REST services. Spring

Sergey Morenets

September, 29th 2018

DEVELOPER 12 YEARS

TRAINER 4 YEARS

WRITER 3 BOOKS

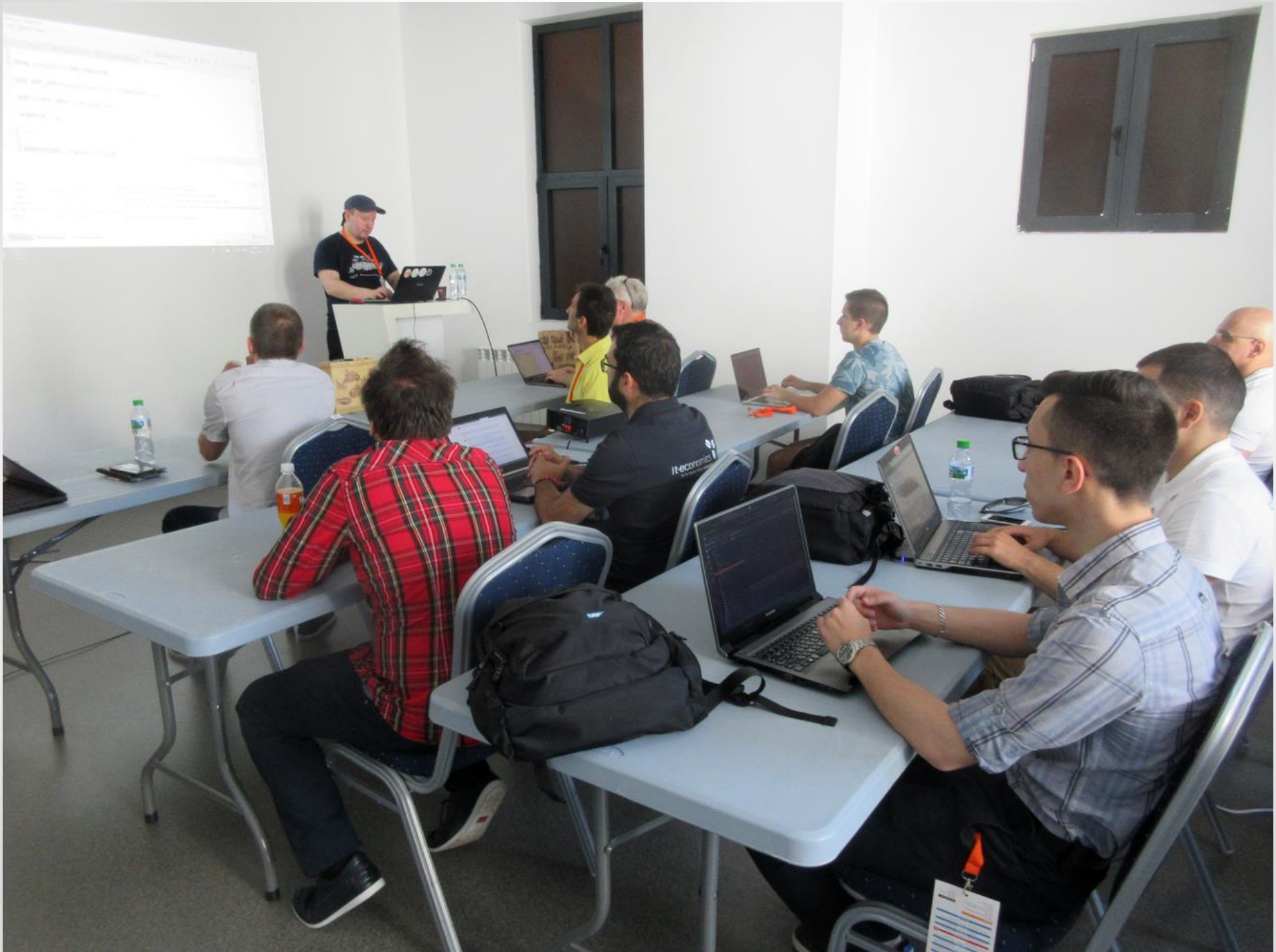
FOUNDER

ITSimulator

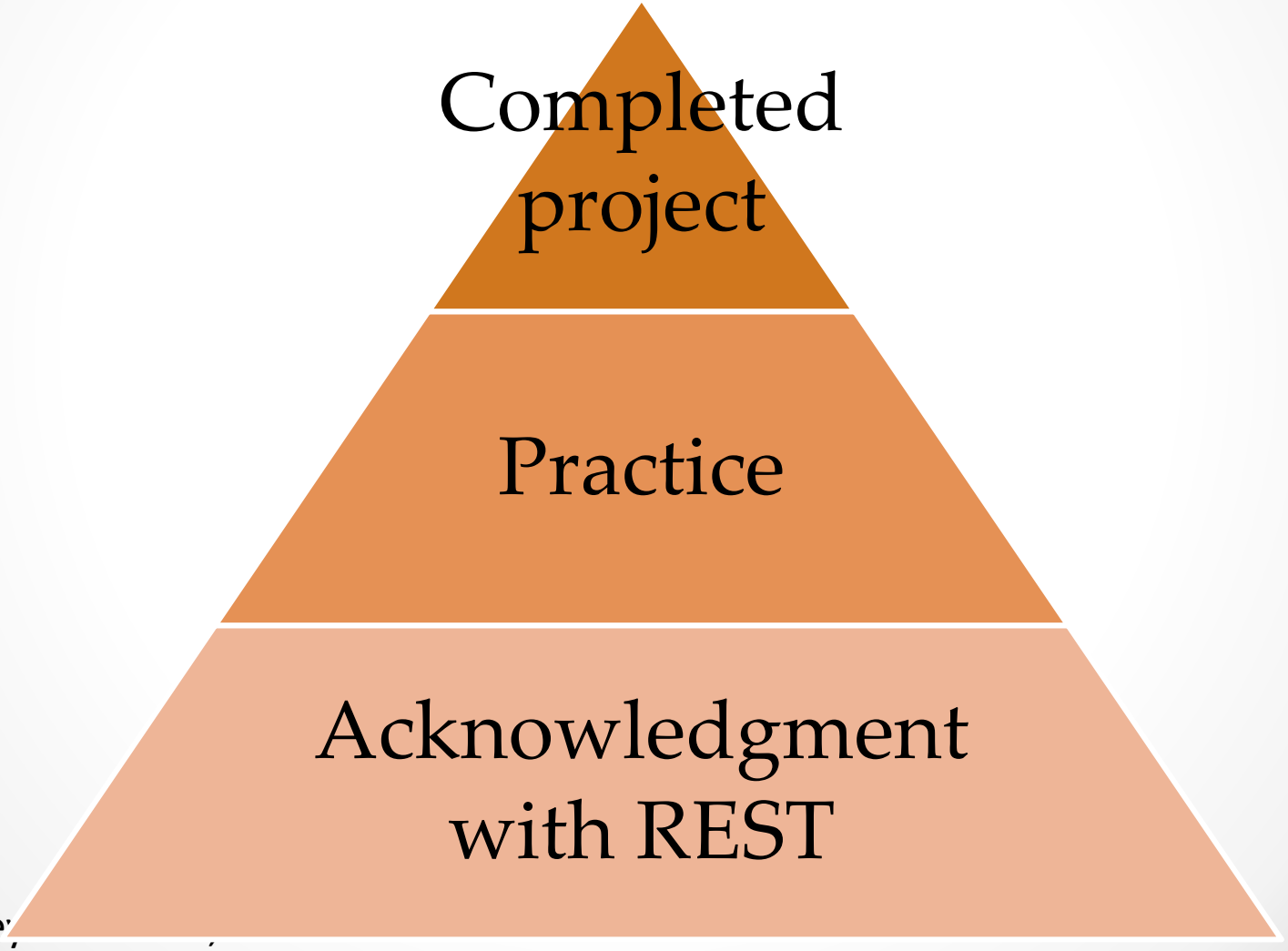
SPEAKER

Java
frameworks
day


@sergey-morenets
DEVOXX
POLAND

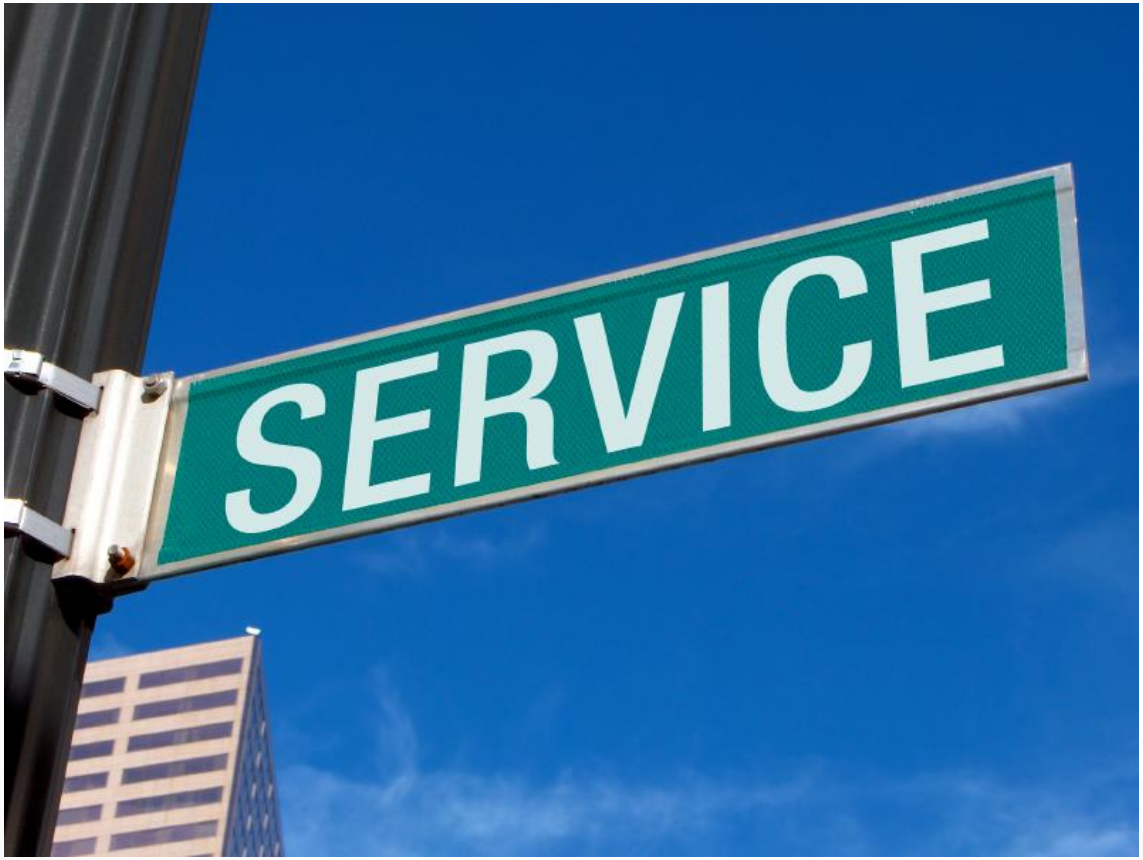

it-economics

Goals



Agenda

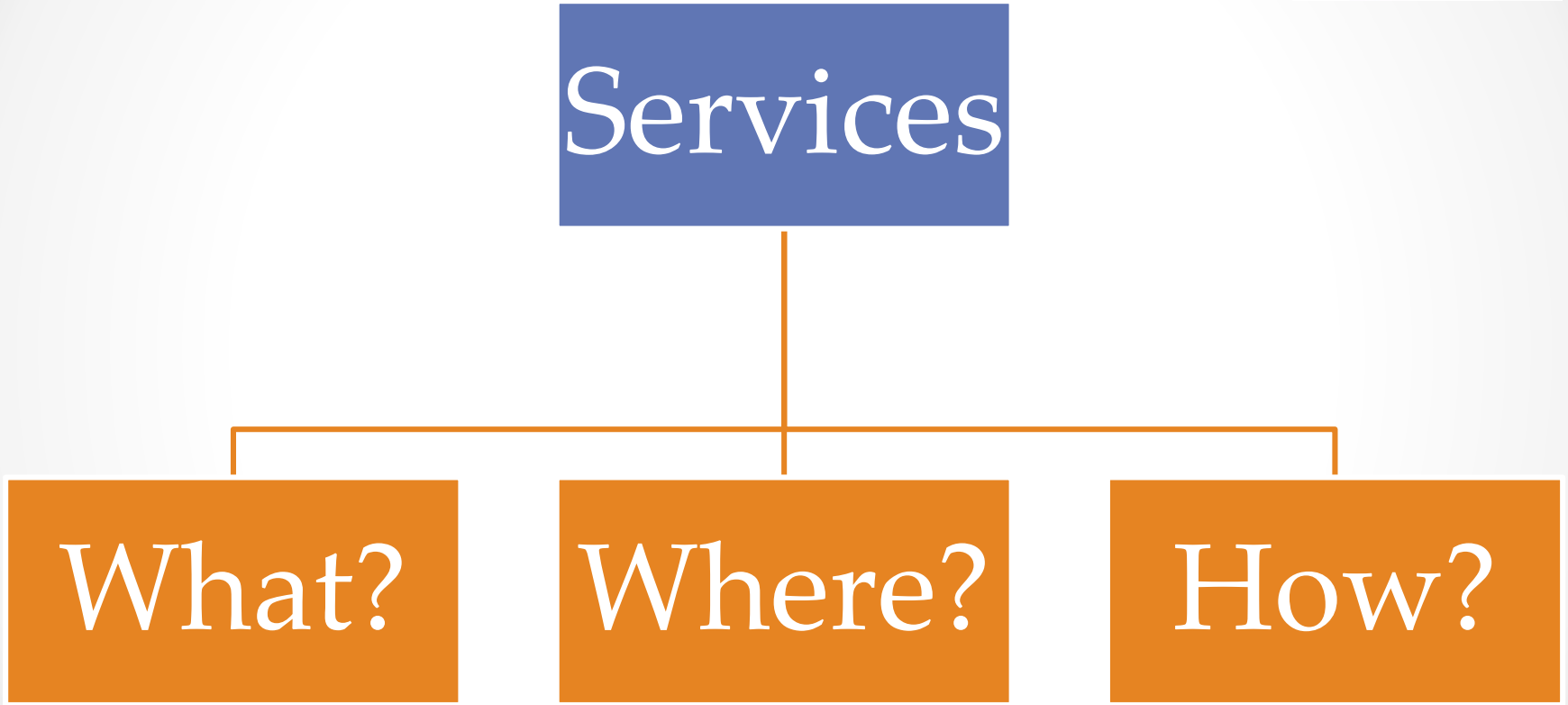
- ✓ REST and REST services
- ✓ REST over HTTP
- ✓ Spring Framework 5.1/Spring Boot 2 usage
- ✓ REST controllers
- ✓ Validation and pagination
- ✓ Service testing
- ✓ Exception handling
- ✓ HATEOAS
- ✓ Scaling and monitoring
- ✓ Caching
- ✓ Performance testing

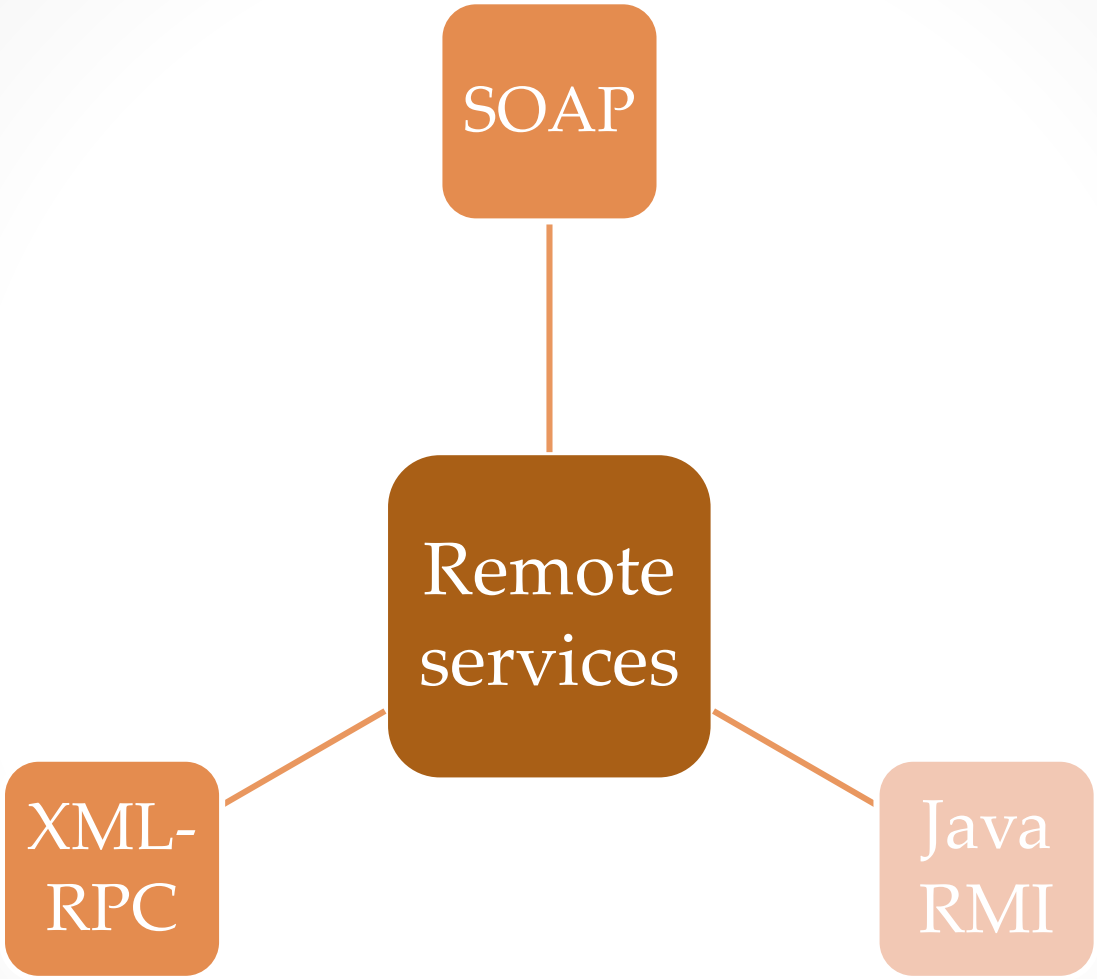


Sergey Morenets, 2018

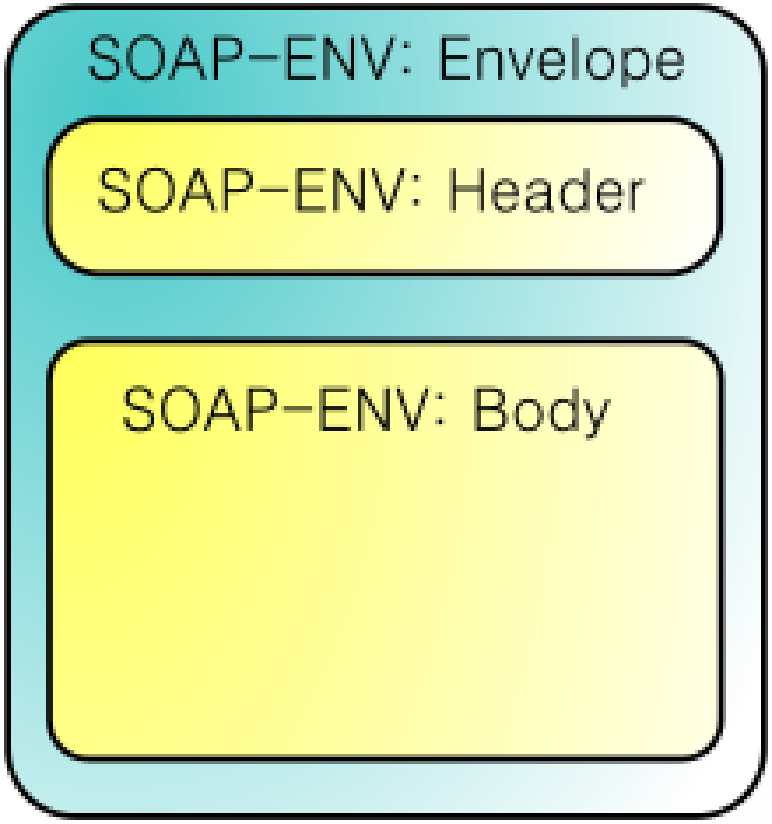
Service

- Sergey Morenets, 20


Services
What?
Where?
How?


SOAP
Remote services
XML-RPC
Java RMI

SOAP



SOAP


```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock/Surya">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

REST

- REST(Representational state transfer) is architectural style
- Introduced by Roy Fielding in 2000
- Mostly used as RESTful web services(over HTTP)

REST popularity

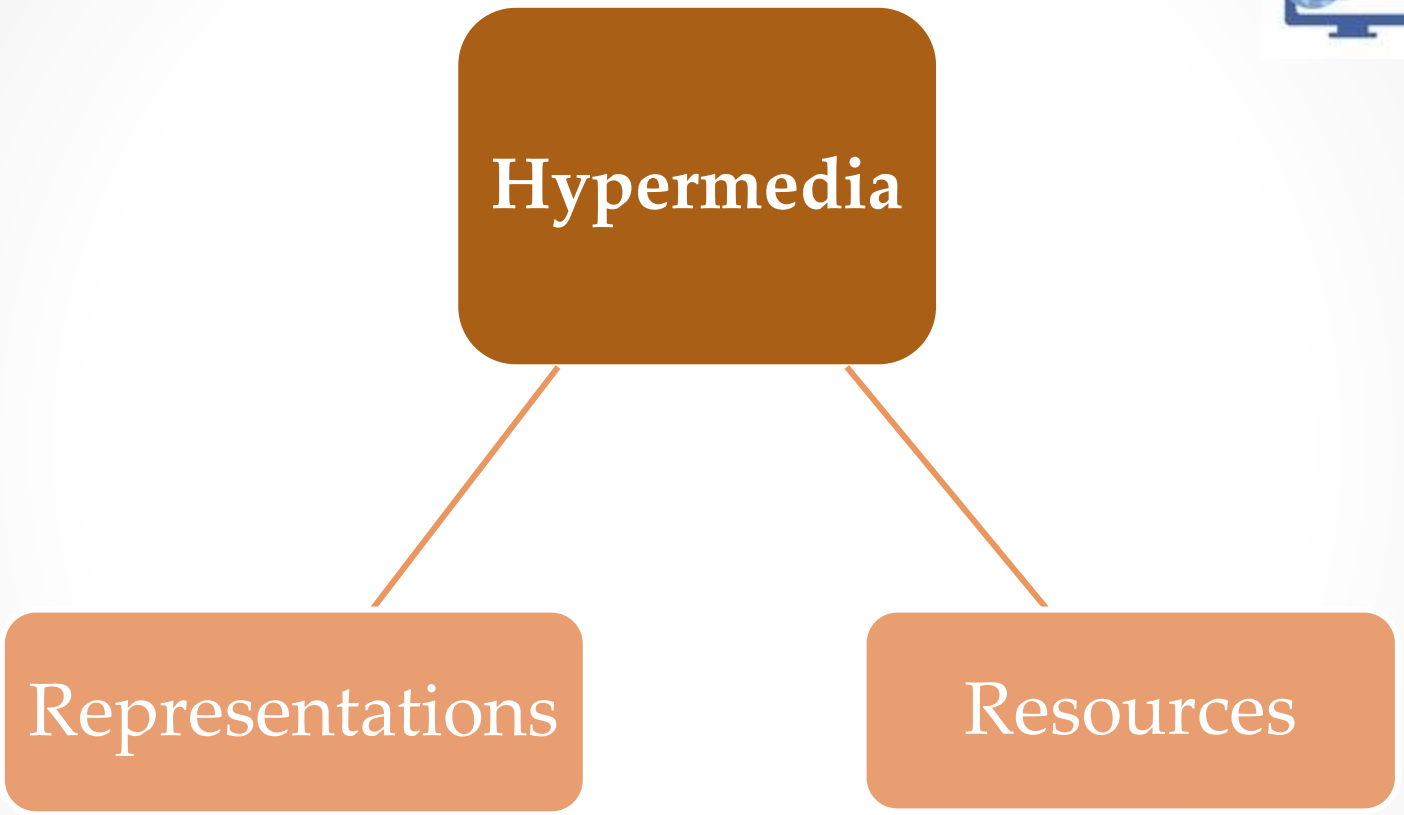
- Sergey Morenets, 2018

Constraints

- ✓ Client-server
- ✓ Stateless
- ✓ Cacheable
- ✓ Layering
- ✓ Uniform interface

REST is not

- ✓ Protocol
- ✓ File format
- ✓ Development framework


Hypermedia
Representations
Resources

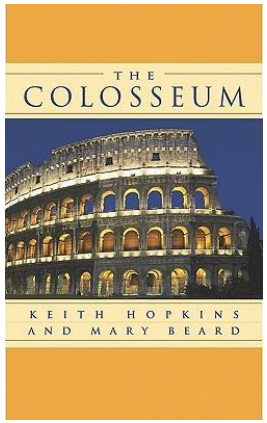
Sergey Morenets, 2018

Colosseum

From Wikipedia, the free encyclopedia

For other uses, see Colosseum (disambiguation).

The **Colosseum** or **Coliseum** (/kɒləˈsiːəm/ *kol-ə-SEE-əm*),



Sergey Morenets, 2018

Resources

- Resource is everything that can be referenced
- Every resource has unique URL

- **Samples**:
- Documents
- Tables or rows
- Files

Representations

- Representation is current state of the resource in the specific format
- Representations contains information about resource
- One resource can have multiple representations

- **Samples**:
- XML document
- JSON document
- Link to document

Representations

GET /hello

```
{
    "text":"helloWorld"
}
```

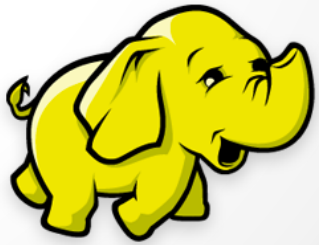
```
<document>
      <text>helloWorld</text>
</document>
```

Representations

Apache Thrift ™

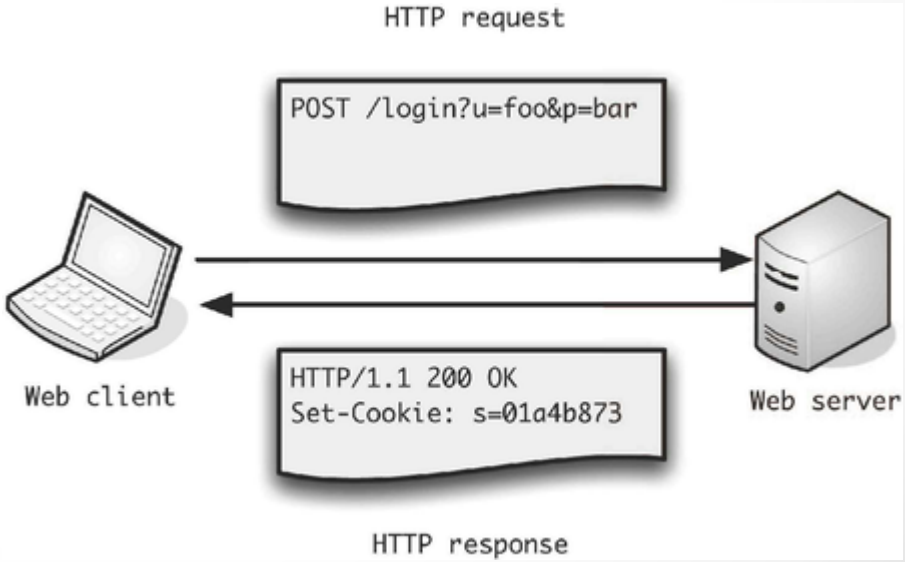
Protocol Buffers

Google



Protocol

- In RESTful systems client and server interacts using predefined protocol
- In RESTful web services protocol is **HTTP** and messages are HTTP requests/responses



Success

- ✓ SOAP/WSDL is complex and difficult
- ✓ HTTP perfectly matches for REST services due to scalability, caching and distribution
- ✓ JSON data format is best choice for light-weight/front-end clients

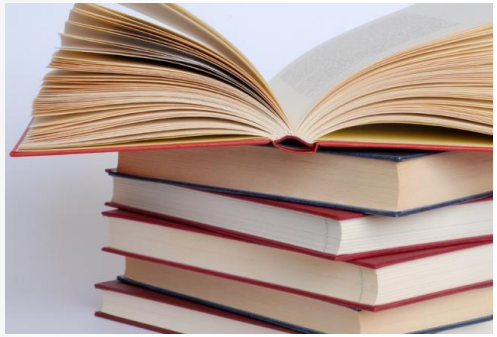
Goals

- ✓ Identify resources
- ✓ Identity endpoints
- ✓ Identifies actions
- ✓ Identify responses

Samples

GET /getAllBooks	**Return all books**
GET /books?id=1	**Return book with specified id**
GET /books/1/buy	**Buy a book**
GET /books/1/orders	**Return all orders for given book**
GET /books/sorted/name	**Return all books sorted by name**
GET /books/1	**Return book with specified id or create new book if it doesn't exist**

HTTP endpoints



/books

/cars

/phones

HTTP method

Name	Description	Sample
GET	Get representation of the resource	GET /books GET /books/1
POST	Creates new resource	POST /books
PUT	Replaces state of the resource	PUT /books/1
DELETE	Deletes resource	DELETE /books/1
HEAD	Get information about resource	HEAD /books/1
PATCH	Partial resource modification	PATCH /books/1
OPTIONS	Information about methods	OPTIONS /books

Add book

POST /books

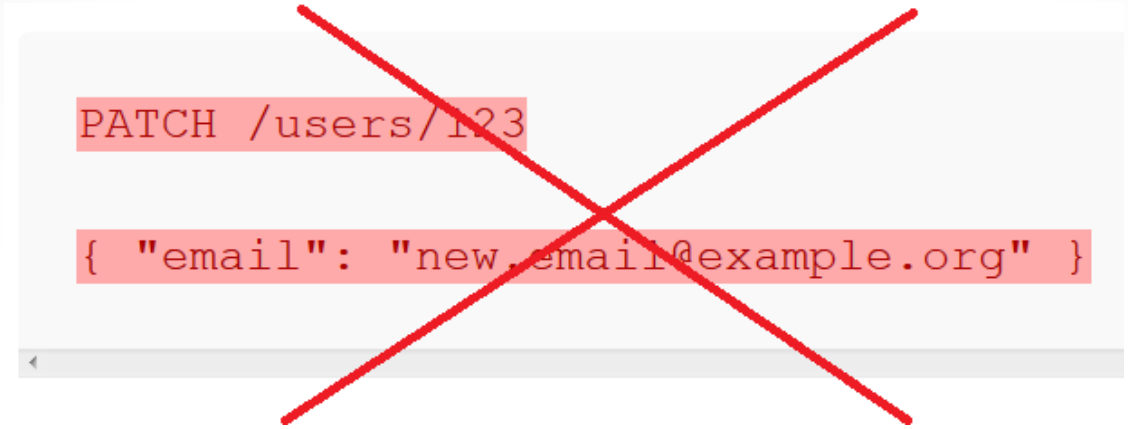
PUT /books

GET /books/add

HTTP method

- **HEAD**, **GET** and **OPTIONS** are safe methods
- **HEAD** and **GET** methods may be cacheable
- **POST** method should return location of the created resource
- **PATCH** request should contain list of instructions to modify resource

PATCH




```
PATCH /users/123

[
    { "op": "replace", "path": "/email", "value": "new.email@example.org"
]
```

- Sergey Morenets, 2018

Google

502. That's an error.

The server encountered a temporary error and could not complete your request.

Please try again in 30 seconds. That's all we know.

HTTP status codes

Range	Category	Explanation
100-199	Informational	Request was processed
200-299	Success	Request was accepted and handles successfully
300-399	Continuation	Additional request is required
400-499	Corrections	Client must change his request
500-599	Failures	Server was unable to handle request

HTTP method

Name	Codes	Explanation
GET	200	OK
POST	201	Resource created with its location provided
	202	Request is accepted and resource will be created later
PUT	200	OK
	204	No content is provided
DELETE	204	Resource is deleted and no content is available
	202	Resource will be deleted later
	404	Resource couldn't be found

Task #1. Reviewing REST services

1. Use browser, **Postman** or any other REST client
2. Open http://jsonplaceholder.typicode.com/
3. Try all supported **HTTP** methods
4. Pay attention to response **body** and **headers**

How does client **know** ?

- ✓ request URL
- ✓ request method
- ✓ response content type
- ✓ query parameters
- ✓ if request body is optional
- ✓ format of request body

Hypermedia

- ✓ Strategy implemented in different technologies
- ✓ Defines how server tells client about supported actions
- ✓ Reduces usability issues of Web APIs

Hypermedia

Create

Name	Email Address	Mobile Number	Action
Lisa	lisa@gmail.com	98763123	Read Update Delete
Tom	tom@gmail.com	93813412	Read Update Delete

- Sergey Morenets, 2018

Hypermedia


```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
   <account_number>12345</account_number>
   <balance currency="usd">100.00</balance>
   <link rel="deposit" href="http://somebank.org/account/12345/deposit" />
   <link rel="withdraw" href="http://somebank.org/account/12345/withdraw" />
   <link rel="transfer" href="http://somebank.org/account/12345/transfer" />
   <link rel="close" href="http://somebank.org/account/12345/close" />
 </account>
```

Hypermedia


```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">-25.00</balance>
    <link rel="deposit" href="http://somebank.org/account/12345/deposit" />
</account>
```

Hypermedia

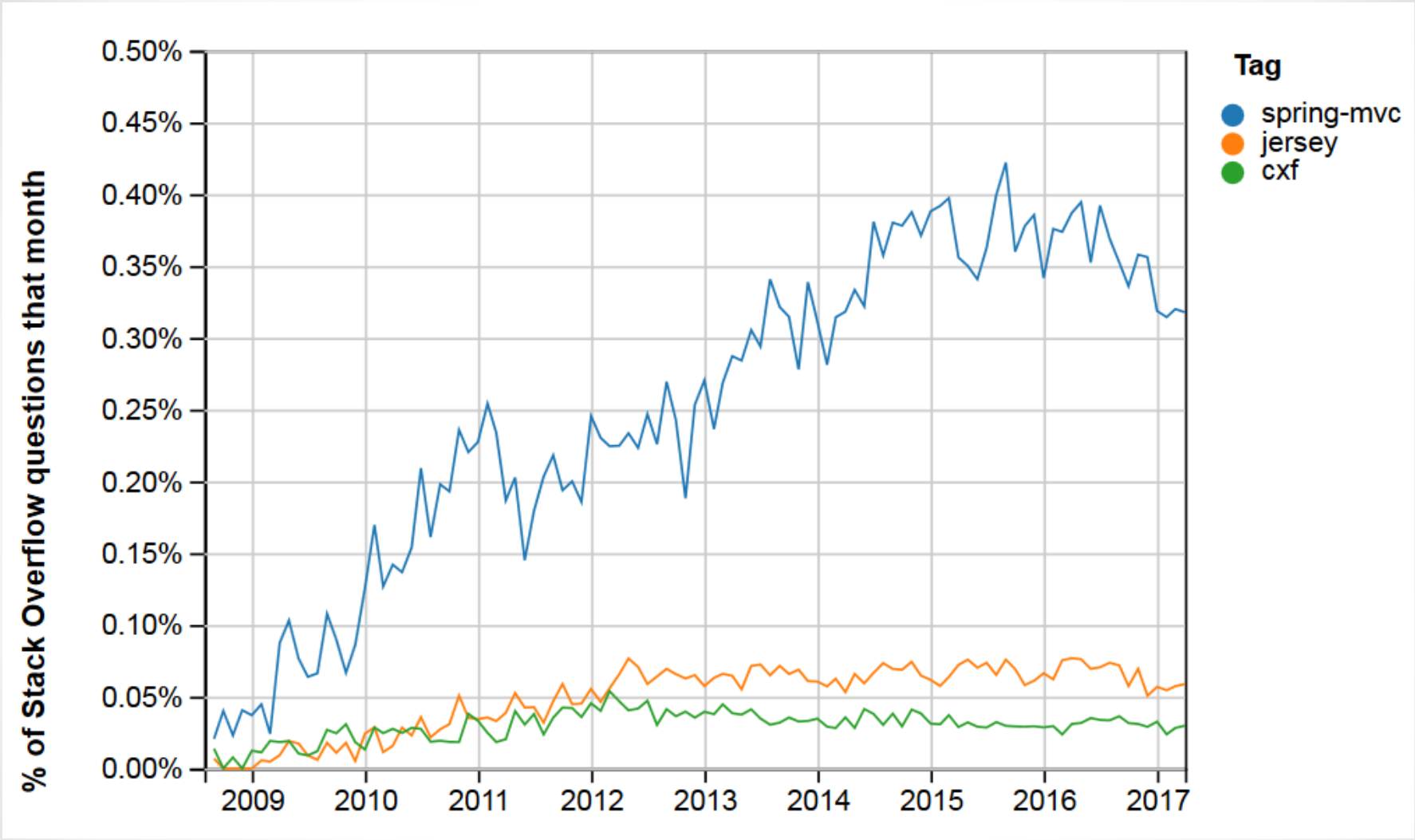

```
{
  "links": [
  {
    "href": "https://api.paypal.com/v1/payments/payment/PAY-1B569607296042 35TKQQIYVY
",
    "rel": "self",
    "method": "GET"
  },
  {
    "href": "https://api.paypal.com/v1/payments/payment/PAY-1B569607296042 35TKQQIYVY
/execute",
    "rel": "execute",
    "method": "POST"
  }]
 }
```

- Sergey Morenets, 2018

Implementations

- Spring MVC
- **JAX-RS** implementations:
- *Apache CXF*
- *Jersey*
- *RestEasy*
- *Restlet*

- Ratpack
- Spark

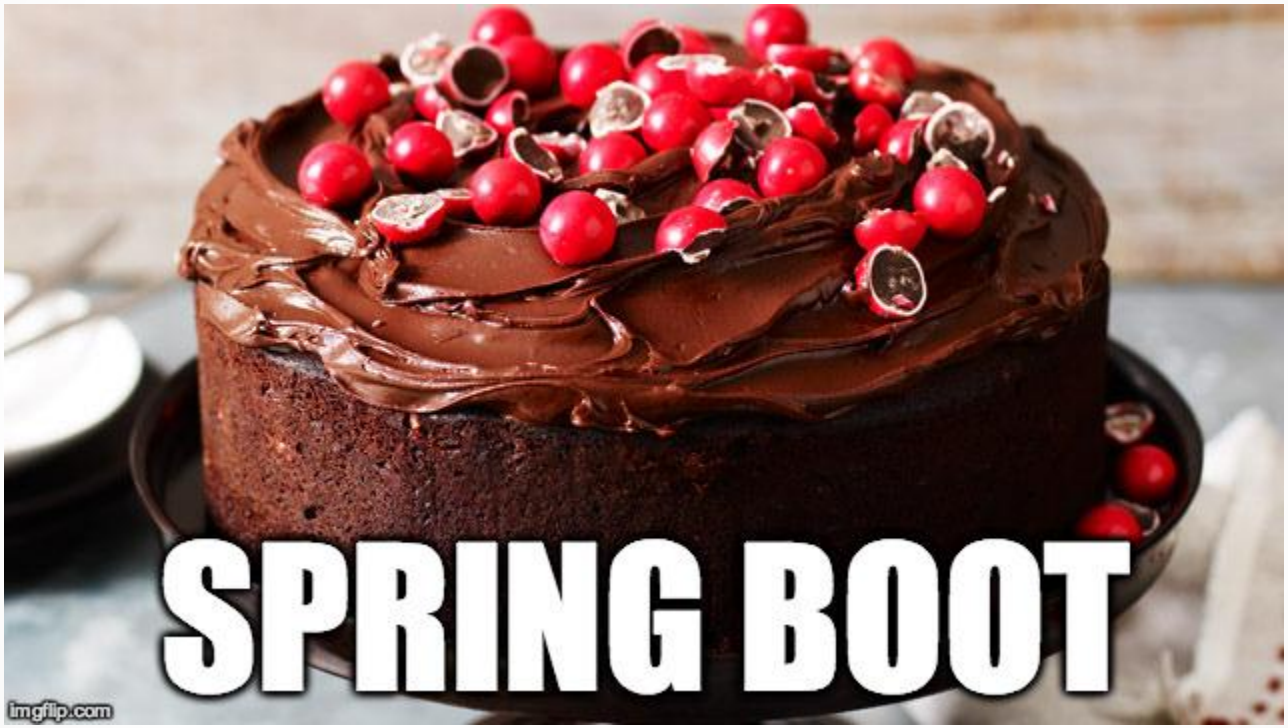


- Sergey Morenets, 2018

Spring Framework

- Analog to **EJB**
- Developed by **SpringSource**(now Pivotal)
- First milestone release in 2003
- **1.0** released in 2004 with Hibernate support
- **2.0** released in 2006
- **2.5** introduced annotations in 2007
- **3.0** released in 2009
- **4.0** supports Java 8, Groovy 2 and Java EE7
- **5.0** includes **Reactive Streams (**Spring WebFlux**) and Junit 5/Java 9** support

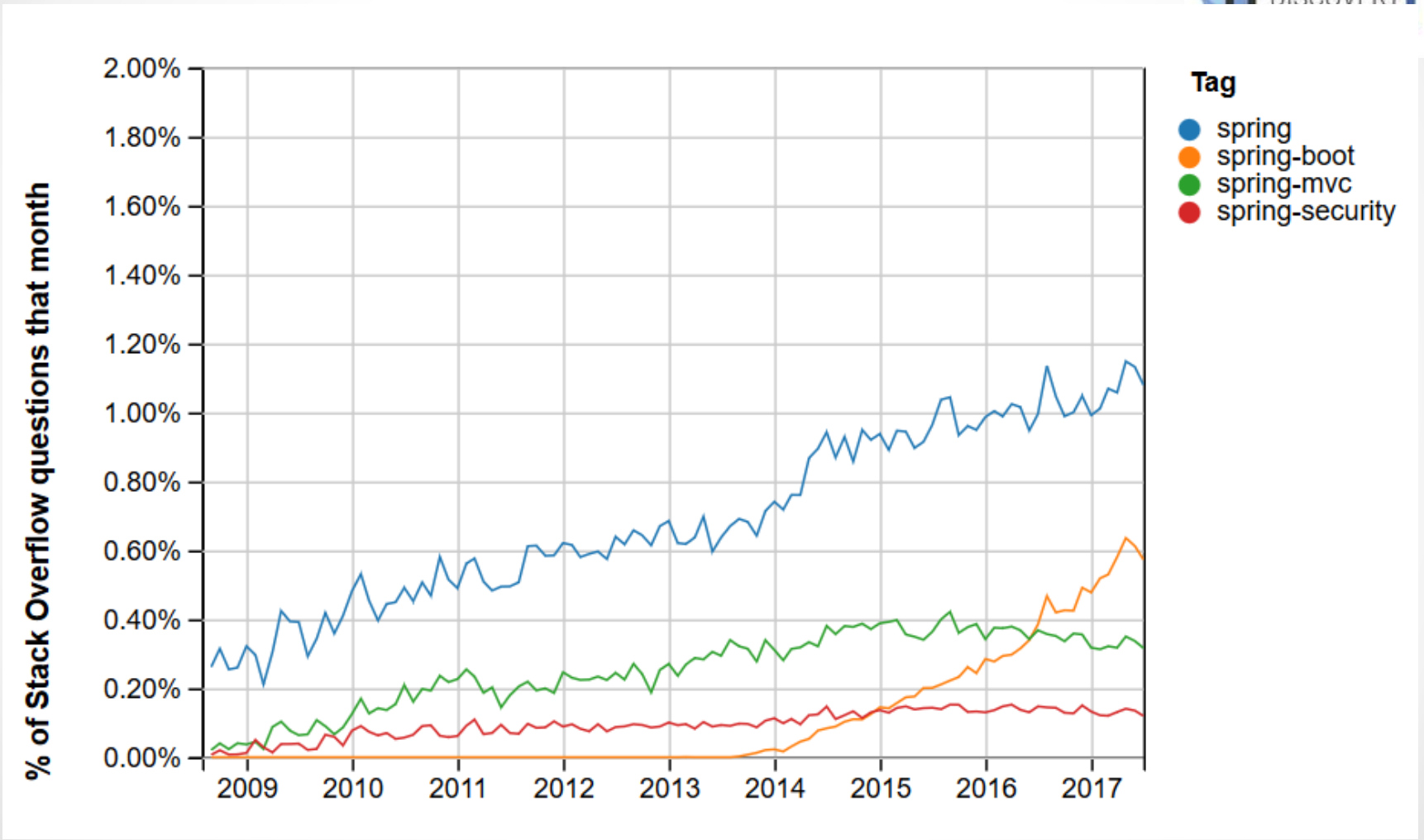
- Sergey Morenets, 2018



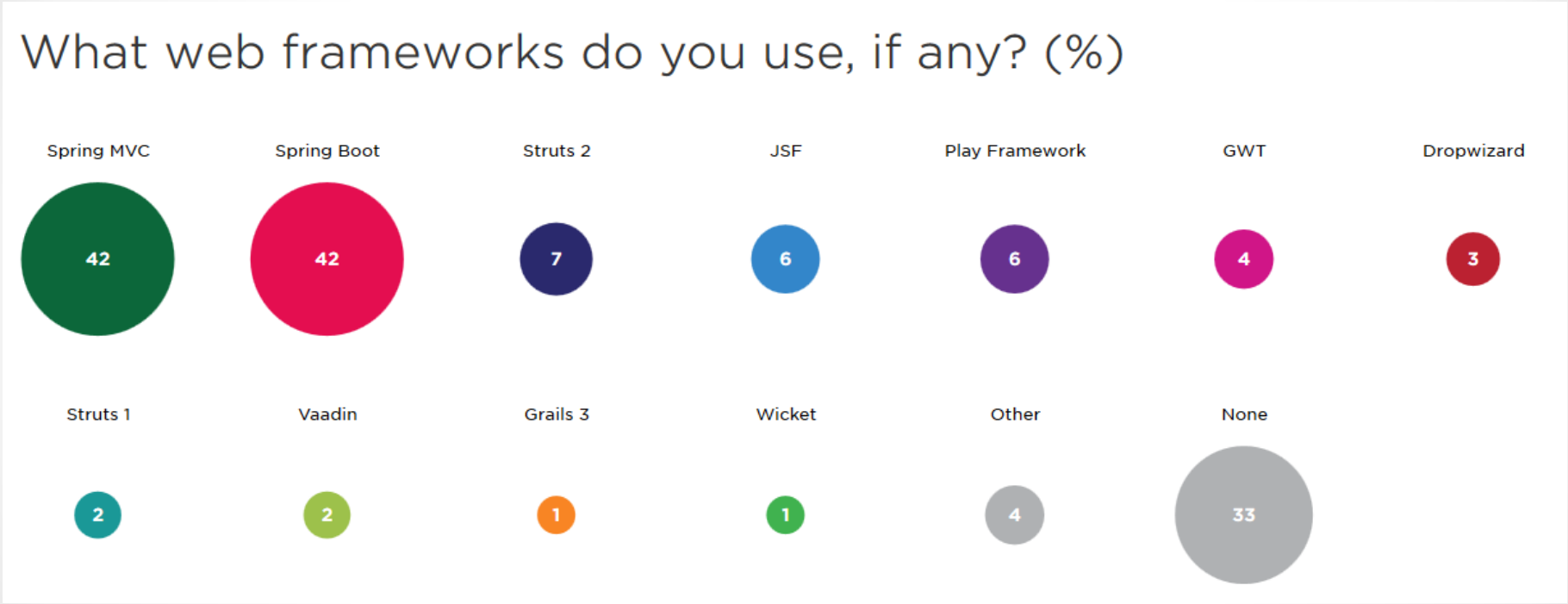
- Sergey Morenets, 2018

Spring Boot

- Stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly
- Automatically Spring configuration
- Convention-over-configuration
- Absolutely no code generation and no requirement for XML configuration
- Focus on business features and less on infrastructure



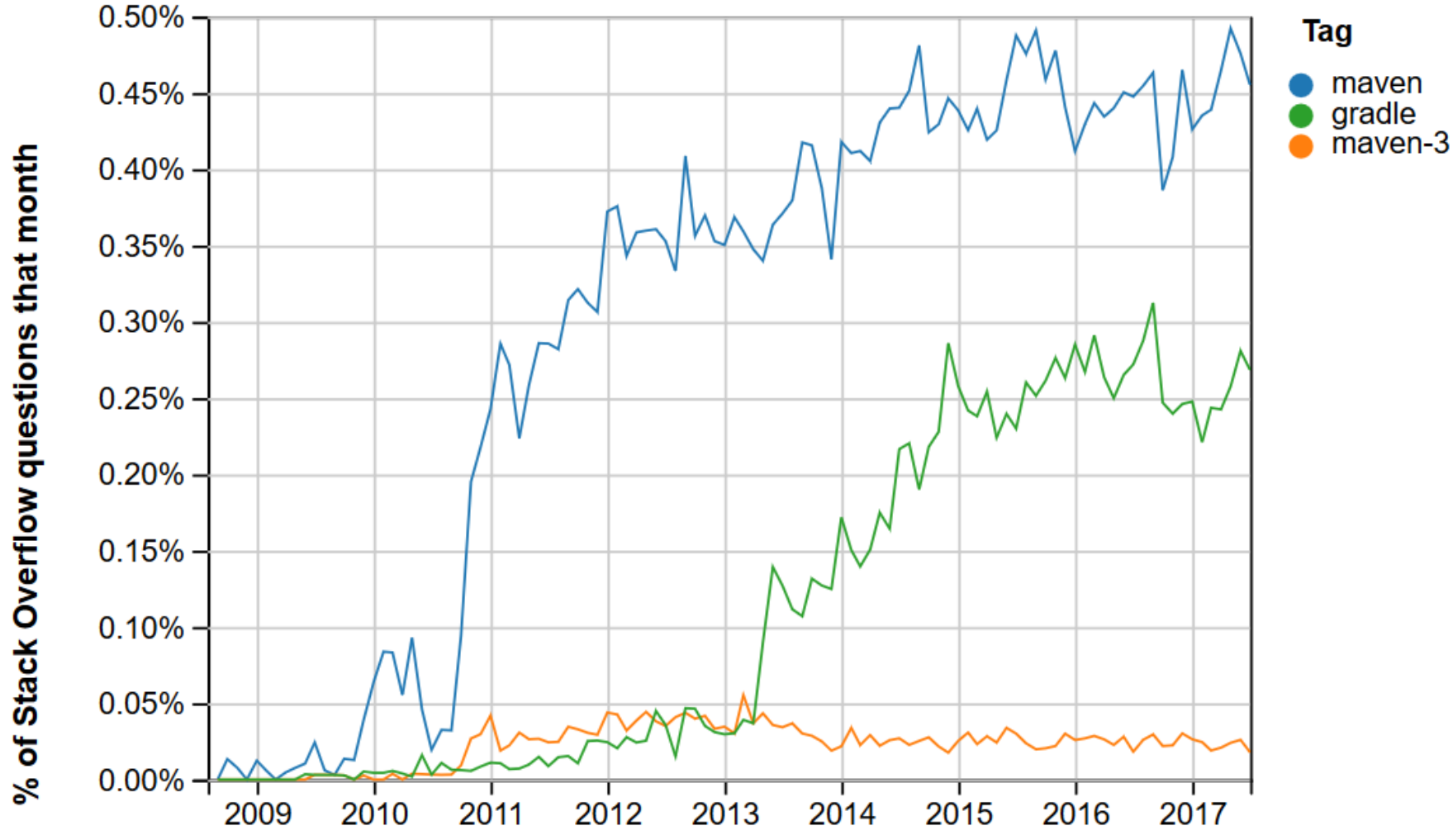
- Sergey Morenets, 2018



- Sergey Morenets, 2018

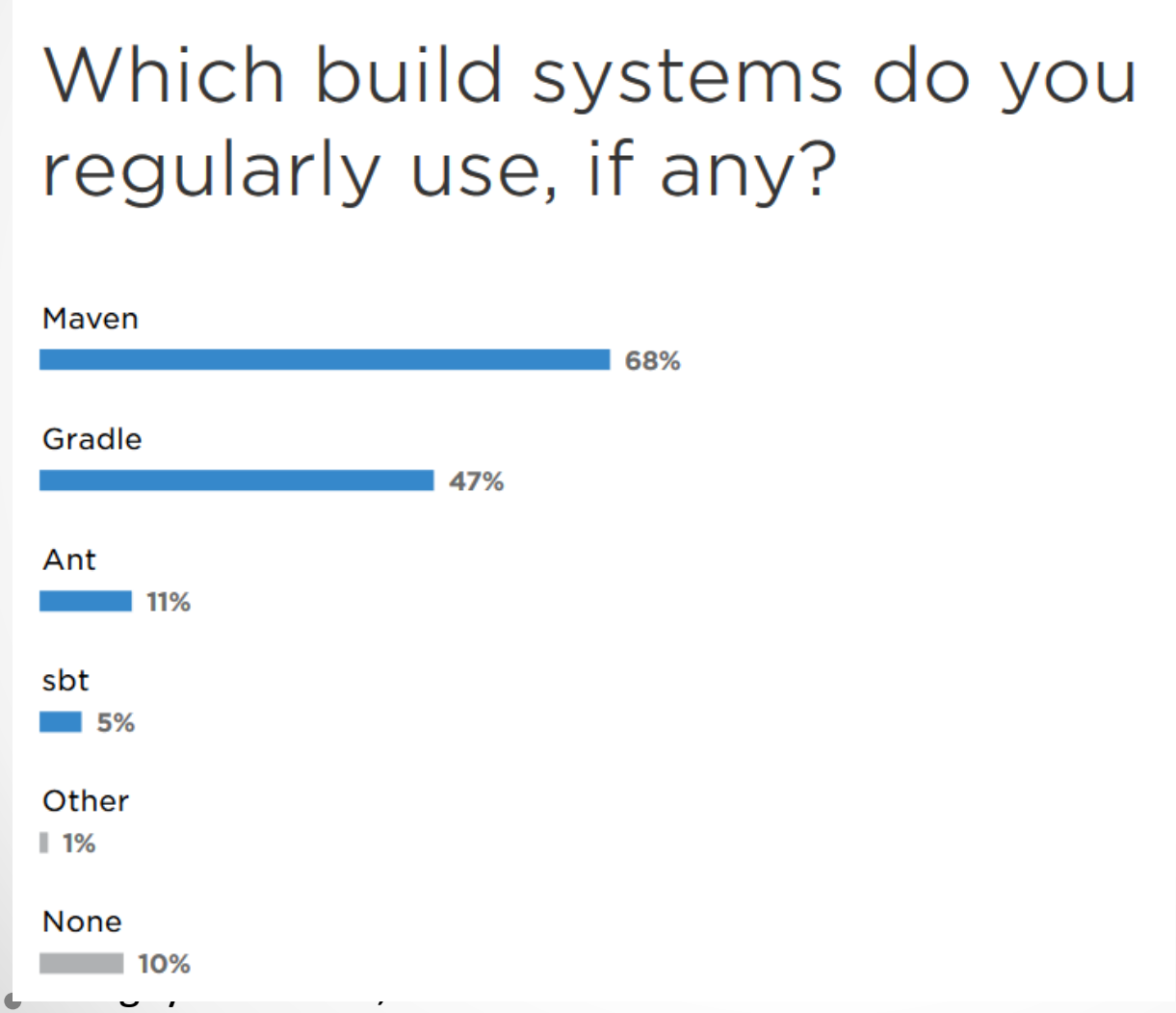
Build management

Maven vs Gradle

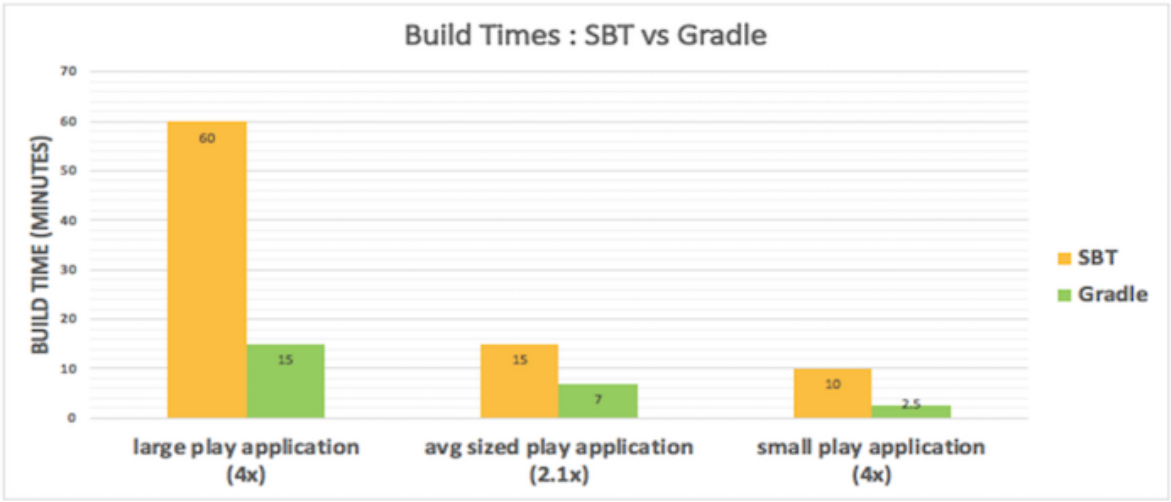


Sergey Morenets, 2018

Maven vs Gradle



Gradle vs SBT

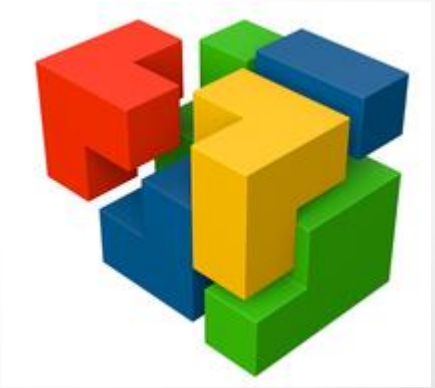


https://engineering.linkedin.com/blog/2018/07/how-we-improved-build-time-by-400-percent

- Sergey Morenets, 2018

Starters concept

- Aggregate modules
- Trusted versions of libraries
- Similar to micro-service architecture



Starters concept

- spring-boot-starter-actuator
- spring-boot-starter-data-jpa
- spring-boot-starter-jdbc
- spring-boot-starter-jersey
- spring-boot-starter-logging
- spring-boot-starter-mobile
- spring-boot-starter-redis
- spring-boot-starter-test
- spring-boot-starter-web

Spring Boot 2

- Released in February 2018
- Based on Java 8 with Java 9 support
- Dropped support for Hibernate 5.1 and earlier, Tomcat 8.x and earlier
- Security configuration by default
- Reactive modules starters
- Actuator supports Spring MVC/Jersey/WebFlux
- HTTP/2 and embedded Netty support
- Micrometer-based metrics

Spring REST

- Based on **Spring MVC**
- Used Dependency Injection(**DI**)
- Integrated with Spring Data(**Spring Data REST**) and Spring Security

Steps

- ✓ Add Spring dependencies
- ✓ Define business domain
- ✓ Define RESTful services
- ✓ Write unit-tests

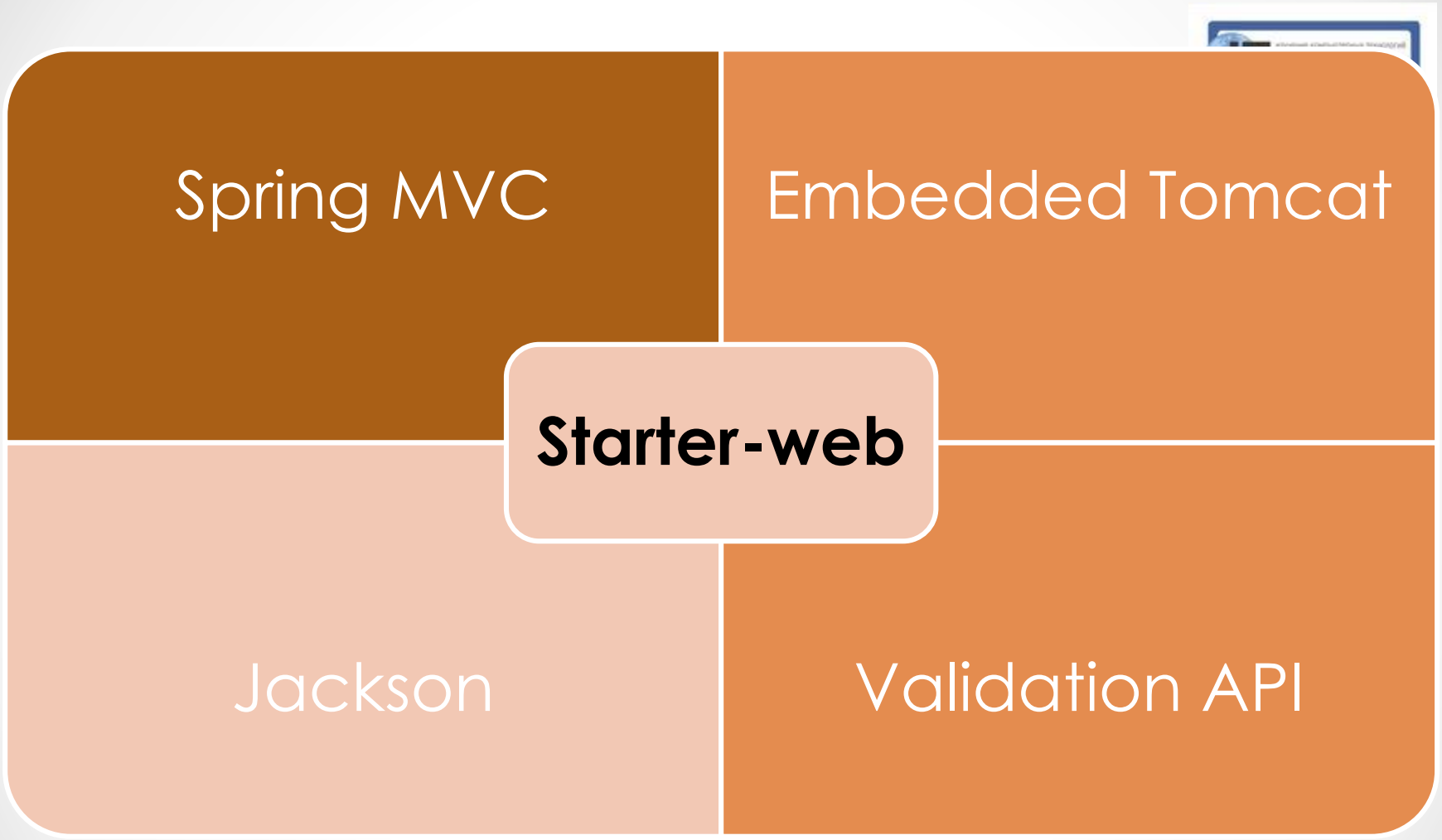
Build dependencies


```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>${spring.boot.version}</version>
</dependency>
```

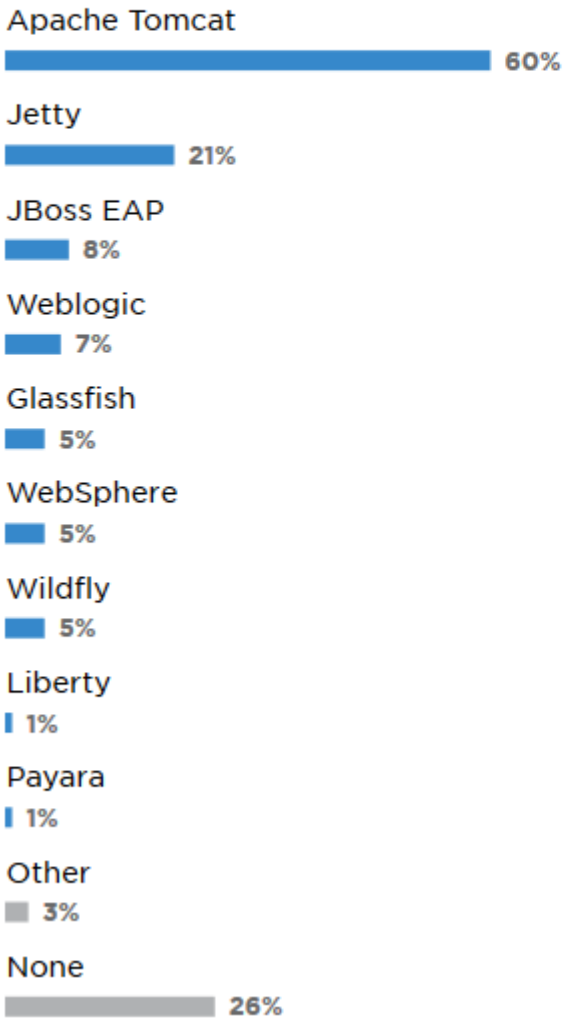
Maven

```
compile("org.springframework.boot:spring-boot-starter-web:" +
        "${springBootVersion}")
```

Gradle


Spring MVC
Embedded Tomcat
Starter-web
Jackson
Validation API

What application servers do you regularly use, if any?

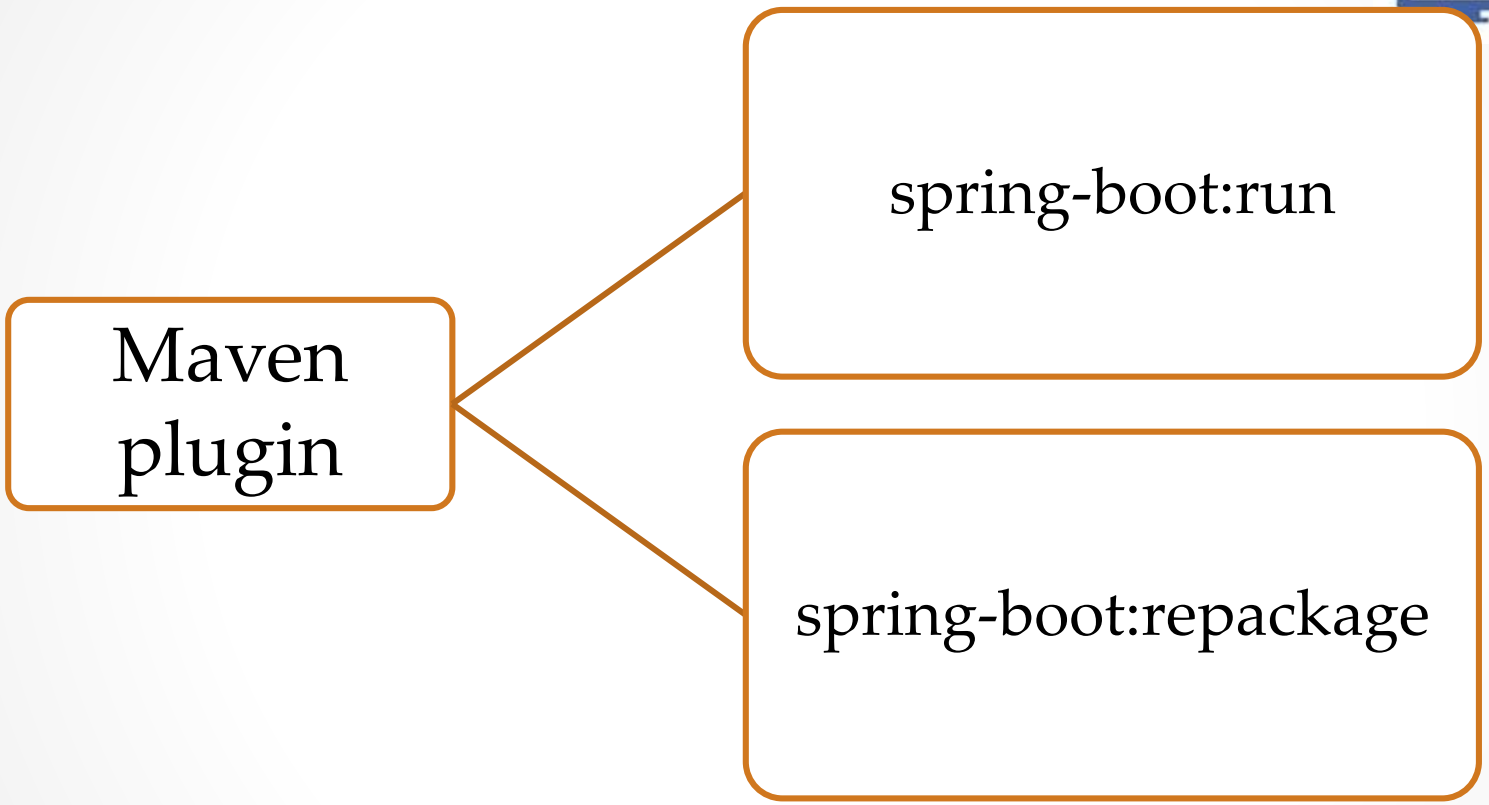


Spring configuration


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

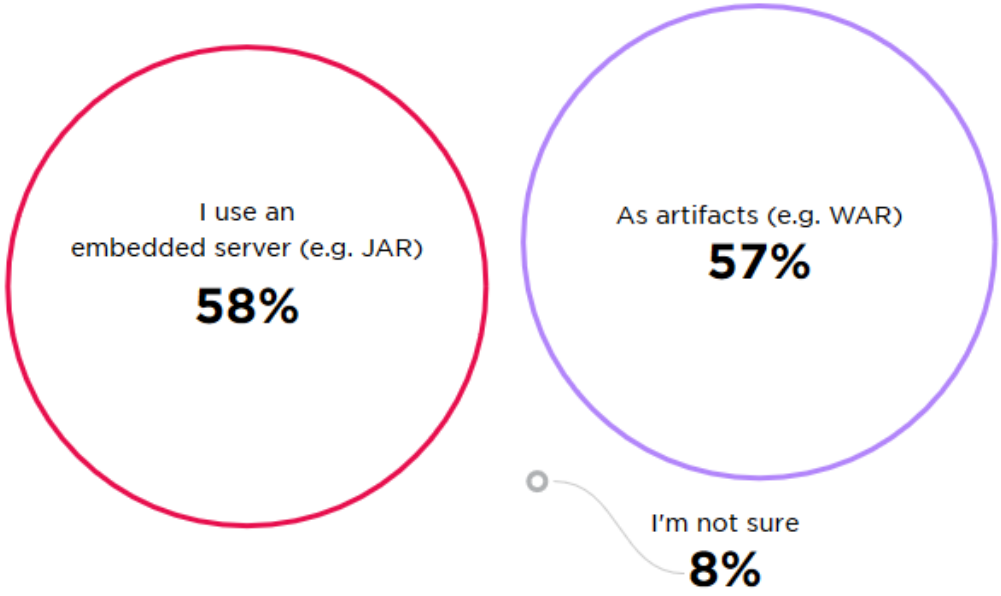
Spring Boot. Maven plugin


```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <version>${spring.boot.version}</version>
    <executions>
        <execution>
            <goals>
                <goal>repackage</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```


Maven plugin
spring-boot:run
spring-boot:repackage

Application packaging

How do you package your web applications?



- Sergey Morenets, 2018

Dependency management. Maven

- ✓ Default compiler level/source encoding
- ✓ Common dependencies management
- ✓ Resource filtering and plugin configurations

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.5.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
</parent>
```

Dependency management. Maven


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

```
<properties>
    <activemq.version>5.15.3</activemq.version>
    <antlr2.version>2.7.7</antlr2.version>
    <appengine-sdk.version>1.9.62</appengine-sdk.vers
    <artemis.version>2.4.0</artemis.version>
    <aspectj.version>1.8.13</aspectj.version>
    <assertj.version>3.9.1</assertj.version>
    <atomikos.version>4.0.6</atomikos.version>
    <bitronix.version>2.1.4</bitronix.version>
    <build-helper-maven-plugin.version>3.0.0</build-h
    <byte-buddy.version>1.7.10</byte-buddy.version>
    <caffeine.version>2.6.2</caffeine.version>
```

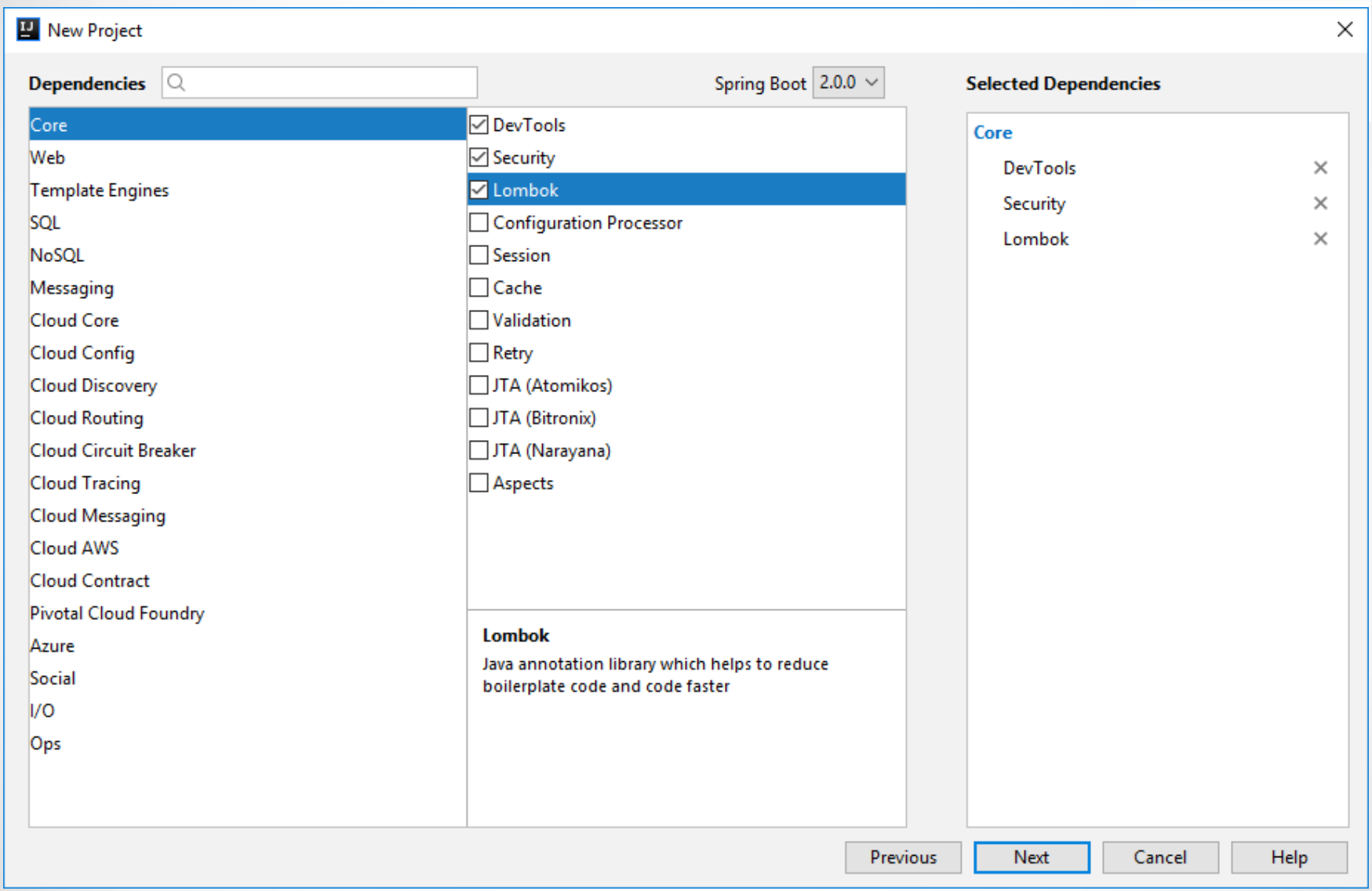
spring-boot-dependencies.pom.xml

Dependency management. Gradle


```
plugins {
    id 'org.springframework.boot' version '2.0.0.RELEASE'
}
```

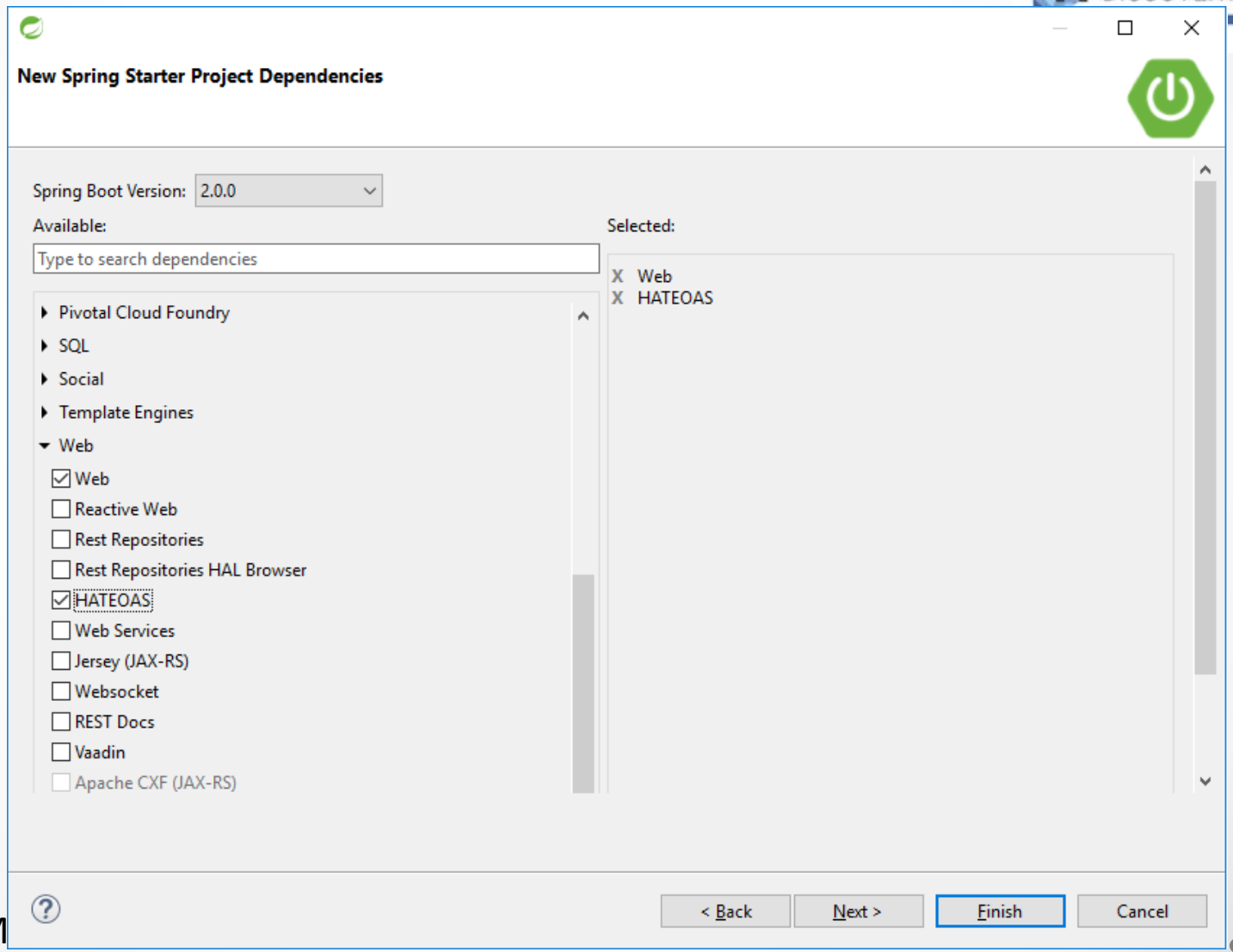
```
dependencies {
    compile 'org.springframework.boot:spring-boot-starter-web'
    compile 'org.apache.commons:commons-lang3'
}
```

IDE



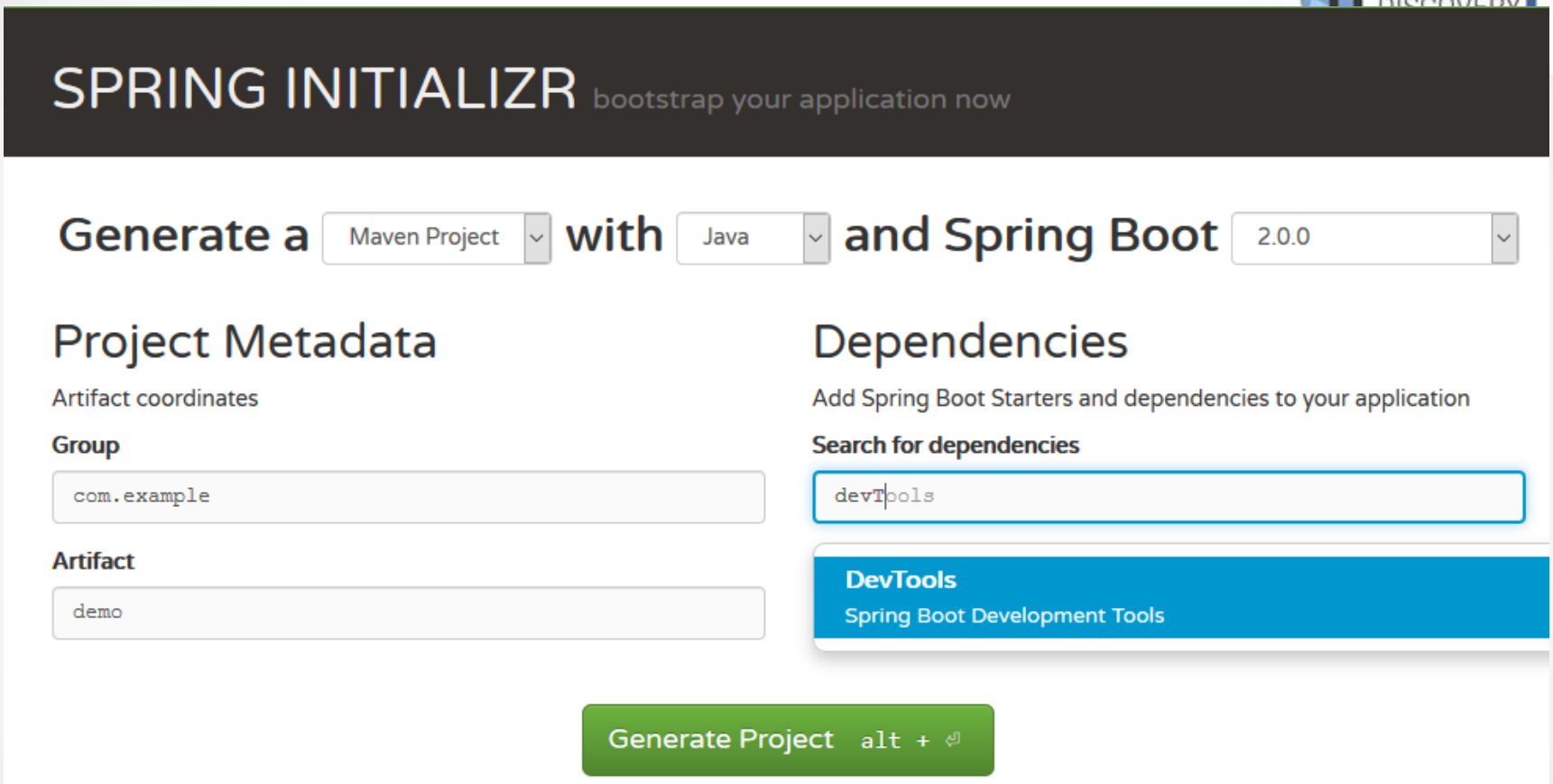
- Sergey Morenets, 2018

Spring Boot Starter. STS



- Sergey M

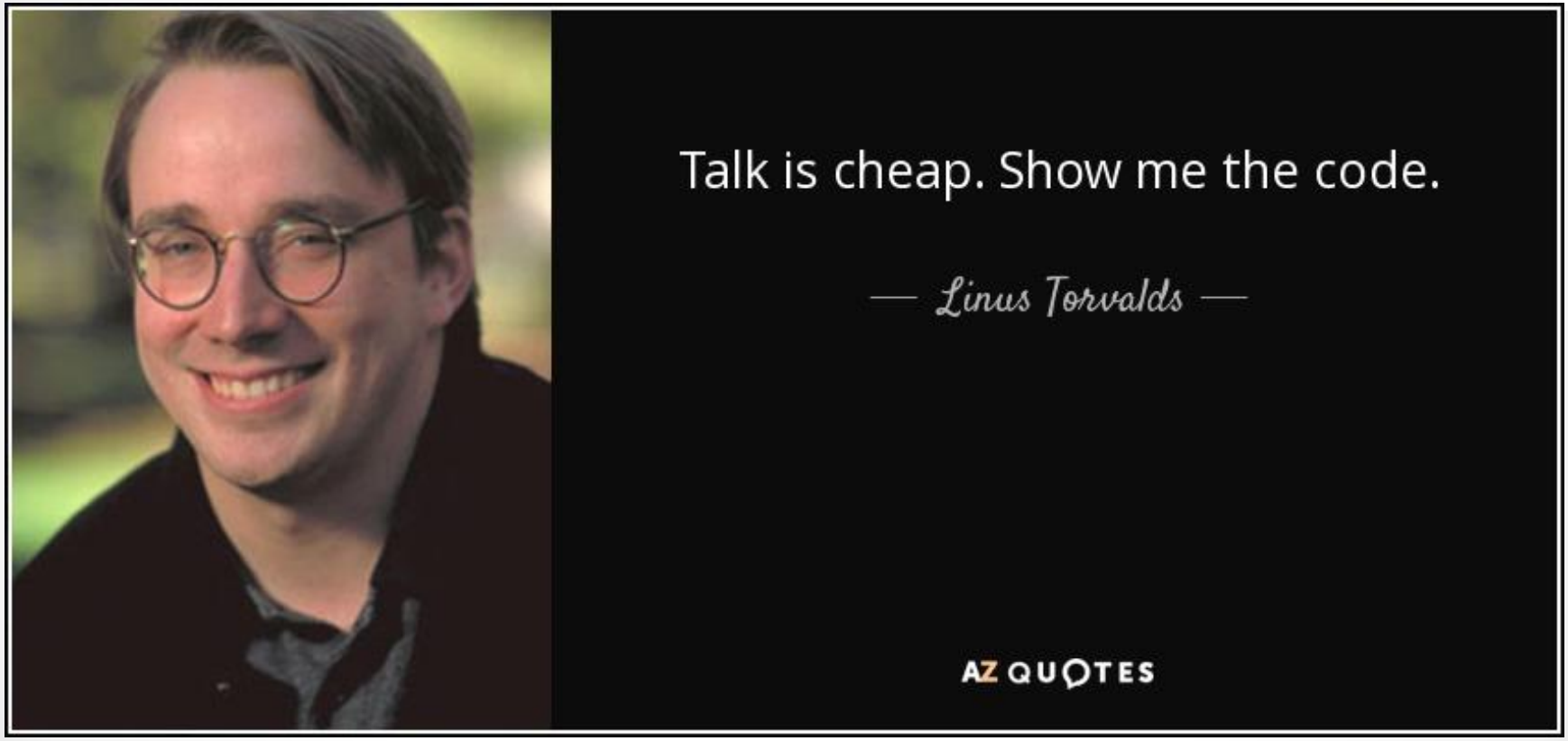
Spring Initializr



https://start.spring.io/

Task 2. Spring Boot and IDE support

1. Create Spring Boot project using **Intellij Idea**
2. Create Spring Boot project using **STS**
3. Create Spring Boot project using http://start.spring.io and open it in IDE
4. **Review** project configuration/contents

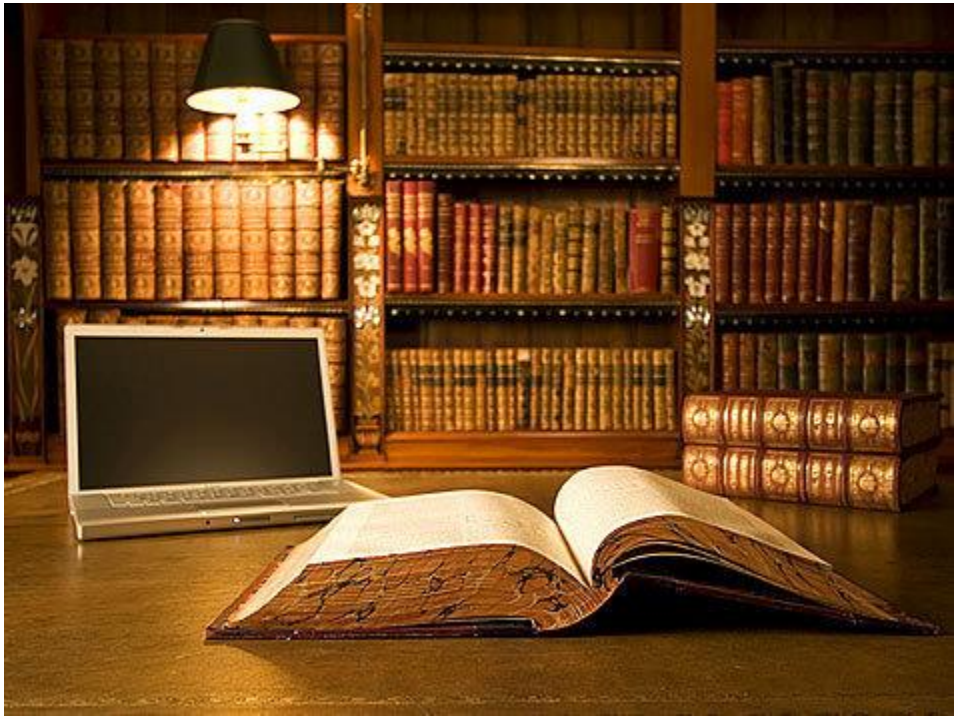


Sergey Morenets, 2018

Task 3. Spring Boot application

1. Import rest-training project into your IDE and open **rest-task3** sub-project
2. Write **RestApplication** bootstrap class
3. Run **RestApplication** and make sure you can open http://localhost:8080 page
4. Review application logs and Spring configuration

Business domain



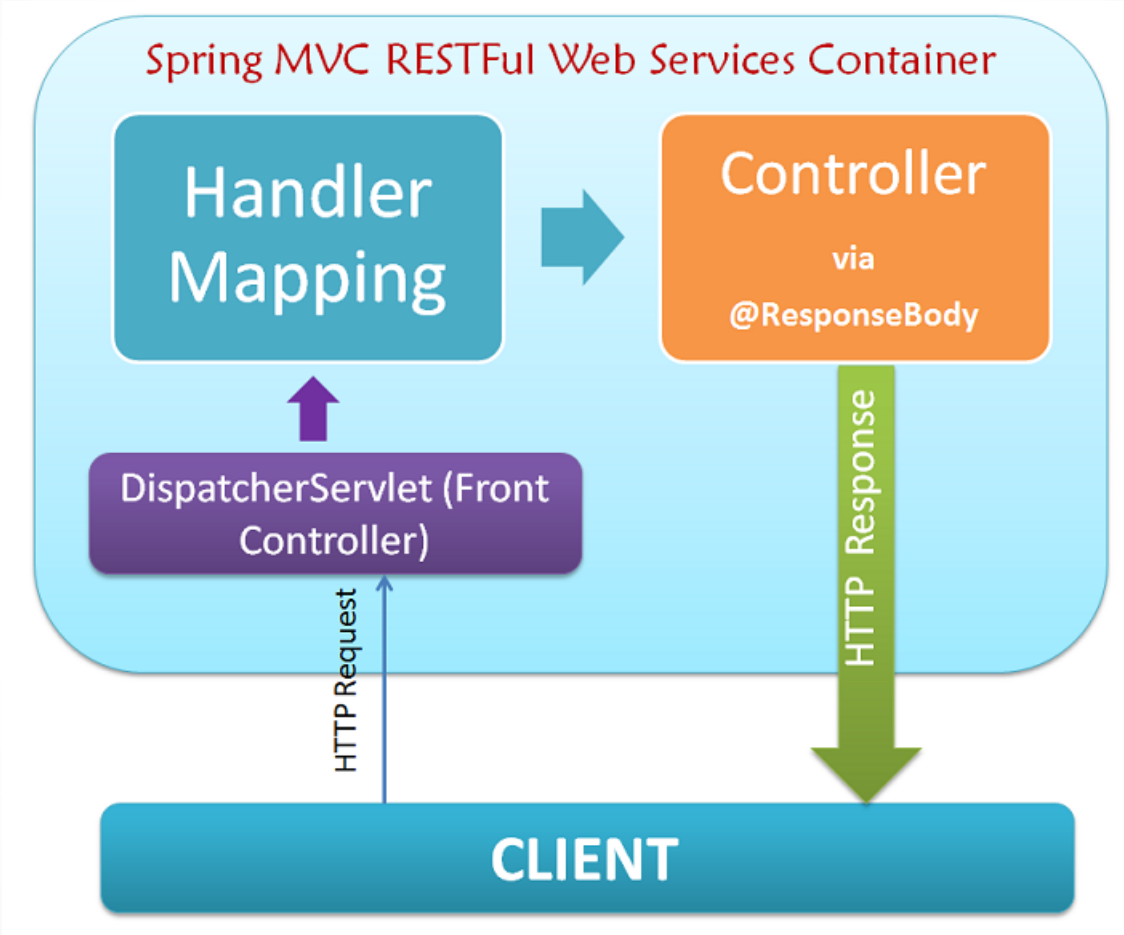
- Sergey Morenets, 2018

Business domain


```
public class Book {
    private int id;

    private String author;

    private String name;

    private int year;
```

Spring MVC. Controllers


```
@Controller
@RequestMapping("/hello")
public class HelloWorldController {

    @RequestMapping
    public String hello() {
        return "hello";
    }

}
```



- Sergey Morenets, 2018

Spring MVC. Controllers


```
@Controller
@RequestMapping("/hello")
public class HelloWorldController {

    @ResponseBody
    @RequestMapping
    public String hello() {
        return "hello";
    }

}
```

GET /hello

Spring MVC


```
@Controller
@RequestMapping("/hello")
public class HelloWorldController {

    @RequestMapping("/world")
    @ResponseBody
    public String hello() {
        return "hello";
    }

}
```

GET /hello/world

Spring Boot. Dev tools

- ✓ Automatic restart when file(s) on a classpath changes
- ✓ **LiveReload** server support
- ✓ Remote application support
- ✓ Dev customization by default

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <version>${spring.boot.version}</version>
    <optional>true</optional>
</dependency>
```

Task 4. First REST service

1. Write two **REST services** that return data in text format
2. First service returns **current date**
3. Second service returns **current time**
4. Check how **dev-tools** technology is working

Mapping annotations

Annotation	Description
@Controller	Indicates MVC controller
@RestController	Indicates MVC controller for REST operations
@RequestMapping	Maps web requests to the controller classes/methods
@PathVariable	Maps method parameter to URL template variables
@RequestParam	Binds method parameter to request parameter
@ResponseBody	Value returned by controller is bound to the response body
@ResponseStatus	Change response status code of the service

RequestMapping

Attribute	Description
name	Mapping name
path	URI mapping
method	Supported HTTP method(s): GET,POST,DELETE, PUT
params	Request parameters to filter requests
headers	Headers values to filter requests
consumes	Specifies input media type(s) of the service
produces	Specifies output media type(s) of the service

Spring 4.3 improvements

- ✓ @GetMapping
- ✓ @PostMapping
- ✓ @PutMapping
- ✓ @DeleteMapping

```
@RequestMapping(method = RequestMethod.POST)
public @interface PostMapping {
```

MediaType

- ✓ *APPLICATION_FORM_URLENCODED_VALUE*
- ✓ *APPLICATION_JSON_VALUE*
- ✓ *APPLICATION_JSON_UTF8_VALUE*
- ✓ *APPLICATION_OCTET_STREAM*
- ✓ *APPLICATION_XML_VALUE*
- ✓ *TEXT_PLAIN_VALUE*
- ✓ *TEXT_HTML_VALUE*

JSON vs XML


```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

JSON vs XML


```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

MediaType. HTTP request


```
POST /blog/posts
Accept: application/json
Content-Type: application/json
Content-Length: 57
```

Response body

Request body

Task 5. Response format

1. Open **rest-task5** sub-project.
2. Open **BookController** class and write REST **GET service** that return sample book instance in **JSON** format
3. Check in Postman that service works correctly
4. Apply **@ResponseStatus** to the method so that it returns another status code (for example **HttpStatus.ACCEPTED**).

Task 6. JSON mapping

1. Try to Google and find out how to change the property names in the response output.
2. Firstly, you need to change (rename) the **JSON** attributes of the book response; for example, “name” -> “title”
3. Secondly, you need to change (rename) the **XML** tag names of the response.

@PathVariable


```
@GetMapping(path = "/{id}")
public Book findById(@PathVariable("id")
                             String bookId) {
    return new Book();
}
```

```
@GetMapping(path = "/{id}")
public Book findById(@PathVariable String id) {
    return new Book();
}
```

@RequestBody


```
@PostMapping
public Book saveBook(@RequestBody Book book) {
    return book;
}
```

Task 7. REST services and CRUD operations

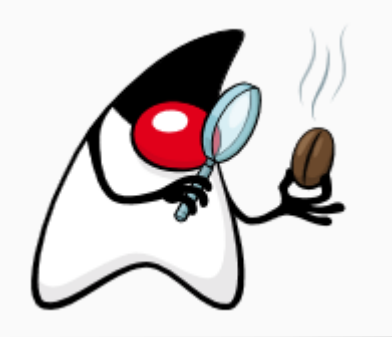

```
@RestController
@RequestMapping("/book")
public class BookController {
    @Autowired
    private BookRepository bookRepository;

}
```

Task 7. REST services and CRUD operations

1. Implement **CRUD** operations in BookController
2. Use **BookRepository** for data access operations
3. Use Postman to send requests

Bean validation API

- Part of Java EE 6 (JSR 380) and later (but can be run on Java SE)
- Expresses predefined constraints on object model
- Custom constraints
- Localization
- Version 2.0 requires Java and supports Optional/Java Time



Maven integration

API

```
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>2.0.1.Final</version>
</dependency>
```

Implementation

```
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.10.Final</version>
</dependency>
```

Bean Validation API

Annotation	Description
@NotEmpty	Element should not be empty
@NotBlank	Element should not be blank
@Min	Specifies minimum value of the element
@Max	Specifies maximum value of the element
@NotNull	Element should not be null
@Pattern	Specifies regular expression pattern
@Email	Element should be email
@Future	Element should be time or instant in the future
@Negative	Element should be negative number
@Size	Element size (length) should be between min and max

Bean Validation API


```
public class Person {

    @NotNull
    private String id;

    @Size(min = 4,max = 20, message = "Username size is between 4
    private String userName;

    @Size(min = 8, max = 16)
    private String password;

    @Min(value = 1900)
    @Max(value = 2018)
    private int birthYear;

    @Email
    private String email;
}
```

Explicit validation


```
ValidatorFactory factory = Validation
        .buildDefaultValidatorFactory();
Validator validator = factory.getValidator();

Person person = new Person();

Set<ConstraintViolation<Person>> violations = validator
        .validate(person);
violations.forEach(violation -> System.out.println(
        violation.getMessage()));
```

Exception in thread "main" javax.validation.ValidationException: HV000183: Unable to initialize 'javax.el.ExpressionFactory'. Check that you have the EL dependencies on the classpath, or use ParameterMessageInterpolator instead

Explicit validation


```
<dependency>
    <groupId>javax.el</groupId>
    <artifactId>javax.el-api</artifactId>
    <version>3.0.0</version>
</dependency>
<dependency>
    <groupId>org.glassfish</groupId>
    <artifactId>javax.el</artifactId>
    <version>3.0.0</version>
</dependency>
```

должно быть задано
должно быть больше или равно 1900

@RequestBody


```
@PostMapping
public Book saveBook(@RequestBody Book book) {
    return book;
}
```

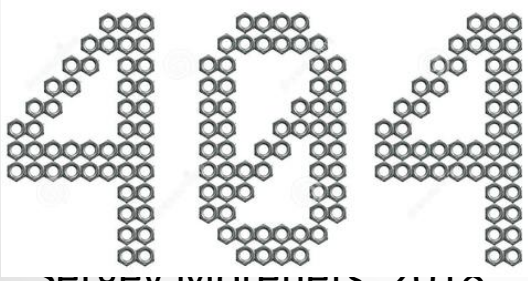
```
@PostMapping
public Book saveBook(@Valid @RequestBody Book book) {
    return book;
}
```

Requires validation provider

Task 8. Validation API

1. Add **Validation API** annotations on the Book class fields.
2. Invoke POST service to create new book and omit required fields. What is the server response?
3. Try to override default messages in Validation API annotations.

Error handling



Error handling


```
@RequestMapping(path = "/{id}",
        produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
public ResponseEntity<Book> findById(
        @PathVariable("id") String id) {
    int bookId = NumberUtils.toInt(id);
    if (bookId <= 0) {
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }

    Book book = bookRepository.findById(bookId);
    if (book == null) {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<>(book, HttpStatus.OK);
}
```

Error handling. Custom exceptions


```
    Book book = bookRepository.findById(bookId);
    validate(book);

    return new ResponseEntity<>(book, HttpStatus.OK);
}

private void validate(Book book) {
    if (book == null) {
        throw new BookNotFoundException();
    }
}
```

Error handling. Custom exceptions

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class BookNotFoundException
                    extends RuntimeException {

    private static final long serialVersionUID =
            3591666710943050205L;

}
```

Error handling. Controller advice


```
@ControllerAdvice
public class RestExceptionHandler extends
        ResponseEntityExceptionHandler {

    @ExceptionHandler(value = { BookNotFoundException.class})
    protected ResponseEntity<Object> handleConflict(
            BookNotFoundException ex,
            WebRequest request) {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
}
```

Task 9. Error handling

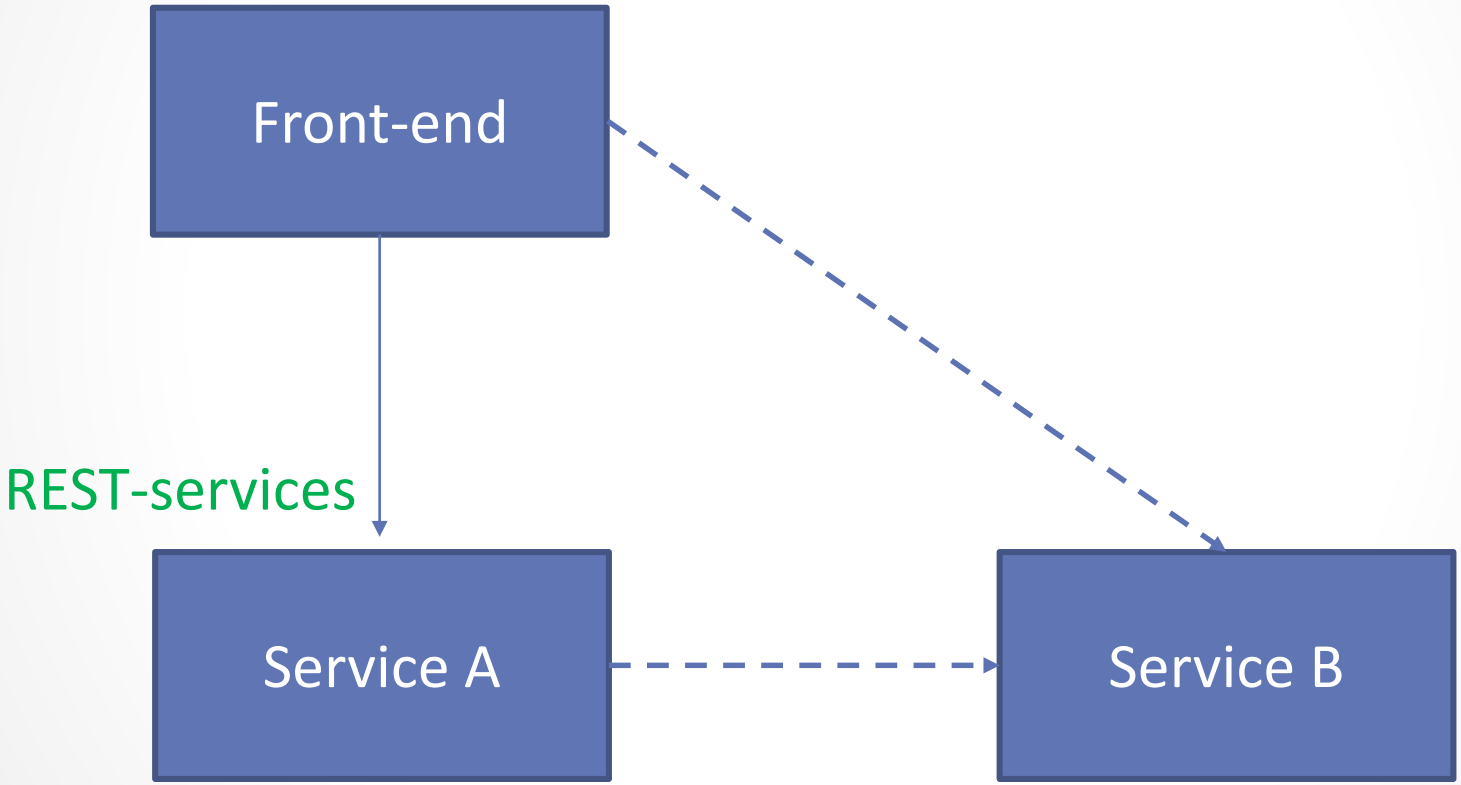
1. Add common error handling in your controller using three approaches:

- ➢ Manual error handling
- ➢ Use **@ResponseStatus** annotation
- ➢ Use **@ControllerAdvice** annotation
- ✓ Compare all the approaches

REST support on client side

Sergey Morenets, 2018

Services communication



Apache HTTP components

- ✓ **java.net** alternative
- ✓ Implementation of HTTP 1.0/1.1 and all HTTP methods
- ✓ **Low-level** components for HTTP communication
- ✓ Blocking and non-blocking I/O
- ✓ Client-side authentication
- ✓ Connection **pools**

Apache HTTP components


```
String uri = "http://example.com/hotels/1/bookings";

PostMethod post = new PostMethod(uri);
String request = // create booking request content
post.setRequestEntity(new StringRequestEntity(request));

httpClient.executeMethod(post);

if (HttpStatus.SC_CREATED == post.getStatusCode()) {
    Header location = post.getRequestHeader("Location");
    if (location != null) {
        System.out.println("Created new booking at :" + location.getValue());
    }
}
```

- Sergey Morenets, 2018

RestTemplate

High-level abstraction over Apache Http components library

HTTP method	RestTemplate method
DELETE	delete
GET	getForObject, getForEntity
POST	postForObject, postForLocation
PUT	Put
any	exchange

RestTemplate. Find


```
public class RestClient {

    private final RestTemplate restTemplate =
            new RestTemplate();

    public ResponseEntity<Book> findBook(int id) {
        return restTemplate.getForEntity(
                "http://localhost:8080/book/" +
                                id, Book.class);
    }
```

???

```
public List<?> findBooks() {
    return restTemplate
            .getForObject("http://localhost:8080/books",
                    List.class);
}
```

RestTemplate. Find


```
List<Map<?,?>> data = restTemplate
        .getForObject("http://localhost:8080/book",
                List.class);
List<Book> books = data.stream()
        .map(item -> objectMapper.convertValue(
                item, Book.class))
        .collect(Collectors.toList());
```

```
ResponseEntity<List<Book>> responseEntity =
        restTemplate.exchange("http://localhost:8080/book",
        HttpMethod.GET, null,
        new ParameterizedTypeReference<List<Book>>() {
        });
return responseEntity.getBody();
```

- Sergey Morenets, 2018

RestTemplate. Save & update


```
public URI saveBook(Book book) {
    return restTemplate.postForLocation(
            "http://localhost:8080/book", book);
}

public void updateBook(Book book) {
    MultiValueMap<String, String> headers =
            new LinkedMultiValueMap<>();
    headers.add(HttpHeaders.CONTENT_TYPE,
            MediaType.APPLICATION_JSON_UTF8_VALUE);
    HttpEntity<Book> request = new HttpEntity<>(book, headers);
    restTemplate.put("http://localhost:8080/book/" +
            book.getId(), request, Void.class);
}
```

RestTemplate. Query params


```
UriComponentsBuilder builder = UriComponentsBuilder
        .fromHttpUrl("http://localhost:8080/books")
        .queryParam("sort", sort)
        .queryParam("asc", asc);

String url = builder.toString();
```

Customization

Autowired

```
@Bean
public RestTemplate restTemplate(
        RestTemplateBuilder builder) {
    return builder.basicAuthorization("admin", "admin")
            .build();
}
```

```
private final RestTemplate restTemplate = new
        RestTemplateBuilder()
        .setConnectTimeout(30)
        .build();
```

Error handling


```
public List<Book> findBooks() {
    return (List<Book>) restTemplate
            .getForObject("http://localhost:8080/book",
                    List.class);
}
```

Unavailable

```
Exception in thread "main" org.springframework.web.client.ResourceAccessException:
I/O error on GET request for "http://localhost:8080/book": connect timed out; nested
exception is java.net.SocketTimeoutException: connect timed out
        at
org.springframework.web.client.RestTemplate.doExecute(RestTemplate.java:743)
```

Catch RestClientException

Custom error handler


```
public class CustomErrorHandler implements ResponseErrorHandler {
    @Override
    public void handleError(ClientHttpResponse response)
            throws IOException {
        System.out.println("Server error: " +
            response.getStatusText());
    }

    @Override
    public boolean hasError(ClientHttpResponse response)
            throws IOException {
        return response.getStatusCode().is5xxServerError();
    }
}
```

```
RestTemplate restTemplate = new RestTemplate();
restTemplate.setErrorHandler(new CustomErrorHandler());
```

Custom error handler


```
public class CustomErrorHandler extends
    DefaultResponseErrorHandler {
    @Override
    public void handleError(ClientHttpResponse response)
            throws IOException {
        System.out.println("Server error: " +
            response.getStatusText());
    }
}
```

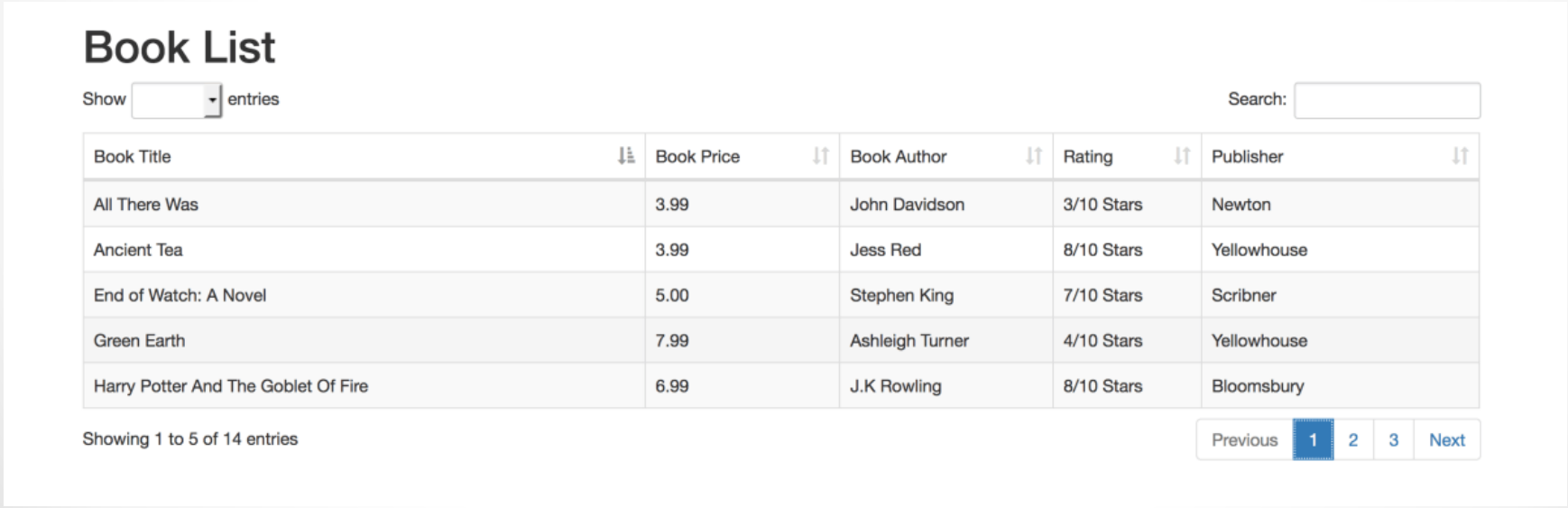
Task 10. RestTemplate

1. Create **BookRestClient** class that will allow to launch following REST services

- ➢ Find a book
- ➢ Find all books
- ➢ Save/update book
- ✓ Verify status codes and returned response

Server-side pagination

Improves performance and usability of your service



- Sergey Morenets, 2018

REST-service


```
@GetMapping(params= {"page", "size"})
public List<Book> searchBooks(@RequestParam int page,
        @RequestParam int size) {
    return bookService.searchBooks(new
            PageCriteria(page, size));
}
```

```
public class PageCriteria {
    private final int page;

    private final int size;

    public PageCriteria(int page, int size) {
        this.page = page;
        this.size = size;
    }
```

Pagination response


```
public class Page {

    private final int totalCount;

    private final List<Book> books;

    public Page(int totalCount, List<Book> books) {
        this.totalCount = totalCount;
        this.books = books;
    }
```

```
@GetMapping(params= {"page", "size"})
public Page searchBooks(@RequestParam int page,
        @RequestParam int size) {
    return bookService.searchBooks(new PageCriteria(page, size));
}
```

Pagination response


```
@GetMapping(params = { "page", "size" })
public ResponseEntity<List<Book>> searchBooks(
        @RequestParam int page, @RequestParam int size) {
    Page pageResponse = bookService.searchBooks(
            new PageCriteria(page, size));

    HttpHeaders headers = new HttpHeaders();
    headers.add("X-Total-Count",
            String.valueOf(pageResponse.getTotalCount()));

    return ResponseEntity.ok().headers(headers)
            .body(pageResponse.getBooks());
}
```

Pagination response


```
@ControllerAdvice
public class TotalCountAdvice implements ResponseBodyAdvice<List<?>> {

    @Override
    public boolean supports(MethodParameter returnType,
            Class<? extends HttpMessageConverter<?>> converterType) {
        return List.class.isAssignableFrom(
                returnType.getParameterType());
    }

    @Override
    public List<?> beforeBodyWrite(List<?> body,
            MethodParameter returnType, MediaType selectedContentType,
            Class<? extends HttpMessageConverter<?>>
            selectedConverterType, ServerHttpRequest request,
            ServerHttpResponse response) {
        response.getHeaders().add("X-Total-Count",
                String.valueOf(body.size()));
        return body;
```

Task 11. Server-side pagination

1. Update **findBooks** method in the BookController class so that it accepts pagination parameters and paginated response (data and total count)
2. Update **BookRepository/SimpleBookRepository** so that it provides pagination functionality when returning books
3. Test your REST-service with different input parameters

Actuator

- Helps manage and monitor applications when pushed to production
- Accessible via HTTP, JMX or remote shell
- You can't manage what you can't measure

Spring Boot Actuator

- Series of endpoints to help manage your Spring application
- Reads properties and spring beans and then returns a JSON view
- Allows direct access to non functional application information without having to open an IDE or a command prompt

Actuator. Spring Boot 2 changes

- ✓ Doesn’t depend directly on Spring MVC
- ✓ Allow to use Spring MVC or WebFlux
- ✓ Only /health and /info endpoints are enabled by default
- ✓ Micrometer is used for metrics

Spring Boot Actuator. Maven


```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-actuator</artifactId>
    <version>${spring.boot.version}</version>
</dependency>
```

Spring Boot Actuator. Endpoints


```
_links:
  self:
      href:        "http://localhost:8080/actuator"
      templated:   false
  health:
      href:        "http://localhost:8080/actuator/health"
      templated:   false
  info:
      href:        "http://localhost:8080/actuator/info"
      templated:   false
```


https://localhost:8080/actuator

Spring Boot Actuator. Endpoints

Endpoint	Description
/actuator/beans	Displays list of Spring beans in the application
/actuator/metrics	Shows application metrics
/actuator/env	Exposes environment variables
/actuator/loggers	Allows to read/change logger settings
/actuator/health	Shows application health information
/actuator/threaddump	Performs thread dump
/actuator/conditions	Displays auto-configuration report with filtering support
/actuator/info	Displays application-related info
/actuator/scheduledtasks	Display scheduled tasks

Endpoints management

management.endpoints.web.exposure.include=*

```
management.endpoint.loggers.cache.time-to-live=50s
```

application.properties

Health information


```
{
    "status": "UP"
}
```

/actuator/health

Indicators	
CassandraHealthIndicator	Checks that Cassandra server is up
DiskSpaceHealthIndicator	Checks for low disk space.
KafkaHealthIndicator	Checks that Kafka is up.
Neo4jHealthIndicator	Checks that Neo4j server is up
MongoHealthIndicator	Checks that a Mongo database is up.
RabbitHealthIndicator	Checks that a Rabbit server is up
RedisHealthIndicator	Checks that a Redis server is up
SolrHealthIndicator	Checks that a Solr server is up
MailHealthIndicator	Checks that a mail server is up

Health


```
@Component
public class StorageHealth implements HealthIndicator {

    @Override
    public Health health() {
        int errorCode = checkStorage();
        if (errorCode != 0) {
            return Health.down().withDetail("Status code",
                    errorCode).build();
        }
        return Health.up().build();
    }
```

- Sergey Morenets, 2018

Health


```
{
    "status": "DOWN",
    "storageHealth": {
        "status": "DOWN",
        "Status code": 90
    },
    "diskSpace": {
        "status": "UP",
        "free": 760162553856,
        "threshold": 10485760
    }
}
```


when-authorized

never (**default**)

```
management.endpoint.health.show-details=always
```

application.properties

Info contributors

Contributors	Description
EnvironmentInfoContributor	Returns all the environment properties for keys started with **info**
GitInfoContributor	Returns information from a git.properties file
BuildInfoContributor	Returns build information from META-INF/build-info.properties file

Application information

```
info.application.name=Spring Boot
info.application.description=Test application
info.application.version=0.1.0
```

application.properties

```
{
    "application": {
        "version": "0.1.0",
        "description": "Test application",
        "name": "Spring Boot"
    }
}
```

/actuator/info

Custom contributor


```
@Component
public class UIInfoContributor implements InfoContributor {

    private final Properties properties;

    public UIInfoContributor() throws IOException {
        Resource resource = new
                ClassPathResource("ui.properties");
        properties = new Properties();
        properties.load(resource.getInputStream());
    }

    @Override
    public void contribute(Info.Builder builder) {
        builder.withDetail("ui", properties);
    }
```

Task 12. Spring Boot Actuator

1. Add **Spring Boot Actuator** dependency:
2. Run application and check global endpoint */actuator* and then **other endpoints**
3. Add new **HealthIndicator** Spring component that will go down if no books are present in the library.
4. Add new **InfoContributor** Spring component that returns all the REST services (paths, methods) defined in your application.

Metrics

- Resides under /actuator/metrics endpoint
- Metrics for all HTTP requests are automatically recorded
- Since Spring Boot 2 based on Micrometer

Metrics

- ✓ System metrics
- ✓ Datasource metrics
- ✓ WebServer metrics
- ✓ Custom metrics

Metrics. Spring Boot 1.5

/application/metrics


```
{
    "mem":185856,
    "mem.free":98617,
    "processors":4,
    "uptime":36557,
    "heap.committed":185856,
    "heap.init":131072,
    "heap.used":87238,
    "heap":1860608,
    "threads.peak":20,
    "threads.daemon":17,
    "threads":19,
    "classes":9122,
    "classes.loaded":9122,
    "classes.unloaded":0,
    "gc.ps_scavenge.count":29,
    "gc.ps_scavenge.time":394,
    "gc.ps_marksweep.count":4,
    "gc.ps_marksweep.time":920
}
```

Metrics. Spring Boot 2.0

/actuator/metrics


```
names:
    0:   "jvm.buffer.memory.used"
    1:   "jvm.memory.used"
    2:   "jvm.gc.memory.allocated"
    3:   "jvm.memory.committed"
    4:   "tomcat.sessions.created"
    5:   "tomcat.sessions.expired"
    6:   "tomcat.global.request.max"
    7:   "tomcat.global.error"
    8:   "jvm.gc.max.data.size"
    9:   "logback.events"
    10:  "system.cpu.count"
    11:  "jvm.memory.max"
```

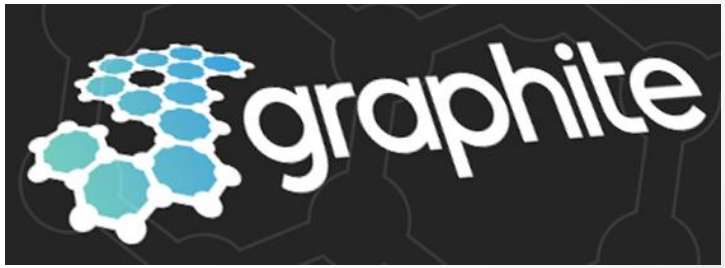
Metrics. Spring Boot 2.0


```
name:                "jvm.memory.used"
measurements:
  0:
      statistic:     "VALUE"
      value:         106108720
availableTags:
  0:
      tag:           "area"
    values:
        0:           "heap"
        1:           "nonheap"
```

/actuator/metrics/jvm.memory.used

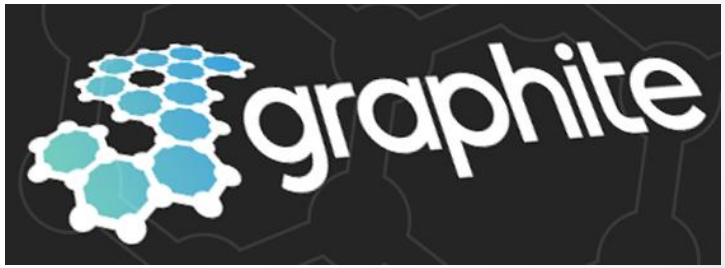
Metrics management

- ✓ Based on Micrometer façade
- ✓ Supports different monitoring solutions:



Graphite

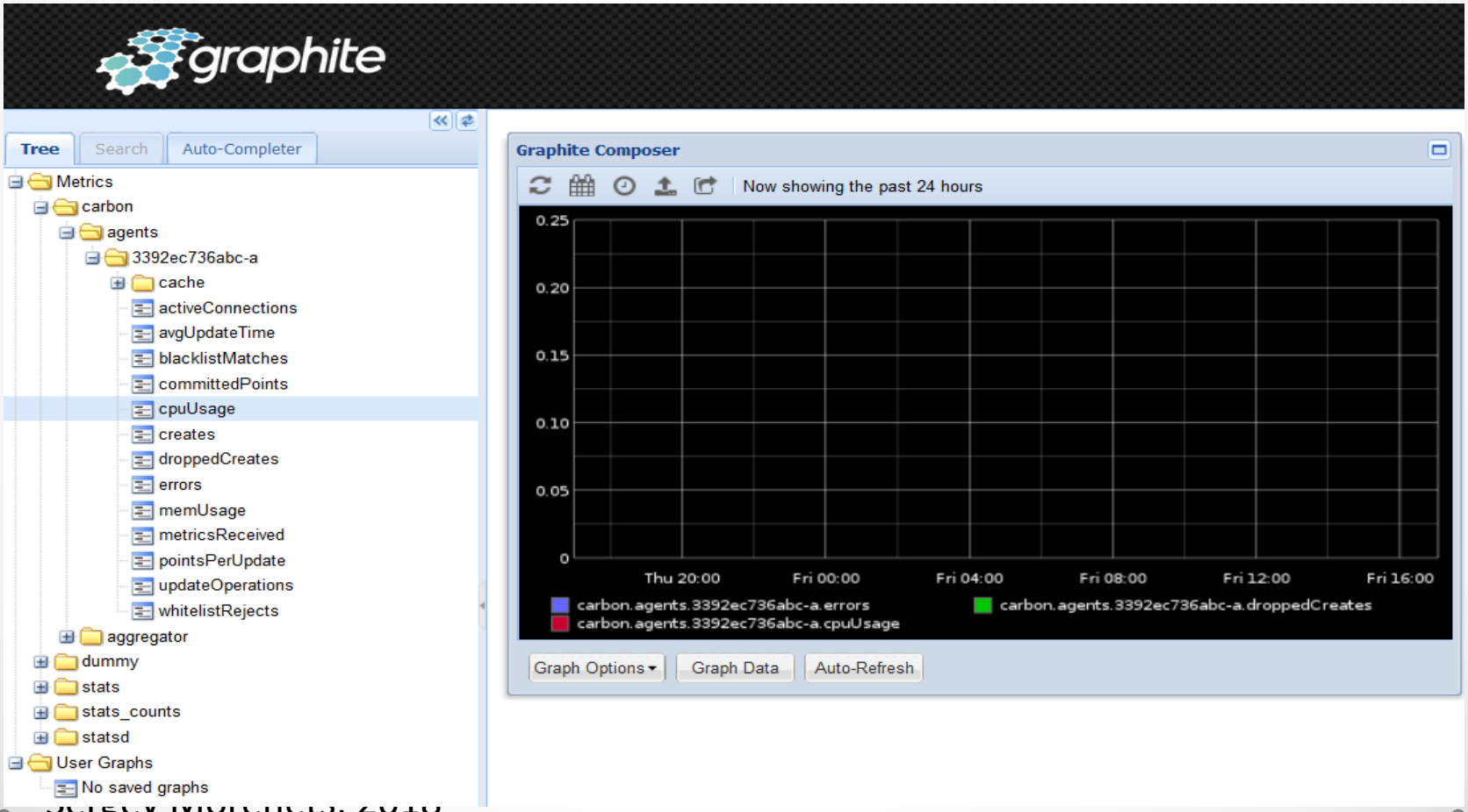
- Open-source monitoring system
- Released in 2008
- Contains UI application (based on Python and Django), Carbon service and Whisper file system
- Integrated with Diamond, Ganglia, Grafana, Graphen and others



Micrometer and Graphite


```
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-graphite</artifactId>
    <version>1.0.0</version>
</dependency>
```

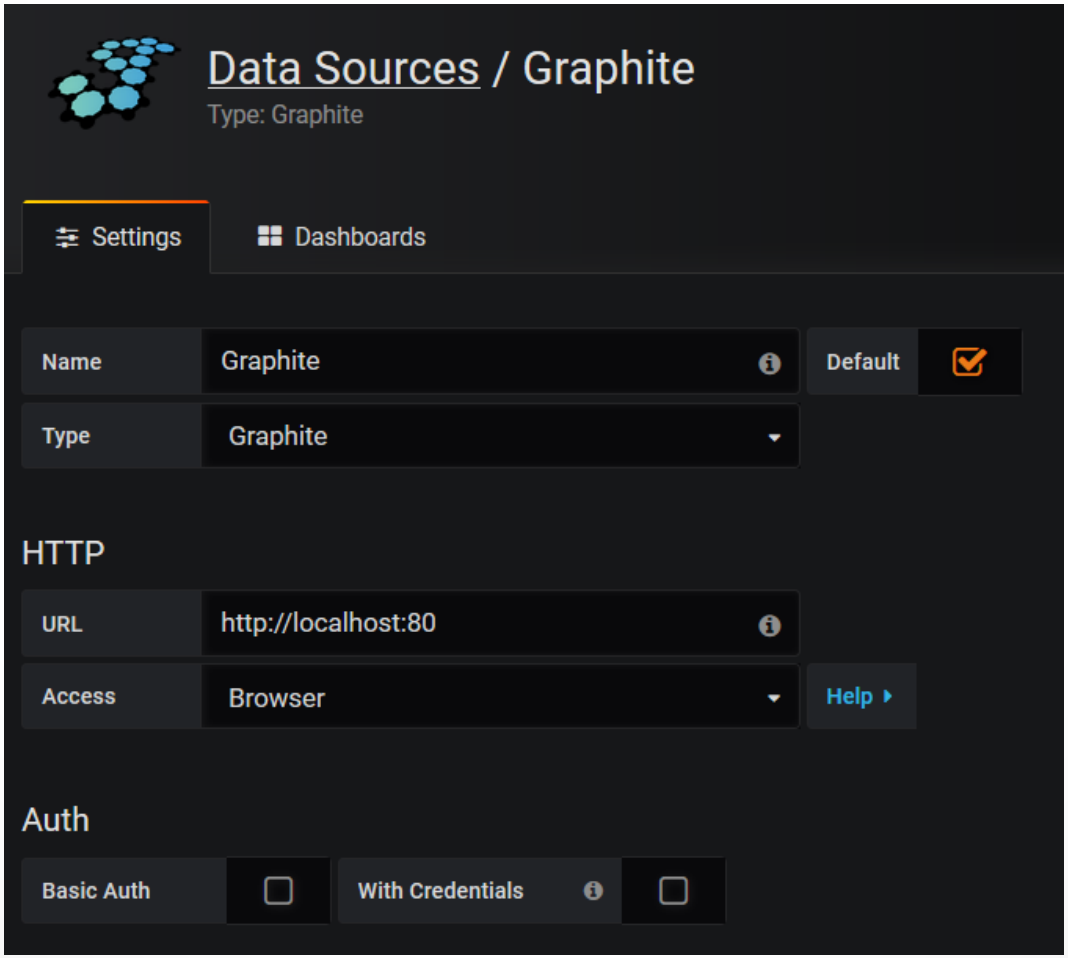
Graphite Dashboard



Grafana

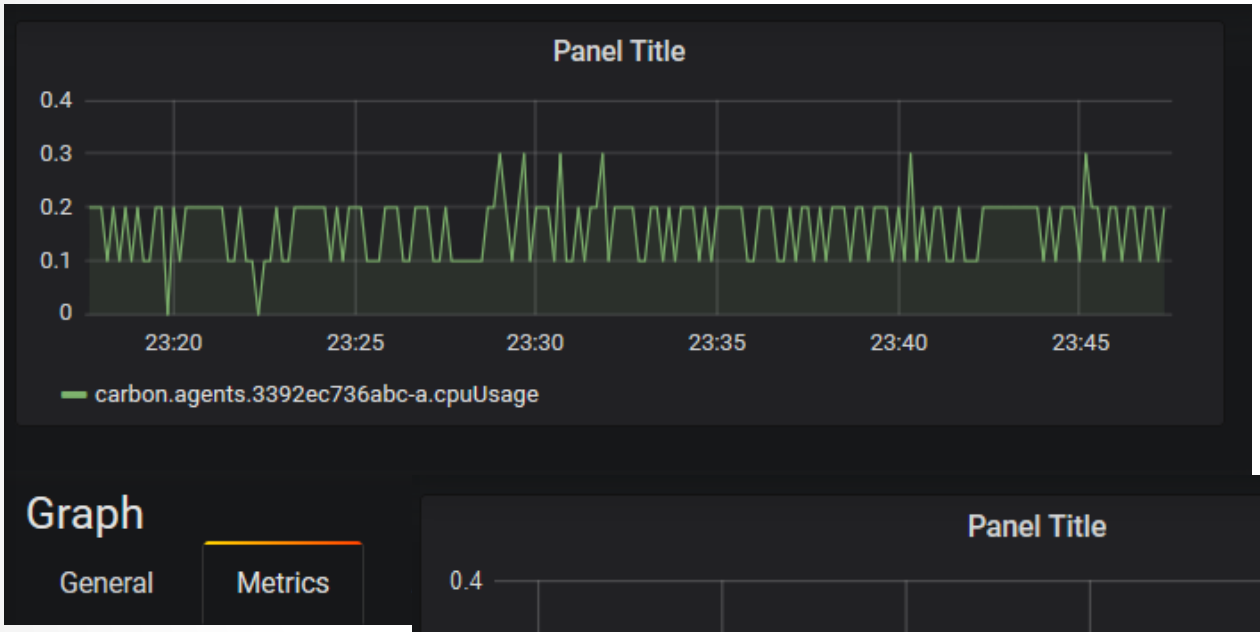
- Data visualization and monitoring tool
- A lot of visualization options
- Extensions using custom plugins
- Supports Prometheus, InfluxDB, Graphite, ElasticSearch and others, Cloudwatch and data-source providers

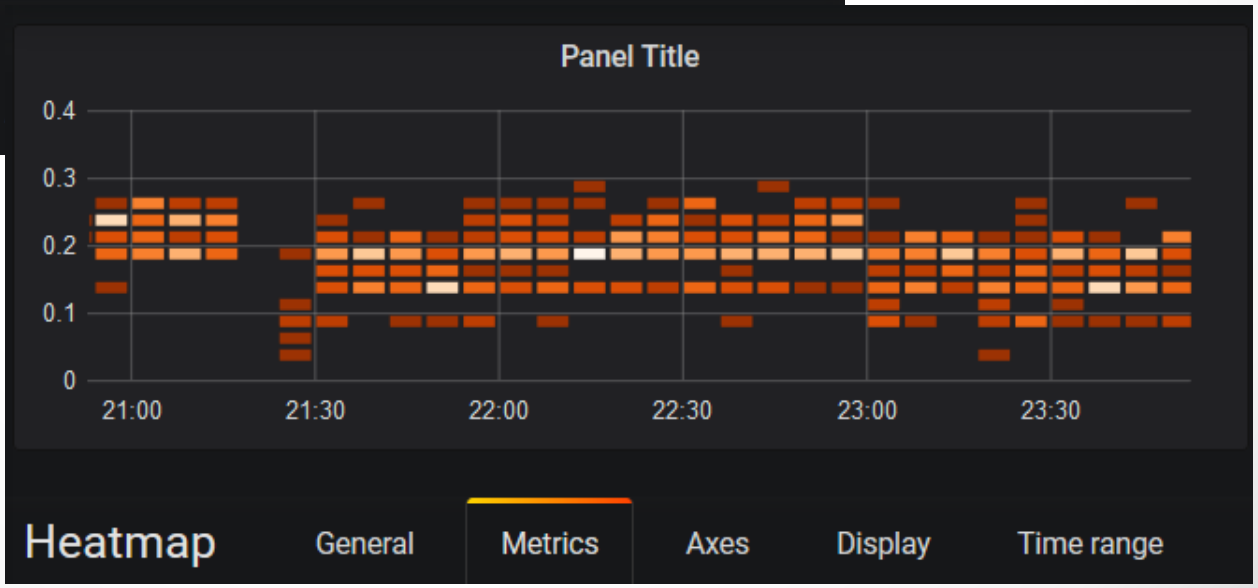
Grafana. Datasources



- Sergey Morenets, 2018

Grafana. Graphs





- Sergey Morenets,

Custom metrics


```
@Component
public class SampleComponent {

    private final Counter invoiceCounter;

    public SampleComponent(MeterRegistry registry) {
        this.invoiceCounter = registry.counter(
                "total.invoices");
    }

    public void handleInvoice(Invoice invoice) {
        this.invoiceCounter.increment();
    }
```

Micrometer

HTTP metrics


```
name:                "http.server.requests"
measurements:
  0:
      statistic:     "COUNT"
      value:         7
  1:
      statistic:     "TOTAL_TIME"
      value:         81.25922700000001
  2:
      statistic:     "MAX"
      value:         44.6276
availableTags:
  0:
      tag:           "exception"
```

Timed metrics


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

    @GetMapping
    @Timed(value="book.findAll")
    public List<Book> findBooks() {
```

Micrometer

Timed metrics


```
name:                "book.findAll"
measurements:
  0:
      statistic:     "COUNT"
      value:         3
  1:
      statistic:     "TOTAL_TIME"
      value:         11.352739
  2:
      statistic:     "MAX"
      value:         7.887477
availableTags:
  0:
      tag:           "exception"
```

Custom timed metrics


```
private final MeterRegistry meterRegistry;

public String status()  {
    Sample timer = Timer.start(meterRegistry);
    try {
        Thread.sleep(1);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    Builder builder = Timer.builder("status.time");
    timer.stop(builder.tags("status", "ok")
            .register(meterRegistry));
    return "ok";

}
```

Custom timed metrics


```
name:                "status.time"
measurements:
  0:
      statistic:     "COUNT"
      value:         9
  1:
      statistic:     "TOTAL_TIME"
      value:         0.011096008
  2:
      statistic:     "MAX"
      value:         0.001988022
availableTags:
  0:
      tag:           "status"
    values:
        0:           "ok"
```


/actuator/metrics/status.time

Task 13. Metrics

1. Open URL */actuator/metrics* and review existing metrics
2. Add **@Timed** annotation to some of your REST-services and verify that new metric is created
3. Create two metrics: number of saved books and number of deleted books using **MeterRegistry** bean
4. Create new timed metric **book.save.requests** that will be updated each time new book is created.

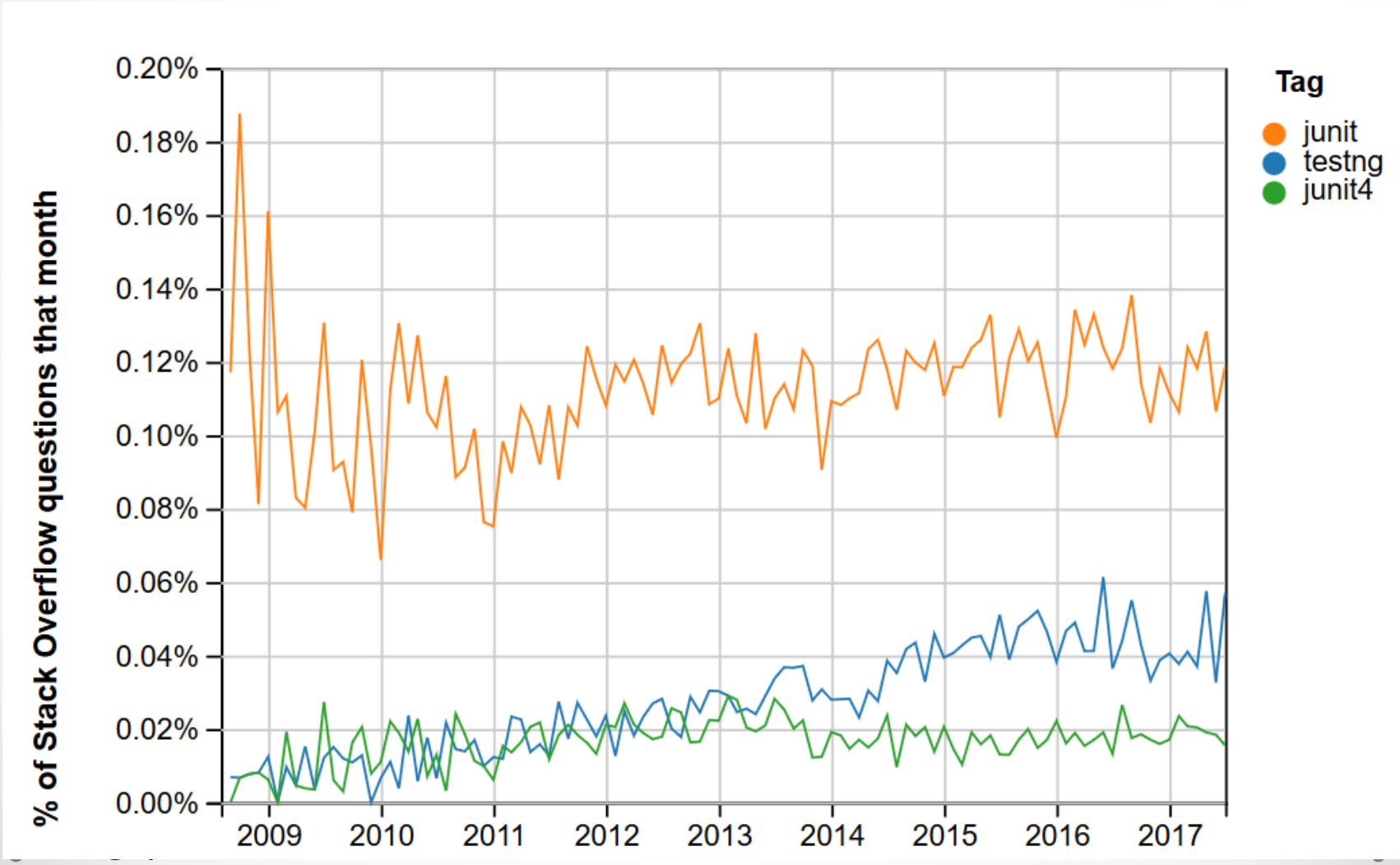
Do you write tests for your projects?

Sergey Morenets, 2018

Spring MVC Test framework

- Part of **TestContext** framework
- Based on mock conception
- Allows to write integration tests
- Don’t require servlet container (memory consumption, startup time)
- Spring MVC - High-level abstractions over mocked requests/responses (since Spring 3.2)

Popularity of test libraries



JUnit 5

- ✓ Separation of concerns
- ✓ API improvements, test extension model
- ✓ Supports repeated, parametrized and dynamic tests
- ✓ New assumptions concept
- ✓ Java 8 - based

JUnit 5

A first test case

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

class FirstJUnit5Tests {

    @Test
    void myFirstTest() {
        assertEquals(2, 1 + 1);
    }

}
```

Spring Test Framework. Maven


```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <scope>test</scope>
</dependency>
```

Spring Test. Maven plugin


```
<plugin>
    <artifactId>maven-surefire-plugin</artifactId>
    <dependencies>
        <dependency>
            <groupId>org.junit.platform</groupId>
            <artifactId>junit-platform-surefire-provider</artifactId>
            <version>1.3.1</version>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.3.1</version>
        </dependency>
    </dependencies>
</plugin>
```

Spring MVC Test. JUnit 5


```
@SpringJUnitWebConfig(BookApplication.class)
@AutoConfigureMockMvc
public class BookControllerTest {
    @Autowired
    private MockMvc mockMvc;
```

Spring Boot startup class

Spring MVC bridge

Spring MVC Test example

import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;

```
@Test
public void getBooks_RepositoryEmpty_NoBooksReturned()
        throws Exception  {
    //Given
    //When
    ResultActions actions = mockMvc.perform(get("/book"));
    //Then
    actions.andExpect(status().isOk())
            .andExpect(content().contentType(
                    MediaType.APPLICATION_JSON_UTF8_VALUE))
            .andExpect(jsonPath("$", Matchers.hasSize(0)));
}
```

import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

Java Hamcrest

Rule	Description
equalTo	Checks that two objects are logically equal
instanceOf	Checks that current object is an instance of the specified type
nullValue	Checks that current object is null
hasProperty	Checks that current object has specified property
hasSize	Checks that current collection has specified size

Jayway JsonPath

A Java DSL for reading JSON documents

```
<dependency>
    <groupId>com.jayway.jsonpath</groupId>
    <artifactId>json-path</artifactId>
    <version>2.4.0</version>
</dependency>
```

```
compile 'com.jayway.jsonpath:json-path:2.4.0'
```

Jayway JsonPath

Operator/ function	Description	Sample
$	Root element of the query	$
.<name>	Child node reference	$.id
[index, (index)]	Array index(indexes)	$.tags[0]

```
{
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}
```

- Sergey Morenets, 2018

Jayway JsonPath

Operator/ function	Description	Sample
@	The current processing node	$.[?(@.id > 10)]
*	Wildcard	$.[*].id
length()	Returns length of the array	$.length()

```
[
    {
        "id": 2,
        "name": "An ice sculpture",
        "tags": ["cold", "ice"],
        "warehouseLocation": {
            "latitude": -78.75,
            "longitude": 20.4
        }
    }
]
```

Logging


```
@Test
public void getBooks_RepositoryEmpty_NoBooksReturned()
        throws Exception  {
    //Given
    //When
    ResultActions actions = mockMvc.perform(get("/book"))
            .andDo(MockMvcResultHandlers.print());
    //Then
    actions.andExpect(status().isOk())
            .andExpect(content().contentType(
                    MediaType.APPLICATION_JSON_UTF8_VALUE))
            .andExpect(jsonPath("$", Matchers.hasSize(0)));
}
```

Testing POST method


```
private static final ObjectMapper MAPPER = new ObjectMapper();

@Test
public void saveBook_validBook_BookIsReturned() throws Exception {
    //Given
    Book book = new Book();
    book.setName("Java 8");
    //When
    ResultActions resultActions = mockMvc.perform(post("/book")
            .contentType(MediaType.APPLICATION_JSON_UTF8_VALUE)
            .content(MAPPER.writeValueAsString(book)));
    //Then
    resultActions.andExpect(status().isOk())
            .andExpect(content().contentType(MediaType.APPLICATION_JSON_
            .andExpect(jsonPath("$.name", Matchers.equalTo("Java 8")));
}
```

Task 14. Writing integration tests

1. Copy all the changed files from rest-task5 to **rest-task14** sub-project and open rest-task14 sub-project.
2. Review **BookControllerTest** class. Add integration tests (using **Junit 5** API) for your REST service operations (GET, POST, PUT, DELETE)
3. Verify **status codes** and returned response in the unit-tests

Rest Assured

- Allows to test REST-services in a simpler way
- Inspired by Groovy/Ruby simplicity
- Integrates with Spring Test/MVC
- Contains its own implementation of JsonPath and XmlPath
- Supports basic, digest , form and OAuth 1/2 authentication

REST-assured

Sergey Morenets, 2018

Rest Assured. Dependencies


```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>${rest-assured.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>spring-mock-mvc</artifactId>
    <version>${rest-assured.version}</version>
    <scope>test</scope>
</dependency>
```

REST-assured

Rest Assured. Initialization


```
@SpringJUnitWebConfig(BookApplication.class)
@AutoConfigureMockMvc
public class BookControllerTest {
    @Autowired
    private MockMvc mockMvc;

    @BeforeEach
    public void setup() {
        RestAssuredMockMvc.mockMvc(mockMvc);
    }
```

Rest Assured. Integration test


```
@Test
public void saveBook_validBook_BookIsReturned() {
    Book book = new Book();
    book.setTitle("Java 8");

    given().
        contentType(ContentType.JSON).body(book)
    .when().post("/book")
    .then().body(containsString("Java 8"))
           .statusCode(HttpStatus.SC_OK);
}
```

```
import static io.restassured.module.mockmvc.RestAssuredMockMvc.*;
```

- Sergey Morenets, 2018

Rest Assured. JsonPath


```
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>json-path</artifactId>
    <version>${rest-assured.version}</version>
    <scope>test</scope>
</dependency>
```

```
given().
    contentType(ContentType.JSON).body(book)
.when().post("/book")
.then().body("title", equalTo("Java 8"))
        .statusCode(HttpStatus.SC_OK);
```

Task 15. Rest Assured

1. Add Rest Assured dependencies to rest-task14 project
2. Update **BookControllerTest** class and rewrite tests using Rest Assured give-then-when syntax.
3. Verify **status codes** and returned response in the unit-tests

Testing improvements

- ✓ Introduced in Spring Boot 1.4
- ✓ Pure unit-testing
- ✓ Added **AssertJ** and **Mockito**
- ✓ Added annotations:

@MockBean

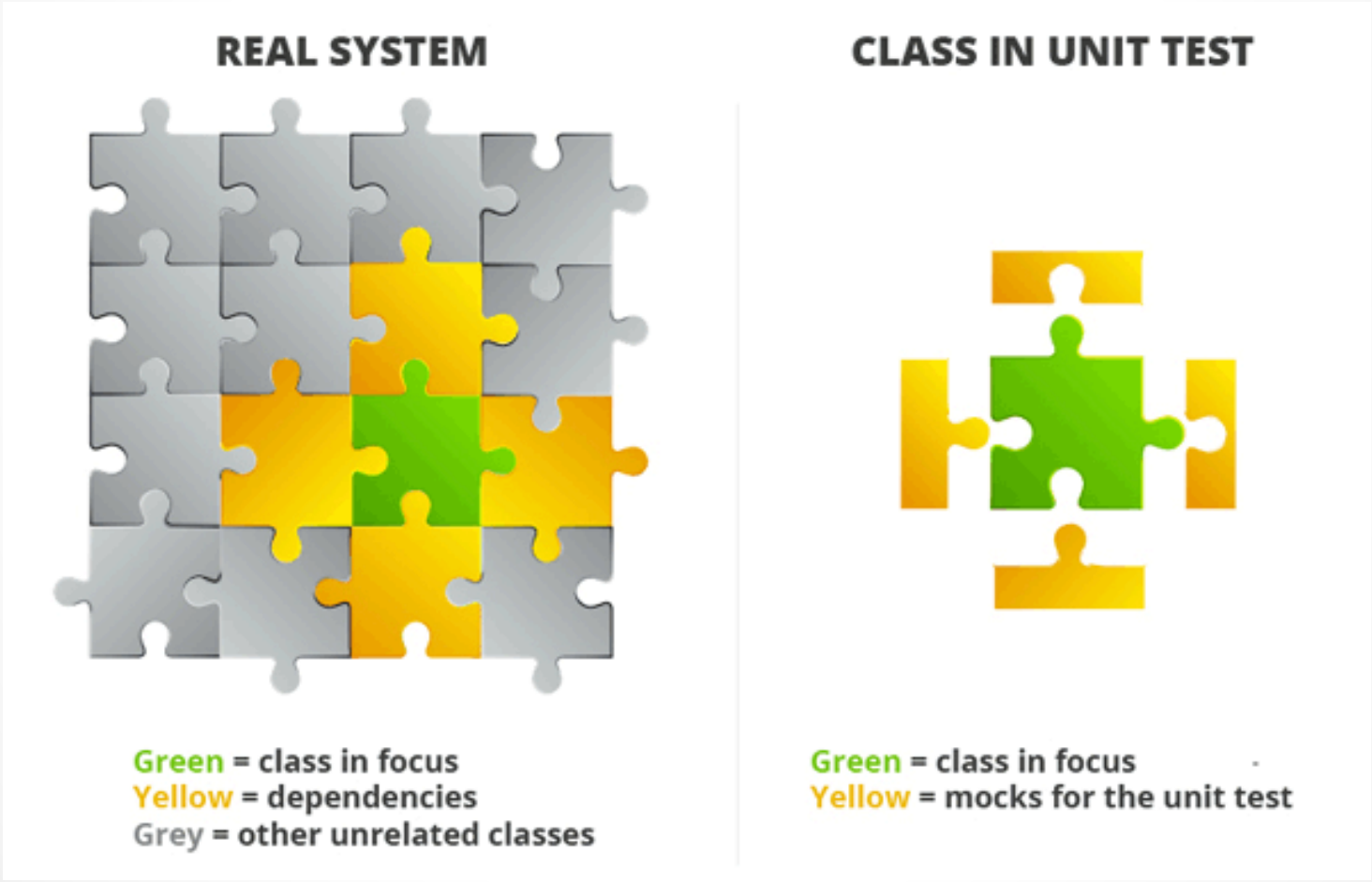
@SpringBootTest

@WebMvcTest

@JsonTest

spring boot

Mocks



- Sergey Morenets, 2018

Mockito


```
interface Execute {
    String execute();

    void process();
}
```

```
Execute execute = Mockito.mock(Execute.class);
Mockito.when(execute.execute()).thenReturn("Hello!");
```

- Sergey Morenets, 2018

Mocking dependencies


```java
@SpringJUnitWebConfig(RestApplication.class)
@AutoConfigureMockMvc
public class BookControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private BookRepository bookRepository;
```

Mocked instance

Unit-tests with mocking


```
@SpringJUnitWebConfig(RestApplication.class)
@AutoConfigureJsonTesters
public class BookControllerTest {
    @Autowired
    private JacksonTester<Book> json;
```

```
ResultActions resultActions = mockMvc.perform(post("/book")
.contentType(MediaType.APPLICATION_JSON_UTF8_VALUE)
.content(json.write(book).getJson()));
```

JsonContent

Unit-tests with mocking


```
@Test
public void findBooks_StorageIsNotEmpty_OneBookReturned()
        throws Exception {
    //Given
    given(bookRepository.findAll()).
            willReturn(Arrays.asList(new Book()));
    //When
    ResultActions resultActions = mockMvc.perform(get("/book"));
    //Then
    resultActions.andExpect(status().isOk())
            .andExpect(content().contentType(
                    MediaType.APPLICATION_JSON_UTF8_VALUE))
            .andExpect(jsonPath("$", Matchers.hasSize(1)));
}
```

BDDMockito

System tests


```
@SpringJUnitConfig(RestApplication.class)
@SpringBootTest(webEnvironment = SpringBootTest
    .WebEnvironment.RANDOM_PORT)
public class BookControllerSystemTest {

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    public void findAll_StorageEmpty_NoBooksReturned() {
        List<Book> books = this.restTemplate
                .getForObject("/books", List.class);
        assertTrue(books.isEmpty());
    }
}
```

Task 16. Unit-tests and system tests

1. Add mocking for the repository using **@MockBean** annotation
2. You can use import static statements to minimize source code:

   import static org.mockito.BDDMockito.;*
3. Provide default behavior for the repository methods.
4. Verify status codes and returned response in the unit-tests

Spring HATEOAS

- ✓ Document your REST API
- ✓ Make your API dynamic and flexible
- ✓ Decouples client and server
- ✓ Lower costs of changes

Spring HATEOAS. Maven


```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-hateoas</artifactId>
    <version>${spring.boot.version}</version>
</dependency>
```

BookController. Resource class


```
import static org.springframework.hateoas.mvc.ControllerLinkBuilder.*;
```

```
@GetMapping
public List<Resource<Book>> findBooks() {
    List<Book> books = bookRepository.findAll();

    List<Resource<Book>> resources = new ArrayList<>();
    for (Book book : books) {
        Resource<Book> resource = new Resource<Book>(book);
        resource.add(linkTo(methodOn(BookController.class).
                findById(book.getId()))
                .withSelfRel());
        resources.add(resource);
    }

    return resources;
}
```

Spring HATEOAS. Response body

GET /books


```
[
  {
    "id":1,
    "author":"Ryan Gosling",
    "year":2002,
    "title":"Core Java",
    "links":[
      {
        "rel":"self",
        "href":"http://localhost:8080/book/1",
        "hreflang":null,
        "media":null,
        "title":null,
        "type":null,
        "deprecation":null
      },
```

BookController. Two links


```
@GetMapping
public List<Resource<Book>> findBooks() {
    List<Book> books = bookRepository.findAll();

    List<Resource<Book>> resources = new ArrayList<>();
    for (Book book : books) {
        Resource<Book> resource = new Resource<Book>(book);
        resource.add(linkTo(methodOn(BookController.class).
                findById(String.valueOf(book.getId())))
                .withSelfRel());
        resource.add(linkTo(methodOn(BookController.class).
                buyBook(String.valueOf(book.getId())))
                .withRel("buy"));
        resources.add(resource);
    }

    return resources;
}
```

Spring HATEOAS. Response


```
[ ⊖
   { ⊖
      "id":1,
      "author":"Gerbert Schildt",
      "name":"Java 8",
      "year":2014,
      "links":[ ⊖
         { ⊖
            "rel":"self",
            "href":"http://localhost:8080/book/1"
         },
         { ⊖
            "rel":"buy",
            "href":"http://localhost:8080/book/buy/1"
         }
      ]
   },
```

Book resource


```
public class BookResource extends ResourceSupport {
    private String author;

    private String name;

    public BookResource(Book book) {
        author = book.getAuthor();
        name = book.getName();
        add(linkTo(methodOn(BookController.class)
                .findById(book.getId())).withSelfRel());
    }
```

BookController

```
@GetMapping
public List<BookResource> findBooks() {
    return bookRepository.findAll().stream()
            .map(BookResource::new)
            .collect(Collectors.toList());
}
```

Spring HATEOAS. Response


```
[ ⊖
  { ⊖
    "author":"James",
    "name":"Spring REST",
    "links":[ ⊖
      { ⊖
        "rel":"self",
        "href":"http://localhost:8080/book/1",
        "hreflang":null,
        "media":null,
        "title":null,
        "type":null,
        "deprecation":null
```

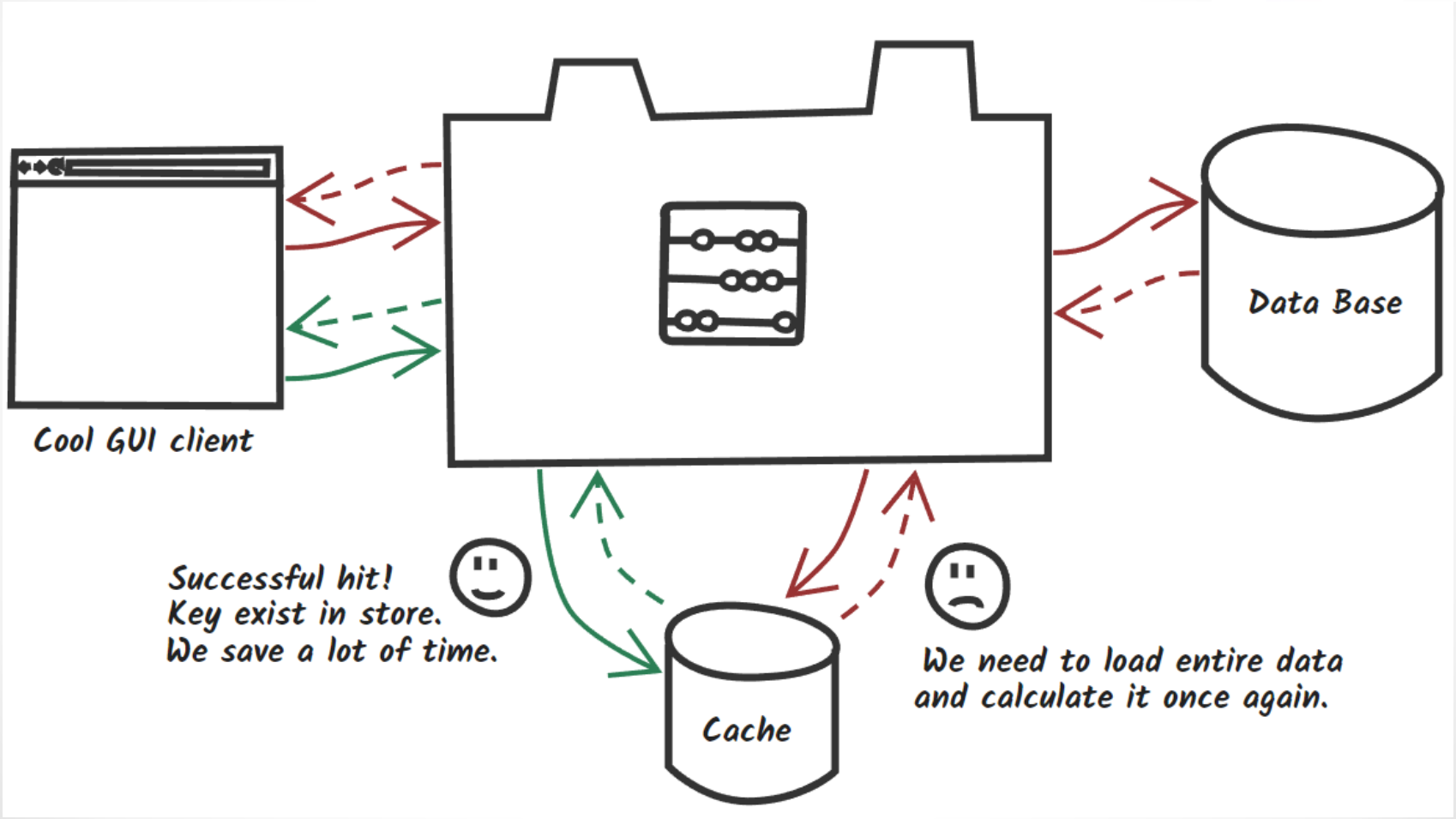
Task 17. Spring HATEOAS

1. Add REST service that allows to **rent** a single book
2. Update **findById** and **findAll** methods in the controller so that it returns links to get current book and rent a book

Caching

- ✓ Internal implementation by Spring
- ✓ **JCache (JSR-107)** is supported
- ✓ Various 3rd party implementations (EHCache, HazelCast, Infinispan, Couchbase, Redis ,Caffeine)

Caching



- Sergey Morenets, 2018

Caching. Maven dependencies

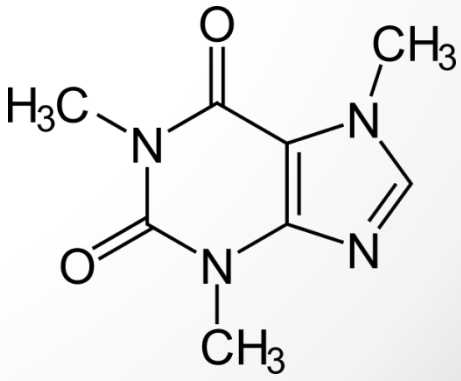

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
    <version>${spring.boot.version}</version>
</dependency>
```

Generic provider

```
<dependency>
    <groupId>com.github.ben-manes.caffeine</groupId>
    <artifactId>caffeine</artifactId>
    <version>2.6.2</version>
</dependency>
```

Caffeine

- High-performance caching library based on Java 8
- Inspired by **Guava**-based cache and ConurrentHashMap
- Asynchronous cache loading
- Sized-base eviction
- Time-based expiration
- Eviction notification
- Writes propagation
- Cache access statistics
- JCache support



Spring annotations

Name	Description
@EnableCaching	Enables annotation-driven cache management
@CacheConfig	Allows to config cache parameters
@Cacheable	Indicate that method(or all methods) result should be cached depending on the method arguments
@CachePut	Indicate that method result should be put into cache whereas method should be always invoked
@CacheEvict	Indicates that specific(or all) entries in the cache should be removed

Enable caching


```
@SpringBootApplication
@EnableCaching
@CacheConfig(cacheNames= {"orders", "payments"})
public class RestApplication {
```

Optionally specify cache names

```
@Cacheable("books")
@GetMapping(path = "/{id}")
public Book findById(@PathVariable int id) {
    return bookRepository.findById(id);
}
```

Update cache

Calls method and update cache

```
@PostMapping
@CachePut("books")
public void saveBook(@Valid @RequestBody Book book) {
    bookRepository.save(book);
}

@DeleteMapping("/{id}")
@CacheEvict("books")
public void deleteBook(@PathVariable int id) {
    bookRepository.delete(id);
}
```

Calls method and remove cache entry

Advanced caching


```
@GetMapping(path = "/{id}")
@Cacheable(cacheNames= {"books", "storage"},
        condition="#id!='1'")
public Book findById(@PathVariable int id) {
    return bookRepository.findById(id);
}
```

Task 18. Caching

1. Add **@EnableCaching** annotation to the **RestApplication** class.
2. Add Spring Boot starter cache dependency:
3. Add **@Cacheable** annotation to the **GET** REST-services. How does it affect its behavior?
4. Add new REST service that would clear the cache and put **@CacheEvict** annotation on it. Verify its behavior.

Data Transfer Object (DTO)


```java
public class Book {
    private int id;

    private String author;

    private String name;

    private int year;

    private List<String> pages;
```

Domain object

```java
public class BookDTO {
    private int id;

    private String name;
```

DTO

```java
@JsonIgnore
private List<String> pages;
```

Data Transfer Object. Pro and cons

- ✓ DTO classes are value objects without business logic
- ✓ We can easily change DTO without affecting domain/persistent objects (especially for public API)
- ✓ Using DTO you avoid put mapping annotations on domain objects
- ✓ Different DTO for each version of your API and different media types
- ✓ Easier support for HATEOAS links with DTO
- ✓ You need to map DTO and domain objects
- ✓ You need to maintain both DTO and domain objects

Mapping libraries

- Apache Commons
- Dozer
- ModelMapper
- Jmapper
- MapStruct
- Orika

Dozer

- Java bean to bean mapper
- Simple and complex type mapper
- Bi-directional mapper
- Implicit/explicit/recursive mapping
- Automatic conversion between types
- XML and annotation configuration

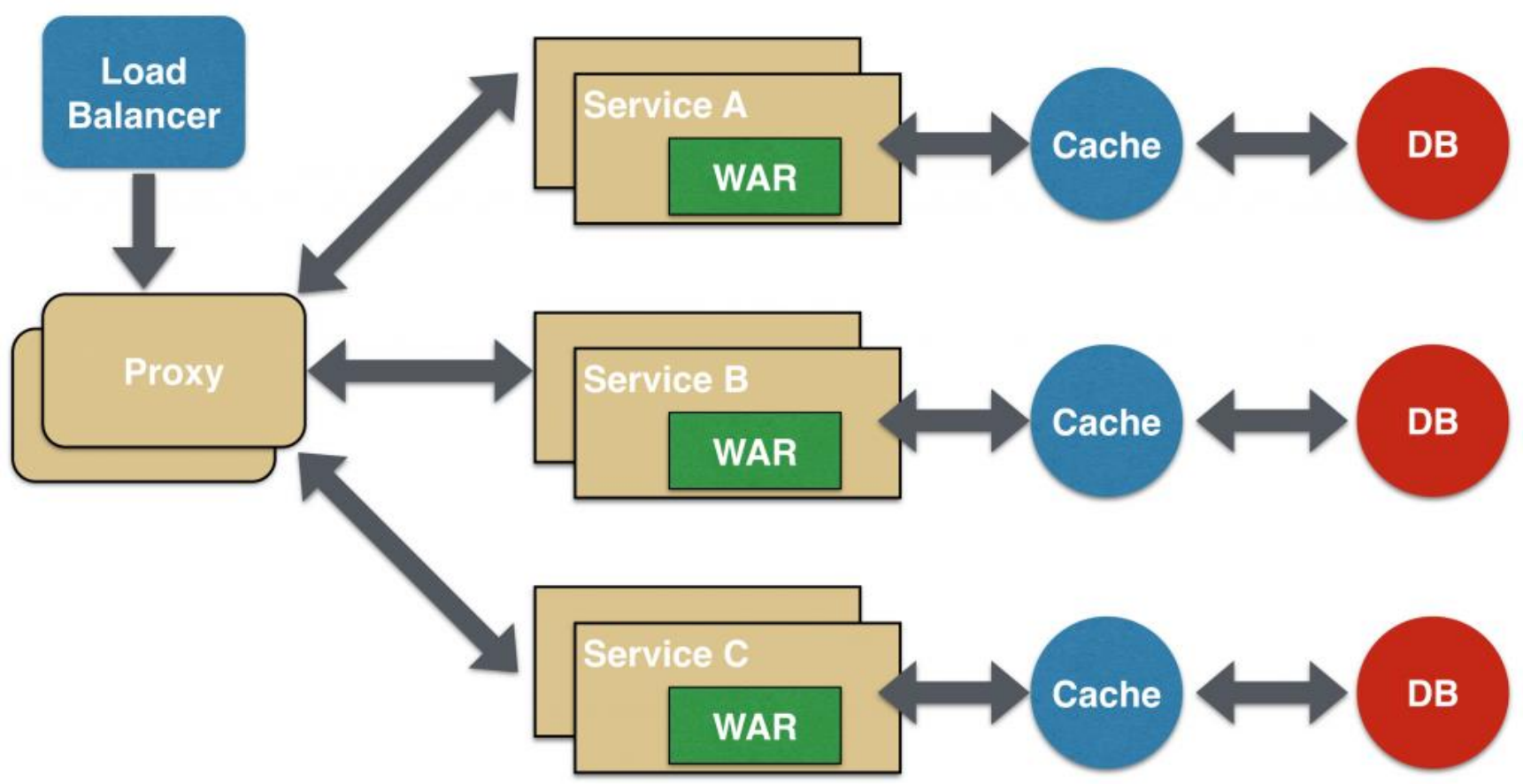
```
<dependency>
    <groupId>com.github.dozermapper</groupId>
    <artifactId>dozer-core</artifactId>
    <version>6.4.1</version>
</dependency>
```

Dozer. Examples


```
Book book = new Book();
book.setId(1);
book.setName("Spring");

Mapper mapper = DozerBeanMapperBuilder.buildDefault();
BookDTO dto = mapper.map(book, BookDTO.class);
```

```
public class BookDTO {
    private int id;

    @Mapping("name")
    private String title;
```

Scaling



- Sergey Morenets, 2018

Redis

- Created in 2009 by Salvatore Sanfillipo as **Re**mote **D**isctionary **S**erver
- In-memory database, message broker and cache provider
- Super-lightweight and easy to use
- 6 data types, 180+ commands
- Optional durability with snapshots or journals
- Provides automatic partitioning with **Redis Cluster**
- Supports monitoring and metrics
- Atomic, isolated and consistent

redis

Redis. Functionality

- ✓ Eviction algorithms are **LRU** (Last Recently Used) and **LFU** (Last frequently used)
- ✓ Durability supported with point-in-time snapshots or persistent logs for every write operation

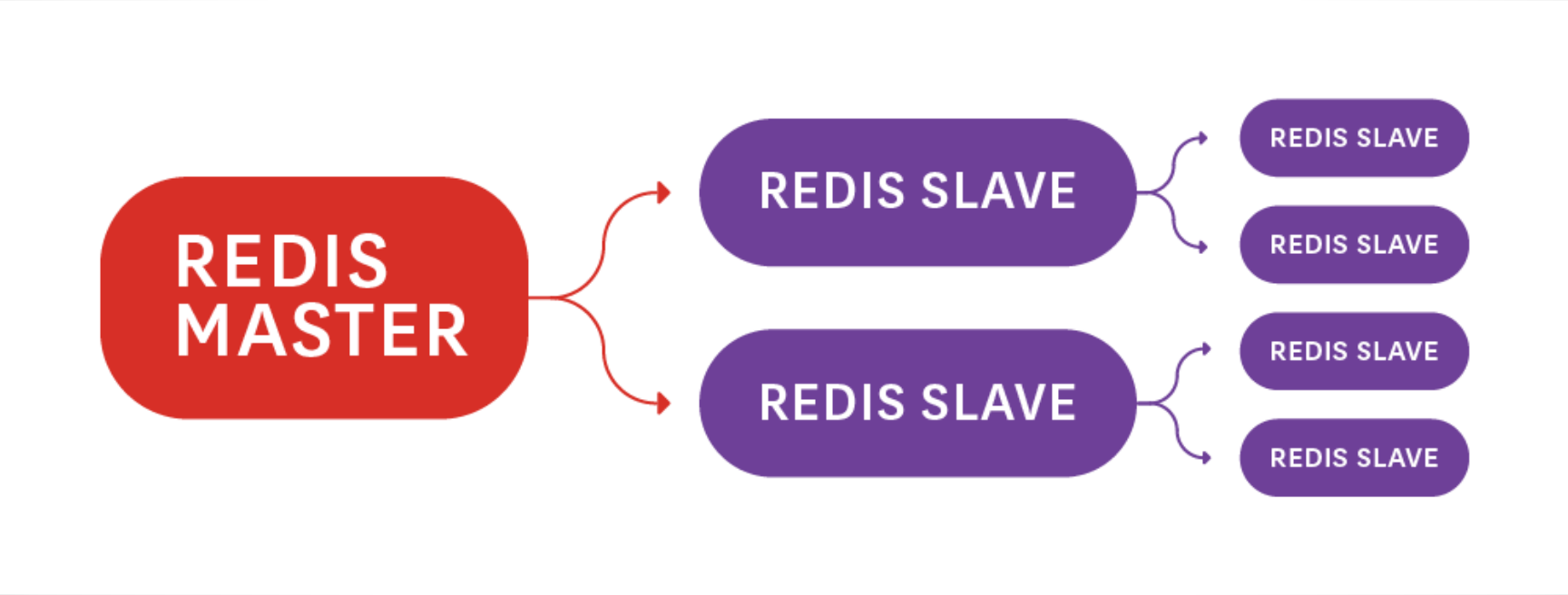
redis

Redis. Popularity

- Competes with **Memcached**(multi-threaded vs single-threaded)
- Sometimes called "**Memcached on steroids**"
- No memory limit for 64-bit systems
- Supports up to **512M** for key/value size (Memcached supports 250 bytes)
- #1 key-value database
- #4 NoSQL database

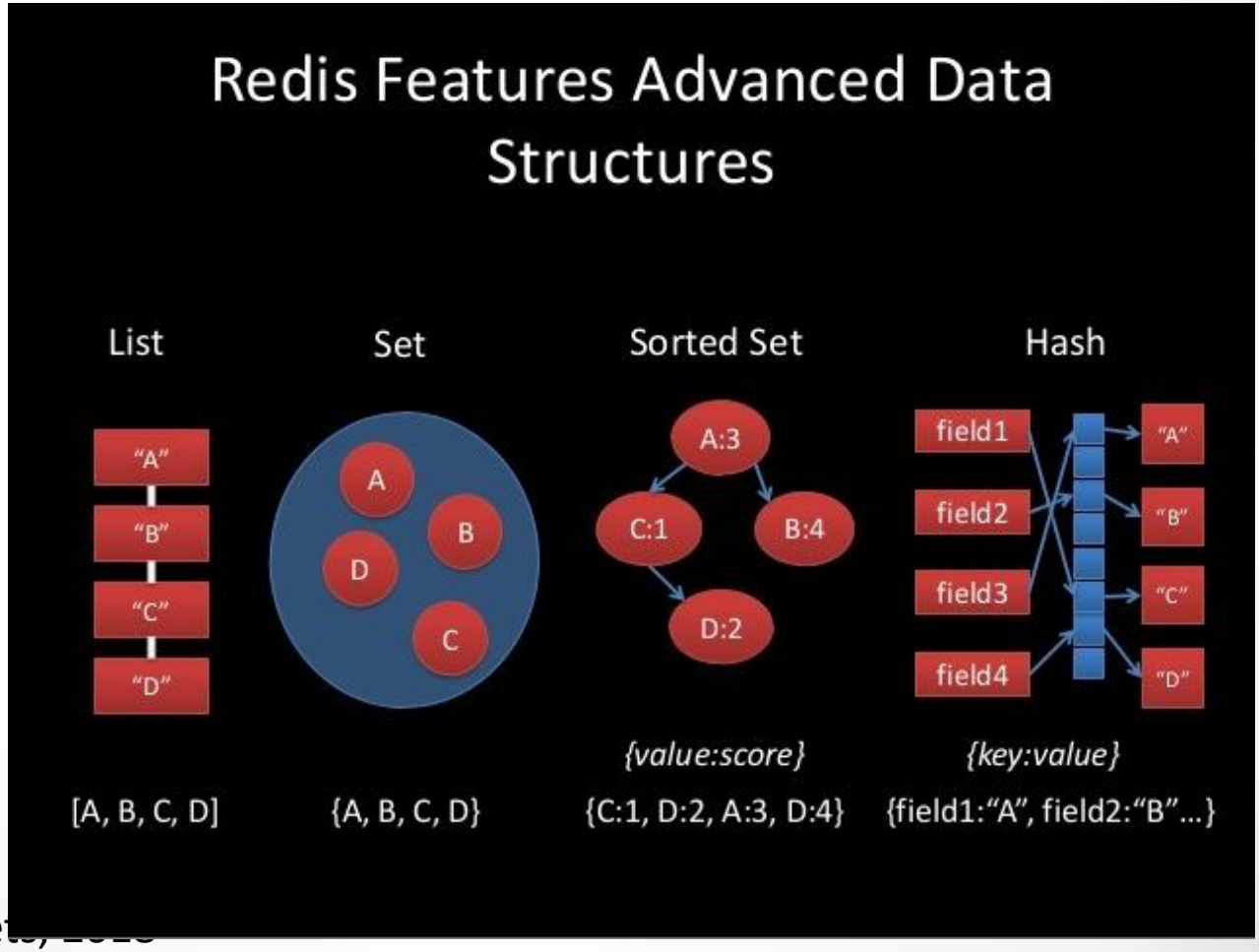
redis

Redis replication



- Sergey Morenets, 2018

Redis data types



- Sergey Morenets, 2013

Task 19. Redis configuration

1. Download and install Redis.
2. Review **redis.conf** configuration file (or **redis.windows.conf** on Windows platform).
3. Start Redis command-line using **redis-cli** command.
4. Starts another Redis console
5. Try to print all the configuration settings
6. Run **redis-cli --stat** command to view active connections to **Redis** server.

JCache and implementation


```
<dependency>
    <groupId>org.ehcache</groupId>
    <artifactId>ehcache</artifactId>
    <version>3.5.0</version>
</dependency>
<dependency>
    <groupId>javax.cache</groupId>
    <artifactId>cache-api</artifactId>
    <version>1.0.0</version>
</dependency>
```

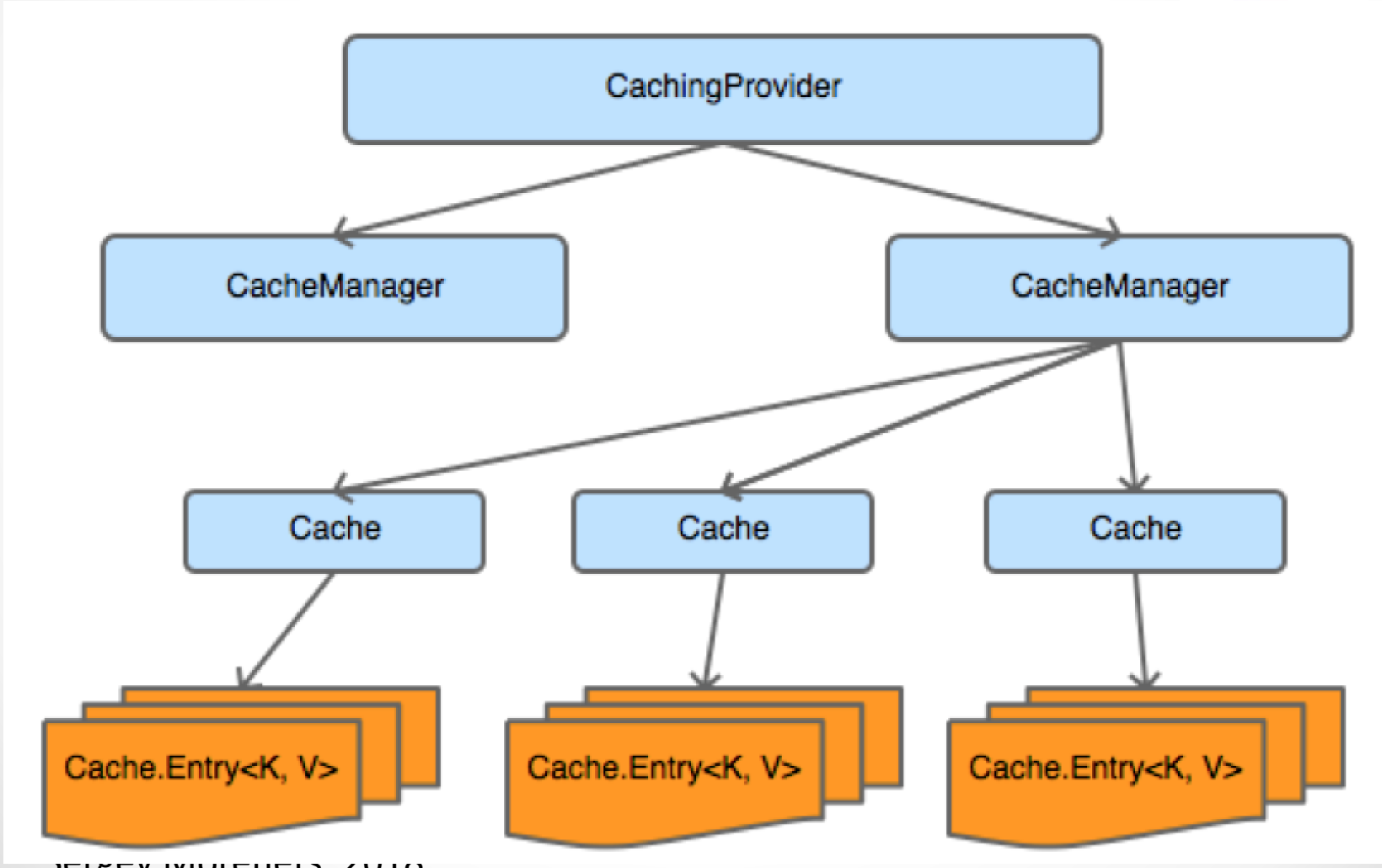
EHCache as caching provider

JSR-107

Redis as caching provider

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

JCache API


CachingProvider
CacheManager
CacheManager
Cache
Cache
Cache
Cache.Entry<K, V>
Cache.Entry<K, V>
Cache.Entry<K, V>

JCache annotations


```
@RestController
@RequestMapping("/book")
@CacheDefaults(cacheName="books")
public class BookController {
```

```
@GetMapping("/greet/{name}")
@CacheResult(cacheName="greeting")
public String greet(@PathVariable String name) {
    return "Hello," + name;
}

@GetMapping("/greet-nocache/{name}")
@CachePut(cacheName="greeting")
public String greetNoCache(@PathVariable
        @CacheValue String name) {
    return "Hello," +name;
}
```

Redis cache configuration


```
spring.cache.cache-names=orders

spring.cache.redis.key-prefix=KEY_
spring.cache.redis.time-to-live.seconds=5
```

Task 20. JCache and Redis

1. Add **JCache** and Spring Data Redis dependencies:
2. Replace @Cacheable with @CacheResult and @CacheEvict with @CacheRemove/ @CacheRemoveAll. Verify that new annotations are processed by Spring.
3. Use **@CacheDefaults** annotation to provide cache name for the entire controller
4. Try to test REST services and then check that **Redis** server contains new cache entries.

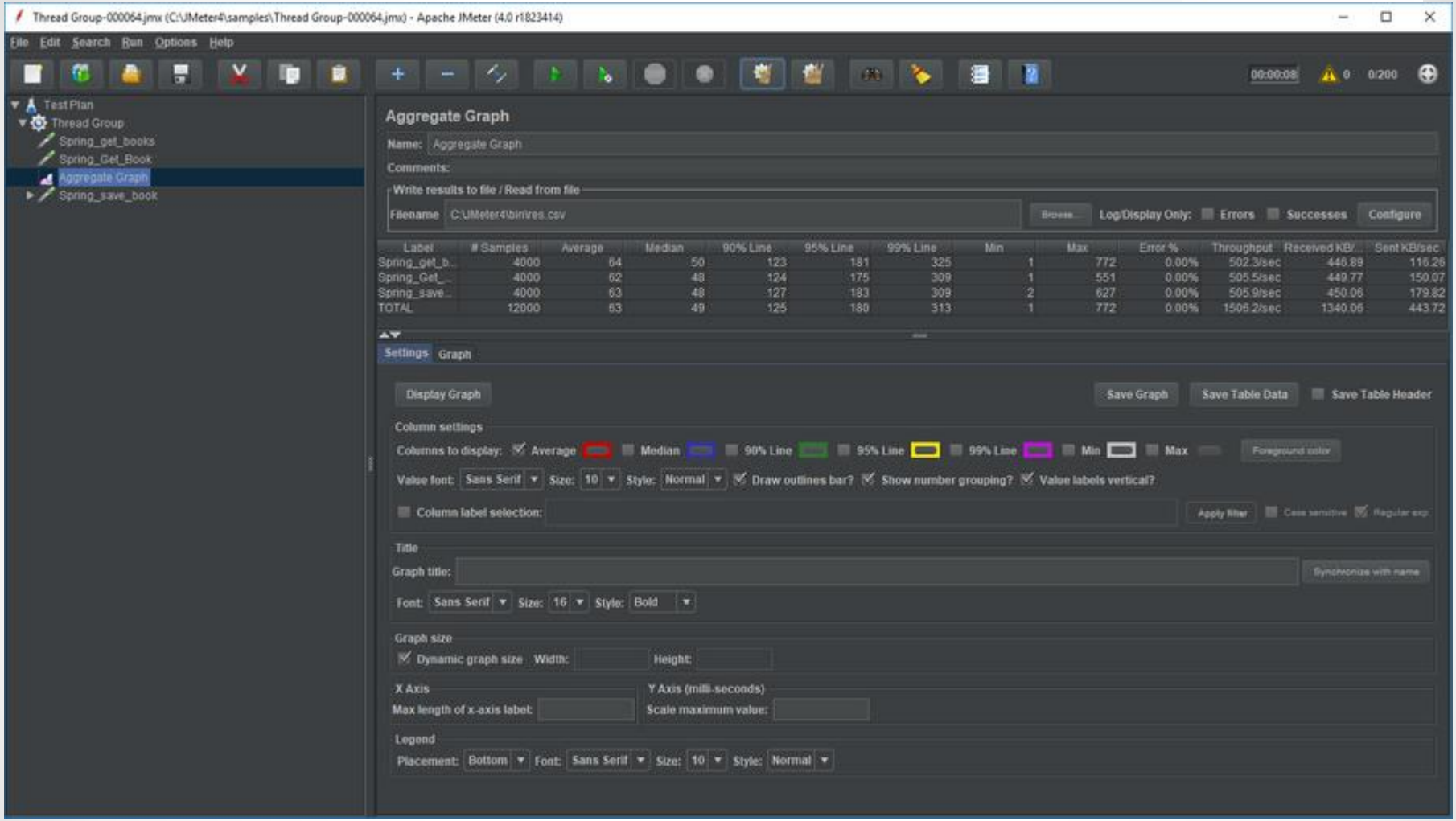
Performance/load testing

- ✓ Gatling/Jmeter
- ✓ Flood.io

- Sergey Morenets, 2018

JMeter

- Development kit for performance/stress testing
- Supports HTTP/JDBC/JMS connections
- Plugin-oriented architecture
- Supports distributed testing
- Current version 4.0 (Java 8 +)
- Allows GUI or command-line mode



- Sergey Morenets, 2018

Work use-case

- ✓ Create test plan
- ✓ Run tests (command-line mode)
- ✓ Observer report

Task 21. Performance testing

1. Download and install Apache JMeter
2. Run **jmeter** executable in the **bin** folder. Review its functionality and interface
3. Click on **Test Plan** node and add new Thread Group. Specify number of threads (users), ramp-up time (5 seconds is recommended) and loop count (10 or 20 is recommended).

Open API. REST documentation

- Provides language-agnostic interface for describing REST API services without knowing implementation details
- Allows to implement code generation, interactive documentation and test automation
- Doesn’t require changing your services
- Defines structure of OpenAPI document (definition)
- Current version is 3.0.1

Service documentation

- Every service should be documented
- Swagger is specification for API design/development/documentation
- Provides human-readable format (JSON, YAML) of your REST API
- Includes Swagger Inpector/Hub/UI

Swagger documentation


```yaml
swagger: "2.0"
info:
  version: "1.0"
  title: "Hello World API"
paths:
  /hello/{user}:
    get:
      description: Returns a greeting to the user!
      parameters:
        - name: user
          in: path
          type: string
          required: true
          description: The name of the user to greet.
      responses:
        200:
          description: Returns the greeting.
          schema:
            type: string
        400:
          description: Invalid characters in "user" w
```

Springfox

- SpringFox is automated JSON API documentation for Spring projects
- Based on original swagger-springmvcproject
- Supports Swagger (1.x/2.x), RAML and JsonAPI

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.8.0</version>
</dependency>
```

http://localhost:8080/v2/api-docs

Springfox configuration


```
@Configuration
@EnableSwagger2
public class SwaggerConfig {

}
```

http://localhost:8080/v2/api-docs

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2).select()
                .apis(RequestHandlerSelectors.basePackage(
                        "com.example.demo"))
                .paths(PathSelectors.any()).build();
    }
}
```

JSON specification


```
{
   "swagger":"2.0",
   "info":{
      "description":"Api Documentation",
      "version":"1.0",
      "title":"Api Documentation",
      "termsOfService":"urn:tos",
      "contact":{

      },
      "license":{
         "name":"Apache 2.0",
         "url":"http://www.apache.org/li
      }
   },
   "host":"localhost:8080",
   "basePath":"/",
   "tags":[
      {
         "name":"book-controller",
         "description":"Book Controller"
      }
   ],
   "paths":{
      "/book":{
         "get":{
            "tags":[
               "book-controller"
            ],
            "summary":"searchBooks",
            "operationId":"searchBooksUsingGET",
            "produces":[
               "*/*"
```


http://localhost:8080/v2/api-docs

Bean validation support

- ✔ Support for annotations @NotNull, @Size, @Email and others

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-bean-validators</artifactId>
    <version>2.9.2</version>
</dependency>
```

Override descriptors. Operation


```
@GetMapping
@ApiOperation(value = "Returns all the books",
    notes = "For pagination purposes use /search service")
public List<Book> findBooks() {
```

```
@GetMapping(params = { "page", "size" })
public List<Book> searchBooks(
        @ApiParam(name="page", value="Page index(zero-based)",
        required = true, example="0") @RequestParam int page,
        @ApiParam(name="size", value="Page size",
        required = true, example="20") @RequestParam int size) {
    Page pageResponse = bookService.searchBooks(
            new PageCriteria(page, size));
```

Override descriptors. Model


```
@ApiModel(value="Element of the library")
public class Book  {
    @ApiModelProperty(name = "Unique book identifier")
    private int id;

    @ApiModelProperty(name = "Book title", required = true)
    private String title;
```

Override descriptors. Response


```
@GetMapping("/{id}")
@ApiOperation(value = "Returns book by identifier")
@ApiResponses({@ApiResponse(code=200, message="Book is found"),
    @ApiResponse(code=404, message="Non-existing id")})
public Book findBookById(@PathVariable int id) {
```

```
@Bean
public Docket api() {
    return new Docket(DocumentationType.SWAGGER_2)
            .useDefaultResponseMessages(false)
            .apiInfo(new ApiInfoBuilder().version("v1")
                    .description("Sample REST API")
                    .title("Library project").build()).select()
            .apis(RequestHandlerSelectors.basePackage(
                    "com.example.demo"))
            .build();
}
```

Remove default response codes

Task 22. Swagger documentation

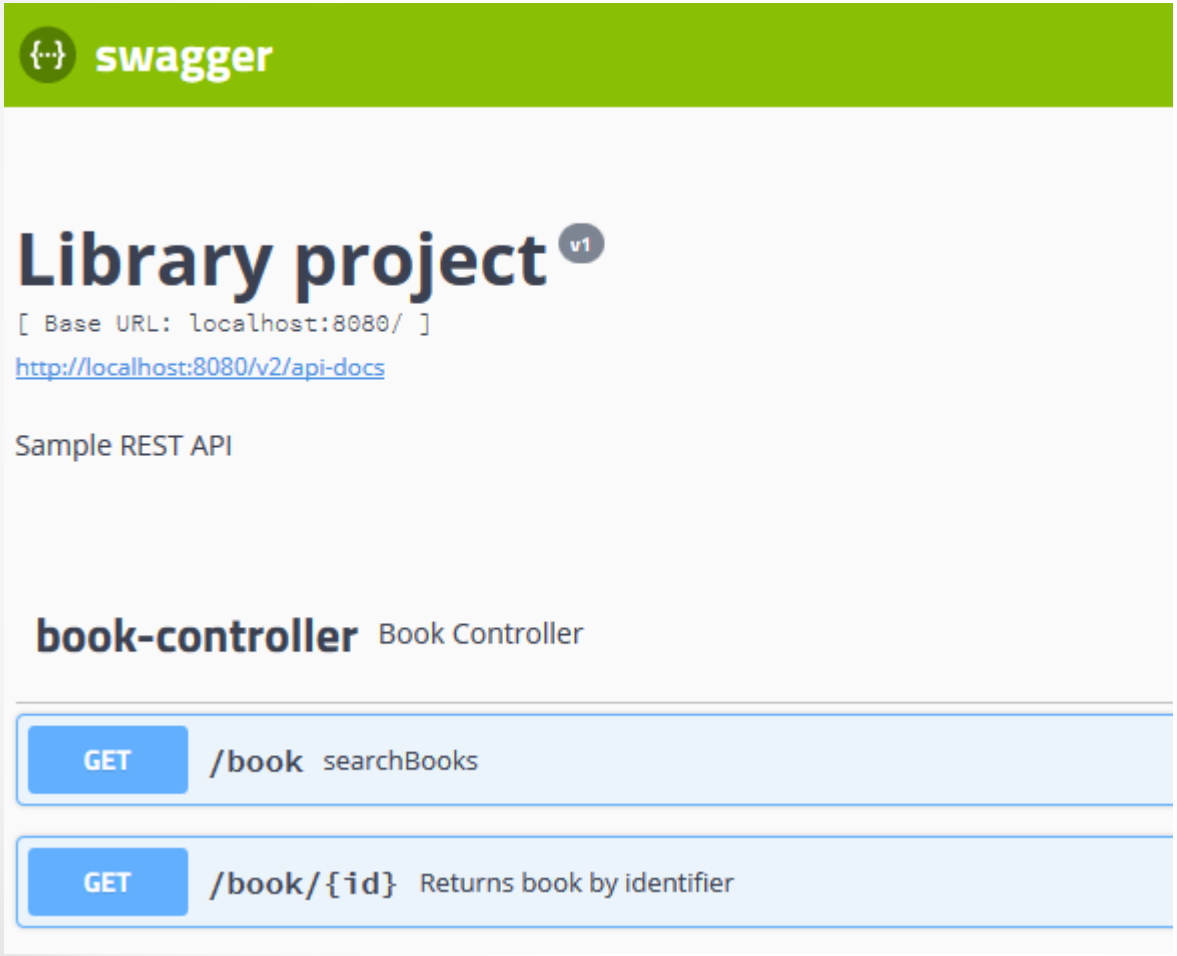
1. Add Springfox dependency into your project:
2. Create new Java class SwaggerConfiguration and put **@Configuration/ @EnableSwagger2** annotations on it.
3. Added new **@Bean** method into SwaggerConfiguration class and specify base package and API project information
4. Add Swagger-core annotations: **@ApiOperation/@ApiParam/@ApiResponses** on your REST services.

Swagger UI

- Swagger UI is a tool to visualize and test your REST services
- Web application written in React
- Automatically generated based on Swagger specification
- Supports security configuration

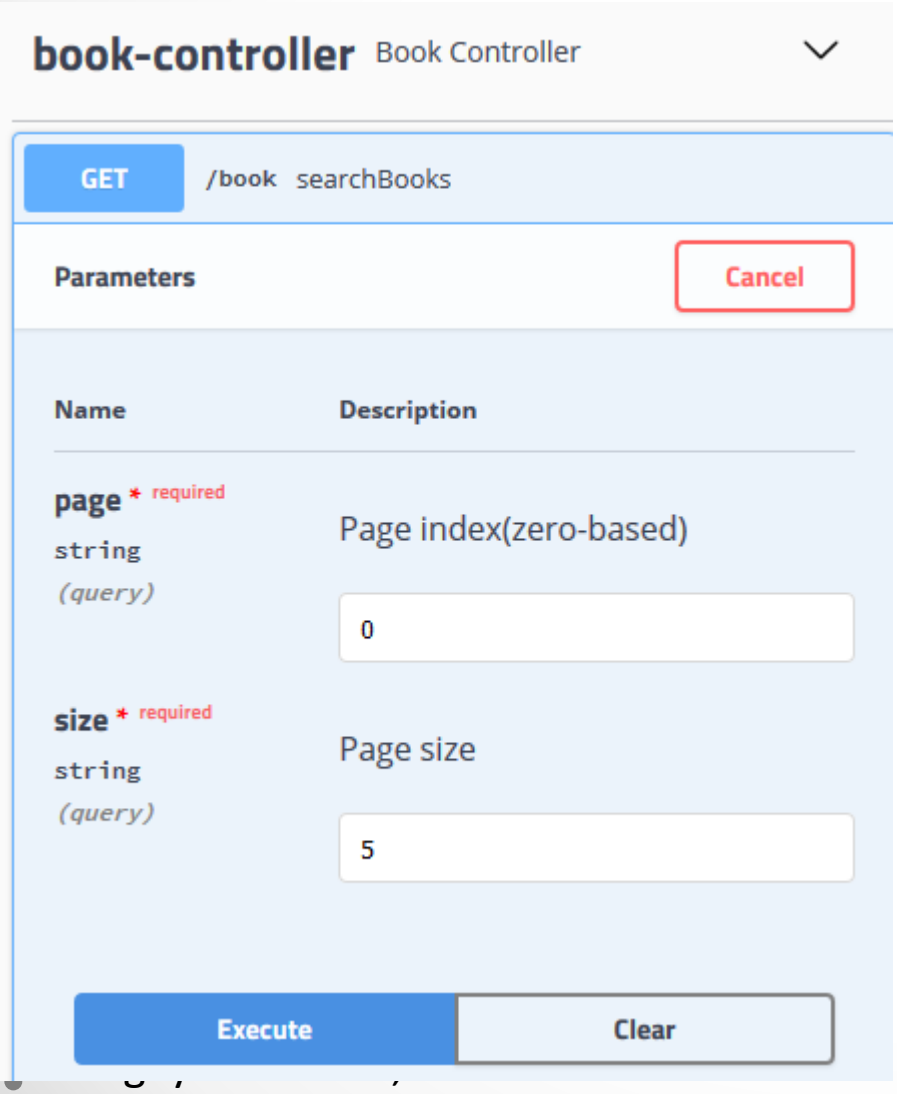
```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

Swagger UI



http://localhost:8080/swagger-ui.html

Swagger UI



http://localhost:8080/swagger-ui.html

Task 23. Swagger documentation

1. Add Springfox Swagger UI dependency into your project:
2. Start your application and open URL http://localhost:8080/swagger-ui.html. Review displayed services and their parameters.
3. Try to call some of your services.
4. How would you change Swagger UI default page?

Versioning

- ✓ **URI** versioning (LinkedIn, Yahoo, Salesforce)
- ✓ **URI parameter** versioning (Netflix, Facebook)
- ✓ **Accept header** versioning (GitHub)
- ✓ **Custom header** versioning (Microsoft Azure)

Versioning


```
/api/v1/article/1234
```

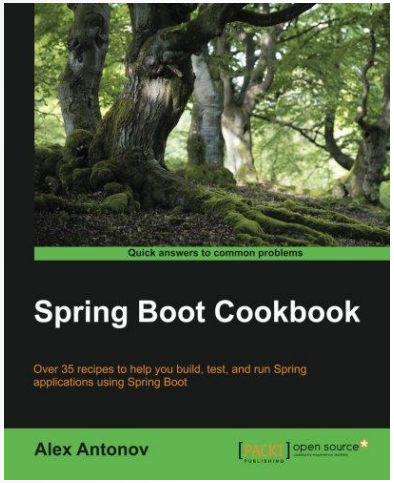
```
/api/v2/article/1234
```

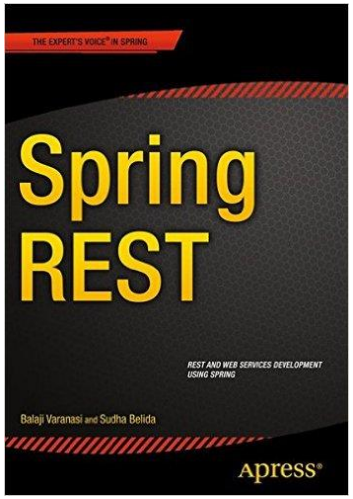
```
GET /api/article/1234 HTTP/1.1
Accept: application/vnd.api.article+xml; version=1.0
```

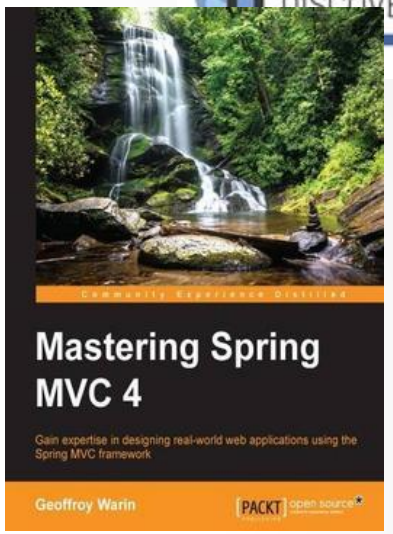
Future

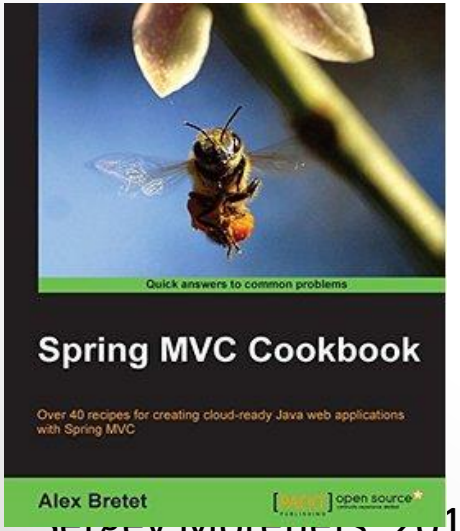
- ✓ Spring Data REST
- ✓ Server-side events

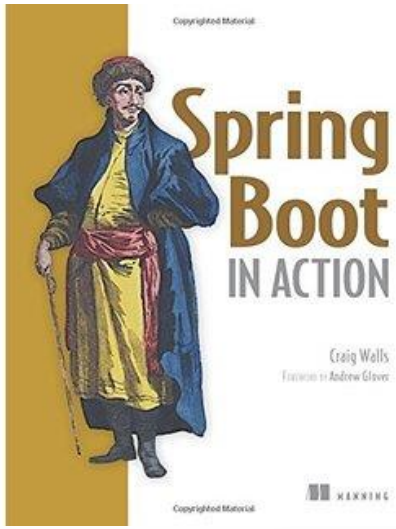
Books


Quick answers to common problems
Spring Boot Cookbook
Over 35 recipes to help you build, test, and run Spring applications using Spring Boot
Alex Antonov
[PACKT] open source


THE EXPERT'S VOICE® IN SPRING
Spring REST
REST AND WEB SERVICES DEVELOPMENT USING SPRING
Balaji Varanasi and Sudha Belida
Apress®


Community Experience Distilled
Mastering Spring MVC 4
Gain expertise in designing real-world web applications using the Spring MVC framework
Geoffroy Warin
[PACKT] open source


Quick answers to common problems
Spring MVC Cookbook
Over 40 recipes for creating cloud-ready Java web applications with Spring MVC
Alex Bretet
[packt] open source


Copyrighted Material
Spring Boot IN ACTION
Craig Walls
Copyrighted Material
MANNING


Spring Security
Eugen Paraschiv
[PACKT] VIDEO

Tutorials

https://github.com/Microsoft/api-guidelines/blob/master/Guidelines.md

✓ Sergey Morenets, sergey.morenets@gmail.com