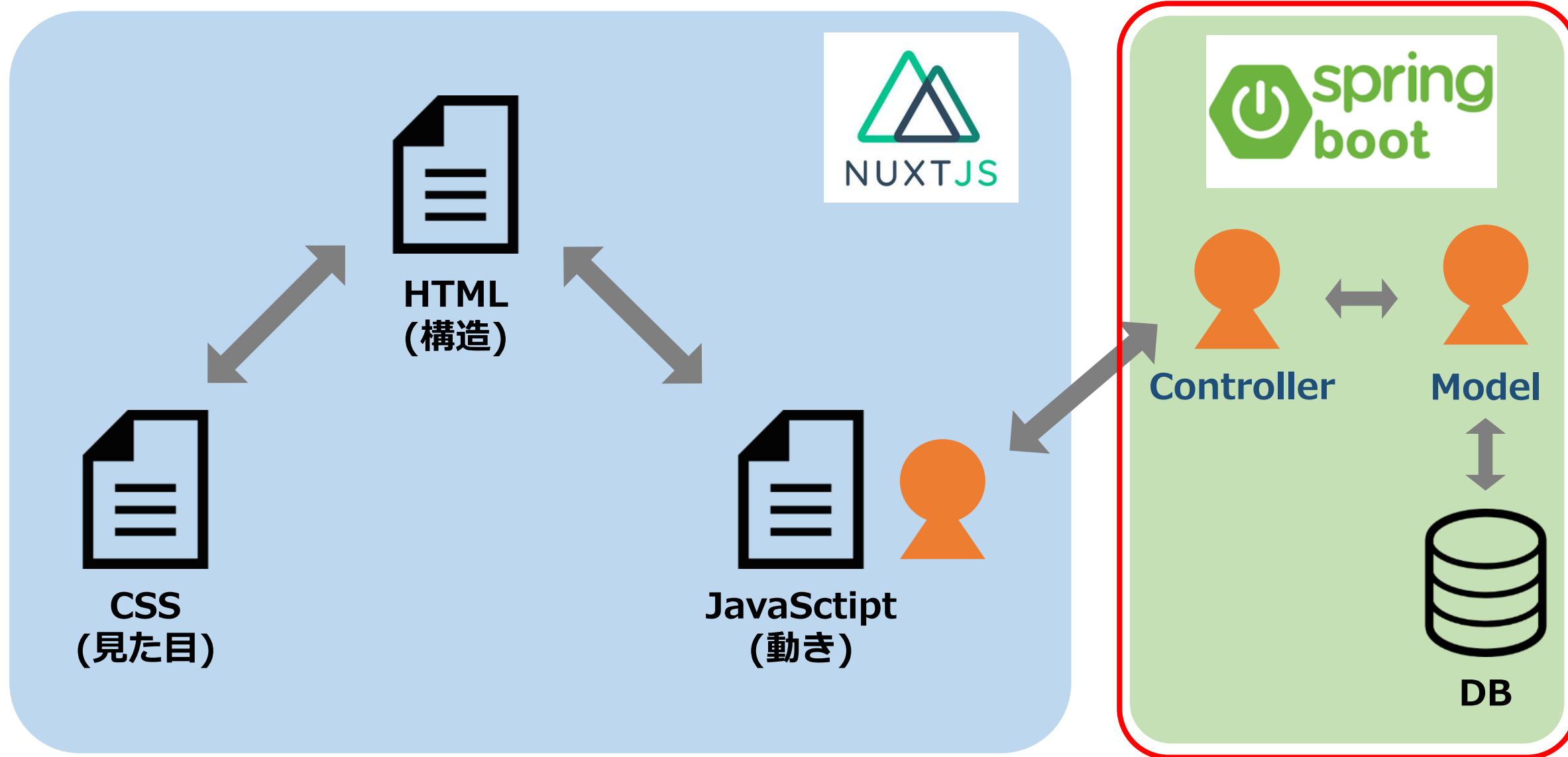
RESTでは、ReadがGET、CreateがPOSTとなる点に注意しましょう

# フロントエンドとバックエンド



# REST APIを利用したシステムの全体

## View







A faint, light blue world map is visible in the background of the slide, centered behind the title text.

# REST APIでのCRUD操作

# アプリケーション作成時の選択

項目	目的
Spring Boot DevTools	開発を便利にするツール (コード変更時の自動再起動など)
Thymeleaf	テンプレートエンジン
Spring Web	Spring MVCを使う
Spring Data JPA	JPAライブラリを使う
JDBC API	JDBCライブラリを使う
H2 Database	データベースにH2を使う
検証	バリデーション
Lombok	Javaの記述を簡素にする
<b>Restリポジトリ</b>	<b>REST APIを作る</b>

新規 Spring スターター・プロジェクト依存関係

Spring Boot バージョン: 2.4.5

使用頻度高:

☒ H2 Database

☐ MySQL Driver

☒ Spring Web

☒ JDBC API

☒ Spring Boot DevTools

☐ Thymeleaf

☒ Lombok

☒ Spring Data JPA

☒ 検証

使用可能:

選択済み:

▶ Alibaba

▶ Amazon Web サービス

▶ 開発ツール

▶ Google Cloud Platform

▶ I/O

▶ メッセージング

▶ Microsoft Azure

▶ NoSQL

▶ Observability

▶ Ops

▶ SQL

▶ セキュリティー

▶ Spring Cloud

▶ Spring Cloud Circuit Breaker

▶ Spring Cloud Config

X Spring Boot DevTools

X Lombok

X 検証

X JDBC API

X Spring Data JPA

X H2 Database

X Spring Web

X Rest リポジトリ

デフォルトにする

選択をクリア

?

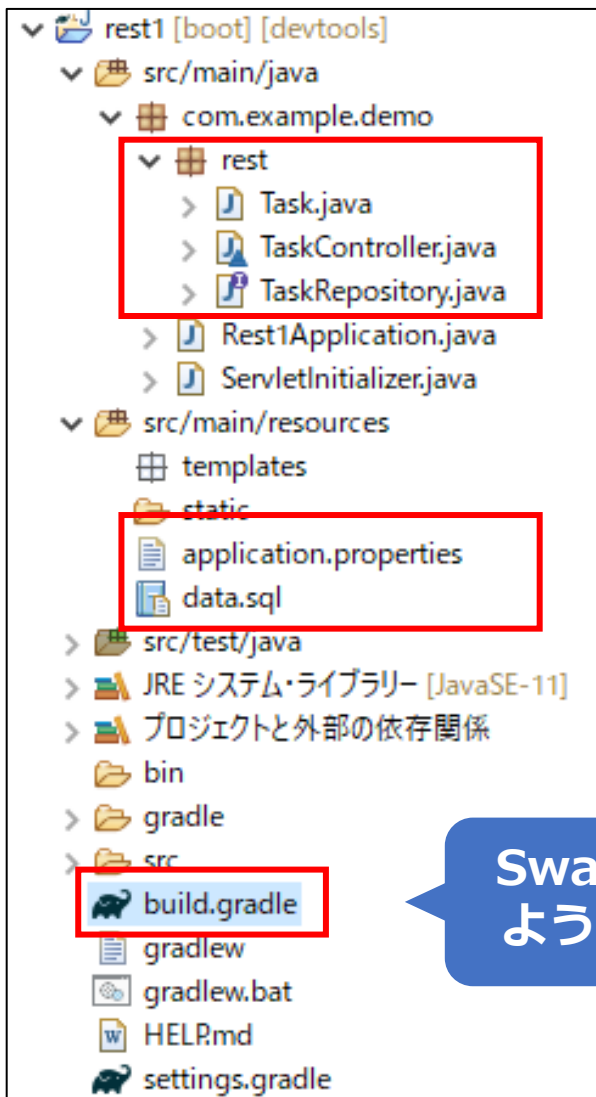
< 戻る(B)

次へ(N) >

完了(E)

キャンセル

# ファイル構成



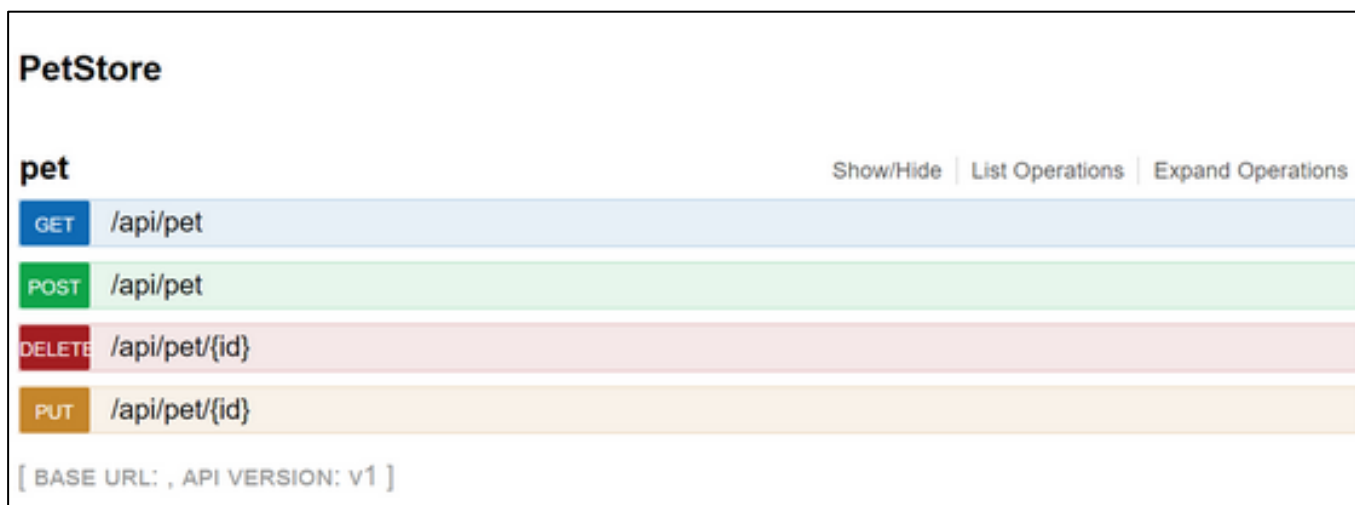
Swagger UIを使える  
ようにする指定を追記

# Swagger UI

Swagger UI(スワッガー ユーアイ)とは、Open APIドキュメントを視覚化できるツールです。これを使ってブラウザからAPIの仕様を確認することができます。

Swaggerは、Open APIの原型であり、Swaggerが3.0になる時に、OpenAPIという名称へ変更されました。

UIやEditorなどのツール群は、Swaggerという名称のまま使われています。



# データベースの設定

## **application.properties**

```
spring.datasource.driverClassName:org.h2.Driver  
spring.datasource.url:jdbc:h2:mem:test  
spring.datasource.username:sa  
spring.datasource.password:  
spring.h2.console.enabled:true
```

## **data.sql**

```
INSERT INTO task (id, name, completed) VALUES (1, 'タスク 1', false);  
INSERT INTO task (id, name, completed) VALUES (2, 'タスク 2', true);
```

※**schema.sql**は作成しない

# Swagger UIを使うための設定

## build.gradle

```
:
dependencies {
    implementation 'org.springdoc:springdoc-openapi-ui:1.5.2'
    implementation 'org.springdoc:springdoc-openapi-data-rest:1.5.2'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-data-rest'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'com.h2database:h2'
    annotationProcessor 'org.projectlombok:lombok'
    providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

# エンティティの指定

## Task.java

```
:  
  
@Entity  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
public class Task {  
    @Id  
    GeneratedValue(strategy = GenerationType.IDENTITY)  
    private long id;  
  
    @NotBlank  
    @Size(max = 255)  
    private String name;  
  
    @NotNull  
    private Boolean completed;  
}
```

# Model(DAO)の指定

## TaskRepository.java ※インターフェイスです

```
package com.example.demo.rest;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource
public interface TaskRepository extends JpaRepository<Task, Long>{

}
```



# Controllerの指定 1

TaskController.java

アクセスするURLを「localhost:8080/api1/…」とする指定

:

**@RequestMapping("/api1")**

後のスライドで詳しく紹介

**@RequiredArgsConstructor**

**@RestController**

このシステムをREST APIにするための指定

class TaskController {

private final TaskRepository repository;

**@Operation(summary = "タスクのテスト")**

Swagger UIの画面で出すメモ書き

**@RequestMapping("/")**

Task test() {

return new Task((long)1,"タスクのサンプル",false);

}

「localhost:8080/api1/」で実行する部分

※次のスライドにて記述

}

# Controllerの指定 2

## TaskController.java ※中の部分

```

:
@Operation(summary = "タスクの全件取得")
@GetMapping("/view")
List<Task> findAll() {
    return repository.findAll();
}

@Operation(summary = "タスクの登録")
@PostMapping("/comp")
Task save(@RequestBody Task task) {
    return repository.save(task);
}
}

```

「localhost:8080/api1/view」で実行する部分

「localhost:8080/api1/comp」で実行する部分

# Controllerの指定 1 の詳細解説

## TaskController.java

```
:  
@RequestMapping("/api1")  
@RequiredArgsConstructor  
@RestController  
class TaskController {  
    private final TaskRepository repository;  
  
    @Operation(summary = "タスクのテスト")  
    @RequestMapping("/")  
    Task test() {  
        return new Task((long)1,"タスクのサンプル",false);  
    }  
  
    ※次のスライドにて記述  
}
```

```
:  
class TaskController {  
    private final TaskRepository repository;  
  
    @Autowired  
    public TaskController(TaskRepository repository) {  
        this.repository = repository;  
    }  
:
```

**@RequiredArgsConstructorは、  
この部分を省略することができる  
lombokのアノテーション**

# 実行結果

最後の/(スラッシュ)忘れない

http://localhost:8080/api1/ にアクセス

## ▼Chrome



## ▼Firefox

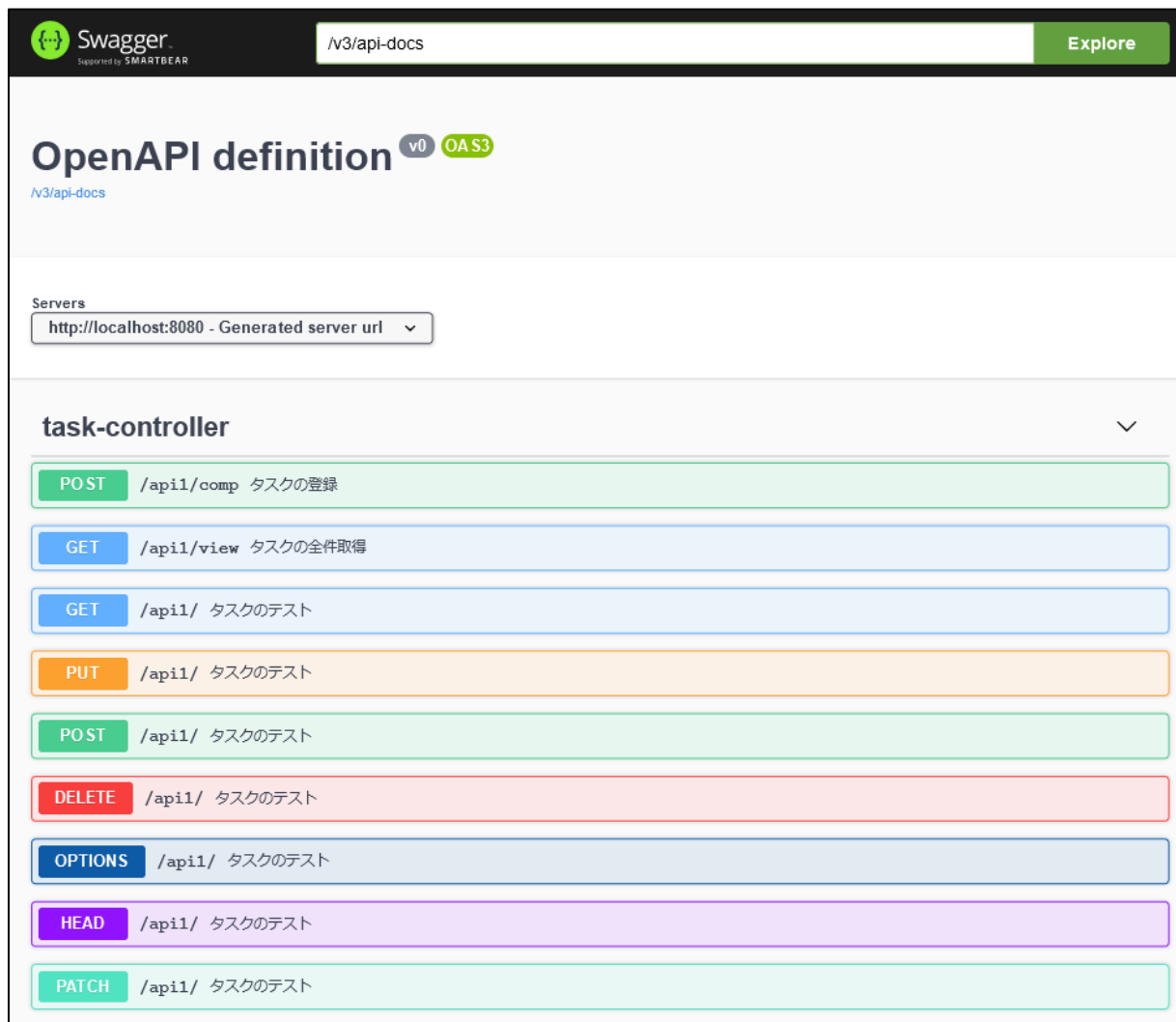


## ！ポイント

Firefoxだと、結果がわかりやすく表示されます

# 実行結果(Swagger UI)

Swagger UIの画面を出すために「<http://localhost:8080/swagger-ui.html>」 にアクセスします



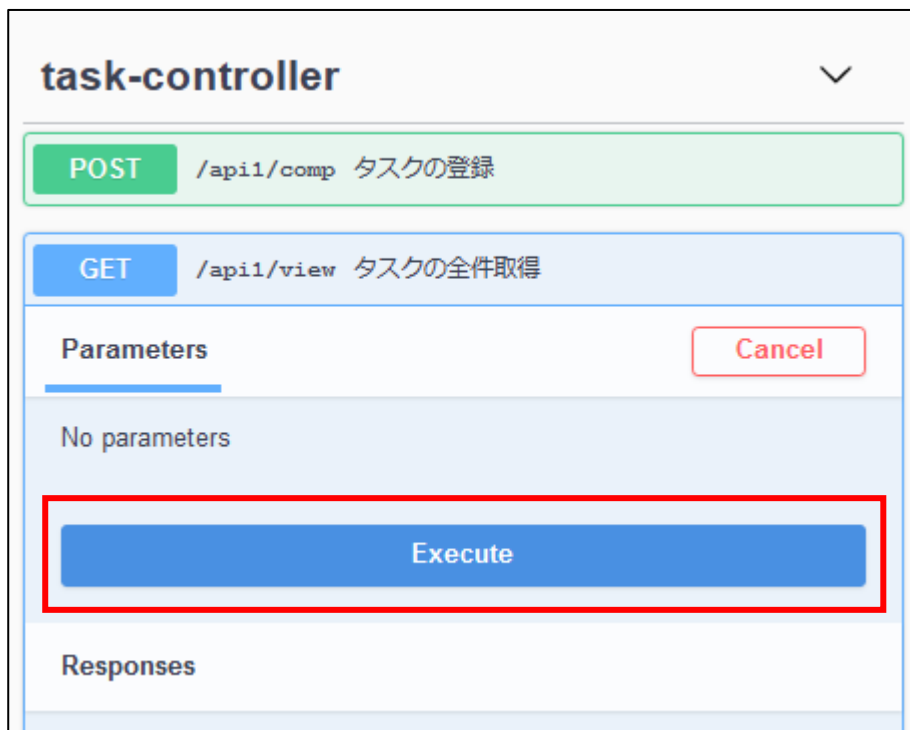
@PostMapping("/comp")

@GetMapping("/")

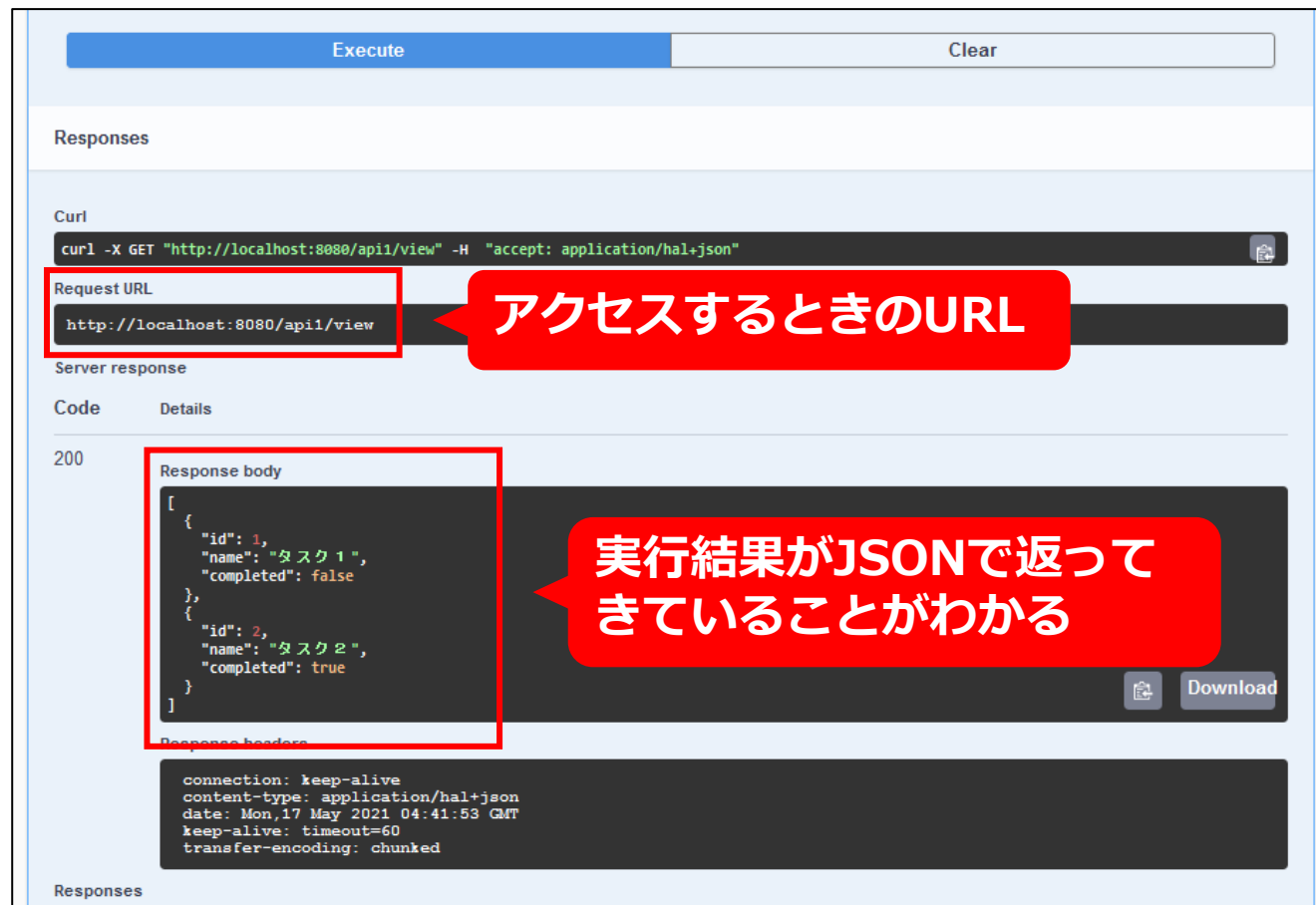
@RequestMapping("/")

# 実行結果(Swagger UI : 一覧表示)

http://localhost:8080/swagger-ui.html にアクセス



GETの項目「/api/view」を開いて、Executeをクリック



# 実行結果(Swagger UI : データ登録)

http://localhost:8080/swagger-ui.html にアクセス

POST /api1/comp タスクの登録

Parameters Cancel

No parameters

Request body required application/json

```
{  "name": "あいうえお",  "completed": true}
```

Execute

登録するデータを入力



Execute Clear

Responses

Curl

```
curl -X POST "http://localhost:8080/api1/comp" -H "accept: application/hal+json" -H "Content-Type: application/json" -d '{"name":"あいうえお","completed":true}'
```

Request URL

http://localhost:8080/api1/comp

Server response

Code	Details
200	<p>Response body</p> <pre>{  "id": 3,  "name": "あいうえお",  "completed": true}</pre>

Response headers

```
connection: keep-alive
content-type: application/hal+json
date: Mon, 17 May 2021 04:50:11 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	OK	No links

アクセスするときのURL

Codeが200と表示されており、正常に動作している

# エラーの例 1

## TaskController.javaのリクエスト指定を間違えていた

### ▼TaskController.java

```
    :  
    :  
    @Operation(summary = "タスクの登録")  
    @GetMapping("/comp")  
    Task save(@RequestBody Task task) {  
        return repository.save(task);  
    }  
}
```

処理内容がCreateなので、GETではなく  
POSTで受け取る必要ありだが、間違っている  
→@PostMapping("/comp")が正解



# エラーの例 1

実行したときに表示されるエラー内容

GET

/api1/comp タスクの登録

Parameters

Cancel

Name	Description
task * required	
object	
(query)	<pre>{   "name": "あいうえお",   "completed": true }</pre>

Execute

Clear



Responses

Curl

```
curl -X GET "http://localhost:8080/api1/comp?name=%E3%81%82%E3%81%84%E3%81%86%E3%81%88%E3%81%8A&completed=true" -H "accept: application/hal+json"
```

Request URL

```
http://localhost:8080/api1/comp?name=%E3%81%82%E3%81%84%E3%81%86%E3%81%88%E3%81%8A&completed=true
```

Server response

Code	Details
400 <small>Undocumented</small>	Error: Response body

```
{
  "timestamp": "2021-05-17T06:22:27.503+00:00",
  "status": 400,
  "error": "Bad Request",
  "trace": "org.springframework.http.converter.HttpMessageNotReadableException: Required request body is missing: com.example.demo.rest.Task com.example.demo.rest.TaskController.save(com.example.demo.rest.Task)",
  "message": "Required request body is missing: com.example.demo.rest.Task com.example.demo.rest.TaskController.save(com.example.demo.rest.Task)",
  "path": "/api1/comp"
}
```

Response headers

```
connection: close
content-type: application/hal+json
date: Mon, 17 May 2021 06:22:27 GMT
transfer-encoding: chunked
```

Responses

400番のエラーが発生

REST APIでは、GETはRead、POSTはCreateと役割分担しているため、「@PostMapping("/comp")」と書かなければなりません

## エラーの例 2

### Task.javaのアノテーションの記述が不足

#### ▼Task.java

```
:  
public class Task {  
    @Id  
    @GeneratedValue  
    private long id;  
    :
```

AUTO INCREMENTを有効にするためには、  
「@GeneratedValue(strategy = GenerationType.IDENTITY)」  
と記述する必要がある

## エラーの例 2

実行したときに表示されるエラー内容

GET

/api1/comp タスクの登録

Parameters

Cancel

Name	Description
<b>task</b> * required	
object	
(query)	<pre>{   "name": "あいうえお",   "completed": true }</pre>

Execute

Clear



Responses

Curl

```
curl -X POST "http://localhost:8080/api1/comp" -H "accept: application/hal+json" -H "Content-Type: application/json" -d '{"name":"あいうえお","co
```

Request URL

```
http://localhost:8080/api1/comp
```

Server response

Code	Details
500	Error: Undocumented Response body

500番のエラーが発生

```
{  
  "timestamp": "2021-05-17T06:33:09.712+09:00",  
  "status": 500,  
  "error": "Internal Server Error",  
  "trace": "org.springframework.dao.DataIntegrityViolationException: could not execute statement; SQL [n/a]; constraint [\"PRIMARY KEY ON PUBLIC.TASK(ID) [1, FALSE, STRINGDECODE('\\\\u30bf\\\\u30b9\\\\u30af')\"],  
  \"message\": \"could not execute statement; SQL [n/a]; constraint [\"PRIMARY KEY ON PUBLIC.TASK(ID) [1, FALSE, STRINGDECODE('\\\\u30bf\\\\u30b9\\\\u30af')\"],  
  \"path\": \"/api1/comp\"  
}
```

Response headers

```
connection: close  
content-type: application/hal+json  
date: Mon, 17 May 2021 06:33:09 GMT  
transfer-encoding: chunked
```

Responses

エラーの詳細が表示

アプリケーション内部での処理エラーでは、500番のエラーが出ます