springboot-microservices-2024 [v3.4.0]

1. Return response with link to newly created resource

Project ref: *a2-sboot-ms-social-media-app*

- **Maven / External dependency**
  - Below required resources are available in Spring web dependency.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-bo   ot-starter-web</artifactId>
</dependency>
```

- **Code changes**
  - imports
    - *import org.springframework.web.servlet.support.ServletUriComponentsBuilder*
  - Build URL to new Resource using current request.
    - *ServletUriComponentsBuilder.fromCurrentRequest().path("{id}").buildAndExpand(savedUser.getId()) .toUri();*
  - Wrap new URL and response object in *ResponseEntity* and return the *ResponseEntity* object.
    - *return ResponseEntity.created(location).body(savedUser);*
  - Controller method

```
@PostMapping("/users")
public ResponseEntity<User> createUser(@RequestBody User user) {

    logger.debug("User to save : {}", user);
    User savedUser = userDaoService.save(user);

    URI location = ServletUriComponentsBuilder.fromCurrentRequest().
        .buildAndExpand(savedUser.getId()).toUri();

    return ResponseEntity.created(location).body(savedUser);
}
```

2. Property, method param or Return type validation

Project ref: *a3-sboot-ms-validation*

- **Maven / External dependency**
  - Add spring validation dependency.

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

- **Code changes**
  - imports
    - `import jakarta.validation.Valid;`
  - Annotate the method parameter for validation.

```
@PostMapping("/users")
public ResponseEntity<User> createUser(@Valid @RequestBody User user

    logger.debug("User to save : {}", user);

    var savedUser = userDaoService.save(user);

    var location = ServletUriComponentsBuilder.fromCurrentRequest().
            .toUri();
    return ResponseEntity.created(location).body(savedUser);
}
```

  - imports
    - `import jakarta.validation.constraints.Past;`
    - `import jakarta.validation.constraints.Size;`
  - Add validation in the properties of the bean.

```
public class User {

    private Integer id;

    @Size(min = 3, max = 20, message = "Name must be more than 2 cha
    private String name;

    @Past(message = "Birth date should be in past.")
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-
    private LocalDate birthDate;

    // constructors, setter-getters and other methods.
}
```

- **Notes:**
  - Spring internally usages `jakarta-validation` API.
  - Check `jakarta.validation.constraints.*` for more validation classes.
    - `@Valid` annotation:
      - Marks a property, method parameter or method return type for validation cascading.
      - Constraints defined on the object and its properties are validated when the property, method parameter or method return type is validated.
    - `@Size` annotation
      - Validates property value to match defined size constraints.
    - `@Past` annotation
      - Validates date value for must be a past date.

---

3. API documentation using openAPI, swagger-ui

Project ref: *a4-sboot-ms-springdoc-swagger-openapi*

- **Maven / External dependency**
  - Add spring validation dependency.

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.7.0</version>
</dependency>
```

- **Code changes**
  - None.
- **Swagger URL:**
  - http://localhost:8080/swagger-ui
  - http://localhost:8080/swagger-ui/index.html
- **Notes:**
  - `No code change required to enable swagger documentation.`
  - It's enabled by default if the ependency is present in `POM.xml`
- **References:**
  - `https://github.com/springdoc/springdoc-openapi/blob/main/springdoc-openapi-starter-webmvc-ui`
  - `https://springdoc.org/#getting-started`

---

4. Content negotiation for Response parsing

Project ref: *a5-sboot-ms-content-negotiation*

- **Maven / External dependency**
  - Add following dependency in POM.xml

```
<!-- XML conversion -->
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

- **Code changes**
  - None
- **Notes:**
  - If client requests for `Accept: application/xml` header.
  - Spring willl internally check `jackson-dataformat-xml` API dependency, if found bean will be transformed to xml.

---

5. Internationalization (i18n)

Project ref: *a6-sboot-ms-content-i18n*

- **Maven / External dependency**
  - Required API is available as part of spring-context dependency.
  - This is imported with spring web dependency internally.

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

- **Code changes**
  - imports
    - `import org.springframework.context.MessageSource;`
  - Autowire the MessageSource in required class.

```
@RestController
public class HelloWorldI18n {

    private MessageSource messageSource;

    // autowire (constructor injection) messageSource.
    public HelloWorldI18n(MessageSource messageSource) {
        super();
        this.messageSource = messageSource;
    }

    @GetMapping("/say-hello-i18n")
    public String sayHello() {

        var locale = LocaleContextHolder.getLocale();

        return this.messageSource.getMessage("good.morning", null, "
    }
}
```

  - Create property files for different locale
    - File: `messages[-<locale>].properties`
    - Here's some examples:
      - Default: `messages.properties`
      - Spanish: `messages_es.properties`
      - German: `messages_ger.properties`

- **Notes:**
  - Default message files name prefix is `messages` & suffix is `.properties`.
  - Spring reads the value of `Accept-Language` Header from `HTTP Request`and replaces it with `<locale>` when locating `messsages[-<locale>].properties` file.

---

6. Microservice API Versioning

Project ref: *a7-sboot-ms-api-versioning*

- **Maven / External dependency**
  - No dependency required.
  - API versioning is HTTP architectural style.
    - None
- **Common Code**
  - Person bean v1

```
public class PersonV1 {

    private String name;

    public PersonV1(String name) {
        super();
        this.name = name;
    }
    // getter-setters
}
```

  - Person bean v2

```
public class PersonV2 {

    private Name name;

    public PersonV2(Name name) {
        super();
        this.name = name;
    }
    // getter-setters
}
```

  - Name bean

```
public class Name {

    private String firstName;
    private String lastName;

    public Name(String firstName, String lastName) {
        super();
        this.firstName = firstName;
        this.lastName = lastName;
    }
    // getters-setters
}
```

- **URI Versioning**
  - In this versioning style, a version number is appended in URL to create new URL version.
  - **Twitter** also follows same versoning strrateegy.
    - **Ref:** https://developer.x.com/en/docs/x-ads-api/versioning
  - **Drawback**
    - Polluting URL
  - **Controller Code changes**
    - imports
      - `import org.springframework.web.bind.annotation.GetMapping;`
    - Here's two versions of API defined.

```java
@RestController
public class UriVersioningPersonController {

    /**
    * Version 1
    * @return
    */
    @GetMapping("/v1/person")
    public PersonV1 getPersonV1() {
        return new PersonV1("URI Versioning v1");
    }

    /**
    * Version 2
    * @return
    */
    @GetMapping("/v2/person")
    public PersonV2 getPersonV2() {
        return new PersonV2(new Name("URI", "Versioning V2"));
    }
}
```

- **Request Param Versioning**
  - In this versioning style, a request param is sent with API version number.
  - Base URL remains unchanged.
  - **Amazon** also follows same versoning strrateegy.
    - **Ref:** https://
  - **Drawback**
    - Polluting URL
  - **Controller Code changes**
    - imports
      - `import org.springframework.web.bind.annotation.GetMapping;`
    - Here's two versions of API defined.

```java
@RestController
public class RequestParamVersioningController {

    /**
    * Version 1
    * @return
    */
    @GetMapping(path = "/person/param", params = "version=1")
    public PersonV1 getPersonV1() {
        return new PersonV1("Request Param versioning v1");
    }

    /**
    * Version 2
    * @return
    */
    @GetMapping(path = "/person/param", params = "version=2")
    public PersonV2 getPersonV2() {
        return new PersonV2(new Name("Request Parama", "Versio
    }
}
```

- **Custom Header Versioning**
  - In this versioning style, a custom HTTP header is sent with API version number.
  - Base URL remains unchanged.
  - **Microsoft** also follows same versoning strrateegy.
    - **Ref:** https://
  - **Drawback**
    - Misuse of HTTP Headers
  - **Controller Code changes**
    - imports
      - `import org.springframework.web.bind.annotation.GetMapping;`
    - Here's two versions of API defined.

```java
@RestController
public class CustomHeaderVersioning {

    /**
    * Version 1
    * @return
    */
    @GetMapping(path = "/person/header", headers = "X-API-VERS
    public PersonV1 getPersonV1() {
        return new PersonV1("Custom Header Versioning v1");
    }

    /**
    * Version 2
    * @return
    */
    @GetMapping(path = "/person/header", headers = "X-API-VERS
    public PersonV2 getPersonV2() {
        return new PersonV2(new Name("Custom Header", "Version
    }
}
```

- **Content Negotiation (Media type) Versioning**
  - a.k.a `Content negotiation` or `Accept header` versioning.
  - In this versioning style, a custome media type is sent in `Accept` HTTP header.
  - Which API matches the header value request is forwarded to it.
    - e.g. `Accept: application/vnd.comp.app-v2+json`
  - Base URL remains unchanged.
  - **Github** also follows same versoning strrateegy.
    - **Ref:** https://
  - **Drawback**
    - Misuse of HTTP Headers
  - **Controller Code changes**
    - imports
      - `import org.springframework.web.bind.annotation.GetMapping;`
    - Here's two versions of API defined.

```
@RestController
public class MediaTypeVersioning {

    /**
    * Version 1
    * @return
    */
    @GetMapping(path = "/person/accept", produces = "applicati
    public PersonV1 getPersonV1() {
        return new PersonV1("Mediatype Versioning v1.");
    }

    /**
    * Version 2
    * @return
    */
    @GetMapping(path = "/person/accept", produces = "applicati
    public PersonV2 getPersonV2() {
        return new PersonV2(new Name("Media type", "Versioning
    }
}
```

---

a8-sboot-ms-hateoas

- Maven dependency

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

- Spring resources

```
import org.springframework.hateoas.EntityModel;
import org.springframework.hateoas.server.mvc.WebMvcLinkBuilder;
```

- API Refactoring

```
/**
 * Retrieve the users.
 *
 * @return
 */
@GetMapping(path = "/users/{id}", produces = {"application/json", "applicati
public EntityModel<User> retrieveUser(@PathVariable Integer id) {
    User user = userDaoService.findById(id);

    if (user == null) {
        throw new UserNotFoundException(String.format("No user exists with i
    }

    // Create link to method
    var link = WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.methodOn(this.get(

    // EntityModel object supports Model and allows to add links
    final EntityModel<User> entityModel = EntityModel.of(user);
    entityModel.add(link.withRel("all-users"));

    return entityModel;
}
```

a9-sboot-ms-static-filtering

Static filtering

- Resource

```
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
```

- Static filtering code

```
/**
 * Do not send field2, field4, field6
 */
@JsonIgnoreProperties(value = {"field2", "field4"})
public class SomeBean {

    private String field1;
    private String field2;
    private String field3;
    private String field4;
    private String field5;

    //Ignore in json response
    @JsonIgnore
    private String field6;

    // constructor

    // getters

    // toString()
}
```

Dynamic filtering

- Resource

```
import com.fasterxml.jackson.databind.ser.FilterProvider;
import com.fasterxml.jackson.databind.ser.impl.SimpleBeanPropertyFilter;
import com.fasterxml.jackson.databind.ser.impl.SimpleFilterProvider;
```

- Dynamic Filtering code

```
@RestController
public class DynamicFilteringController {

    @GetMapping("/dyna-filtering")
    public SomeBeanDynamicFilter filtering() {
        SomeBeanDynamicFilter SomeBeanDynamicFilter = new SomeBeanDynamicFil
                "Value-4", "Value-5", "Value-6");

        // Dynamic filtering
        final SimpleBeanPropertyFilter simpleBeanPropertyFilter = SimpleBean
                "field4", "field6");

        final SimpleFilterProvider simpleFilterProvider = new SimpleFilterPr
                simpleBeanPropertyFilter);

        final MappingJacksonValue mappingJacksonValue = new MappingJacksonVa
        mappingJacksonValue.setFilters(simpleFilterProvider);

        return SomeBeanDynamicFilter;
    }

    @GetMapping("/dyna-filtering-list")
    public MappingJacksonValue filteringList() {
        List<SomeBeanDynamicFilter> SomeBeanDynamicFilterList = Arrays.asLis
                new SomeBeanDynamicFilter("Value-1", "Value-2", "Value-3", "
                new SomeBeanDynamicFilter("Value-11", "Value-22", "Value-33"
                new SomeBeanDynamicFilter("Value-111", "Value-222", "Value-3
                        "Value-666"));

        // Dynamic filtering
        SimpleBeanPropertyFilter simpleBeanPropertyFilter = SimpleBeanProper
                "field3", "field5", "field6");
        FilterProvider simpleFilterProvider = new SimpleFilterProvider().add
                simpleBeanPropertyFilter);

        final MappingJacksonValue mappingJacksonValue = new MappingJacksonVa
        mappingJacksonValue.setFilters(simpleFilterProvider);

        return mappingJacksonValue;
    }
```

- Dynamic filterig bean

-Resource

```
import com.fasterxml.jackson.annotation.JsonFilter;
```

- Bean class

```
/**
 * Dynamically exclude properties as per the specified filter.
 */
@JsonFilter("SomeBeanDynamicFilter")
public class SomeBeanDynamicFilter {

    private String field1;
    private String field2;
    private String field3;
    private String field4;
    private String field5;
    private String field6;

    // constructor

    // getters

    // toString()
}
```

a11-sboot-ms-hal-explorer

- Dependency

```
<!-- Spring boot HAL explorer -->
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-rest-hal-explorer</artifactId>
</dependency>
```

- Default URL

```
- http://localhost:8080/explorer
- http://localhost:8080/explorer/index.html#
```

##a13-sboot-ms-mysql-jpa

- Launch MySQL as Docker container

```
docker run --detach --env MYSQL_ROOT_PASSWORD=dummypassword --env MYSQL_USEI
```

- mysqlsh commands

```
mysqlsh
\connect social-media-user@localhost:3306
\sql
use social-media-database
select * from user_details;
select * from post;
\quit
```

- /pom.xml Modified

```
<!-- Use this for Spring Boot 3.1 and higher -->
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
</dependency>

<!-- Use this if you are using Spring Boot 3.0 or lower
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>
-->
```

- /src/main/resources/application.properties

```
#spring.datasource.url=jdbc:h2:mem:testdb
spring.jpa.show-sql=true
spring.datasource.url=jdbc:mysql://localhost:3306/social-media-database
spring.datasource.username=social-media-user
spring.datasource.password=dummypassword
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

##a14-sboot-sc-basic-authentication

- Dependency

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

- Note: If facing any issue while starting the application, try following - Stop the server. - Update maven project (Alt + f5). - Start the server.
- Default user is 'user'.
- Get auto generated password from log.
    - Search in logs for "Using generated security password: " text to get the auto generated password.
- Configuring user and password in application properties

```
spring.security.user.name=vivek
spring.security.user.password=welcome
```

- Customizing default authentication
    - Create a Configuration class to override default authetication

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.web.builders.HttpSecur
import org.springframework.security.web.SecurityFilterChain;

@Configuration
public class SpringSecurityConfiguration {

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throv

        /*
         * All requests must be authorized.
         *
         * Else return HTTP 403, it doesn't prompt for user creds.
         */
        httpSecurity.authorizeHttpRequests(
                authorizationManagerRequestMatcherRegistryCustomizer -> auth
                        .anyRequest().authenticated());

        /*
         * Prompt for authentication if request is not authorized.
         *
         * Using default customizer
         */
        httpSecurity.httpBasic(Customizer.withDefaults());

        /*
         * Disabling CSRF as it may cause issue with HTTP methods - POST & P
         *
         * if enabled, Keep prompting for user credentials for post request.
         */
        httpSecurity.csrf(csrf -> csrf.disable());

        return httpSecurity.build();
    }
}
```

- X