Western Norway
University of
Applied Sciences

EXAM

Exam code: DAT152

Course name: Advanced Web Applications

Date: November 28, 2022

Type of examination: Written exam

Time: 4 hours (0900-1300)

Number of questions: 6

Number of pages: 19 (including this page and appendices):

Appendices: The last 7 pages

Exams aids: Bilingual dictionary

Academic coordinator: Bjarte Kileng (909 97 348), Atle Geitung (482 42 851), Tosin Daniel Oyetoyan (405 70 403), Lasse Jenssen (922 33 948)

Notes: None

Question 1 – Globalization (15% ~ 36minutes)

a) Explain, in your own words, the terms globalization (g11n), internationalization (i18n) and localization (l10n). Also explain the relationship between the concepts.

Given the following jsp:

```
<body>
        November 15, 2022, the human population passed
        8,000,000,000 humans.<br>
        The population is expected to grow by 30% until
        it peaks in the 2080s.<br>
</body>
```

b) Internationalize the jsp and localize it to English and one other language. You must add and replace code to the jsp. Also, you need to write the properties files for English and the other language.

Question 2 – Custom tags (10% ~ 24minutes)

We want to have a tag called "formattedText". The purpose is to create a formatted text like this:

Before (without the tag):

```
<p><b>Type: Written digital school exam</b></p>
```

Using the tag:

```
<p><dat152:formattedText name="Type">
  Written digital school exam
</dat152:formattedText></p>
```

Both should give the following result on the web page:

Type: Written digital school exam

a) Implement the "formattedText" tag using SimpleTagSupport in Java. You do not need to write xml-code for the tld-file. You do not need import-sentences.

You will need to override and implement this:

```
@Override
public final void doTag() throws JspException, IOException {...}
```

b) Implement the "formattedText" tag using a tag-file.

A tag-file starts with this line:

```
<%@ tag language="java" pageEncoding="UTF-8"%>
```

c) Add the attribute “type” to the “formattedText” tag. The “type” attribute can be either “normal”, “bold” or “italic” and tells how to show the name. Default should be “bold”. Examples (jsp and result on web page):

```
<p><dat152:formattedText name="Type">
    Written digital school exam
</dat152:formattedText></p>
<p><dat152:formattedText type="bold" name="Type">
    Written digital school exam
</dat152:formattedText></p>
<p><dat152:formattedText type="italic" name="Type">
    Written digital school exam
</dat152:formattedText></p>
<p><dat152:formattedText type="normal" name="Type">
    Written digital school exam
</dat152:formattedText></p>
```

Result on the web page:

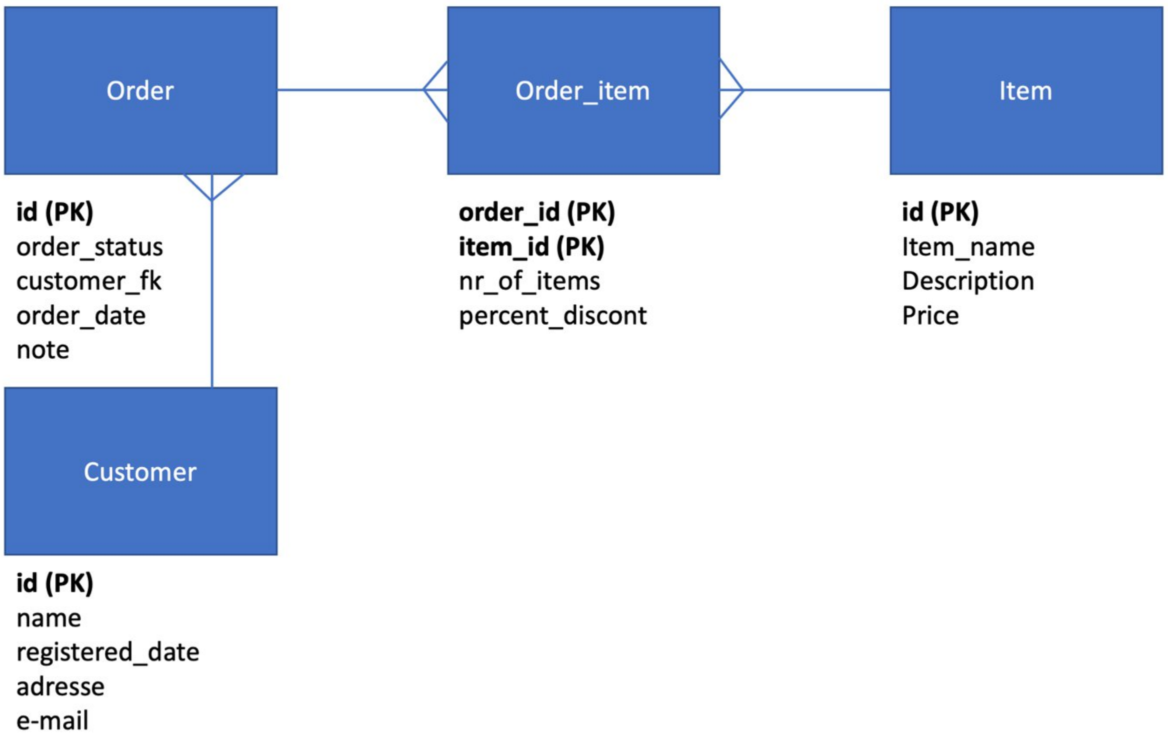
Type: Written digital school exam

Type: Written digital school exam

Type: Written digital school exam

Type: Written digital school exam

Question 3 – Web APIs and Framework (20% ~ 48minutes)



a) The data model above is designed for a startup web store.
The web applications to be built expects two types of users (Actors):

- Customers: Registering (creating a customer account), Ordering items
- Company employees: Handling customers, items and orders

Task: Create the URI path design for the needed REST web services for both types of users (actors). The URI path design should include:

- **HttpMethod**
- **URI (from root, shown in example above)**
- **A short description of the REST web service**

Notes!

- **You do NOT need to add filters (except for the path variables such as {id} shown in the example below).**
- **The URI path design should follow level 2 in the Richardson Maturity model.**

Examples:

Method	URI	Description
GET	/items	List all item in the store
GET	/items/{id}	Get an item in the store (given by the item id)

b) A skeleton for a REST controller class for handling “Customers” is already started on (see below). The project is using the Spring Framework with both Spring Boot and Spring Web MVC.

```
@RestController
@RequestMapping(CustomerController.BASE_URL)
public class CustomerController {

  final static String BASE_URL = "/customers";

  @Autowired
  private CustomerRepositoryImpl repository;

  <you need to write the controller methods below>
}
```

The controller class is using a “CustomerRepositoryImpl” class for accessing data. The repository class implements the following interface:

```
public interface CustomerRepository {
  List<Customer> findAll();
  Optional<Customer> findById(Long id);
  Customer save(Customer customer);
  void delete(Customer customer);
  List<Order> findCustomerOrdersByStatus(
                      Long customerId, String status);
}
```

Task: Write the controller methods for the functionalities described in the list below. The methods should return a “ResponseEntity” and include proper HTTP statuses.

1) **Create a new customer.**
   Statuses: Status OK (200) or Bad Request (400).
2) **Get a customer given by the customer id (primary key).**
   Statuses: OK (200) or Not Found (404).
3) **Get all orders with a given status for a given customer.**
   Statuses: OK (200) or Not Found (404).

Question 4 – Universal Design (10% ~ 24 minutes)

a) Who is WCAG 2.1 (Web Content Accessibility Guidelines) meant for and why?

b) What does WCAG 2.1 cover?

c) One of the five principles in WCAG 2.1 is “Operable” (mulig å betjene). Discuss briefly how to meet this principle. Also, give and explain an examples of failure to adhere to this principle.

Question 5 – Web security (25% ~ 60 minutes)

a) **Part 1** - Multiple choice (Some questions may contain multiple correct answers. In this case, choose all the correct options)

1) Can it be a serious security risk to rely only on client-side validation of critical functions?
    1. No, because we can make the validation strong enough and enable all security policies
    2. Yes, because it is possible to inject untrusted data from the client-side
    3. No, because the client can validate the response from the server
    4. Yes, because anything coming from the client should not be trusted

2) A user has requested to reset his/her password and received the password in plaintext by email. In what possible way was the user's password stored in the database?
    1. stored in plaintext
    2. stored using a two-way encryption
    3. stored using a one-way hash function

3) In a single sign on (SSO) solution:
    1. An identity provider is responsible for issuing and managing authentication tokens
    2. A service provider is responsible for issuing and managing authentication tokens
    3. Service providers must use the certificate from the identity provider to verify authentication tokens
    4. A service provider must issue an authentication token to the identity provider which is verified by the identity provider

4) A web service uses a combination of access and refresh tokens. The access token has expired. Is it possible to use any refresh token to request for a new access token?
    1. Yes, if there is an association between the access and refresh tokens
    2. No, if there is an association between the access and refresh tokens
    3. No, if there is no association between the access and refresh tokens
    4. Yes, if there is no association between the access and refresh tokens

5) A Json Web Token (JWT) contains a header, payload and signature. Which part of the token is usually signed?
    1. The header and signature
    2. The payload
    3. The header, payload, and signature
    4. The header and payload

Part 2

b) Consider the code snippet below. The sql query is vulnerable to SQL injection. The ***asckey*** corresponds to either ‘shoppingitem’ or ‘datetime’ column in the ShoppingCart table. Write a mitigation for this code.

```
public List<Cart> doSearch(String userid, String asckey) throws SQLException, NoSuchAlgorithmException {

   List<Cart> items = new ArrayList<Cart>();

      try {

         String sqlQuery = "select * from ShoppingCart where userid='"+userid+"' ORDER BY “+asckey+” ASC";

     Connection conn = Db.getConnection();
     Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(sqlQuery);

         if (!rs.next()) {
                    throw new SecurityException("UserId incorrect");
         } else {
      // proceed and store result in items
         }

      } finally {
         try {
           conn.close();
         } catch (SQLException x) {
            //
         }
      }
   return items;
 }
```

c) Inspect the two code snippets (snippet1 and snippet2) below.

1. Are they vulnerable to XSS?
2. If no, justify your answer.
3. If yes, justify your answer and what is/are the type(s) of XSS?

```
private void snippet1(HttpServletRequest request, HttpServletResponse response) throws Throwable {
      String data = “”;

      Cookie cookies = request.getCookies();

      if (cookies != null)
         data = cookies[0].getValue();

      if (data != null)
      {
         response.getWriter().println("<br> data = " + data);
      }

}
```

```
private void snippet2(HttpServletRequest request, HttpServletResponse response) throws Throwable {
    String data;

    data = request.getParameter("name");

    if (data != null)
    {
      response.getWriter().println("<br> data = " + data.replaceAll("(<script>)", ""));
    }

}
```

d) Describe the different approaches to store passwords.

1. Among the approaches you have discussed, which approach is the best to protect against various attacks and why?
2. What are the challenges with the other approaches?
3. Describe briefly the attack techniques against password?

e) In the web security obligatory assignment (Oblig3), the service provider (DAT152BlogApp) recieves an authentication token (id_token) from the identity provider (DAT152WebSearch). The id_token is a Json web token (see example below) with some claims and it is used to grant access to the blog website and determine the role (privilege) of the user.

HEADER:
```
{
  "alg": "RS256"
}
```
PAYLOAD:
```
{
  "iss": "http://localhost:9092/DAT152WebSearch",
  "sub": "47544B959729E79BDCA8766A87A8971C",
  "aud": "http://localhost:9091/blogapp/callback",
  "iat": 1632243868,
  "exp": 1633243868,
  "role": "user"
}
```
SIGNATURE:

Answer the following questions:

1. Explain clearly how to manipulate the id_token such that it can be exploited to elevate the privilege of the user.
2. If the attack in 1. would succeed, what vulnerability will be present in the service provider endpoint?

f) Describe briefly how the 'trust' relationship between an identity provider and a service provider is established.

Question 6 – JavaScript (20% ~ 48 minutes)

a) A web application includes a file *util.js* with the following code:

```
export default {
    // Returns a number, length in feet
    // Input parameter is a number, length in metres
    metrestofeet(metres) { /* JavaScript code */ },

    // Returns a number, length in metres
    // Input parameter is a number, length in feet
    feettometres(feet) { /* JavaScript code */ },

    // Returns a string, the name of unit in languge
    // Input parameters are strings
    // Parameter unit is a unit name in “en-US”
    unitInLanguage(unit, language) { /* JavaScript code */ }
}
```

Observe, you are not asked to implement any of the the code of *util.j*s.

i. What are the consequences of the *export default* statement at the beginning of the file?

ii. Demonstrate how to import *utils.js* from another JavaScript module, and how to call the method *metrestofeet* of *util.j*s from that module. You can choose the file folder structure as you wish.

b) An HTML custom tag **LENGTH-READER** adds a GUI component to the web application that lets the user input a length value.

The custom tag is based on an HTML **TEMPLATE** element that is determined by the current browser language settings. The HTML code below shows the **TEMPLATE** elements to be used for the languages “en-GB” and “nb-NO”:

```
<template data-name="length-reader" data-language="en-GB">
  <fieldset>
    <legend>Length in <span data-units></span></legend>
    <input type="number" value="" min="0" placeholder="Enter length"/>
  </fieldset>
</template>

<template data-name="length-reader" data-language="nb-NO">
  <fieldset>
    <legend>Lengde i <span data-units></span> </legend>
    <input type="number" value="" min="0" placeholder="Angi lengde"/>
  </fieldset>
</template>
```

If no **TEMPLATE** element corresponds to the current browser language setting, the application should default to the first **TEMPLATE** element with attribute *data-name* set to “length-reader”. If the browser does not report a language, the language should default to “en-GB”.

The HTML code below demonstrates how to use the custom tag **LENGTH-READER**:

```
<length-reader data-units="foot"></length-reader>
```

For language “en-GB”, the HTML code above should produce the view as is shown in Figure 1.

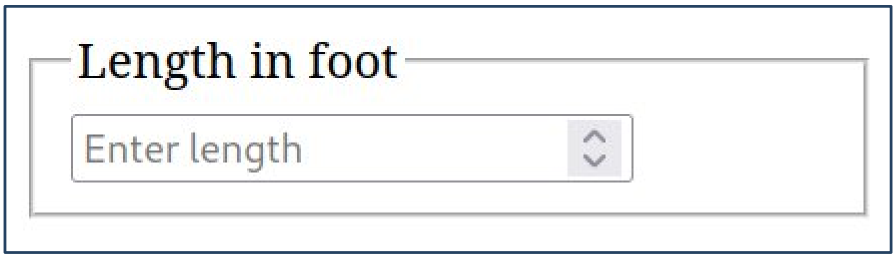


Figure 1: Length-reader in language “en-GB”

For language “nb-NO” the HTML code above with **LENGTH-READER** should produce the view as is shown in Figure 2.

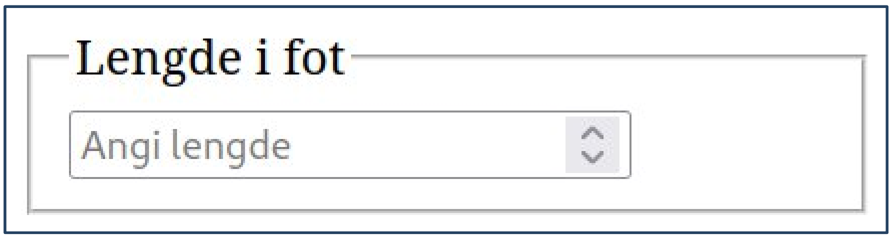


Figure 2: Length-reader in language “nb-NO”

The HTML element class of **LENGTH-READER** has two public methods only:

- *setValue(length)*: The method will set *length* as the value of the HTML **INPUT** element, i.e. display the value.

  Parameters:

  - In parameter: Number
  - Return value: None

- *addCallback(callback)*: The method will add a callback that is run at event *input* on the HTML **INPUT** element.

  Parameters:

  - In parameter: Function
  - Return value: Not required

  The method can return a value to identify the callback, but that is not required.

  When *callback* is run at event *input*, it must be run with the current value of the HTML **INPUT** element as parameter.

i. What is shadow DOM, what are the consequences of using shadow DOM and what differentiates the *open* and *closed* modes of shadow DOM?

ii. Use shadow DOM in closed mode to implement a JavaScript HTML element for the custom tag **LENGTH-READER** as described above.

You can assume a method *unitInLanguage* of the JavaScript module *util.js*. This method lets you convert a unit name from language “en-US” to any language, e.g. “foot” to “fot” if “nb-NO”, “pied” if “fr-FR” or “πόδι” if “el-GR”.

Tip: See the appendix.

c) A GUI component has functionality to convert between lengths specified in metres and lengths specified in feet. The GUI component can be added to the web application with the HTML custom tag **LENGTH-CONVERTER**.

The custom tag **LENGTH-CONVERTER** is based on the HTML **TEMPLATE** element that is shown below:

```
<template data-name="converter">
    <length-reader data-units="meter"></length-reader>
    <length-reader data-units="foot"></length-reader>
</template>
```

The HTML code below demonstrates how to use the custom tag **LENGTH-CONVERTER**:

```
<length-converter></length-converter>
```

With the browser language configured as “en-GB”, and after user input for a length in feet, the HTML code above should produce the view as is shown in Figure 3.

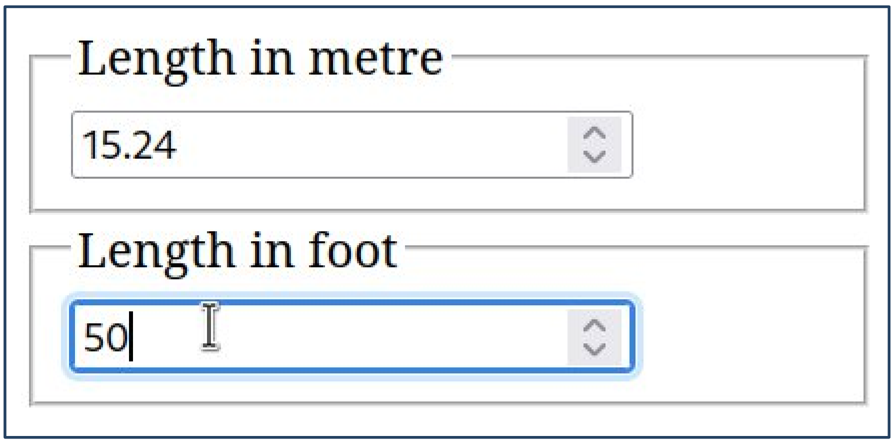


Figure 3: Metric-imperial length converter in language "en-GB" after user input for length in foot

Observe: On user input in one of the **LENGTH-READER** elements, the corresponding value should immediately be calculated and displayed in the other **LENGTH-READER** element. That is, if the user inputs a length i metres, the length in feet should immediately be shown. Similarly, an input in feet should immediately show the corresponding length in metres.

You can assume the methods *metrestofeet* and *feettometre* of the JavaScript module *util.js*. These methods let you convert between feet and metres.

i. Use shadow DOM in closed mode to implement a JavaScript HTML element for the custom tag **LENGTH-CONVERTER** as described above.

ii. The HTML file of the converter is named *index.html*, and all JavaScript code of the application is included as JavaScript modules, using *import* and *export* statements.

- What JavaScript files would should use for the application, and in what files would you put the *import* and *export* statements? Remember also to import the file *utils.js*.

- Show the HTML **SCRIPT** tag(s) with all attributes and values that are necessary to include the JavaScript code of the application. Where in the file *index.html* would you put the HTML **SCRIPT** tag(s)? You must explain your answer.

- Demonstrate how to use the *customElements.define* method to create the custom tags **LENGTH-READER** and **LENGTH-CONVERTER**. In what files would you put the call to *customElements.define*? You must explain your answer.

Tip: See the appendix.

Appendix

Help for question 1 (JSTL fmt)

Tag Summary	
requestEncoding	Sets the request character encoding
setLocale	Stores the given locale in the locale configuration variable
timeZone	Specifies the time zone for any time formatting or parsing actions nested in its body
setTimeZone	Stores the given time zone in the time zone configuration variable
bundle	Loads a resource bundle to be used by its tag body
setBundle	Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable
message	Maps key to localized message and performs parametric replacement
param	Supplies an argument for parametric replacement to a containing <message> tag
formatNumber	Formats a numeric value as a number, currency, or percentage
parseNumber	Parses the string representation of a number, currency, or percentage
formatDate	Formats a date and/or time using the supplied styles and pattern
parseDate	Parses the string representation of a date and/or time

Help for question 3

Package org.springframework.http

Class ResponseEntity<T>

java.lang.Object

org.springframework.http.HttpEntity<T>

org.springframework.http.ResponseEntity<T>

Type Parameters:

T - the body type

```
public class ResponseEntity<T> extends HttpEntity<T>
```

Extension of `HttpEntity` that adds an `HttpStatusCode` status code. Used in `RestTemplate` as well as in `@Controller` methods.

…

Constructor Details

ResponseEntity

`public ResponseEntity(HttpStatusCode status)`

Create a `ResponseEntity` with a status code only.

Parameters: `status` - the status code

ResponseEntity

`public ResponseEntity(@Nullable T body, HttpStatusCode status)`

Create a `ResponseEntity` with a body and status code.

Parameters: body - the entity body, `status` - the status code

ResponseEntity

`public ResponseEntity(MultiValueMap<String,String> headers, HttpStatusCode status)`

Create a `ResponseEntity` with headers and a status code.

Parameters:

`headers` - the entity headers
`status` - the status code

ResponseEntity

```
public ResponseEntity(@Nullable T body, @Nullable MultiValueMap<String,String> headers,
        HttpStatusCode status)
```

Create a `ResponseEntity` with a body, headers, and a status code.

Parameters:

body - the entity body
`headers` - the entity headers
`status` - the status code

ResponseEntity

```
public ResponseEntity(@Nullable T body, @Nullable MultiValueMap<String,String> headers,
        int rawStatus)
```

Create a `ResponseEntity` with a body, headers, and a raw status code.

Parameters:

body - the entity body
`headers` - the entity headers
`rawStatus` - the status code value

Since: 5.3.2

Method Details

getStatusCode

`public HttpStatusCode getStatusCode()`

Return the HTTP status code of the response.

Returns: the HTTP status as an HttpStatus enum entry

getStatusCodeValue

`@Deprecated(since="6.0") public int getStatusCodeValue()`

Deprecated.

as of 6.0, in favor of `getStatusCode()`

Return the HTTP status code of the response.

Returns: the HTTP status as an int value

Since: 4.3

equals

```
public boolean equals(@Nullable
       Object other)
```

Overrides: `equals` in class `HttpEntity<T>`

hashCode

`public int hashCode()`

Overrides: `hashCode` in class `HttpEntity<T>`

toString

`public String toString()`

Overrides: `toString` in class `HttpEntity<T>`

status

`public static ResponseEntity.BodyBuilder status(HttpStatusCode status)`

Create a builder with the given status.

Parameters:

`status` - the response status

Returns: the created builder

Since: 4.1

status

`public static ResponseEntity.BodyBuilder status(int status)`

Create a builder with the given status.

Parameters:

`status` - the response status

Returns: the created builder

Since: 4.1

ok

`public static ResponseEntity.BodyBuilder ok()`

Create a builder with the status set to OK.

Returns: the created builder

Since: 4.1

ok

`public static <T> ResponseEntity<T> ok(@Nullable T body)`

A shortcut for creating a `ResponseEntity` with the given body and the status set to OK.

Parameters:

body - the body of the response entity (possibly empty)

Returns: the created `ResponseEntity`

Since: 4.1

of

`public static <T> ResponseEntity<T> of(Optional<T> body)`

A shortcut for creating a `ResponseEntity` with the given body and the OK status, or an empty body and a NOT FOUND status in case of an Optional.empty() parameter.

Returns: the created `ResponseEntity`

Since: 5.1

of

`public static ResponseEntity.HeadersBuilder<?> of(ProblemDetail body)`

Create a new `ResponseEntity.HeadersBuilder` with its status set to `ProblemDetail.getStatus()` and its body is set to `ProblemDetail`.

Note: If there are no headers to add, there is usually no need to create a `ResponseEntity` since `ProblemDetail` is also supported as a return value from controller methods.

Parameters: body - the problem detail to use

Returns: the created builder

Since: 6.0

created

`public static ResponseEntity.BodyBuilder created(URI location)`

Create a new builder with a CREATED status and a location header set to the given URI.

Parameters: `location` - the location URI

Returns: the created builder

Since: 4.1

accepted

`public static ResponseEntity.BodyBuilder accepted()`

Create a builder with an ACCEPTED status.

Returns:the created builder

Since: 4.1

noContent

`public static ResponseEntity.HeadersBuilder<?> noContent()`

Create a builder with a NO_CONTENT status.

Returns: the created builder

Since: 4.1

badRequest

`public static ResponseEntity.BodyBuilder badRequest()`

Create a builder with a BAD_REQUEST status.

Returns: the created builder

Since: 4.1

notFound

`public static ResponseEntity.HeadersBuilder<?> notFound()`

Create a builder with a NOT_FOUND status.

Returns: the created builder

Since: 4.1

Interface ResponseEntity.BodyBuilder

All Superinterfaces:

`ResponseEntity.HeadersBuilder<ResponseEntity.BodyBuilder>`

Enclosing class:

`ResponseEntity<T>`

Method Details

contentType

`ResponseEntity.BodyBuilder contentType(MediaType contentType)`

Set the media type of the body, as specified by the `Content-Type` header.

Parameters: `contentType` - the content type

Returns: this builder

body

`<T> ResponseEntity<T> body(@Nullable T body)`

Set the body of the response entity and returns it.

Type Parameters: `T` - the type of the body

Parameters: body - the body of the response entity

Returns: the built response entity

Package org.springframework.http

Interface HttpStatusCode

Method Summary

Modifier and Type	Method	Description
`boolean`	`is1xxInformational()`	Whether this status code is in the Informational class (1xx).
`boolean`	`is2xxSuccessful()`	Whether this status code is in the Successful class (2xx).
`boolean`	`is3xxRedirection()`	Whether this status code is in the Redirection class (3xx).
`boolean`	`is4xxClientError()`	Whether this status code is in the Client Error class (4xx).
`boolean`	`is5xxServerError()`	Whether this status code is in the Server Error class (5xx).
`boolean`	`isError()`	Whether this status code is in the Client or Server Error class
`int`	`value()`	Return the integer value of this status code

Enum Class HttpStatus

java.lang.Object

java.lang.Enum<HttpStatus>

org.springframework.http.HttpStatus

All Implemented Interfaces:

Serializable, Comparable<HttpStatus>, Constable, HttpStatusCode

Enum Constants

Enum Constant	Description
ACCEPTED	202 Accepted.
BAD_REQUEST	400 Bad Request.
CREATED	201 Created.
FORBIDDEN	403 Forbidden.
FOUND	302 Found.
INTERNAL_SERVER_ERROR	500 Internal Server Error.
NO_CONTENT	204 No Content.
NOT_FOUND	404 Not Found.
OK	200 OK.
UNAUTHORIZED	401 Unauthorized.

Help for question 6

The current language, e.g. “nb-NO” is available in the browser as *navigator.language*.

The code below demonstrates how to use an HTML **TEMPLATE** element for a GUI component with shadow DOM.

```
class MyElement extends HTMLElement {
    #shadow;

    constructor() {
        super();
        this.#shadow = this.attachShadow({ mode: 'closed' });
        const template = document.querySelector("template");
        const content = template.content.cloneNode(true);
        this.#shadow.appendChild(content);
    }
}
```

Good Luck!