

Spring

Content

1. About Framework
2. Spring
 - 2.1. About Spring
 - 2.2. Type Of Spring
 - 2.3. Flow Of Spring
 - 2.4. Environment Setup & Folder Structure
 - 2.5. Spring Annotation (@)
 - 2.6. Spring Mybatis
 - 2.7. Debug
 - 2.8. JUnit
 - 2.9. Deploy (heroku)

1. About Framework

Why Framework? It is quite difficult to develop Java applications with only Core Java. It is absolutely necessary to connect the screen and the processing on the back-end well, and to create a project with good efficiency and quality.

There is not much difference in the concept and usage of Java, but each framework has a different development method.

Java has the following framework.

1. Spring
2. Hibernate
3. Struts
4. JSF etc

2. Spring

2.1. About Spring

Java programs are complex and feature many heavyweight components. Heavyweight means the components are dependent on the underlying operating system (OS) for their appearance and properties.

Spring is considered to be a secure, low-cost and flexible framework. Spring improves coding efficiency and reduces overall application development time because it is lightweight -- efficient at utilizing system resources -- and has a lot of support.

Spring removes tedious configuration work so that developers can focus on writing business logic. Spring handles the infrastructure so developers can focus on the application.

2.2. Type Of Spring

Spring can create various types of applications. Such as

- a) Spring Web
- b) Spring Security
- c) Spring API
- d) Spring Batch

a)Spring Web

b)Spring Security

It automatically handles app authentication and security measures.

c)Spring API

API for interacting with mobile apps and external systems. Request and Response with JSON data.

c)Spring Batch

You can easily develop a batch's function. Such as CVS download, upload, large data transfer etc ...

2.3. Flow Of Spring

There are three types of layers.

a)View Layer :

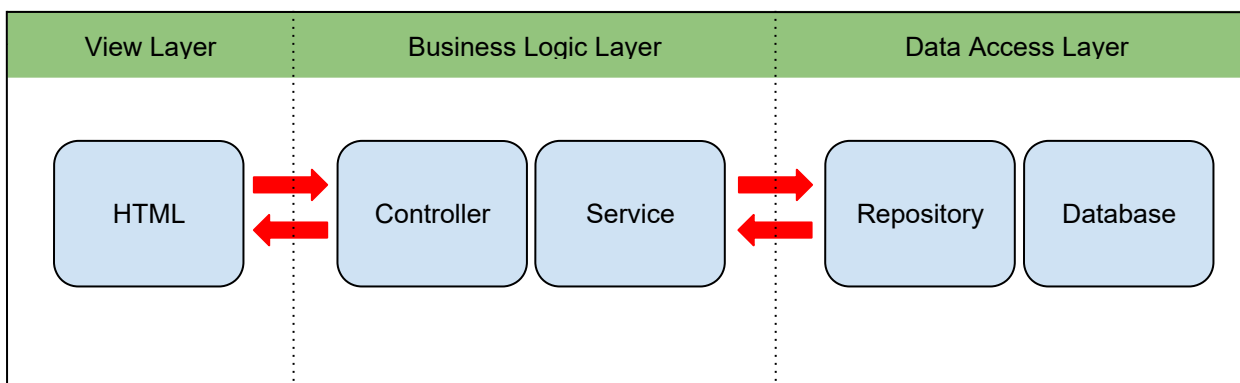
The outer layer that shows content and interaction with the user. (html, jsp)

b)Business Logic Layer :

The central layer that deals with the logic of a program.

c)Data Access Layer :

The deep layer that deals with data retrieval from databases.

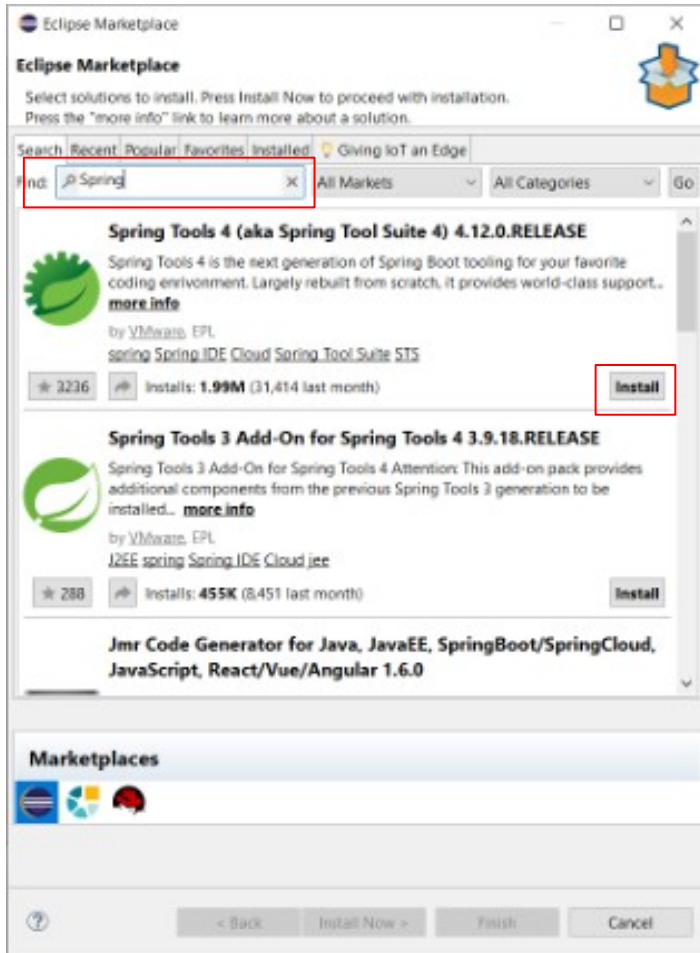


2.4. Environment Setup & Folder Structure

Step 1 :

Let's put the Spring Plugin into the existing Eclipse.

Launch Eclipse IDE and press Help > Eclipse Marketplace.



Step 2 :

Restart Eclipse, File > New > Other > Spring Boot > Spring Starter Project

The screenshot shows the 'New Spring Starter Project' dialog box in Eclipse. The dialog is titled 'New Spring Starter Project' and features a green power button icon in the top right corner. The fields and options are as follows:

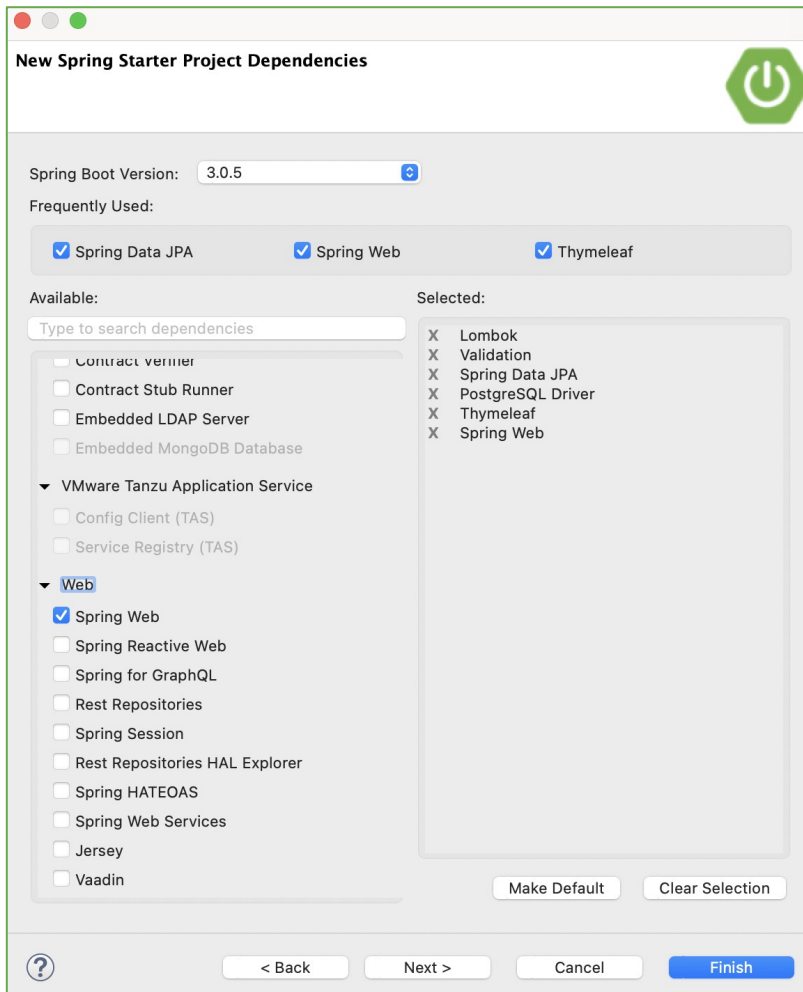
- Service URL:** `https://start.spring.io`
- Name:** `HelloWorld` (Annotated with 'Project Name')
- Use default location:** ☒
- Location:** `/Users/kyawwinthu/Downloads/プロジェクト/dev/develop/kizuna/006. so` (Annotated with 'Select maven')
- Type:** `Maven` (Annotated with 'Jar or War')
- Packaging:** `Jar` (Annotated with 'Jar or War')
- Java Version:** `17` (Annotated with 'Installed java')
- Language:** `Java` (Annotated with 'Installed java')
- Group:** `co.jp.hello` (Annotated with 'Folder structure')
- Artifact:** `HelloWorld`
- Version:** `0.0.1-SNAPSHOT`
- Description:** `This is HelloWord Project.`
- Package:** `co.jp.hellq` (Annotated with 'Package structure')

The 'Working sets' section at the bottom includes:

- ☐ Add project to working sets
- Working sets:** (Annotated with 'Package structure')
- Buttons:** `New...`, `Select...`, `< Back`, `Next >`, `Cancel`, `Finish`

Step 3 :

Choose dependencies.



The image shows a 'New Spring Starter Project Dependencies' dialog box. At the top, it has a title bar with standard window controls and a green power icon. Below the title bar, the 'Spring Boot Version' is set to '3.0.5'. Under 'Frequently Used:', three dependencies are checked: 'Spring Data JPA', 'Spring Web', and 'Thymeleaf'. The 'Available:' section on the left contains a search bar and a list of dependencies. The 'Web' category is expanded, showing 'Spring Web' checked. The 'Selected:' section on the right lists the chosen dependencies: 'Lombok', 'Validation', 'Spring Data JPA', 'PostgreSQL Driver', 'Thymeleaf', and 'Spring Web'. At the bottom of the dialog are buttons for '< Back', 'Next >', 'Cancel', and 'Finish'.

New Spring Starter Project Dependencies

Spring Boot Version: 3.0.5

Frequently Used:

- ☒ Spring Data JPA
- ☒ Spring Web
- ☒ Thymeleaf

Available:

Type to search dependencies

- ☐ Contract verifier
- ☐ Contract Stub Runner
- ☐ Embedded LDAP Server
- ☐ Embedded MongoDB Database
- ▼ VMware Tanzu Application Service
 - ☐ Config Client (TAS)
 - ☐ Service Registry (TAS)
- ▼ Web
 - ☒ Spring Web
 - ☐ Spring Reactive Web
 - ☐ Spring for GraphQL
 - ☐ Rest Repositories
 - ☐ Spring Session
 - ☐ Rest Repositories HAL Explorer
 - ☐ Spring HATEOAS
 - ☐ Spring Web Services
 - ☐ Jersey
 - ☐ Vaadin

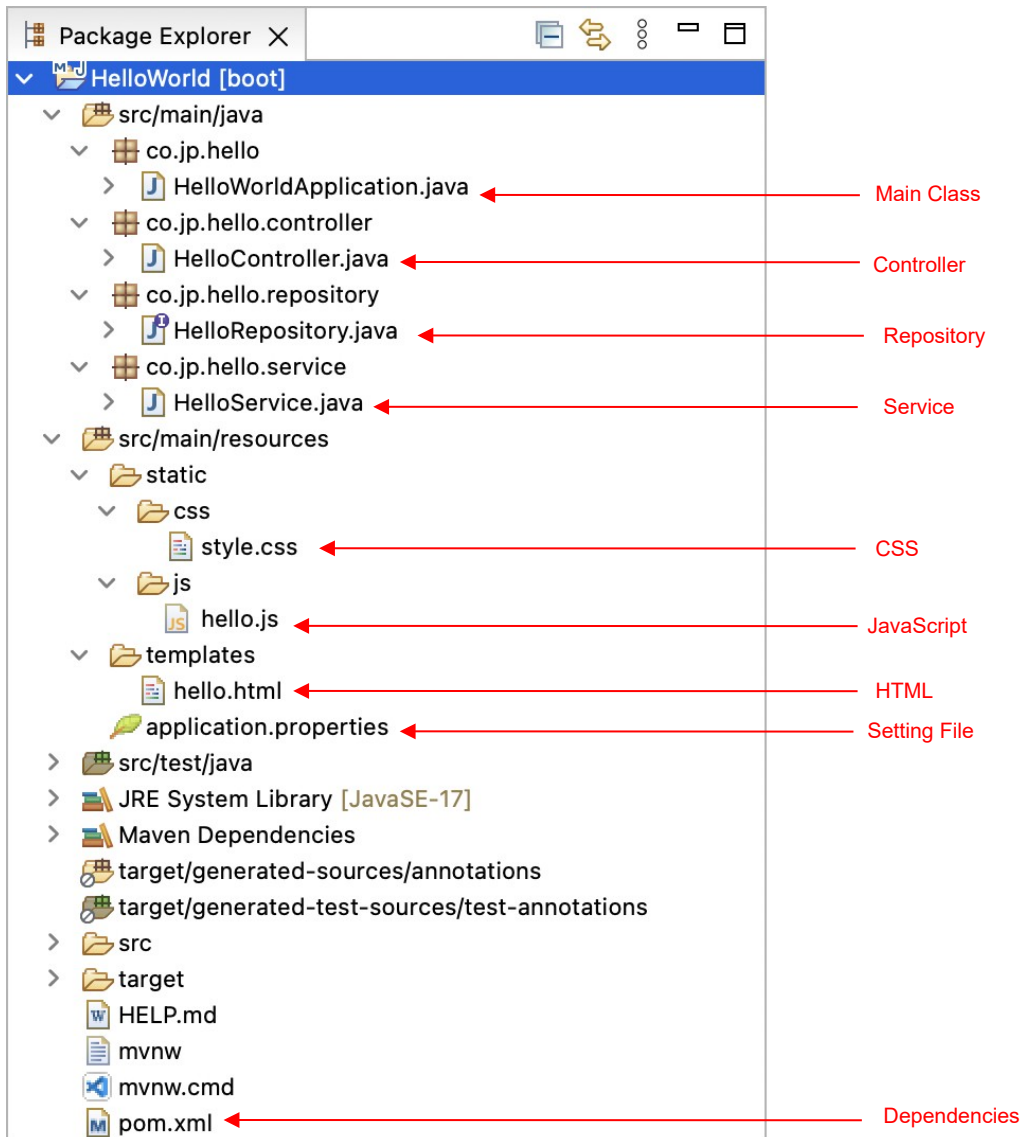
Selected:

- X Lombok
- X Validation
- X Spring Data JPA
- X PostgreSQL Driver
- X Thymeleaf
- X Spring Web

Make Default Clear Selection

? < Back Next > Cancel Finish

Step 4 : Folder Structrue



2.5. Spring Annotation (@)

All the parts of Spring work with annotation. Annotation defines the meaning of Class and its properties.

Spring MVC Annotations

- 1) **@SpringBootApplication** : Main function.
- 2) **@Controller** : Web request handler.
- 3) **@RequestMapping** : Map the HTTP request route.
- 4) **@ModelAttribute** : Pass the value to View.

Main Class

```
package co.jp.hello;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class HelloWorldApplication {
    public static void main(String[] args) {
        SpringApplication.run(HelloWorldApplication.class, args);
    }
}
```

Controller

```
package co.jp.hello.controller;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import co.jp.hello.entity.Person;
@Controller
public class HelloController {

    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String index(Model model) {
        return "index";
    }

    @RequestMapping(value = "/greeting", method = RequestMethod.POST)
    public String greeting(Model model, @ModelAttribute("form") Person p) {
        model.addAttribute("name", p.getName());
        return "hello";
    }
}
```

Entity


```

package co.jp.hello.entity;

public class Person {

    String name;

    public String getName() {
        return name;
    }

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

```

Index.html

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<div style="text-align:center;">
    <form th:action="@{/greeting}" method="post" th:object="${person}">
        <h1>Enter Your Name</h1>
        <input type="text" name="name">
        <input type="submit" value="Enter">
    </form>
</div>
</body>
</html>

```

hello.html

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<div style="text-align:center">
    <h1>Hello</h1>
    <span th:text="${name}"></span><br>
</div>
</body>
</html>

```

Result:

Insert title here

×

+

▼

←

→

↻

localhost:8080/hello

🔗

☆

⚙️

🖱️

k

Update

⋮

Enter Your Name

Insert title here

×

+

▼

←

→

↻

localhost:8080/greeting

🔗

☆

⚙️

🖱️

k

Update

⋮

Hello

山田

- 1) **@Entity** : Class represents Database's Table.
- 2) **@Data** : No need to write getter and setter.(Lombok)
- 3) **@Table** : Table name.
- 4) **@Id** : Parameter represents Table's ID.
- 5) **@GeneratedValue** : Id's value is auto generate and auto increase.
- 6) **@Column** : Column name.
- 7) **@Autowired** : It injects object dependency, bean, service, repository etc.
- 8) **@PathVariable** : Get the value which passed the value from route.
- 9) **@RequestParam** : Get the value from the request object.
- 10) **@Service** : Write the business logic.
- 11) **@Repository** : Data Access Object (DAO) that accesses the database directly.
- 12) **@Transactional** : Service uses transactional and can rollback if fail.

Step1 : Create Database and table

PostgreSQL (PgAdmin)

```
CREATE TABLE IF NOT EXISTS public.person
(
  id integer NOT NULL DEFAULT nextval('person_id_seq'::regclass),
  name character varying(255) COLLATE pg_catalog."default" NOT NULL,
  CONSTRAINT person_pkey PRIMARY KEY (id)
)
```

Step2 : set Database Connection

application.properties

```
spring.jpa.database=POSTGRESQL
spring.datasource.url=jdbc:postgresql://localhost:5432/helloworld
spring.datasource.username=postgres
spring.datasource.password=postgres
```

Step3 : Entity

```

package co.jp.hello.entity;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.Data;
@Entity
@Data
@Table(name="person")
public class Person {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    int id;
    @Column(name="name")
    String name;
}

```

Setp4: Controller

```

package co.jp.hello.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import co.jp.hello.entity.Person;
import co.jp.hello.service.PersonService;
@Controller
public class PersonController {

    @Autowired
    PersonService personService;
    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String index(Model model) {
        model.addAttribute("personList", this.personService.getPersonList());
        return "personList";
    }

    @RequestMapping(value = "/add", method = RequestMethod.GET)
    public String add(Model model) {
        model.addAttribute("form", new Person());
        return "personAdd";
    }

    @RequestMapping(value = "/add", method = RequestMethod.POST)
    public String addConfirm(Model model, @ModelAttribute("form") Person p) {
        this.personService.addPerson(p);
    }
}

```

```

@RequestMapping(value = "/detail/{id}", method = RequestMethod.GET)
public String detail(Model model, @PathVariable int id) {
    model.addAttribute("form", this.personService.getPerson(id));
    return "personDetail";
}

@RequestMapping(value = "/update", method = RequestMethod.GET)
public String update(Model model, @RequestParam("id") int id) {
    model.addAttribute("form", this.personService.getPerson(id));
    return "personUpdate";
}

@RequestMapping(value = "/update", method = RequestMethod.POST)
public String updateConfirm(Model model, @ModelAttribute("form") Person p) {
    this.personService.updatePerson(p);
    return "redirect:/";
}

@RequestMapping(value = "/delete", method = RequestMethod.GET)
public String delete(Model model, @RequestParam("id") int id) {
    this.personService.deletePerson(id);
    return "redirect:/";
}

```

Step5 : Service

```

package co.jp.hello.service;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.interceptor.TransactionInterceptor;
import co.jp.hello.entity.Person;
import co.jp.hello.repository.PersonRepository;
import jakarta.transaction.Transactional;

@Service
@Transactional
public class PersonService {
    @Autowired
    PersonRepository personRepository;
    /**
     * Get Person List
     *
     * @return
     */
    public List<Person> getPersonList() {
        List<Person> pList = new ArrayList<Person>();
        pList = this.personRepository.findAll();
        return pList;
    }
}

```

```

/**
 * Add Person
 *
 * @param p
 */
public void addPerson(Person p) {
    try {
        this.personRepository.save(p);
    } catch (Exception e) {
        TransactionInterceptor.currentTransactionStatus().setRollbackOnly();
    }
}

/**
 * Get Person Detail
 *
 * @param id
 * @return
 */
public Person getPerson(int id) {
    Optional<Person> p = this.personRepository.findById(id);
    return p.get();
}

/**
 * Update Person
 *
 * @param p
 */
public void updatePerson(Person p) {
    try {
        this.personRepository.save(p);
    } catch (Exception e) {
        TransactionInterceptor.currentTransactionStatus().setRollbackOnly();
    }
}

```

```

/**
 * Delete Person
 *
 * @param id
 */
public void deletePerson(int id) {
    try {
        this.personRepository.deleteById(id);
    } catch (Exception e) {
        TransactionInterceptor.currentTransactionStatus().setRollbackOnly();
    }
}
}

```

```

package co.jp.hello.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import co.jp.hello.entity.Person;
@Repository
public interface PersonRepository extends JpaRepository<Person, Integer> {

}

```

Step7 : PersonList.html

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<div style="text-align:center">
<h1>Person Management</h1><br>
<a th:href="@{/add}">Add</a><br><br>
<table border="1" align="center" >
<tr>
<th width="50px">No.</th>
<th width="100px" align="left">Name</th>
<th width="200px" align="center">Action</th>
</tr>
<tr th:each="person,index : ${personList}">
<td><span th:text="${index.count}"></span></td>
<td align="left"><span th:text="${person.name}"></span></td>
<td>
<a th:href="@{/detail/{id}(id=${person.id})}">Detail</a>&nbsp;&nbsp;&nbsp;
<a th:href="@{/update?id=' + ${person.id}"}>Update</a>&nbsp;&nbsp;&nbsp;
<a th:href="@{/delete?id=' + ${person.id}"}>Delete</a>&nbsp;&nbsp;&nbsp;
</td>
</tr>
</table>
</div>
</body>
</html>

```

Step8 : personAdd.html

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<div style="text-align:center;">
    <form th:action="@{/add}" method="post" th:object="${form}">
        <h1>Person Add</h1>
        <span>Name : &nbsp;</span><input type="text" name="name">
        <input type="submit" value="Enter">
    </form>
</div>
</body>
</html>

```

Step9 : personDetail.html

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<div style="text-align:center;">
    <h1>Person Detail</h1><br>
    <span>ID : <span th:text="${form.id}"></span></span><br>
    <span>Name : <span th:text="${form.name}"></span></span>
</div>
</body>
</html>

```

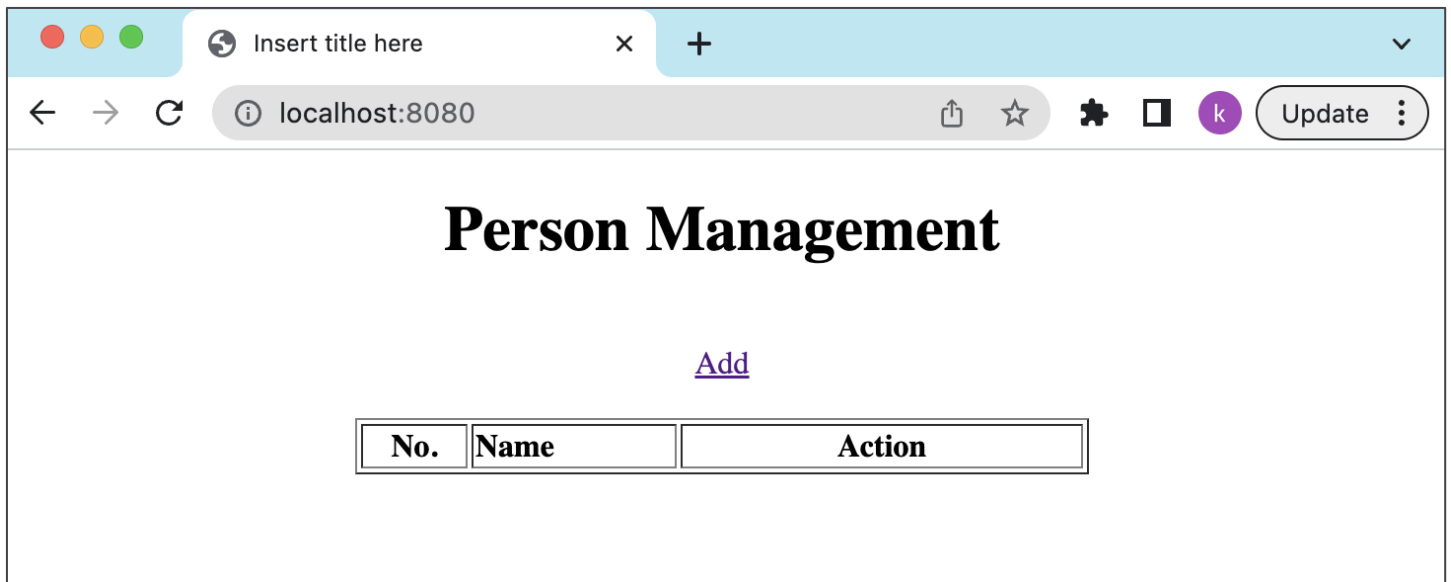
Step10 : personUpdate.html


```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<div style="text-align:center;">
    <form th:action="@{/update}" method="post" th:object="${form}">
        <input type="hidden" th:field="**{id}">
        <h1>Person Update</h1>
        <span>Name : &nbsp;  </span><input type="text" th:field="**{name}">
        <input type="submit" value="Enter">
    </form>
</div>
</body>
</html>

```

Result :



Insert title here

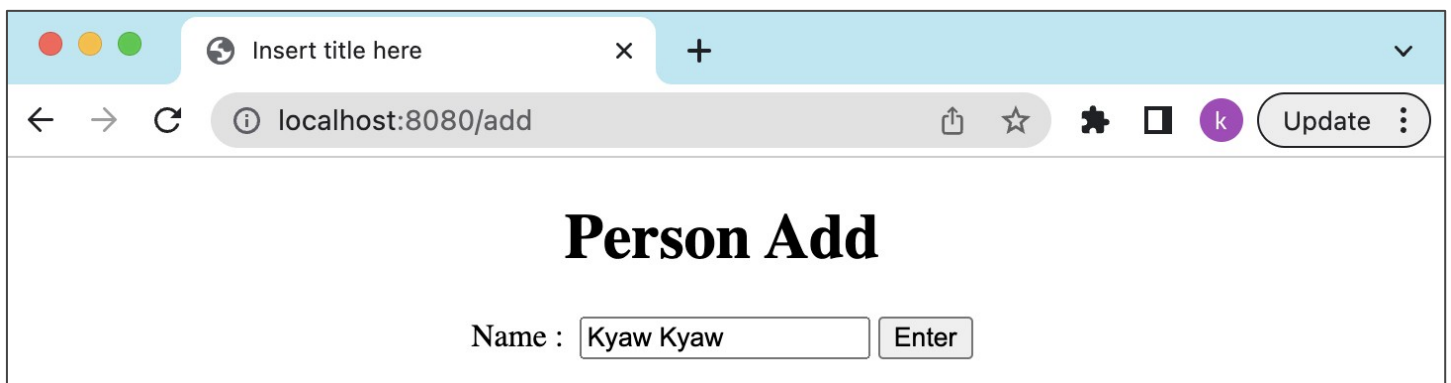
localhost:8080

Update

Person Management

[Add](#)

No.	Name	Action
-----	------	--------



Insert title here

localhost:8080/add

Update

Person Add

Name :

Insert title here

×

+

▼

←

→

↺

localhost:8080

📄

☆

⚙️

🖱️

k

Update

⋮

Person Management

[Add](#)

No.	Name	Action
1	Kyaw Kyaw	Detail Update Delete

Insert title here

×

+

▼

←

→

↺

localhost:8080/detail/6

📄

☆

⚙️

🖱️

k

Update

⋮

Person Detail

ID : 6
Name : Kyaw Kyaw

Insert title here

×

+

▼

←

→

↺

localhost:8080/update?id=6

📄

☆

⚙️

🖱️

k

Update

⋮

Person Update

Name :

- 1) **@Valid** : The request mapping uses Entity's valid.
- 2) **@NotBlank** : Check the String is not null and the length is greater than 0.
- 3) **@NotNull** : Check the annotated Char, Collection, Map, Array, Object is not null.
- 4) **@NotEmpty** : Check the annotated element is not null and not empty.
- 5) **@Size** : Check annotated element's size is between min value and max value.
- 6) **@Min** : Check the annotated value is greater than or equal to the specified minimum value.
- 7) **@Max** : Check the annotated value is smaller than or equal to the specified maximum value.
- 8) **@Pattern** : Check that the annotated string matches the regular expression considering the given flag matches.
- 9) **@AssertTrue** : Check that the annotated element is true.
- 10) **@AssertFalse** : Check that the annotated element is false.
- 11) **@Negative** : Check if the element is strictly smaller than 0.
- 12) **@Positive** : Check if the element is strictly greater than 0.
- 13) **@NegativeOrZero** : Check if the given element is smaller than or equal to 0.
- 14) **@PositiveOrZero** : Check if the given element is greater than or equal to 0.
- 15) **@Email** : Check the string is a valid email.
- 16) **@Range** : Checks that the value is between the specified minimum and maximum values.

```

import jakarta.validation.constraints.AssertTrue;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.Max;
import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotEmpty;
import jakarta.validation.constraints.Pattern;
import jakarta.validation.constraints.Size;
import lombok.Data;
@Data
public class PersonForm {
    int id;
    @NotBlank(message = "名前を入力してください。")
    @Size(max=30, message = "名前は 30 桁以内に入力してください。")
    String name;

    @NotEmpty(message = "住所を入力してください。")
    String address;

    @Pattern(regexp="^[0][1-9][0-9]*$", message = "年齢は数字を入力してください。")
    String age;

    @Min(value=200000 , message = "給料は 200000 円以上入力してください。")
    @Max(value=1000000 , message = "給料は 1000000 円以下入力してください。")
    String salary;

    @NotEmpty(message = "メールを入力してください。")
    @Email(regexp = "[a-zA-Z0-9_!#$%&'*/=?`{}~^.-]+@[a-zA-Z0-9.-]+$",
        message = "メールを正しく入力してください。")
    String email;

    @AssertTrue(message = "同意をチェックしてください。")
    boolean approve;
}

```

Controller

```

@Controller
public class PersonController {

    @RequestMapping(value = "/add", method = RequestMethod.POST)
    public String addConfirm(Model model,@Valid @ModelAttribute("form") PersonForm p, BindingResult result) {

        if(result.hasErrors()) {
            return "personAdd";
        }
        this.personService.addPerson(p);
        return "redirect:";
    }
}

```

personAdd.html

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
</head>
<body>
<div style="text-align:center;">
    <form th:action="@{/add}" method="post" th:object="${form}">
        <h1>Person Add</h1>
        <div align="center">
            <span>Name : &nbsp;</span><input type="text" name="name">
            <span th:if="${#fields.hasErrors('name')}" th:errors="**{name}"></span><br><br>
            <span>Address : &nbsp;</span><input type="text" name="address">
            <span th:if="${#fields.hasErrors('address')}" th:errors="**{address}"></span><br><br>
            <span>Age : &nbsp;</span><input type="text" name="age">
            <span th:if="${#fields.hasErrors('age')}" th:errors="**{age}"></span><br><br>
            <span>Salary : &nbsp;</span><input type="text" name="salary">
            <span th:if="${#fields.hasErrors('salary')}" th:errors="**{salary}"></span><br><br>
            <span>Email : &nbsp;</span><input type="text" name="email">
            <span th:if="${#fields.hasErrors('email')}" th:errors="**{email}"></span><br><br>
            <span>Approve : &nbsp;</span><input type="checkbox" name="approve" value="true">
            <span th:if="${#fields.hasErrors('approve')}" th:errors="**{approve}"></span><br><br>
            <input type="submit" value="Save">
        </div>
    </form>
</div>
</body>
</html>

```

Result :

Insert title here

×

+

▼

←

→

↻

localhost:8080/add

📄

☆

⚙️

🖨️

👤 k

Update

⋮

Person Add

Name :

名前を入力してください。

Address :

住所を入力してください。

Age :

年齢は数字を入力してください。

Salary :

給料は1000000円以下入力してください。
給料は200000円以上入力してください。

Email :

メールを正しく入力してください。
メールを入力してください。

Approve :

☐

同意をチェックしてください。

Save

- 1) **@RestController** : use in REST API controller class.
- 2) **@GetMapping** : a composed version of @RequestMapping(method = RequestMethod.GET)
- 3) **@PostMapping** : a composed version of @RequestMapping(method = RequestMethod.POST)
- 4) **@RequestBody** : accept parameter with JSON format.
- 5) **@ResponseBody** : response parameter with JSON format.
- 6) **@RequestHeader** : To read the Header of an HTTP request in a controller method.
- 7) **@ResponHeader** : An HTTP Response also contains Header information and can include custom key/value pairs in the Response Header.
- 8) **@CrossOrigin** : use for configuring allowed origins.

Download Postman to access data with API.

<https://www.postman.com/downloads/>

Select data from “person” table by using @GetMapping annotation.

“person” table has the following data.

	id [PK] integer	name character varying (255)	address character varying (255)	age character varying (255)	salary integer	email character varying (255)	approve boolean
1	1	nal	Tokyo	30	200000	nal@test.com	true
2	2	suzuki	Tokyo	25	250000	suzuki@test.com	true

PersonController.java

```
@CrossOrigin(origins = "http://localhost:8081")
@RestController
public class PersonController {

    @Autowired
    PersonService personService;

    @GetMapping("/api/person")
    public ResponseEntity<List<Person>> getAllPersons (@RequestParam(required
= false) int id) {
        try {
            Person person = new Person();
            List<Person> persons = new ArrayList<Person>();
            person = this.personService.findById(id);
            persons.add(person);
            if (persons.isEmpty()) {
                return new ResponseEntity<> (HttpStatus.NO_CONTENT);
            }
            return new ResponseEntity<> (persons, HttpStatus.OK);
        }
    }
}
```

PersonService.java

```
@Service
@Transactional
public class PersonService {

    @Autowired
    PersonRepository personRepository;

    public Person findById(int id) {
        Optional<Person> persons = this.personRepository.findById(id);
    }
}
```

Postman

The screenshot shows the Postman interface for a GET request to `http://localhost:8080/api/person?id=1`. The request is successful, returning a 200 OK status with a response time of 8 ms and a size of 359 B. The response body is displayed in JSON format, showing a person object with the following details:

```
{
  "id": 1,
  "name": "nal",
  "address": "Tokyo",
  "age": "30",
  "salary": 200000,
  "email": "nal@test.com",
  "approve": true
}
```


Insert data to “person” table by using @PostMapping annotation.

In PersonController.java, add createPerson() method.

```
@CrossOrigin(origins = "http://localhost:8081")
@RestController
public class PersonController {

    @Autowired
    PersonService personService;

    @GetMapping("/api/person")
    public ResponseEntity<List<Person>> getAllPersons(@RequestParam(required
= false) int id) {
        try {
            Person person = new Person();
            List<Person> persons = new ArrayList<Person>();
            person = this.personService.findById(id);
            persons.add(person);
            if (persons.isEmpty()) {
                return new ResponseEntity<>(HttpStatus.NO_CONTENT);
            }
            return new ResponseEntity<>(persons, HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @PostMapping("/api/person")
    public ResponseEntity<Person> createPerson(@RequestBody Person
createPerson) {
        try {
```

In PersonService.java, add addPerson() method.

```

@Service
@Transactional
public class PersonService {

    @Autowired
    PersonRepository personRepository;

    public Person findById(int id) {
        Optional<Person> persons = this.personRepository.findById(id);
        return persons.get();
    }

    public Person addPerson(Person p) {
        Person person = new Person();
        try {
            person = this.personRepository.save(p);
        } catch (Exception e) {
            TransactionInterceptor.currentTransactionStatus().setRollbackOnly();
        }
    }
}

```

Postman

POST http://localhost:8080/api/person Send

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```

1 {
2   "id": 3,
3   "name": "Takeda",
4   "address": "Saitama",
5   "age": "29",
6   "salary": 300000,
7   "email": "takeda@test.com",
8   "approve": true
9 }

```

Body Cookies (1) Headers (8) Test Results Status: 201 Created Time: 12 ms Size: 370 B Save as Example

Pretty Raw Preview Visualize JSON Copy Search

```

1 {
2   "id": 3,
3   "name": "Takeda",
4   "address": "Saitama",
5   "age": "29",
6   "salary": 300000,
7   "email": "takeda@test.com",
8   "approve": true
9 }

```

Check the database.

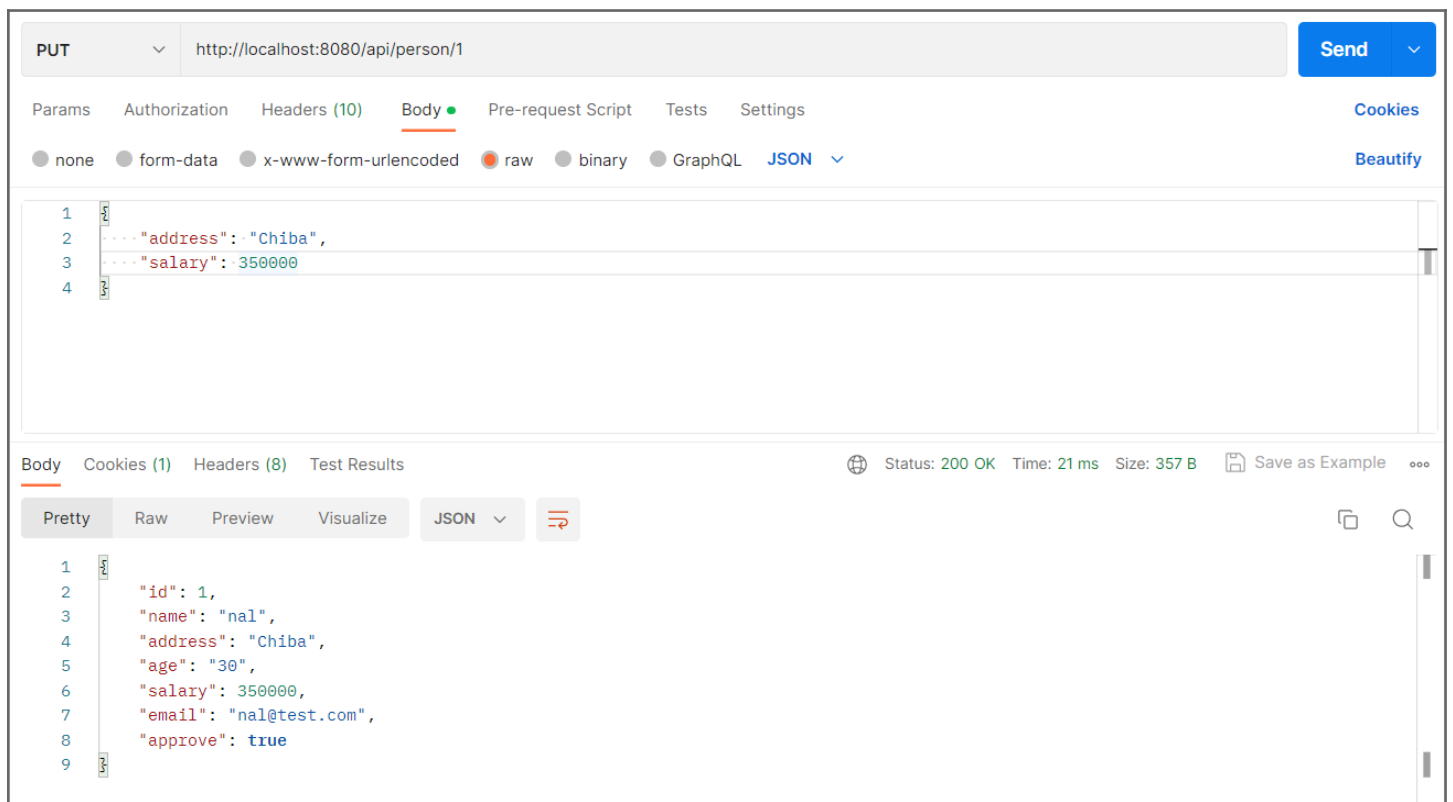
	id [PK] integer	name character varying (255)	address character varying (255)	age character varying (255)	salary integer	email character varying (255)	approve boolean
1	1	nal	Tokyo	30	200000	nal@test.com	true
2	2	suzuki	Tokyo	25	250000	suzuki@test.com	true
3	3	Takeda	Saitama	29	300000	takeda@test.com	true

Update data of person.id=1 by using @PostMapping annotation.

In PersonController.java, add createPerson() method.

```
@PutMapping("/api/person/{id}")
@ResponseBody
public ResponseEntity<Person>
updatePerson(@PathVariable("id") int id,
             @RequestBody Person updatePerson) {
    Optional<Person> personOpt =
this.personService.findById(id);
    if (personOpt.isPresent()) {
        Person person = personOpt.get();
        person.setAddress(updatePerson.getAddress());
    }
}
```

Postman



Check the database.

	id [PK] integer	name character varying (255)	address character varying (255)	age character varying (255)	salary integer	email character varying (255)	approve boolean
1	1	nal	Chiba	30	350000	nal@test.com	true
2	2	suzuki	Tokyo	25	250000	suzuki@test.com	true
3	3	Takeda	Saitama	29	300000	takeda@test.com	true

Delete data of person.id=1 by using @DeleteMapping annotation.

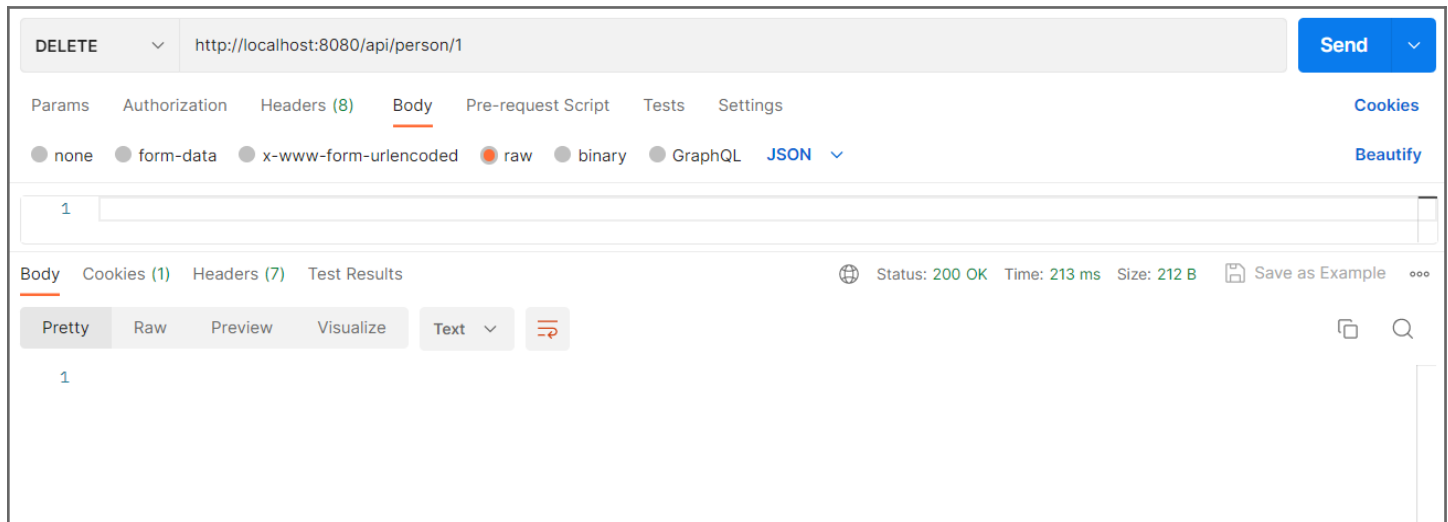
In PersonController.java, add createPerson() method.

```
@DeleteMapping("/api/person/{id}")
public ResponseEntity<HttpStatus>
deleteTutorial(@PathVariable("id") int id) {
    try {
        this.personService.deletePerson(id);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

PersonService.java

```
public void deletePerson(int id) {
    try {
        this.personRepository.deleteById(id);
    } catch (Exception e) {
    }
}
```

Postman



HTTP Request and Response Header

```

@GetMapping("/getBaseUrl")
public ResponseEntity<String> getBaseUrl(@RequestHeader
HttpHeaders headers) {
    InetAddress host = headers.getHost();
    String url = "http://" + host.getHostName() + ":" +
host.getPort();

    HttpHeaders responseHeaders = new HttpHeaders();

```

GET http://localhost:8080/getBaseUrl Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1

Body Cookies (1) Headers (9) Test Results Status: 200 OK Time: 205 ms Size: 303 B Save as Example

Pretty Raw Preview Visualize JSON

1 Base URL = http: [//localhost:8080](http://localhost:8080)

GET http://localhost:8080/getBaseUrl Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1

Body Cookies (1) **Headers (9)** Test Results Status: 200 OK Time: 205 ms Size: 303 B Save as Example

Key	Value
Vary	Origin
Vary	Access-Control-Request-Method
Vary	Access-Control-Request-Headers
Header-Key	Header-Value
Content-Type	application/json
Content-Length	32
Date	Thu, 13 Apr 2023 04:55:42 GMT
Keep-Alive	timeout=60
Connection	keep-alive

- 1) **@Configuration** : It is used as a source of bean definitions and config and indicates that the class has @Bean definition methods.
- 2) **@Bean** : a method-level annotation that is applied on a method to create beans that can be injected into each other and used within a class annotated with @Configuration.
- 3) **@Component** : a class-level annotation that is the same as @Bean but doesn't need to be used with the @Configuration annotation.
- 4) **@ComponentScan** : along with the @Configuration annotation to specify the packages that we want to be scanned.
- 5) **@EnableWebSecurity** : The WebSecurityConfig class is annotated with @EnableWebSecurity to enable Spring Security's web security support and provide the Spring MVC integration.
- 6) **@EnableAutoConfiguration** : to enable an auto-configuration feature in the Spring boot project.

Let's add security

pom.xml

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-security</artifactId>
```

WebSecurityConfig.java

```

@Configuration
@EnableWebSecurity
public class WebSecurityConfig {
    @Bean
    public InMemoryUserDetailsManager
userDetailsService() {
        UserDetails user1 = User.withUsername("user1")
            .password(passwordEncoder().encode("user1Pass
")) )
            .roles("USER")
            .build();
        UserDetails user2 = User.withUsername("user2")
            .password(passwordEncoder().encode("user2Pass
")) )
            .roles("USER")
            .build();
        UserDetails admin = User.withUsername("admin")
            .password(passwordEncoder().encode("adminPass
")) )
            .roles("ADMIN")

```

```

@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

```



ログイン
http://localhost:8080

ユーザー名

パスワード

ログイン キャンセル



ログイン
http://localhost:8080

ユーザー名

パスワード

ログイン キャンセル

Enter Your Name

Other Annotations

- 1) **@Scheduled** : This is a method-level annotation. If we want a method to execute periodically, we can annotate that method with @Scheduled.

```
@Scheduled(cron = "0 */3 * ? * *")
public void runEvery3Minutes() {
    Date d1 = new Date();
    System.out.println("Current datetime is :: " + d1);
}
```

```
Current datetime is :: Fri Apr 14 10:42:16 JST 2023
Current datetime is :: Fri Apr 14 10:45:00 JST 2023
Current datetime is :: Fri Apr 14 10:48:00 JST 2023
```

2.6. Spring Mybatis

Mybatis is an open source framework for implementing SQL database access in Java applications.

2.6.1 Installation

Add mybatis dependency to pom.xml

```
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>3.0.1</version>
```

2.6.2 SQL statement

Write the SQL statement in the xml file.

All of the SQL statements are placed in the <mapper> tag and map repository interface with namespace.

PersonRepository.java

```
@Mapper
public interface PersonRepository {

    List<Person> findAll();

    void save(Person person);

    Optional<Person> findById(int id);

    void updatePerson(Person person);
}
```

PersonRepository.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper
3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="repository file directory">
```

2.6.2.1 Select

Use <select> tag to write a SELECT query.

<select>

id : repository's method name

resultMap : mapping sql results and java class

<resultMap>

id : id maps with <select>'s resultMap

type : java class directory to map with sql results

※ If the db column name and java class's property is same, no need to write column.

```
<resultMap id = "result" type =  
    "helloworld.co.jp.hello.entity.Person">  
    <result property = "id" column = "id"/>  
    <result property = "name" column = "name"/>  
    <result property = "address" column = "address"/>  
    <result property = "age" column = "age"/>  
    <result property = "salary" column = "salary"/>  
    <result property = "approve" column = "approve"/>  
    <result property = "email" column = "email"/>  
</resultMap>  
  
<!-- Get All Person List -->  
<select id = "findAll" resultMap = "result">  
    SELECT * FROM person;
```

2.6.2.1 Insert

```
<!-- Add Person -->  
<insert id="save"  
parameterType="helloworld.co.jp.hello.entity.Person">  
    INSERT INTO person (name, address, age, salary,
```

2.6.2.1 Update

```
<!-- Update Person -->  
<update id="updatePerson"  
parameterType="helloworld.co.jp.hello.entity.Person">  
    UPDATE person  
    SET name = #{name}, address = #{address}, age =
```

2.6.2.1 Delete

```
<!-- Delete Person by ID -->  
<delete id="deleteById" parameterType="int">  
    DELETE FROM person WHERE id = #{id}  
</delete>
```