

# RestController

## Démonstration 1 du module 6

Les objectifs de cette démonstration sont

- La mise en place de l'annotation @RestController
- et le mapping d'une classe POJO en JSON

## Contexte

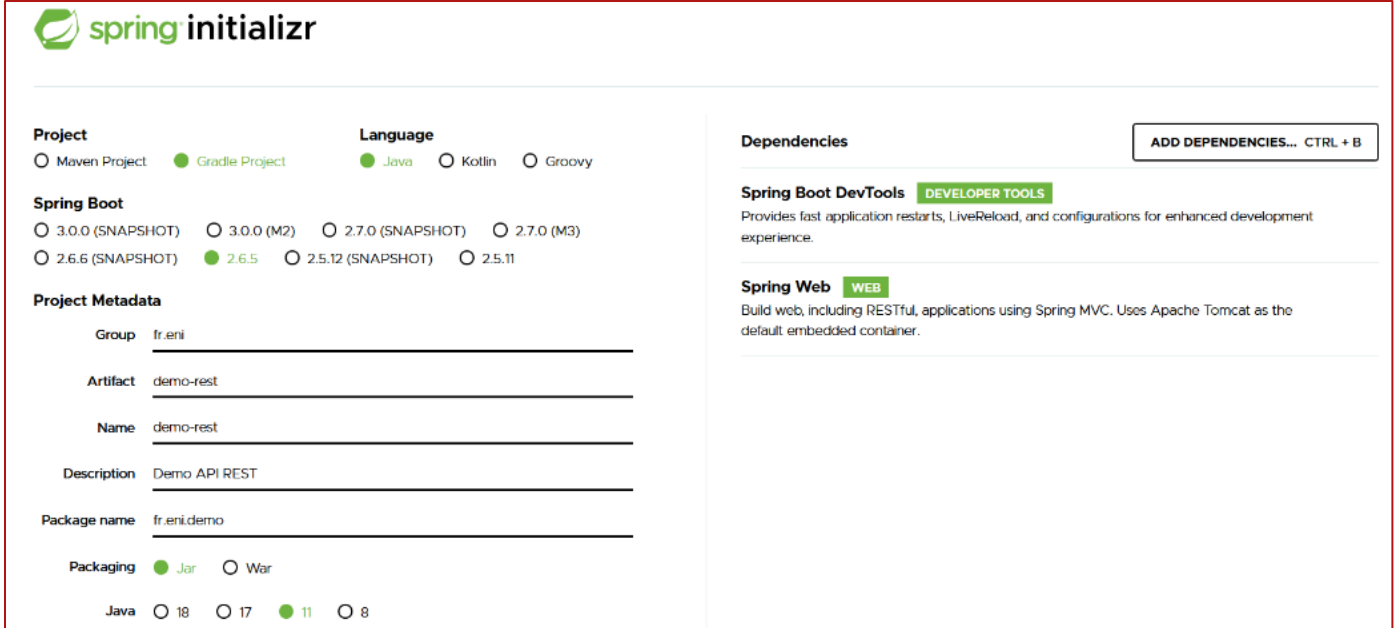
- Nous voulons gérer une application Web en nous appuyant sur l'architecture REST pour un site de vente en lignes
- Pour cette étape, il faut gérer la liste des articles disponibles et permettre dans ajouter, supprimer et modifier.

## Déroulement

### Création d'un nouveau projet avec Spring Boot

- Aller sur le site : <https://start.spring.io/>
- Sélectionner :
  - Gradle Project (Gradle est le gestionnaire de dépendances que nous utilisons)
  - Java 11
  - Prendre la version la plus récente de Spring Boot GA
  - Group : fr.eni
  - Artifact : demo-rest
  - Package name : fr.eni.demo
- Le livrable sera créé sous la forme d'un .jar et non d'un .war ce qui facilite le déploiement

Ajouter les librairies : Spring Web, Spring Boot Dev Tools ( pour le rechargement automatique du serveur suite à des changements de code)



The image shows the Spring Initializr web interface for creating a new project. The 'Project' section has 'Maven Project' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '2.6.5' selected. The 'Project Metadata' section contains the following fields: Group (fr.eni), Artifact (demo-rest), Name (demo-rest), Description (Demo API REST), Package name (fr.eni.demo), and Packaging (Jar). The 'Dependencies' section shows 'Spring Boot DevTools' and 'Spring Web' as selected dependencies.

BO :

- Création d'une classe Article :

```
package fr.eni.demo.bo;

public class Article {
    private long id;
    private String mark;
    private String department;
    private String title;
    private String description;
    private float price;

    public Article() {
    }

    public Article(long id, String mark, String department, String title, String description, float price) {
        this.id = id;
        this.mark = mark;
        this.department = department;
        this.title = title;
        this.description = description;
        this.price = price;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getMark() {
        return mark;
    }

    public void setMark(String mark) {
        this.mark = mark;
    }
}
```

```

public String getDepartment() {
    return department;
}

public void setDepartment(String department) {
    this.department = department;
}

public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public float getPrice() {
    return price;
}

public void setPrice(float price) {
    this.price = price;
}

@Override
public String toString() {
    return "Article [id=" + id + ", mark=" + mark + ", department=" + department + ", title=" +
title
        + ", description=" + description + ", price=" + price + "];"
}
}

```

## BLL :

- Création de l'interface ArticleService :
  - Elle déclare des méthodes pour récupérer
    - l'ensemble des articles,
    - un article par son identifiant,
  - Elle déclare une méthode de sauvegarde et une de suppression

```

package fr.eni.demo.bll;

import java.util.List;

import fr.eni.demo.bo.Article;

public interface ArticleService {
    List<Article> findAll();

    Article findById(Long id);

    void save(Article article);

    void delete(Long id);
}

```

- Création d'une classe bouchon, ArticleServiceMock qui implémente l'interface précédente :

```
package fr.eni.demo.bll.mock;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.springframework.stereotype.Service;

import fr.eni.demo.bo.Article;
import fr.eni.demo.bll.ArticleService;

@Service
public class ArticleServiceMock implements ArticleService {
    // Liste des articles pour gérer la sauvegarde
    private static List<Article> lstArticles;
    private int index;

    public ArticleServiceMock() {
        // Initialisation de la liste
        lstArticles = new ArrayList<Article>();
        index = 1;
        lstArticles.add(new Article(index++, "Decathlon", "Sport", "VTC", "Vélo tout Chemin",
175));
        lstArticles
            .add(new Article(index++, "Boulangier", "Electroménager", "Aspirateur VA",
"Aspirateur sans fil", 199));
    }

    @Override
    public List<Article> findAll() {
        return lstArticles;
    }

    @Override
    public Article findById(Long id) {
        for (Article article : lstArticles) {
            if (article.getId() == id) {
                return article;
            }
        }
        return null;
    }

    @Override
    public void save(Article article) {
        if (article.getId() == 0) {
            //nouvel article
            article.setId(index++);
            lstArticles.add(article);
        } else {
            //Mise à jour
            Article current = findById(article.getId());
            current.setMark(article.getMark());
            current.setDepartment(article.getDepartment());
            current.setTitle(article.getTitle());
            current.setDescription(article.getDescription());
            current.setPrice(article.getPrice());
        }
    }

    @Override
    public void delete(Long id) {

```

```

        Iterator<Article> it = lstArticles.iterator();
        while (it.hasNext()) {
            Article current = it.next();
            if (current.getId() == id) {
                it.remove();
            }
        }
    }
}

```

## Contrôleur de l'application Web REST :

- Création d'une classe @RestController permettant de mettre à disposition les méthodes liés aux articles :
  - Get
  - Post
  - Put
  - Delete

```

package fr.eni.demo.rest.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import fr.eni.demo.bo.Article;
import fr.eni.demo.bll.ArticleService;

@RestController
@RequestMapping("/articles")
public class ArticleController {
    private ArticleService articleService;

    @Autowired
    public ArticleController(ArticleService articleService) {
        this.articleService = articleService;
    }

    // expose '/articles' and return list of articles
    @GetMapping
    public List<Article> findAll() {
        return articleService.findAll();
    }

    // expose with Get '/articles/{id}' and return article
    @GetMapping("/{id}")
    public Article findById(@PathVariable long id) {
        Article article = articleService.findById(id);
        if (article == null) {
            throw new RuntimeException("Article id not found - " + id);
        }
        return article;
    }
}

```

```

// expose with Post '/articles}'
@PostMapping
public Article addArticle(@RequestBody Article theArticle) {
    System.out.println("addArticle");
    System.out.println(theArticle);
    // Also just in case they pass an id in JSON...
    // Set id to 0 this is to force a save of new itemm..
    // instead of update
    theArticle.setId(0);
    articleService.save(theArticle);
    return theArticle;
}

// expose with Put '/articles - update existing article'
@PutMapping
public Article putArticle(@RequestBody Article theArticle) {
    System.out.println("addArticle");
    System.out.println(theArticle);
    articleService.save(theArticle);
    return theArticle;
}

// expose with Delete '/articles/{id}' - delete article
@DeleteMapping("/{id}")
public String deleteArticle(@PathVariable long id) {
    Article article = articleService.findById(id);
    if (article == null) {
        throw new RuntimeException("Article id not found - " + id);
    }
    articleService.delete(id);
    return "Deleted article id - " + id;
}
}

```

## Utilisation de ReqBin pour tester les requêtes :

- <https://reqbin.com/>
- C'est un plugin qui s'installe sous Chrome

The screenshot shows the ReqBin website interface. On the left is a sidebar with a list of request examples: POST Request Example, REST API POST Example, POST JSON Example, GET Request Example, Bearer Token Auth Example, JSON Comments Example, GET JSON Example, JSON Payload Example, JSON Pagination Example, JSON Response Example, Sample API POST Request, POST XML Example, HTML Form POST Example, Send Cookies Example, PUT Request Example, PATCH Request Example, DELETE Request Example, HEAD Request Example, OPTIONS Request Example, REST API GET Example, and Custom Headers Example. The main content area is titled "Online REST & SOAP API Testing Tool" and describes the tool's capabilities. It includes a form to create a new request with fields for URL, method, and authorization. The "Authorization" tab is selected, showing options for Bearer Token, Basic Auth, and Custom. A "Token" field is present. The "Send" button is highlighted. On the right, a section titled "API Testing Made Easy" lists features: Test APIs, websites and web services online; Post requests directly from your browser; Share and discuss your requests online; Load test APIs and websites; Generate PHP, Python, JavaScript/AJAX, Java, C#/.NET, and Curl/Bash code snippets for your requests; Built-in JSON, XML, HTML and CSS validators. At the bottom, it states: "ReqBin API testing tool provides millisecond precision timings for API requests. Run API tests and find performance bottlenecks in your API."

## Exécution :

- Lancer l'application de votre serveur.
- Puis dans ReqBin tester l'ensemble des méthodes
- Get : <http://localhost:8080/articles>

File Generate Code Tools Share Generate Code Debug API Premium

http://localhost:8080/articles GET EXT Send Status: 200 () Time: 147 ms Size: 0.24 kb

Authorization Content Headers Raw (2) Content (15) Headers (6) Raw (8) JSON

☒ Bearer Token ☐ Basic Auth ☐ Custom

Token

The authorization header will be automatically generated when you send the request. Read more about HTTP Authentication.

```
[{
  "id": 1,
  "mark": "Decathlon",
  "department": "Sport",
  "title": "VTC",
  "description": "Vélo tout Chemin",
  "price": 175.0
}, {
  "id": 2,
  "mark": "Boulangier",
  "department": "Electroménager",
  "title": "Aspirateur VA",
  "description": "Aspirateur sans fil",
  "price": 199.0
}]
```

- Get by id : <http://localhost:8080/articles/1>

File Generate Code Tools Share Generate Code Debug API Premium

http://localhost:8080/articles/1 GET EXT Send Status: 200 () Time: 34 ms Size: 0.11 kb

Authorization Content Headers Raw (2) Content (8) Headers (6) Raw (8) JSON

☒ Bearer Token ☐ Basic Auth ☐ Custom

Token

The authorization header will be automatically generated when you send the request. Read more about HTTP Authentication.

```
{
  "id": 1,
  "mark": "Decathlon",
  "department": "Sport",
  "title": "VTC",
  "description": "Vélo tout Chemin",
  "price": 175.0
}
```

- Get by id (pour un identifiant inexistant) : <http://localhost:8080/articles/0>

File Generate Code Tools Share Generate Code Debug API Premium

http://localhost:8080/articles/0 GET EXT Send Status: 500 () Time: 13 ms Size: 5.10 kb

Authorization Content Headers Raw (2) Content (8) Headers (5) Raw (7) JSON

☒ Bearer Token ☐ Basic Auth ☐ Custom

Token

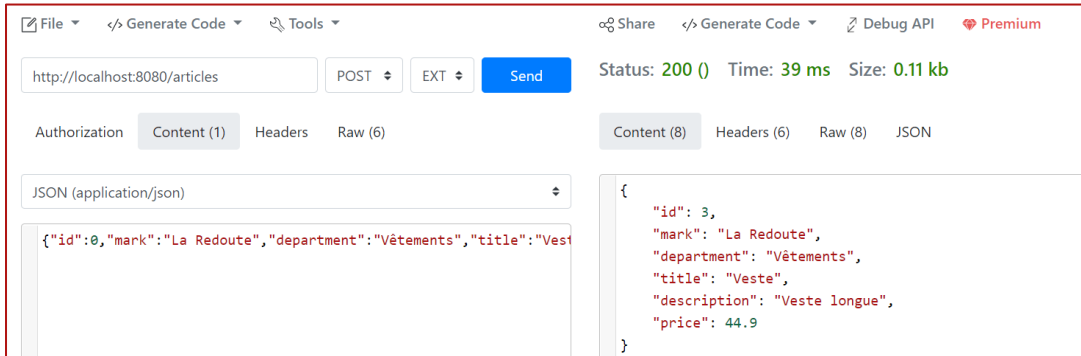
The authorization header will be automatically generated when you send the request. Read more about HTTP Authentication.

```
{
  "timestamp": "2022-03-31T10:14:27.152+00:00",
  "status": 500,
  "error": "Internal Server Error",
  "trace": "java.lang.RuntimeException: Article id not found - 0",
  "message": "Article id not found - 0",
  "path": "/articles/0"
}
```

- Post : <http://localhost:8080/articles>
  - Utiliser le JSON suivant pour créer un nouvel article :

```
{"id":0,"mark":"La Redoute","department":"Vêtements","title":"Veste longue","price":44.9}
```

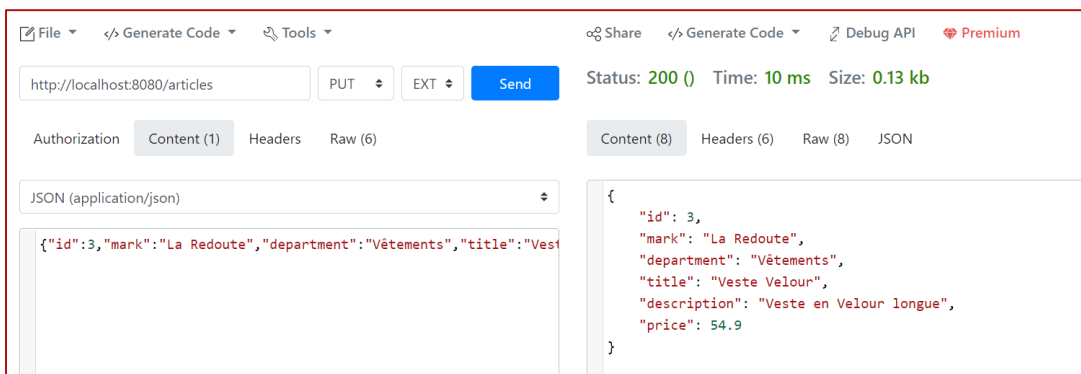
- Mettez bien en POST
- Et sélectionnez bien JSON (application/json)



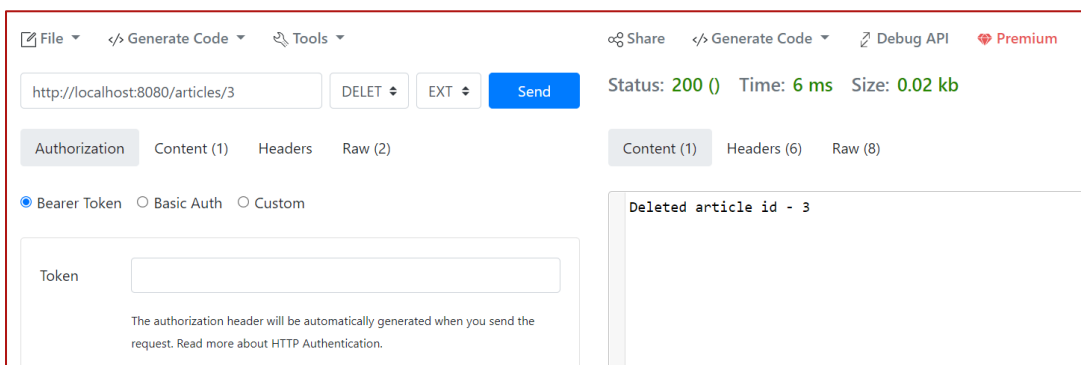
- Put : <http://localhost:8080/articles>
  - Fonctionnement similaire à POST
  - Utiliser le JSON suivant pour mettre à jour l'article précédemment enregistré :

```
{"id":3,"mark":"La Redoute","department":"Vêtements","title":"Veste Velour","description":"Veste en Velour longue","price":54.9}
```

- Mettez bien en PUT
- Et sélectionnez bien JSON (application/json)



- Delete : <http://localhost:8080/articles/3>
  - Sélectionnez DELETE





## Gestion des RuntimeException

- Création d'un contrôleur Spring qui pourra gérer les RuntimeException.

```
package fr.eni.demo.rest.controller.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;

@ControllerAdvice
class NotFoundAdvice {

    @ResponseBody
    @ExceptionHandler(RuntimeException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    String notFoundHandler(RuntimeException ex) {
        return "notFoundHandler - " + ex.getMessage();
    }
}
```

- Exécuter l'application
- Rester : Get by id (pour un identifiant inexistant) : <http://localhost:8080/articles/0>

The screenshot shows a REST client interface. The URL bar contains 'http://localhost:8080/articles/0'. The method is set to 'GET'. The status bar indicates 'Status: 404 ()', 'Time: 18 ms', and 'Size: 0.04 kb'. The 'Content' tab is selected, showing the response body: 'notFoundHandler - Article id not found - 0'. The 'Authorization' section shows 'Bearer Token' selected. A token input field is present, with a note: 'The authorization header will be automatically generated when you send the request. Read more about HTTP Authentication'.

- Maintenant ; l'AOP a capturé la RuntimeException, et notre contrôleur de gestion d'exception a transmis un 404 et juste le message de l'exception.

## Conclusion :

- Avec notre application Web sur l'architecture REST;
- Il est possible de transmettre des données au format JSON, XML, ...
- Et de l'interfacer avec une couche présentation orientée JS.
- Nous avons moyen de sécuriser les accès entre les :
  - URL des contrôleurs
  - La couche Service