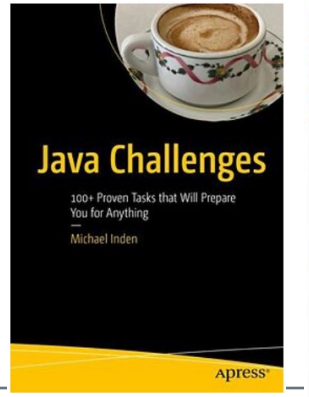
Spring + REST

Michael Inden

Speaker Intro

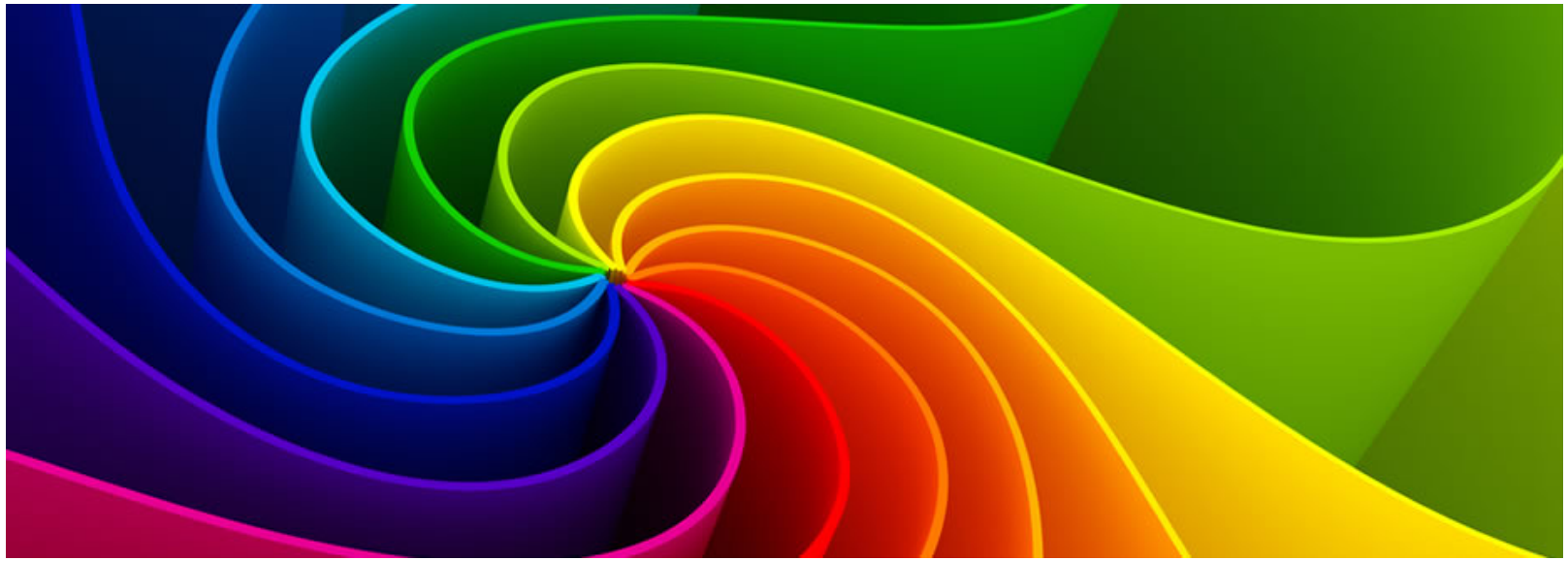
- Michael Inden, Year of Birth 1971
- Diploma Computer Science, C.v.O. Uni Oldenburg
- ~8 ¼ Years **SSE** at Heidelberger Druckmaschinen AG in Kiel
- ~6 ¾ Years **TPL, SA** at IVU Traffic Technologies AG in Aachen
- ~4 ¼ Years **LSA / Trainer** at Zühlke Engineering AG in Zurich
- ~3 Years **TL / CTO** at Direct Mail Informatics / ASMIQ in Zurich
- Independent **Consultant, Conference Speaker and Trainer**
- **Since January 2022 Head of Development at Adcubum in Zurich**
- Author @ dpunkt.verlag and APress

E-Mail: michael_inden@hotmail.com
Blog: https://jaxenter.de/author/minden

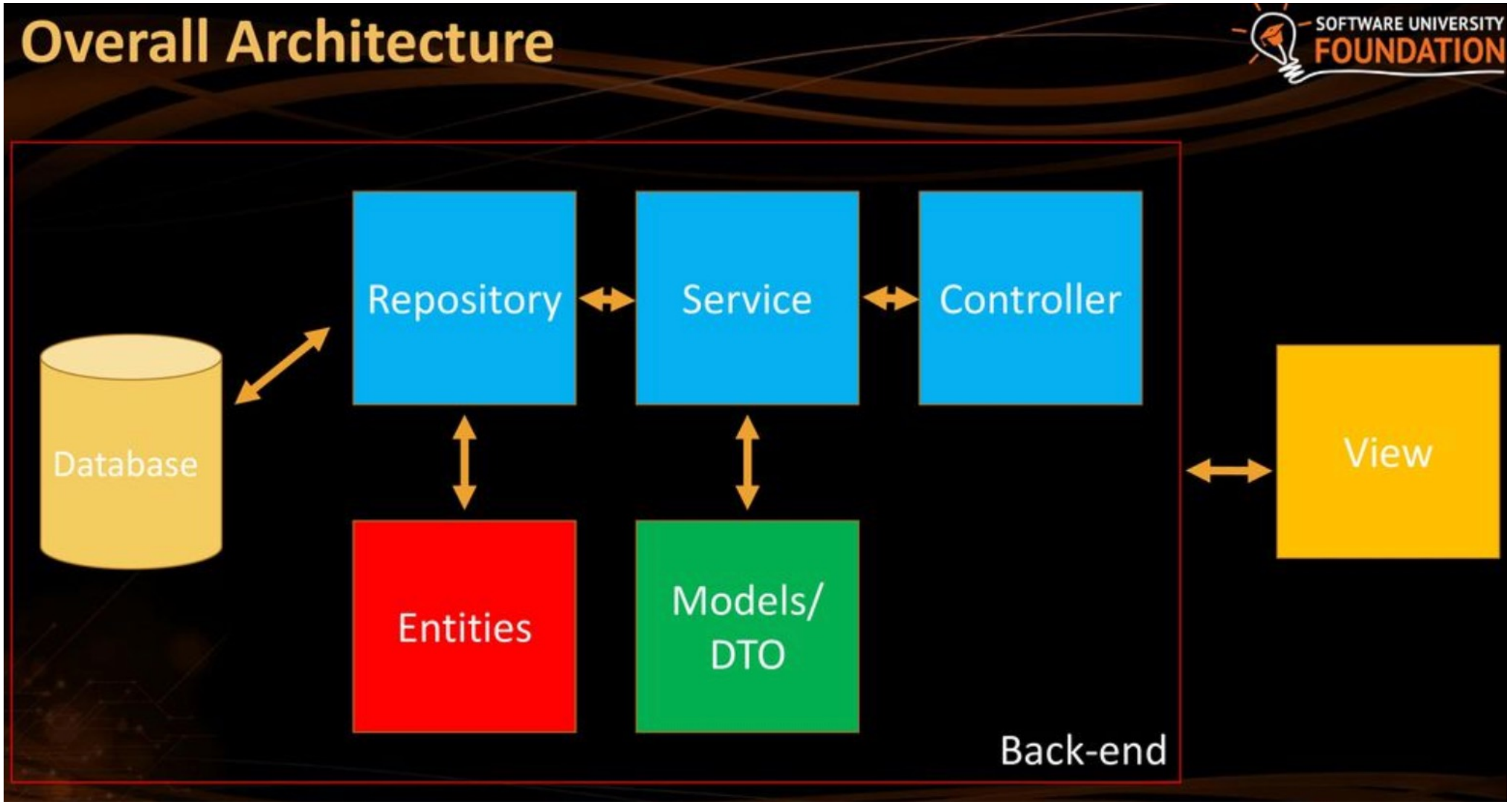


https://github.com/Michaeli71/ADC_BOOTCAMP_SPRING

Architecture in Spring

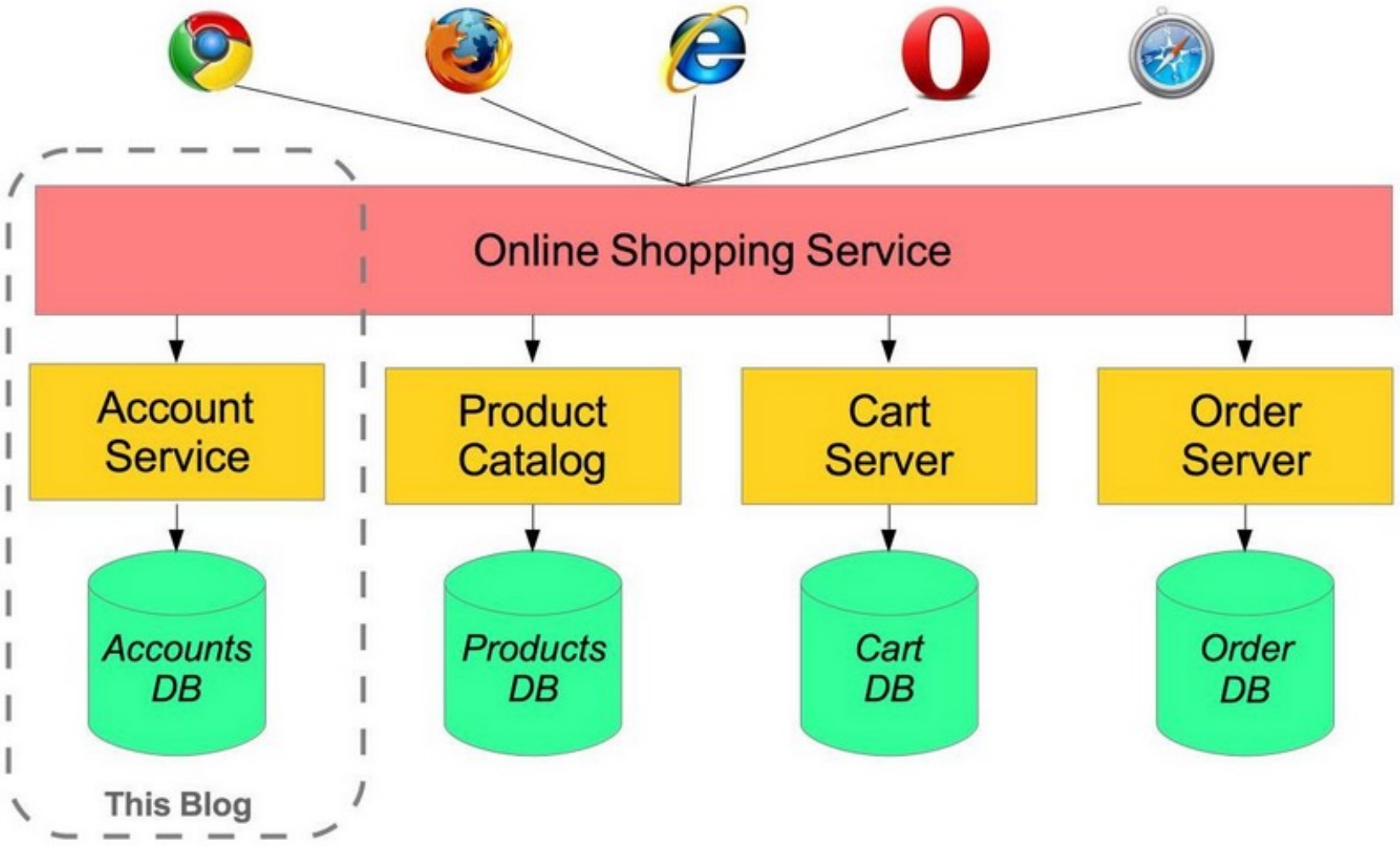


Architecture - Interaction



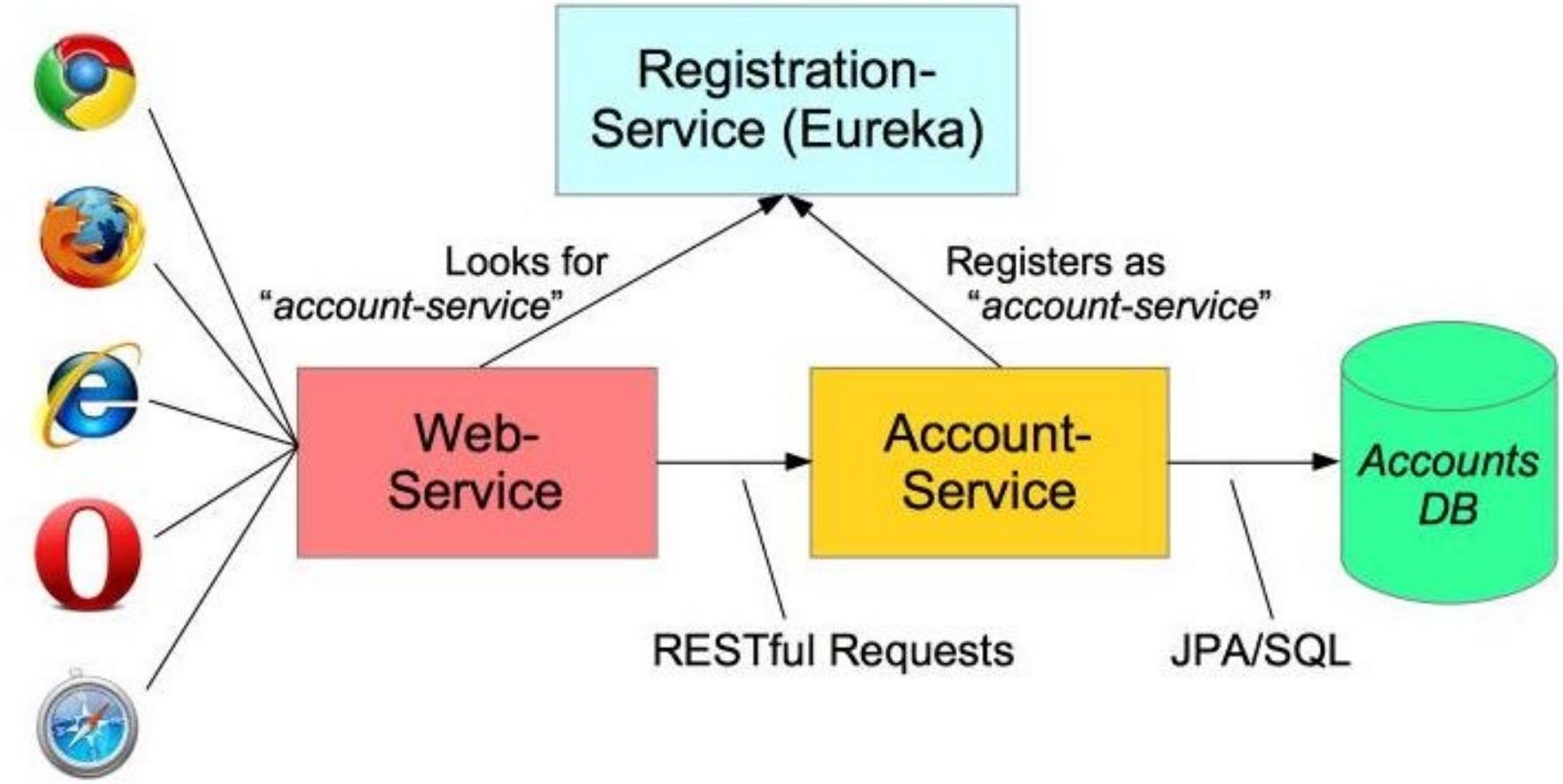
https://slideplayer.com/slide/13335057/

Microservices



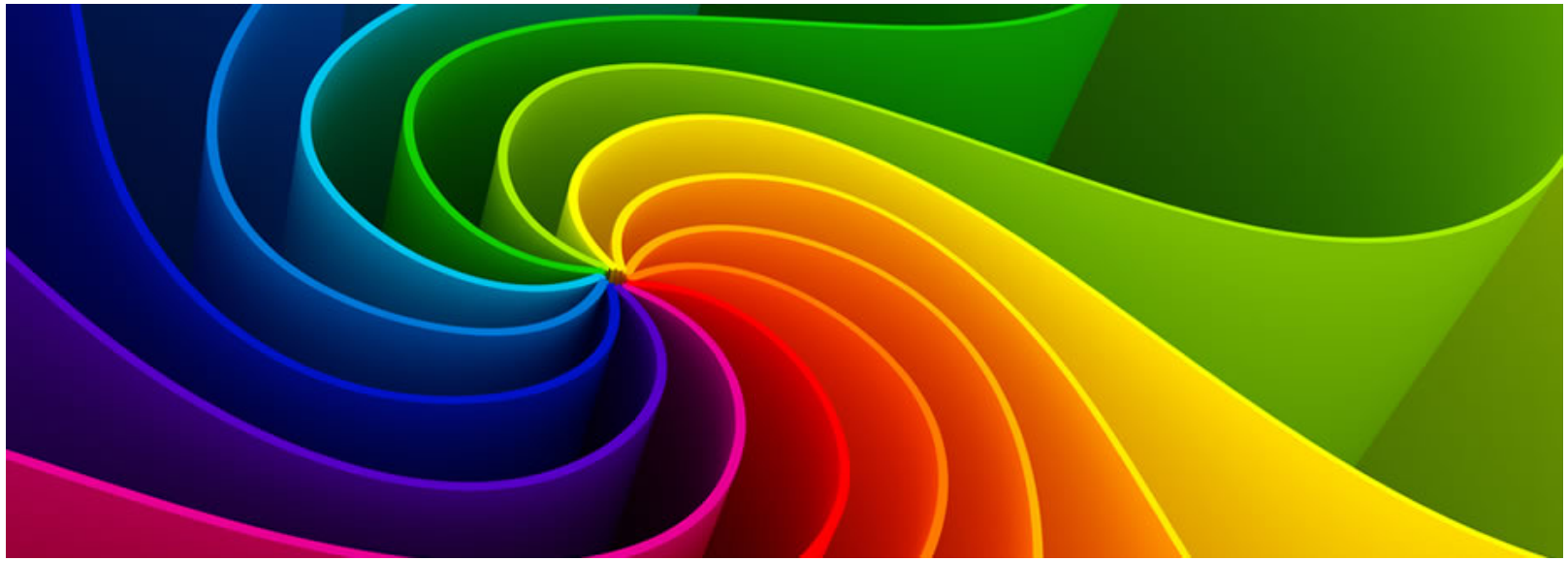
https://spring.io/blog/2015/07/14/microservices-with-spring

Microservices



https://spring.io/blog/2015/07/14/microservices-with-spring

Introduction to REST



Getting Started with REST Services

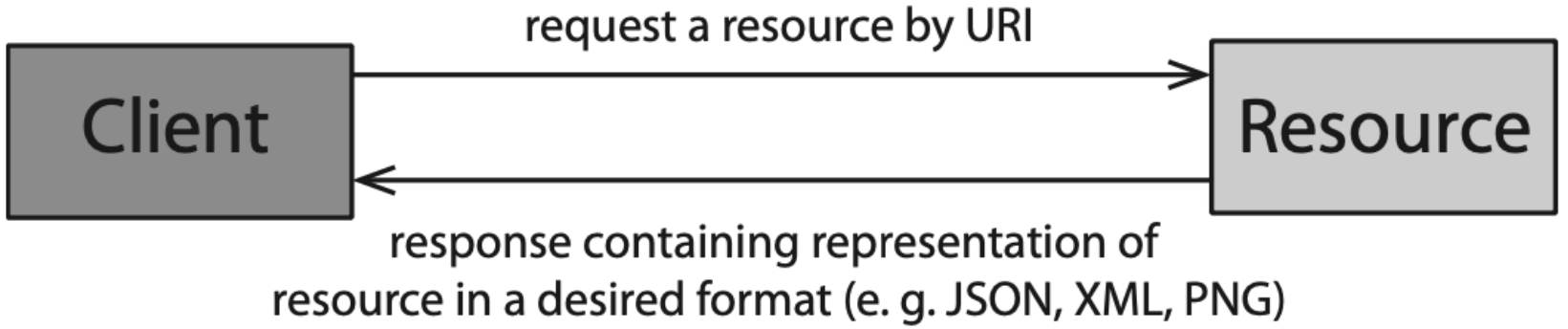
- Nowadays, it is widespread that applications are implemented as distributed systems, and different system components run on other computers or virtual machines (VMs).
- So-called web services can be used to provide functionality via the network, i.e., services that are targeted at the web.
- In contrast to the classic web applications running in a browser, web services are not so much intended for human users but offer functionalities in the form of special APIs (such as Twitter, Google, etc.) that can be accessed by other programs.
- REST = REpresentational State Transfer

Getting Started with REST Services

- REST = REpresentational State Transfer
- **The basic idea is to provide all functionality in the form of addressable resources, not via methods**. A resource can be anything. It can be accessed via a Uniform Resource Identifier (URI).
- **REST uses HTTP for stateless, client-server communication**. Here, a client sends requests to a server, which processes them and then sends back responses.
- **No standard format is defined for message exchange.**
- We can create REST using both XML and JSON. JSON is more common.

Getting Started with REST Services

- We distinguish between a **REST server**, which provides resources, and **REST clients**, which access these resources. For this purpose, each resource has an ID, which is usually modeled as a URI (Uniform Resource Identifier).



- A **URI** is used to address a **REST service** (also called a resource). It consists of a server and port as well as a base path and a path of the resource:

```
http://server:port/basePath/resourcePath/
```

Getting Started with REST Services

- To address a REST service (also called a resource), a URI is used that consists of a server and port as well as a base path and a path of the resource:

```
http://server:port/basePath/resourcePath/
```

- This addresses the REST service registered under it and executes the desired action.

- The communication is mainly based on the four operations:
  - POST – creation of new data
  - GET – read access for one or more resources
  - PUT – modifiaction of existing resource
  - DELETE – deletion of existing resource

Typical REST Calls

HTTP Verb (Command)	URL Path	Description
POST	/customers	Creates a new customer based on the info from the body
GET	/customers	Retrieves all customers
GET	/customers/<id>	Retrieves customer with the passed id
PUT	/customers/<id>	Updates the data set of the customer with the passed id based on the info from the body
DELETE	/customers/<id>	Deletes the customer with the passed id

Typical HTTP-Status-Codes

Status Code	Meaning	Description
200	Ok	The request was successfully processed
201	Created	A resource was successfully sent to the server and created
204	No Content	The request was successfully processed, but the response contains no data. Often used, for example, with DELETE.
400	Bad Request	The request was not properly constructed and the server is missing data
401	Unauthorized	The request requires authentication
403	Forbidden	The client does not have the necessary rights to perform an operation
404	Not Found	The requested resource was not found
500	Internal Server Error	There is an error on the server side and the request cannot be processed properly.

RECAP: RAD – Rapid Application Development

RestService to say “hello“

```
@SpringBootApplication
public class App {

        public static void main(String[] args) {
                SpringApplication. run(App. class, args);
        }
}

@RestController
public class MyRestController{

        @GetMapping("/hello")
        public String hello(){
                return "hello ZAGREB";
        }
}
```

RECAP: More REST actions

```
@GetMapping("/items")
public void getItems(){
      ...
}

@GetMapping("/items/{itemId}")
public void getItems(@PathVariable String itemId){
      ...
}

@PostMapping("/items")
public void putItems(@RequestBody ShoppingItem item){
      ...
}
```

Example: Typical CRUD REST server

```
@RestController
public class ProductController {
    @Autowired
    private IProductService productService;

    @PostMapping(value = "/products")
    public Product create(@RequestBody Product product) {
        return productService.create(product);
    }

    @GetMapping(value = "/products")
    public List<Product> getProduct() {
        return productService.findAll();
    }

…
```

Example: Typical CRUD REST server

…

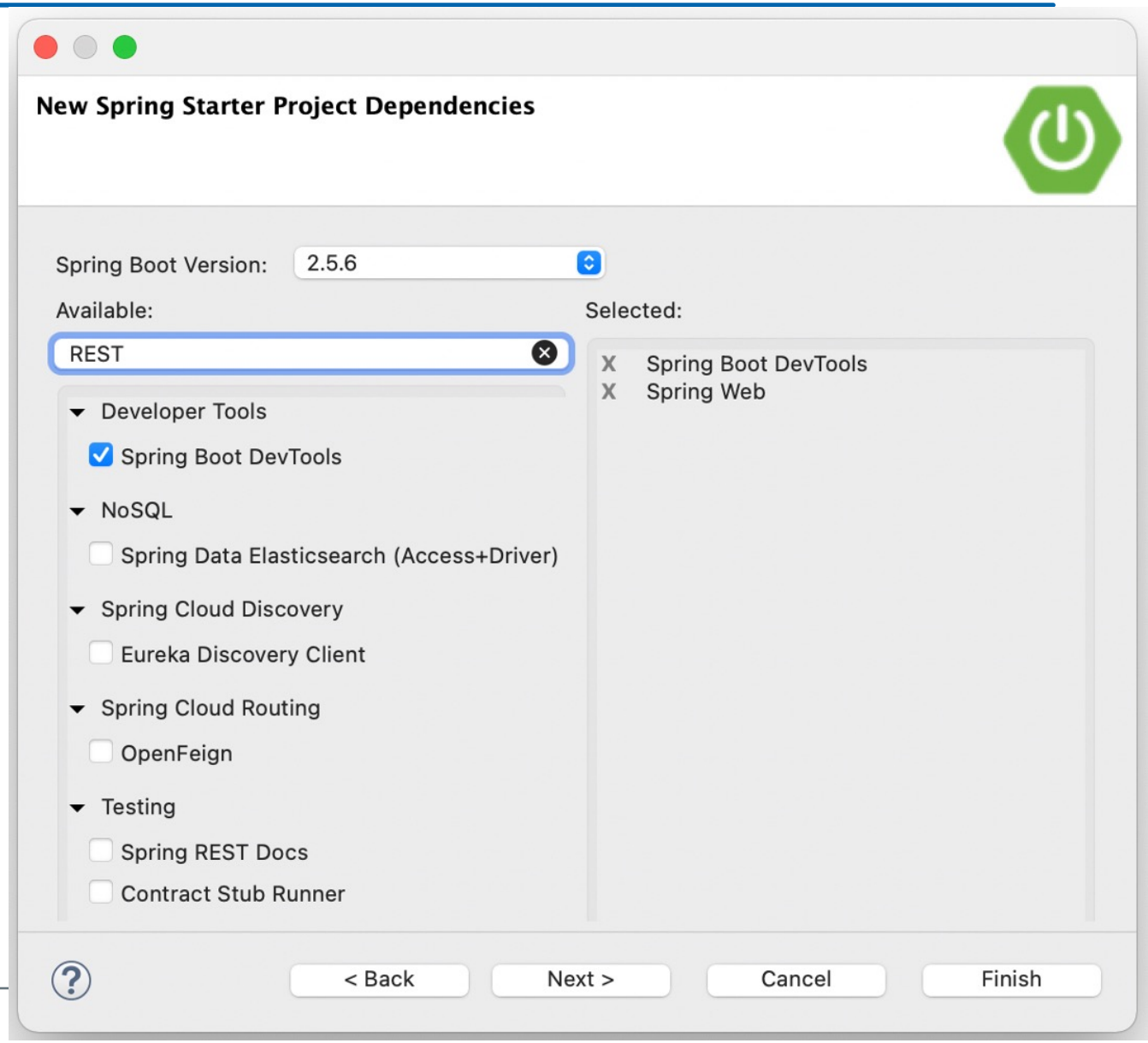
```
    @GetMapping(value = "/products/{id}")
    public Product findById(@PathVariable("id") long id) {
        return productService.findById(id);
    }

    @DeleteMapping(value = "/products/{id}")
    public void deleteById(@PathVariable("id") long id) {
        productService.deleteById(id);
    }
}
```

D E M O

«spring-rest-product-controller»

Example: Spring Boot Rest App

- Management of books
- REST controller with CRUD functionality
- In-memory fake repository
- But with validation!



Example: Spring Boot Rest App for Managing Books

```
@RestController
@RequestMapping("/books")
public class BookController {

    @Autowired
    private BookService service;

    @GetMapping
    public List<Book> findAll() {
        return service.findAll();
    }

    @GetMapping(value = "/{id}")
    public Book findById(@PathVariable("id") long id) {
        return service.findById(id);
    }
…
```

Example: Spring Boot Rest App for Managing Books

```
@RestController
@RequestMapping("/books")
public class BookController {

…

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public Book create(@Valid @RequestBody Book resource) {
        Objects.requireNonNull(resource);

        return service.create(resource);
    }

…
```

D E M O

«BookManagementApp»

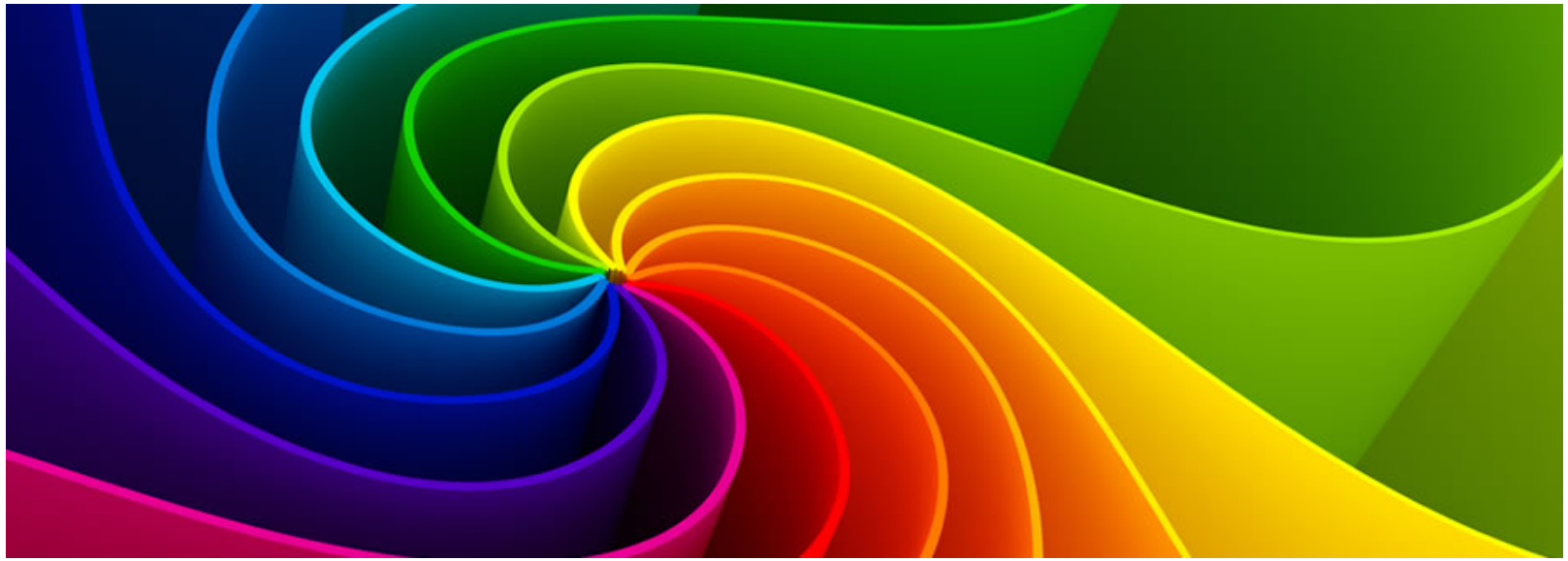
Advantages of REST servers

- RESTful web services are platform independent.
- They can be written in any programming language and run on any platform.
- They provide various data formats such as JSON, text, HTML, and XML.
- The interface is unified and exposes resources.
- The service should have a multi-tier architecture.

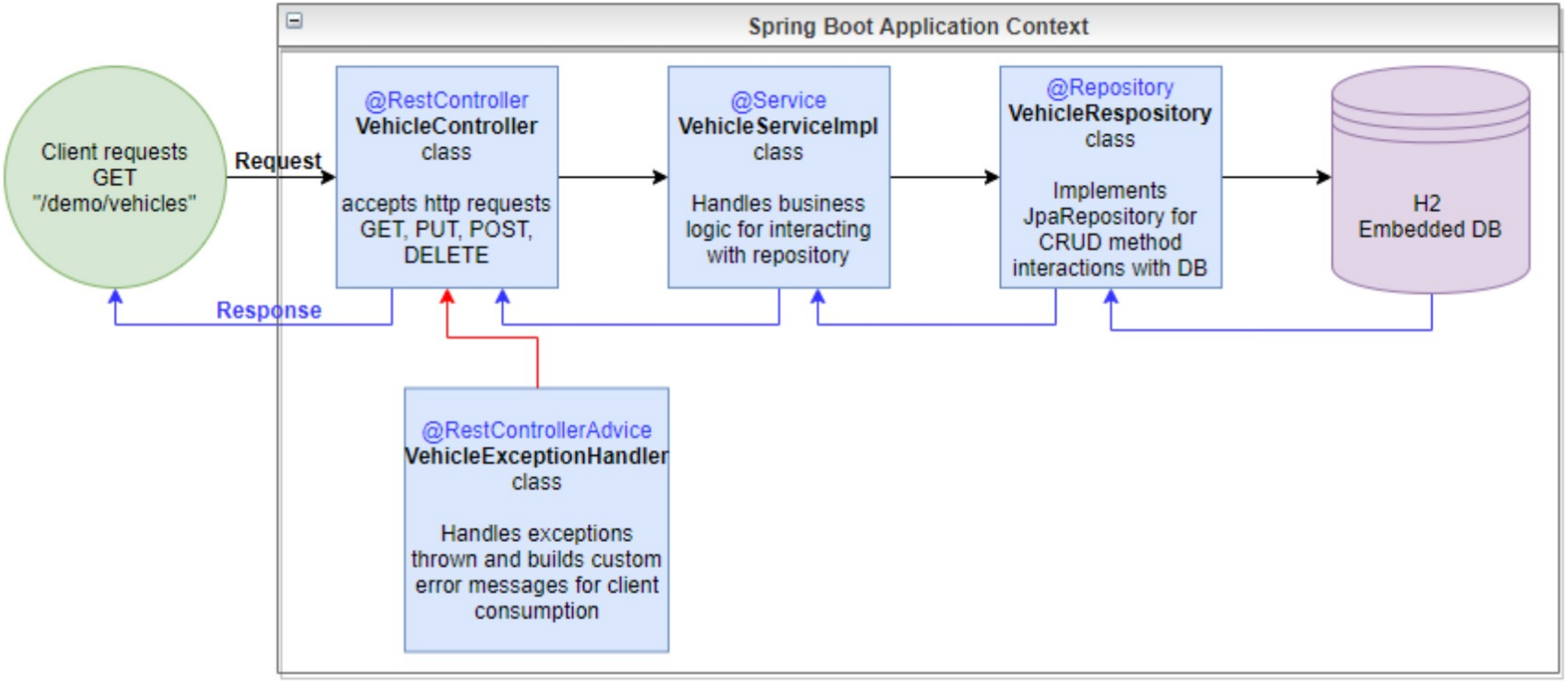
Comparision with JAX-RS

Annotation	Beschreibung
`@Path`	Legt den Pfad fest, unter dem die Ressource ansprechbar ist.
`@POST`	Die annotierte Methode reagiert auf ein HTTP `POST`.
`@GET`	Die annotierte Methode bearbeitet einen HTTP `GET` Request.
`@PUT`	Die annotierte Methode reagiert auf HTTP `PUT`.
`@DELETE`	Die annotierte Methode behandelt HTTP `DELETE` .
`@Produces`	Bestimmt, welche Rückgabeformate von der annotierten Methode produziert werden können.
`@Consumes`	Legt für Eingabeparameter fest, in welchem Format diese von der annotierten Methode entgegengenommen werden können.
`@PathParam`	Beschreibt Parameter, die im Pfad der URL notiert werden.
`@QueryParam`	Beschreibt Parameter, die im Query-Teil der URL angegeben sind, also nach dem `?` als Name-Wert-Paar `name = value` und durch `&` voneinander getrennt.
`@FormParam`	Beschreibt Parameter, die über HTML-Formulare eingegeben werden.

Introduction to REST Client



Big picture


Spring Boot Application Context
Client requests GET "/demo/vehicles"
Request
@RestController
VehicleController
class
accepts http requests GET, PUT, POST, DELETE
@Service
VehicleServiceImpl
class
Handles business logic for interacting with repository
@Repository
VehicleRespository
class
Implements JpaRepository for CRUD method interactions with DB
H2
Embedded DB
Response
@RestControllerAdvice
VehicleExceptionHandler
class
Handles exceptions thrown and builds custom error messages for client consumption

Spring Boot REST Client App

- Goal: Query books from the other app
- RestTemplate for " submitting " REST calls
  - getForObject
  - postForObject
  - …

```
public PhonebookEntry createEntry(PhonebookEntry entry)
{
    final String uri = "http://localhost:8081/api/v1/phonebook";

    RestTemplate restTemplate = new RestTemplate();
    PhonebookEntry result = restTemplate.postForObject( uri, entry, PhonebookEntry.class);
    return result;
}
```

Spring Boot REST Client App

```
@Component
public class BookInfoClient {

    public static final String SERVER_URI = "http://localhost:8888/books";

    // ATTENTION: does not work: @Autowired
    private RestTemplate restTemplate;

    @Autowired
    public BookInfoClient(RestTemplateBuilder builder) {
        this.restTemplate = builder.build();
    }

    @SuppressWarnings("unchecked")
    public List<Book> getAllBooks() {
        List<Book> books = (List<Book>) restTemplate.getForObject(SERVER_URI,
                                List.class);
        System.out.println(books);
        return books;
    }
```

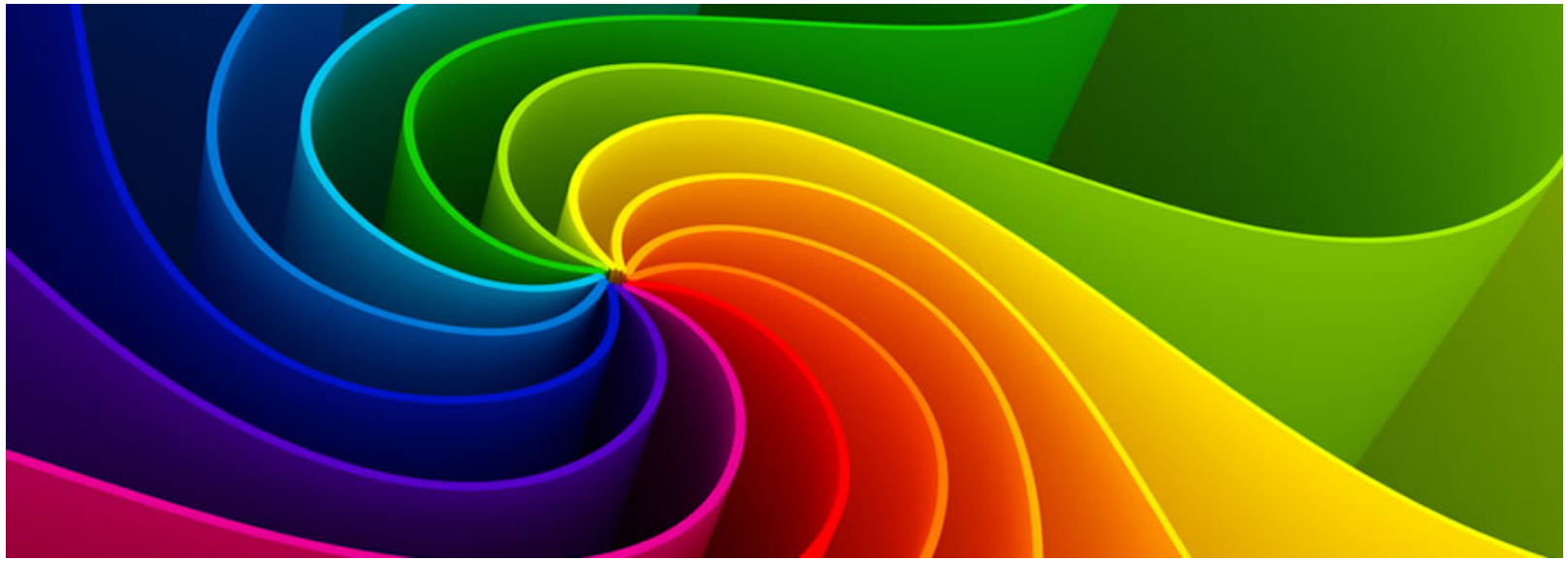
D E M O

«BookInfoClientApp»

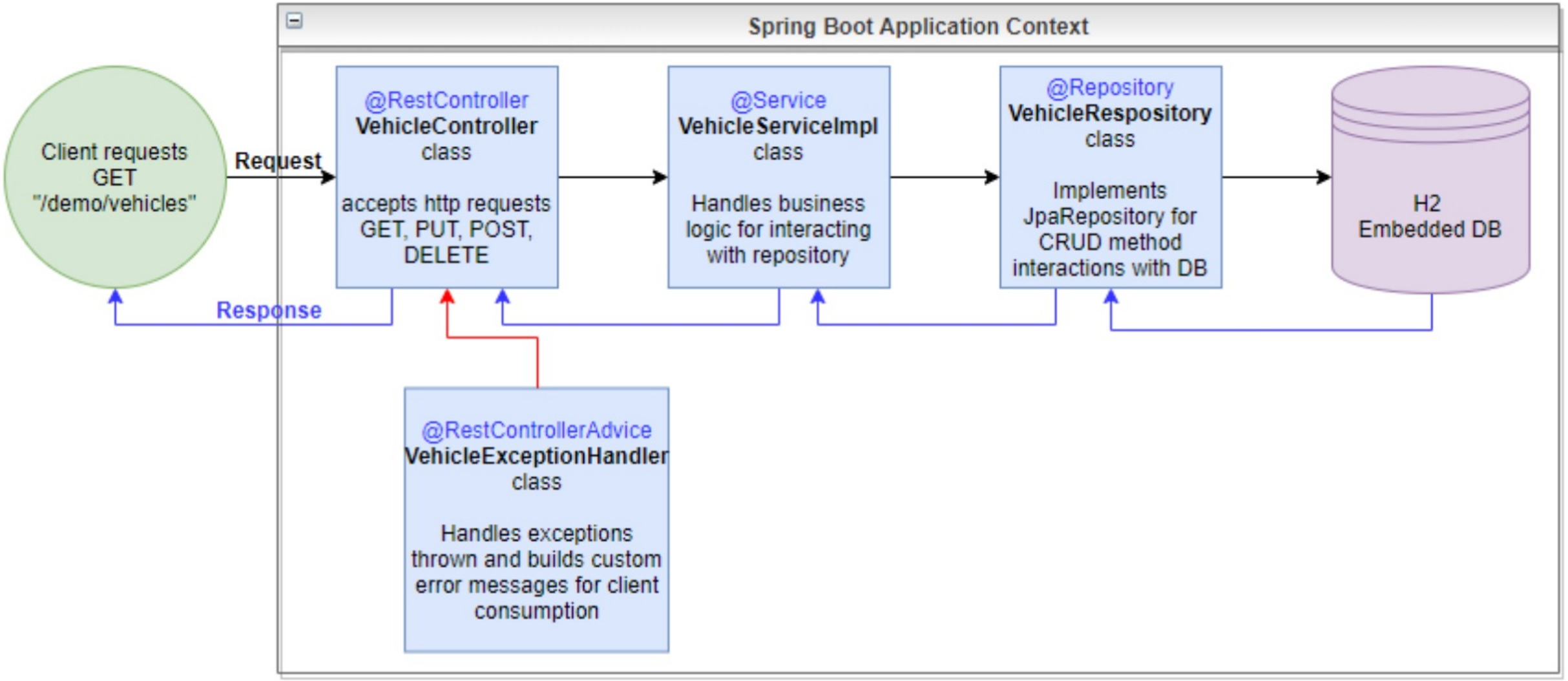
Add On Exercises 1 + 2

https://github.com/Michaeli71/ADC_BOOTCAMP_SPRING

Exception Handler

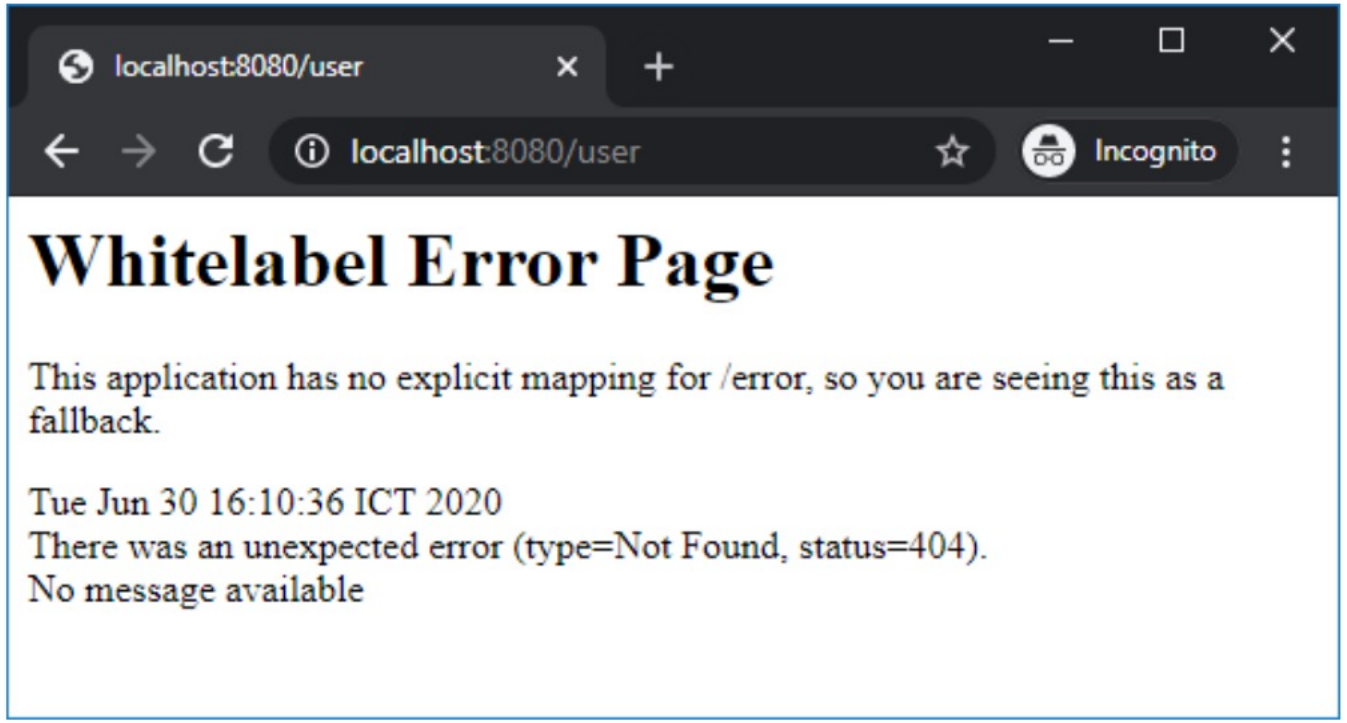


RECAP: More REST actions



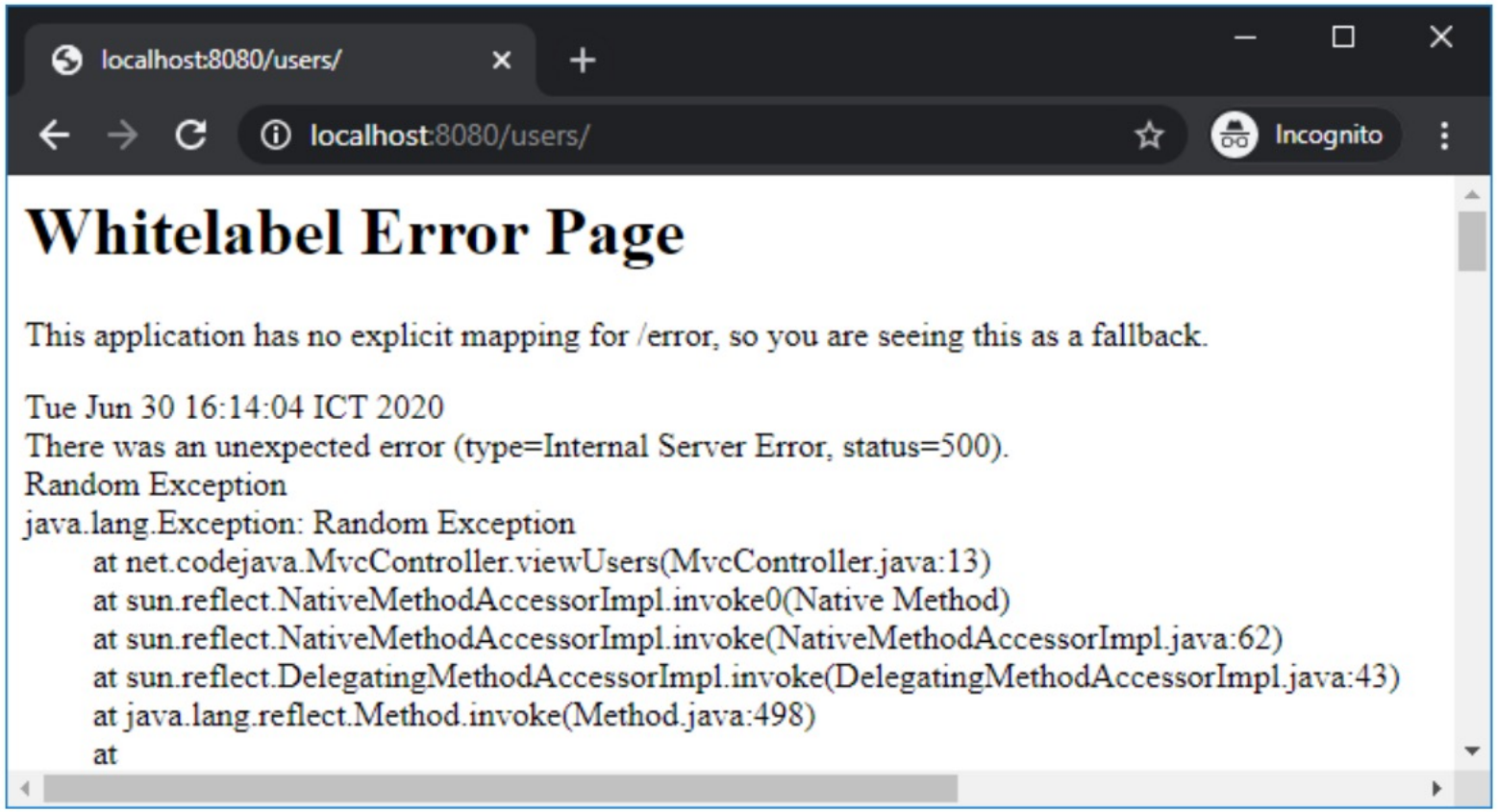
Error Handling

- By default, Spring Boot displays the whitelabel error page when an error occurs. For example, when a page could not be found (HTTP 404 error):



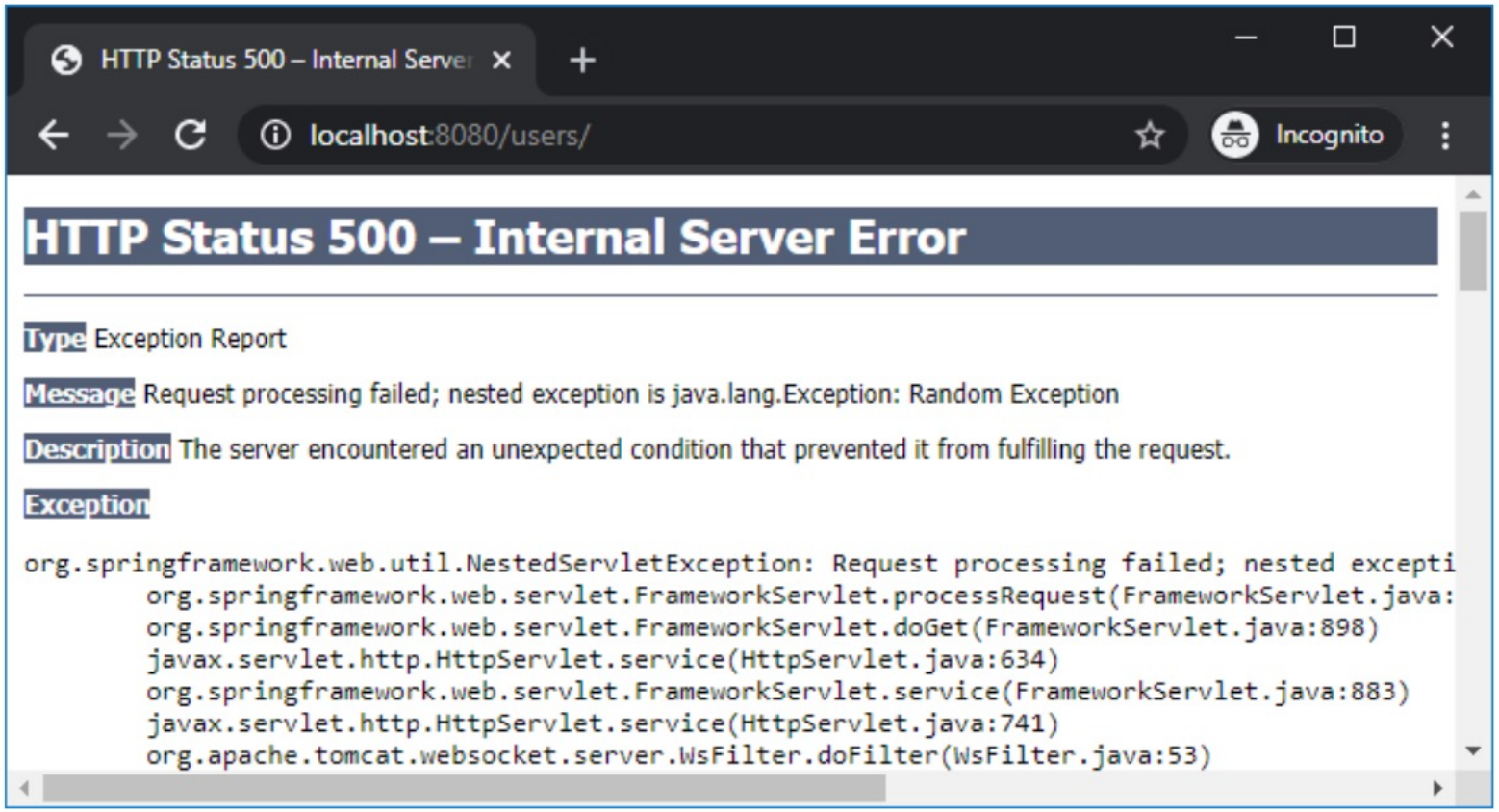
Error Handling

- When an exception occurs that causes an HTTP 500 Internal Server Error, the white-label error page is displayed with the stack trace of the exception:



Error Handling

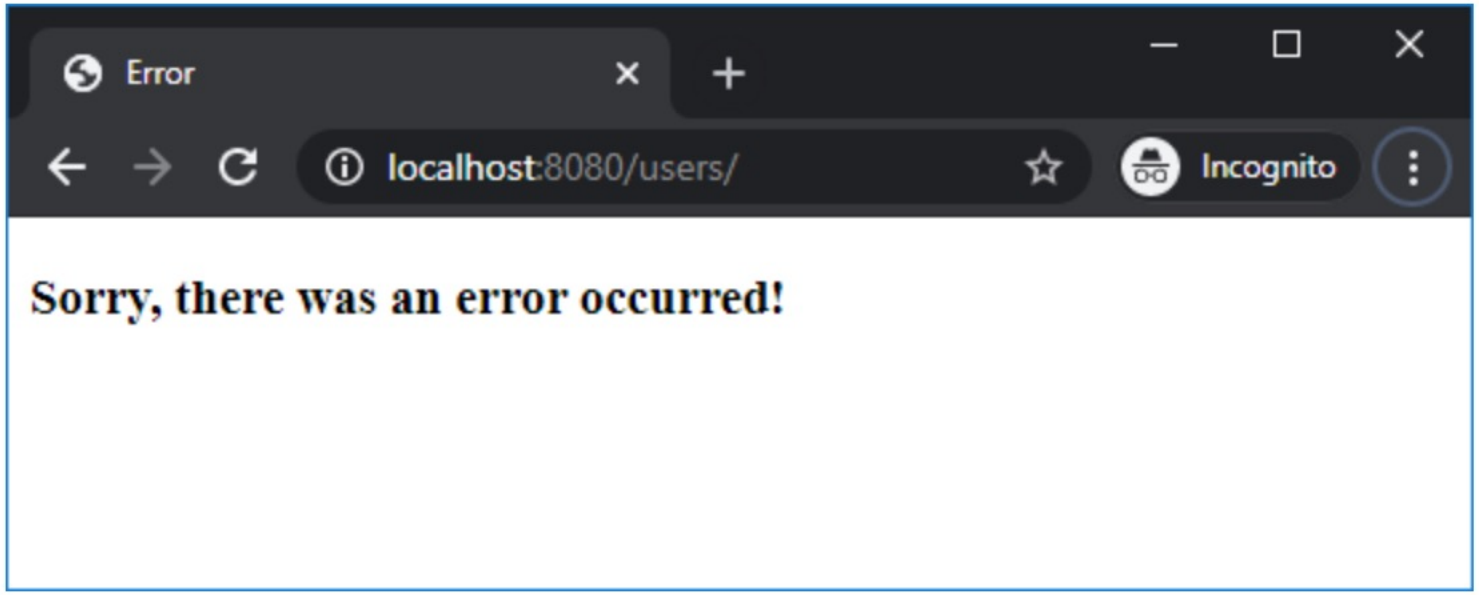
- `server.error.whitelabel.enabled=false`



Custom Error Pages

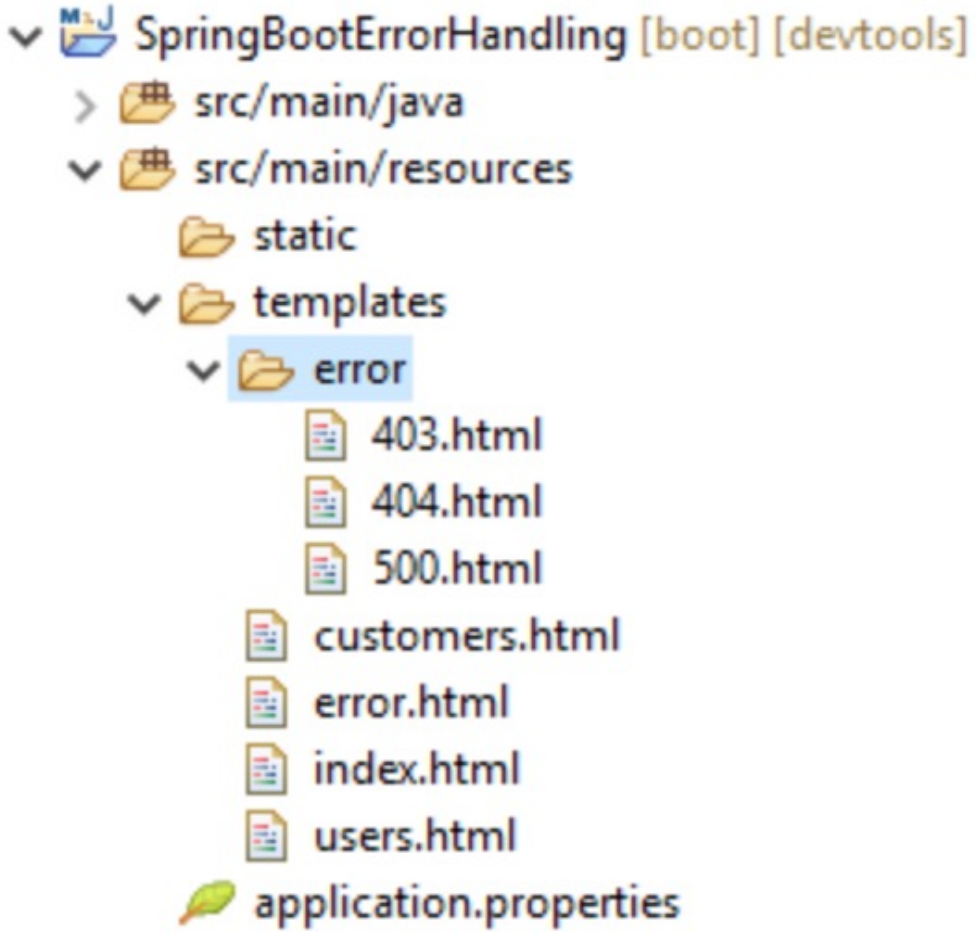
- `server.error.whitelabel.enabled=false`

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Error</title>
</head>
<body>
    <h3>Sorry, there was an error occurred!</h3>
</body>
</html>
```



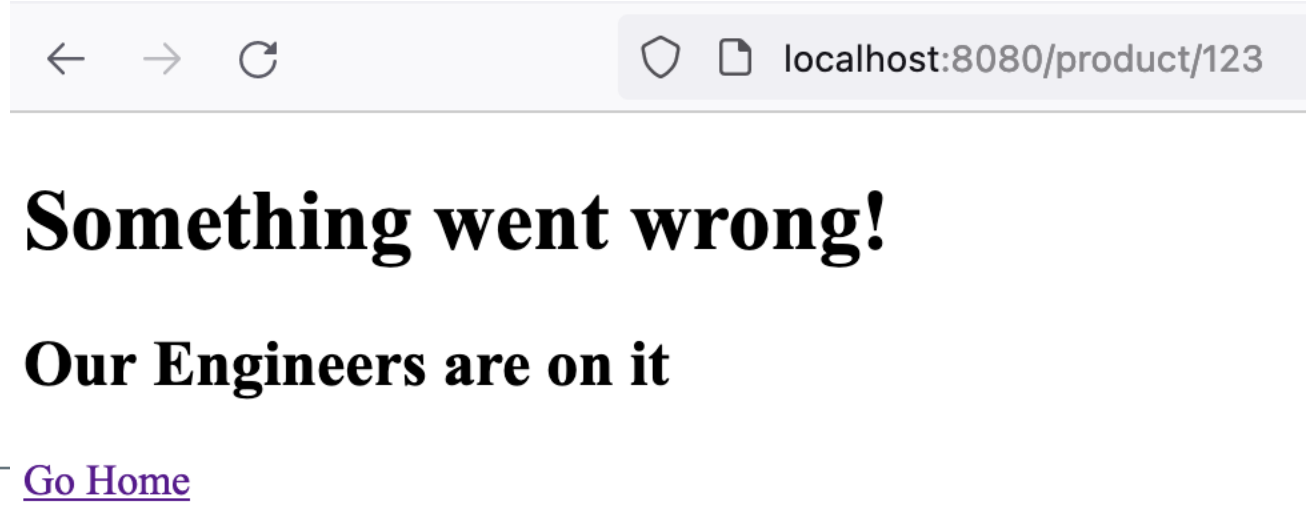
Custom Error Pages

- Spezifische Error-Seiten



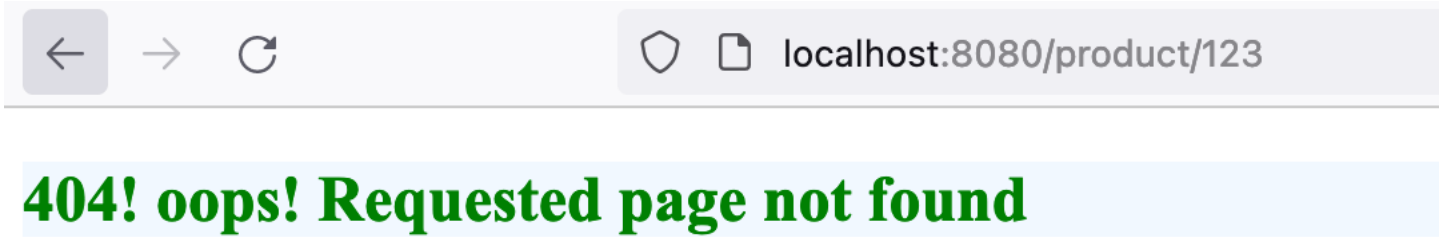
Custom Error Controller

```
<!DOCTYPE html>
<html>
<body>
<h1>Something went wrong! </h1>
<h2>Our Engineers are on it</h2>
<a href="/">Go Home</a>
</body>
</html>
```



Custom Error Controller

```
<html>
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
    <title>400 Bad Request</title>
    <meta name="viewport" content="width=device-width">
    <style>
        #error {
            border-color: darkred;
            background-color: aliceblue;
        }
        h2 {
            color: green;
        }
    </style>
</head>
<body>
<div class="error-page-wrap">
    <div id="error">
        <h2>400! oops! Passed WRONG parameters</h2>
    </div>
</div>
</body>
</html>
```



D E M O

«spring-rest-product-error-controller»

Custom error controller analogous to standard

- If you want to perform some actions before the custom error pages are displayed, consider implementing a controller class that satisfies the ErrorController interface:

```
@RestController
public class CustomErrorController implements ErrorController {
    @RequestMapping("/error")
    public ModelAndView handleError(HttpServletResponse response) {
        ModelAndView modelAndView = new ModelAndView();

        if (response.getStatus() == HttpStatus.BAD_REQUEST.value()) {
            modelAndView.setViewName("error/400");
        } else if (response.getStatus() == HttpStatus.NOT_FOUND.value()) {
            modelAndView.setViewName("error/404");
        } else if (response.getStatus() == HttpStatus.INTERNAL_SERVER_ERROR.value()) {
            modelAndView.setViewName("error/500");
        } else {
            modelAndView.setViewName("error");
        }
        return modelAndView;
    }
}
```

Initial situation: REST controller with exception handling in methods

```
public class DogsController {
    @Autowired private final DogsService service;

    @GetMapping
    public ResponseEntity<List<Dog>> getDogs() {
        List<Dog> dogs;

        try {
            dogs = service.getDogs();
        } catch (DogsServiceException ex) {
            return new ResponseEntity<>(null, null, HttpStatus.INTERNAL_SERVER_ERROR);
        } catch (DogsNotFoundException ex) {
            return new ResponseEntity<>(null, null, HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<>(dogs, HttpStatus.OK);
    }
}
```

https://dzone.com/articles/spring-rest-service-exception-handling-1

2. Solution 1: the Controller-Level *@ExceptionHandler*

The first solution works at the *@Controller* level. We will define a method to handle exceptions and annotate that with *@ExceptionHandler*.

```
public class FooController{

    //...
    @ExceptionHandler({ CustomException1.class, CustomException2.class })
    public void handleException() {
        //
    }
}
```

This approach has a major drawback: T**he *@ExceptionHandler* annotated method is only active for that particular Controller**, not globally for the entire application. Of course, adding this to every controller makes it not well suited for a general exception handling mechanism.

---

https://www.baeldung.com/exception-handling-for-rest-with-spring

4. Solution 3: *@ControllerAdvice*

Spring 3.2 brings support for **a global *@ExceptionHandler* with the *@ControllerAdvice* annotation.**

This enables a mechanism that breaks away from the older MVC model and makes use of *ResponseEntity* along with the type safety and flexibility of *@ExceptionHandler*.

```
@ControllerAdvice
public class RestResponseEntityExceptionHandler
  extends ResponseEntityExceptionHandler {

    @ExceptionHandler(value
      = { IllegalArgumentException.class, IllegalStateException.class })
    protected ResponseEntity<Object> handleConflict(
      RuntimeException ex, WebRequest request) {
        String bodyOfResponse = "This should be application specific";
        return handleExceptionInternal(ex, bodyOfResponse,
          new HttpHeaders(), HttpStatus.CONFLICT, request);
    }
}
```

The*@ControllerAdvice* annotation allows us to **consolidate our multiple, scattered *@ExceptionHandlers* from before into a single, global error handling component.**

---

https://www.baeldung.com/exception-handling-for-rest-with-spring

General exception handling

```
@ControllerAdvice
@Slf4j
public class DogsServiceErrorAdvice {

    @ExceptionHandler({RuntimeException.class})
    public ResponseEntity<String> handleRunTimeException(RuntimeException e) {
        return error(INTERNAL_SERVER_ERROR, e);
    }

    @ExceptionHandler({DogsNotFoundException.class})
    public ResponseEntity<String> handleNotFoundException(DogsNotFoundException e) {
        return error(NOT_FOUND, e);
    }

    @ExceptionHandler({DogsServiceException.class})
    public ResponseEntity<String> handleDogsServiceException(DogsServiceException e){
        return error(INTERNAL_SERVER_ERROR, e);
    }

    private ResponseEntity<String> error(HttpStatus status, Exception e) {
        log.error("Exception : ", e);
        return ResponseEntity.status(status).body(e.getMessage());
    }
}
```

https://dzone.com/articles/spring-rest-service-exception-handling-1

Custom Error Controller

```
@ControllerAdvice
public class ProductNotFoundAdvice {
    @ExceptionHandler
    void handleIllegalArgumentException(IllegalArgumentException e,
                                        HttpServletResponse response)
         throws IOException {

        response.sendError(HttpStatus.CONFLICT.value());
    }

    @ExceptionHandler
    void handleIllegalArgumentException(ProductNotFoundException e,
                                        HttpServletResponse response)
         throws IOException {

        response.sendError(HttpStatus.BAD_REQUEST.value());
    }
}
```

D E M O

«spring-rest-product-error-controller»

Add On Exercises 3 + 4

https://github.com/Michaeli71/ADC_BOOTCAMP_SPRING

Questions?

Links

REST

- https://entwickler.de/spring/spring-boot-tutorial-so-entwickelt-man-rest-services-mit-spring-boot/
- https://spring.io/guides/tutorials/rest/
- https://spring.io/guides/gs/rest-service/
- https://www.techiedelight.com/display-custom-error-pages-in-spring-boot/
- https://www.javadevjournal.com/rest-with-spring-series/
- https://www.javadevjournal.com/spring/exception-handling-for-rest-with-spring/
- https://auth0.com/blog/spring-data-rest-tutorial-developing-rest-apis-with-ease/

Links

EXCEPTION HANDLING

- https://reflectoring.io/spring-boot-exception-handling/
- https://www.bezkoder.com/spring-boot-restcontrolleradvice/
- https://dzone.com/articles/spring-rest-service-exception-handling-1
- https://www.codejava.net/frameworks/spring-boot/spring-boot-error-handling-guide

Thank You