REST services. Spring

Sergey Morenets

September, 29th 2018

DEVELOPER 12 YEARS

TRAINER 4 YEARS

WRITER 3 BOOKS

FOUNDER

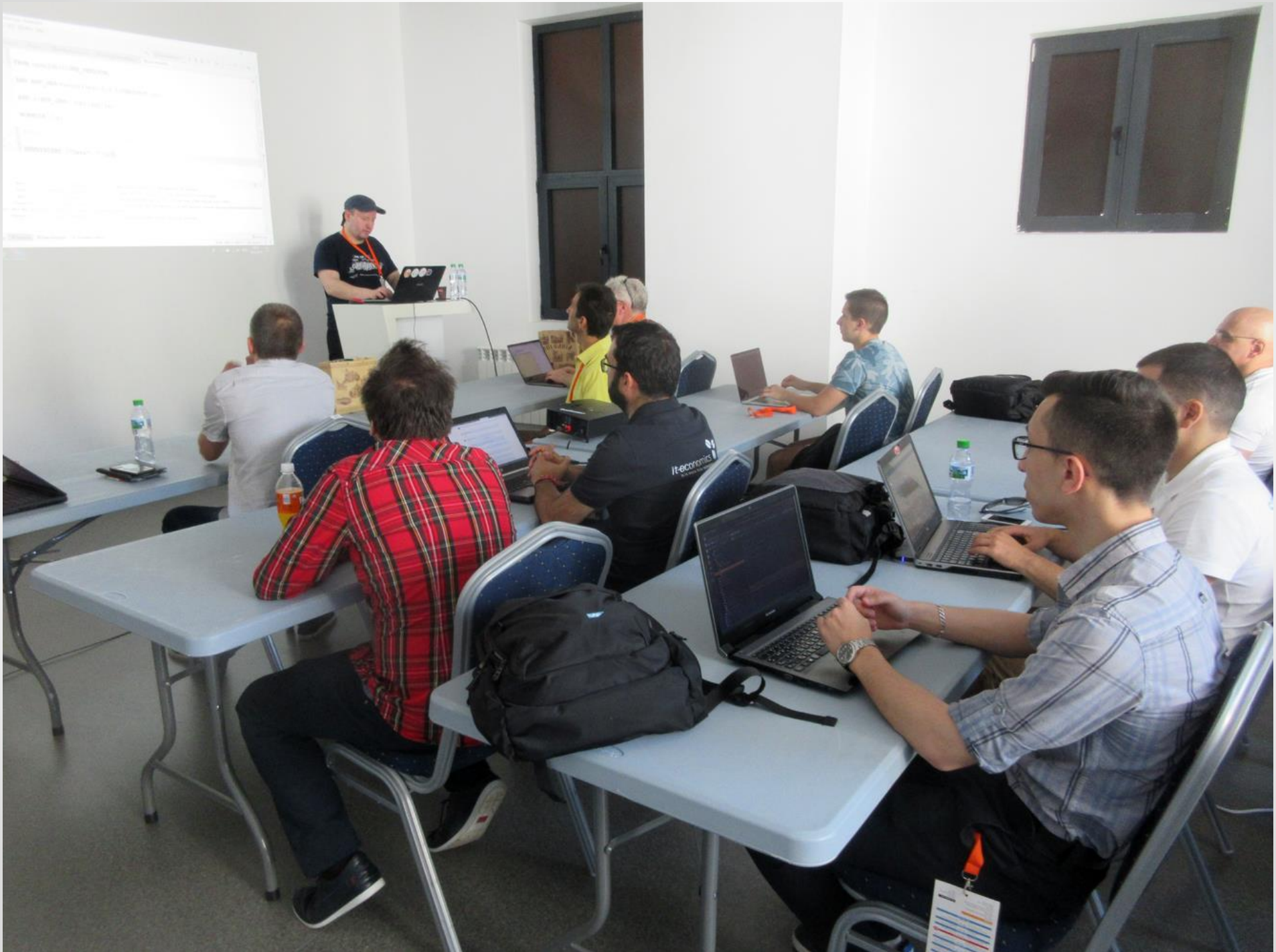
ITSimulator

SPEAKER

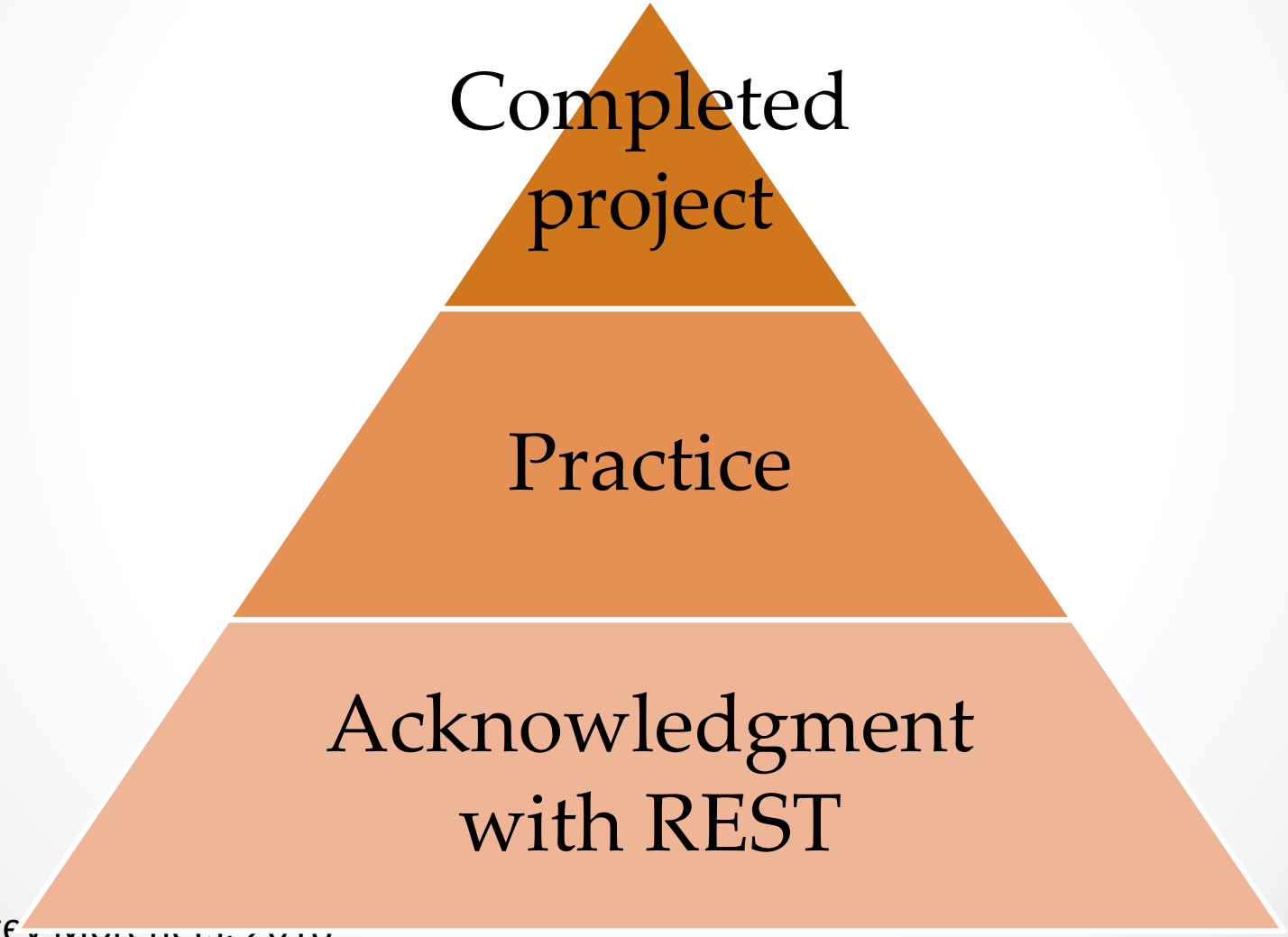
Java
frameworks
day

- Sergey Morenets, 2


@sergey-morenets
DEVOX
POLAND

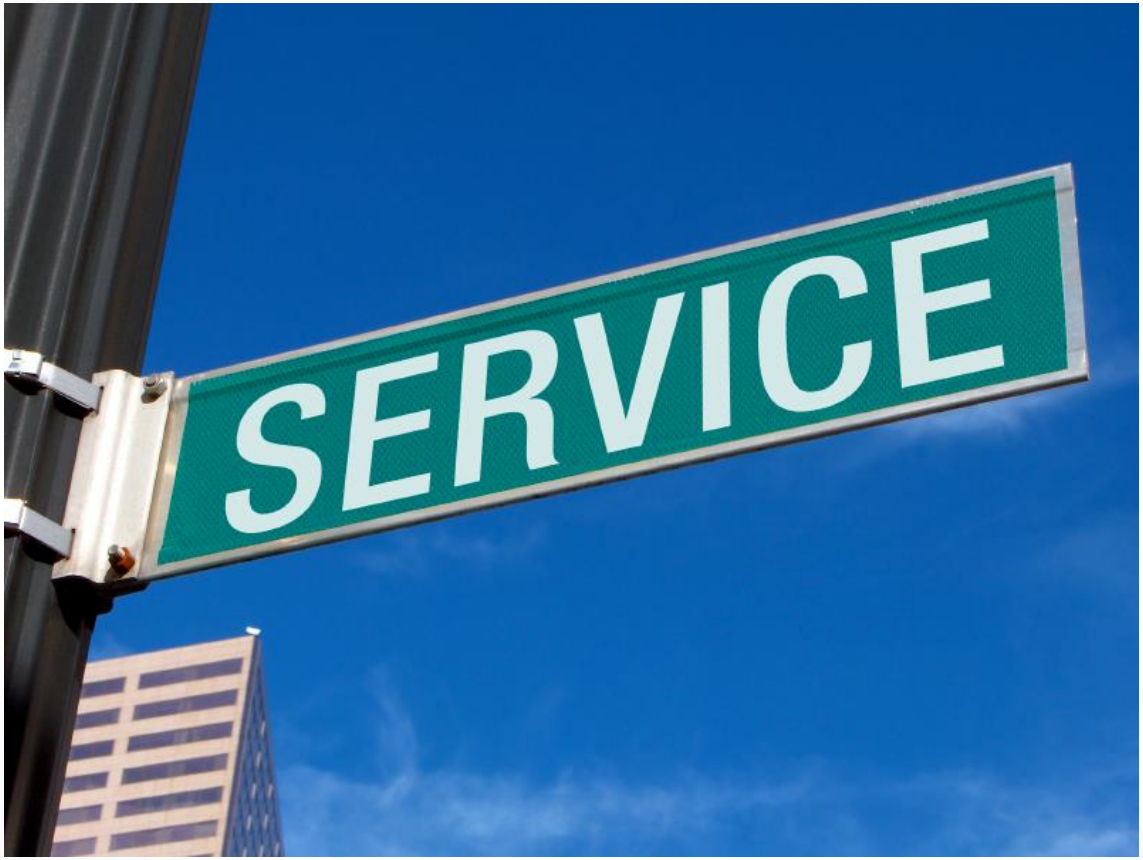

it-economics

Goals



Agenda

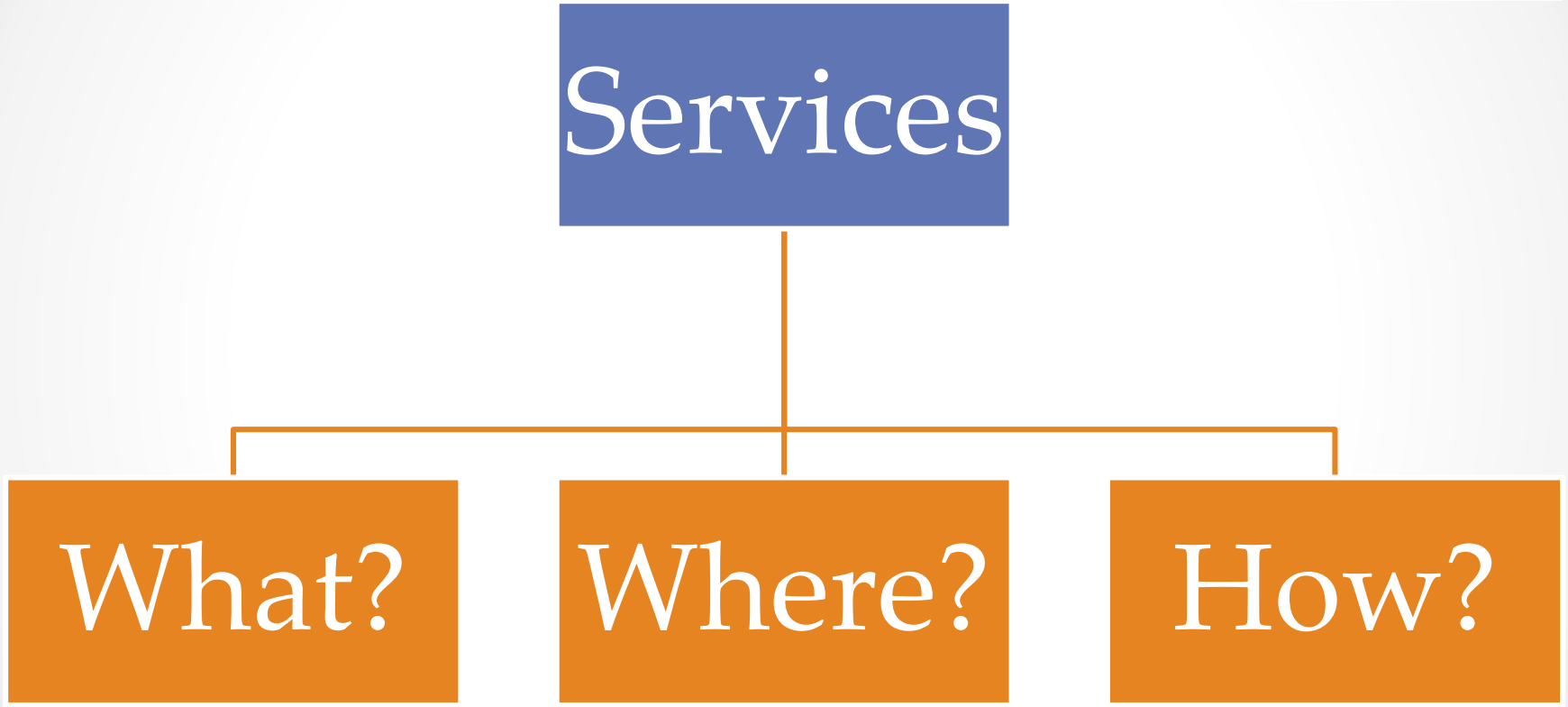
- ✓ REST and REST services
- ✓ REST over HTTP
- ✓ Spring Framework 5.1/Spring Boot 2 usage
- ✓ REST controllers
- ✓ Validation and pagination
- ✓ Service testing
- ✓ Exception handling
- ✓ HATEOAS
- ✓ Scaling and monitoring
- ✓ Caching
- ✓ Performance testing

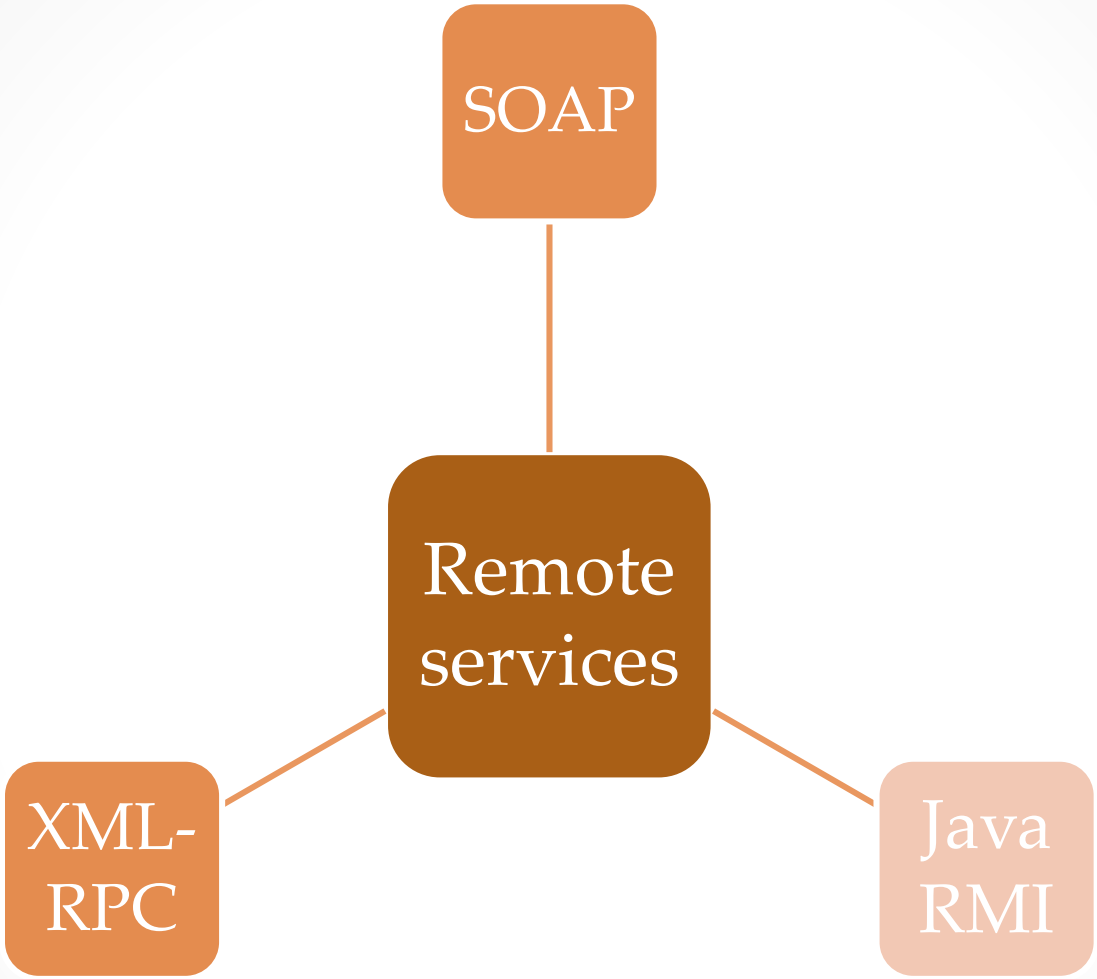


- Sergey Morenets, 2018

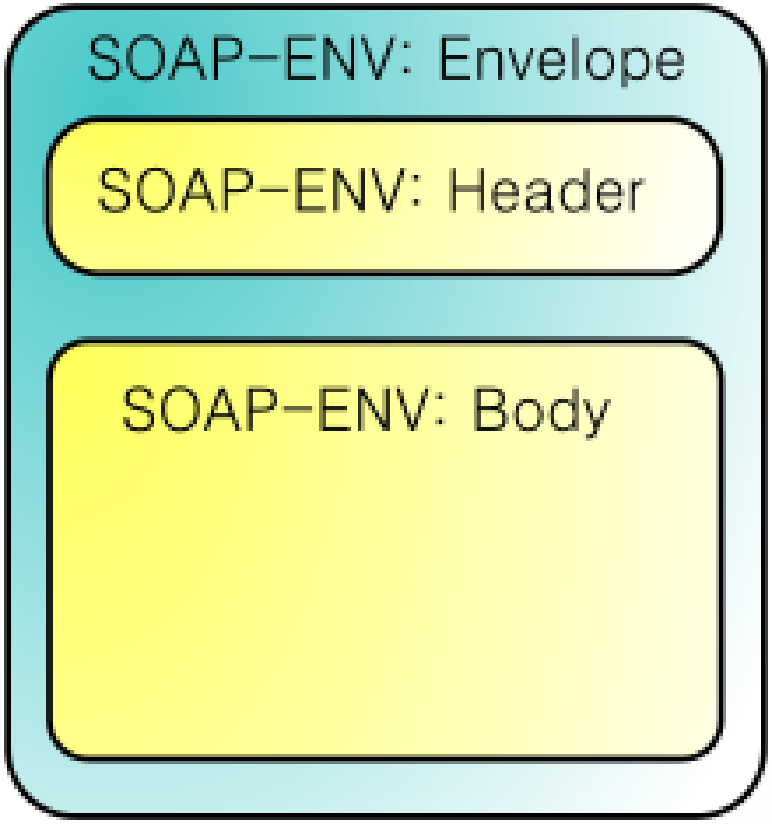
Service

- 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

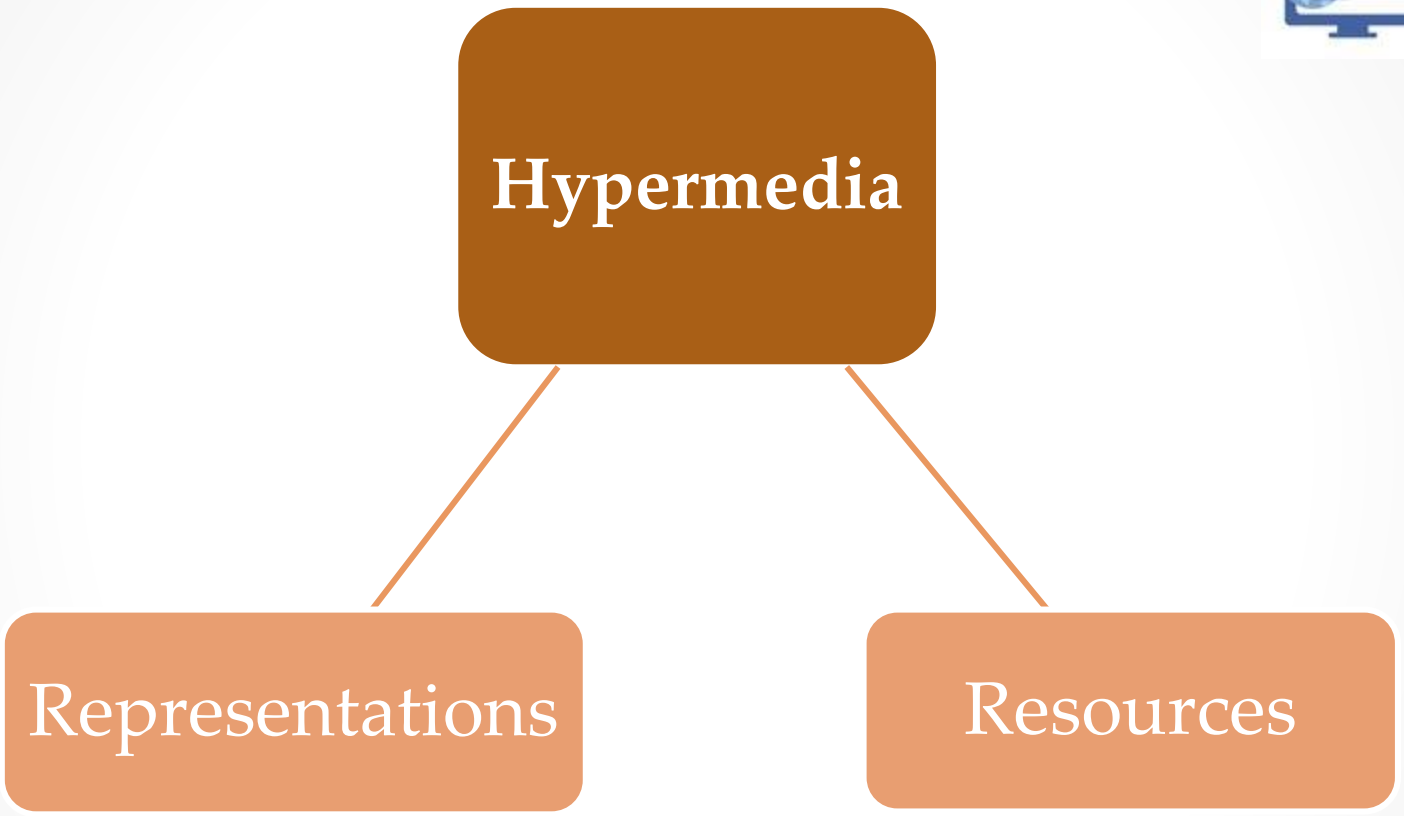
- Sergey Morenets, 2018

Constraints

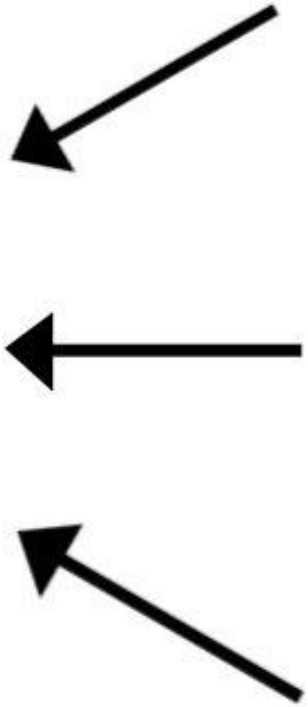
- ✓ Client-server
- ✓ Stateless
- ✓ Cacheable
- ✓ Layering
- ✓ Uniform interface

REST is not

- ✓ Protocol
- ✓ File format
- ✓ Development framework


Hypermedia
Representations
Resources

- Sergey Morenets, 2018

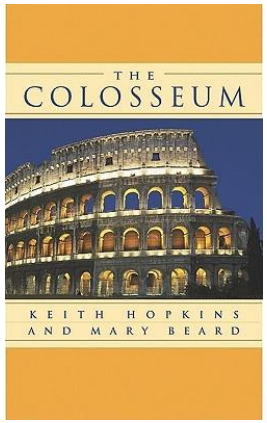


Colosseum

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

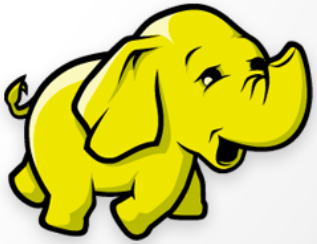
```
{
    "text":"helloWorld"
}
```

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

Representations

Apache Thrift ™

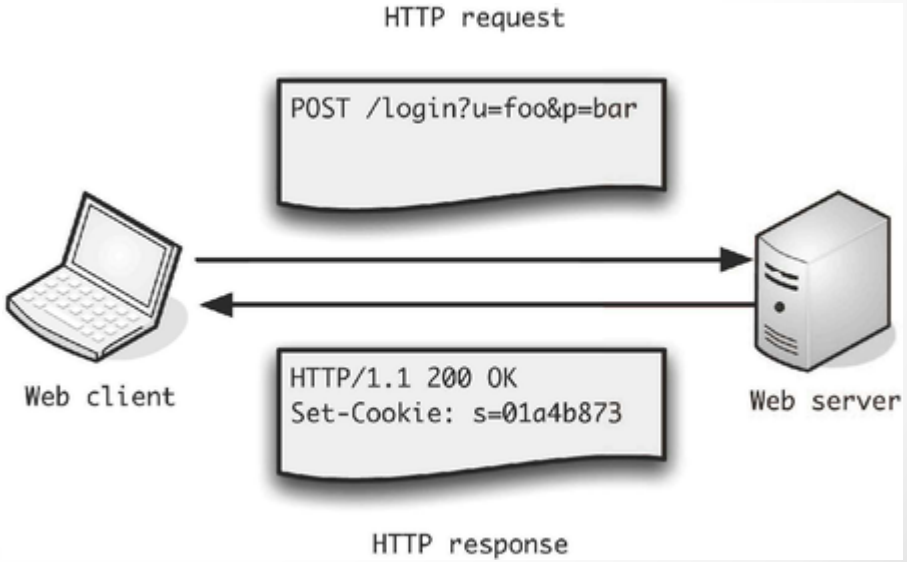
Protocol Buffers



- Sergey Morenets, 2018

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

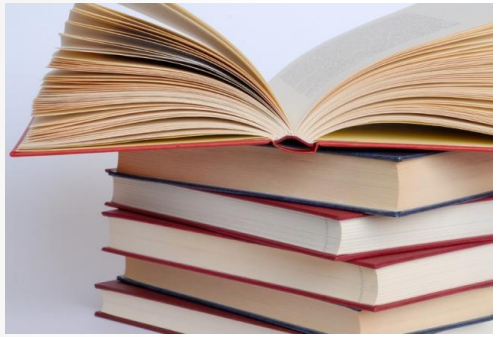
Goals

- ✓ 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

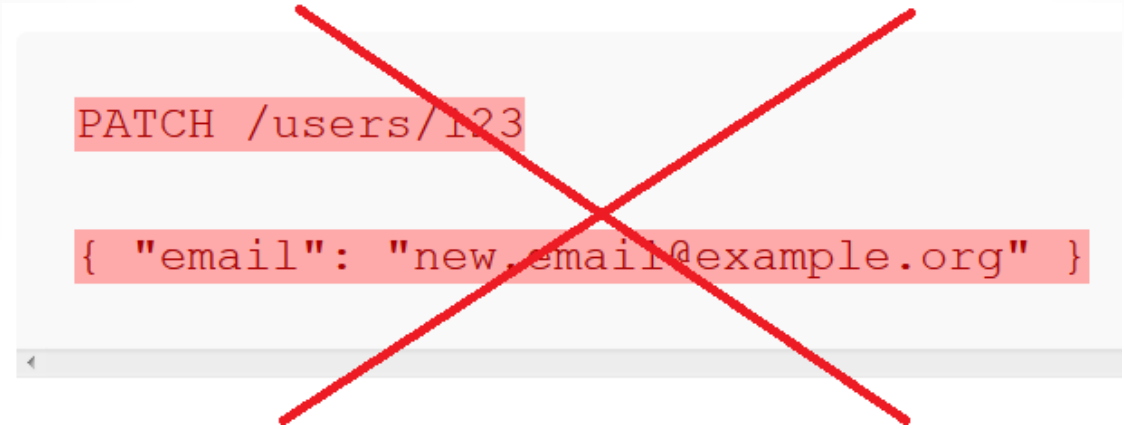
PUT /books

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>
```

- 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">-25.00</balance>
    <link rel="deposit" href="http://somebank.org/account/12345/deposit" />
</account>
```

- Sergey Morenets, 2018

Hypermedia

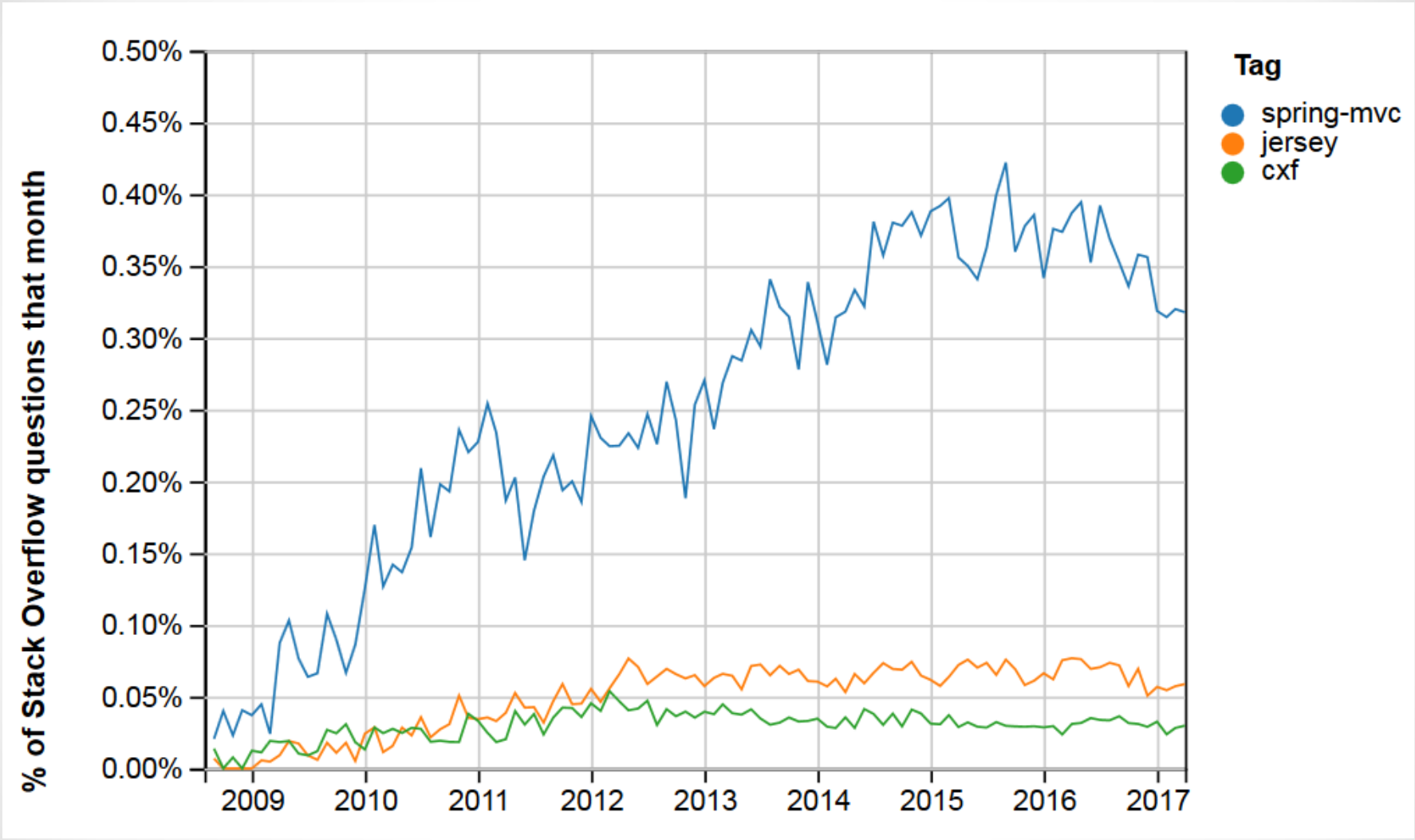

```
{
  "links": [
  {
    "href": "https://api.paypal.com/v1/payments/payment/PAY-1B569607296042 35TKQQIYVY
",
    "rel": "self",
    "method": "GET"
  },
  {
    "href": "https://api.paypal.com/v1/payments/payment/PAY-1B569607296042 35TKQQIYVY
/execute",
    "rel": "execute",
    "method": "POST"
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

- Sergey Morenets, 2018

Implementations

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

- ✓ Ratpack
- ✓ Spark

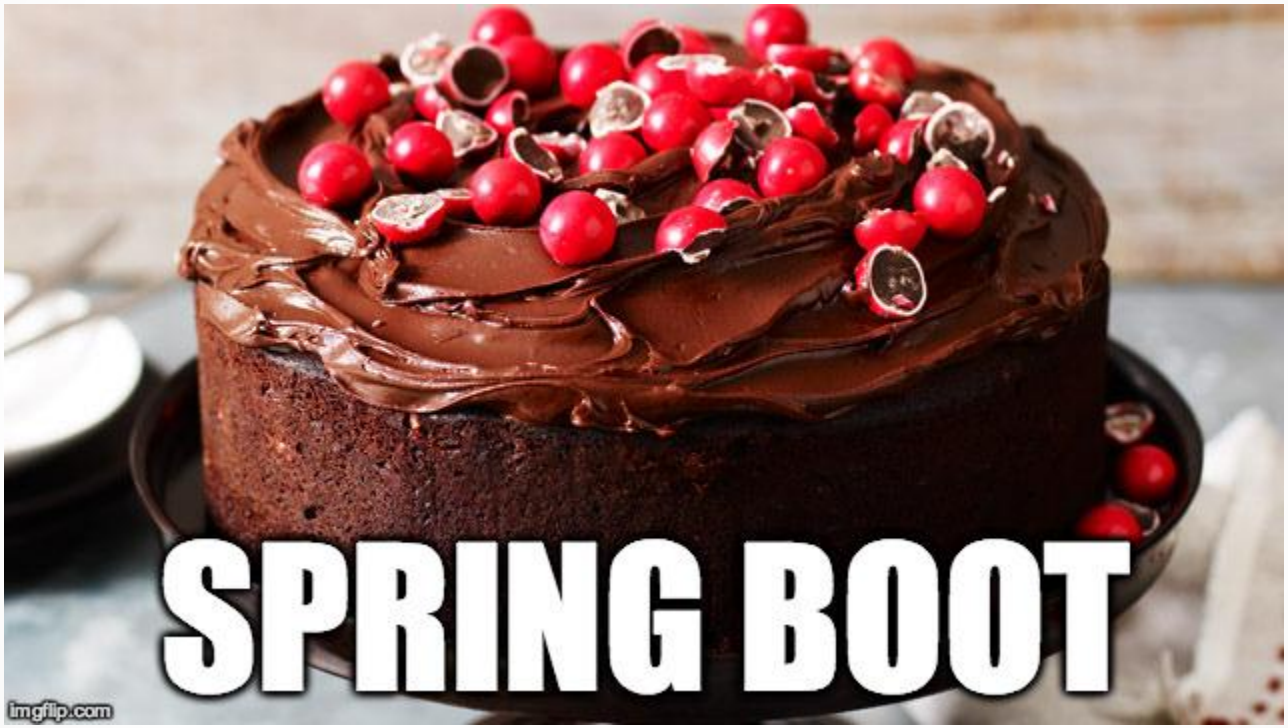


- Sergey Morenets, 2018

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

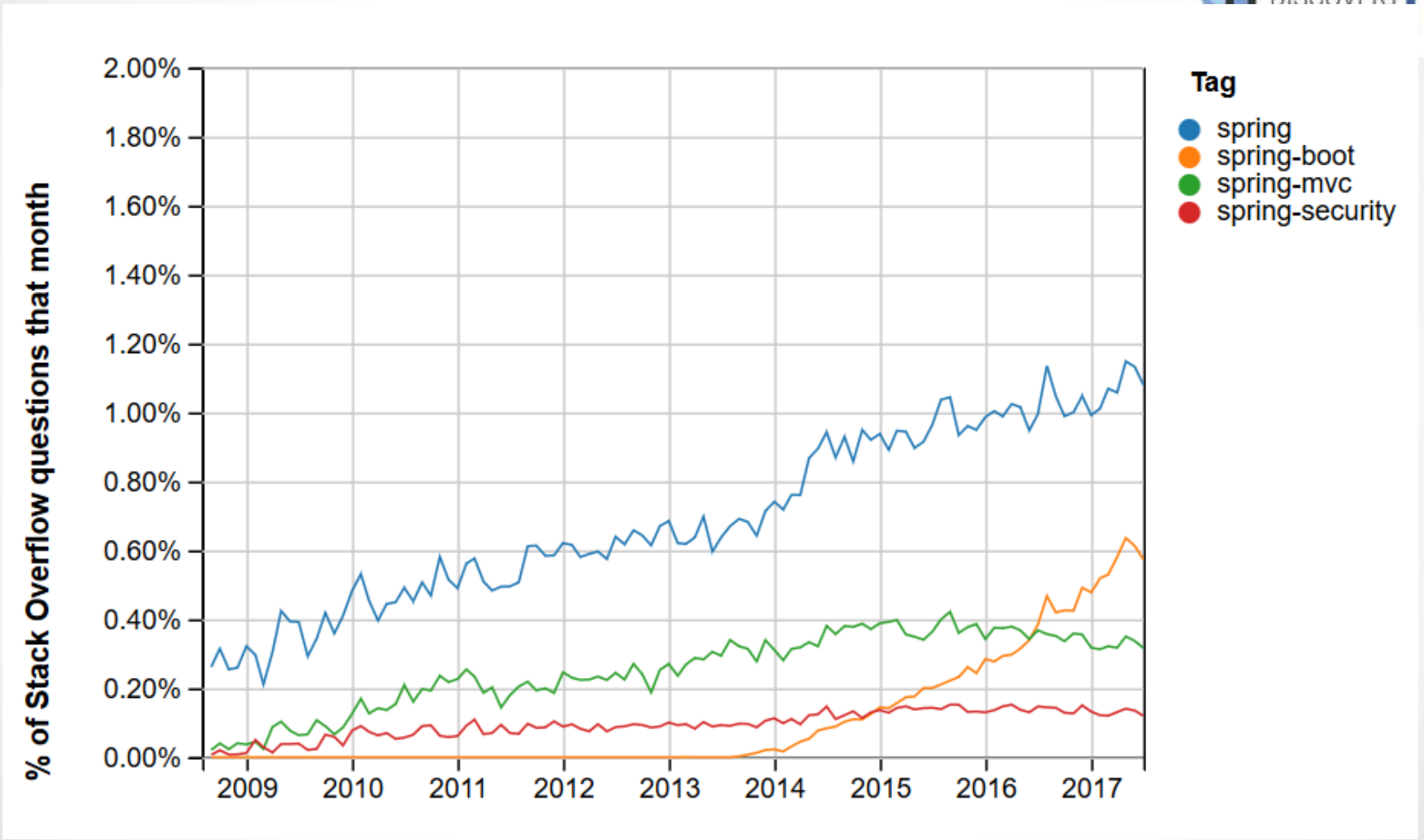
- Sergey Morenets, 2018



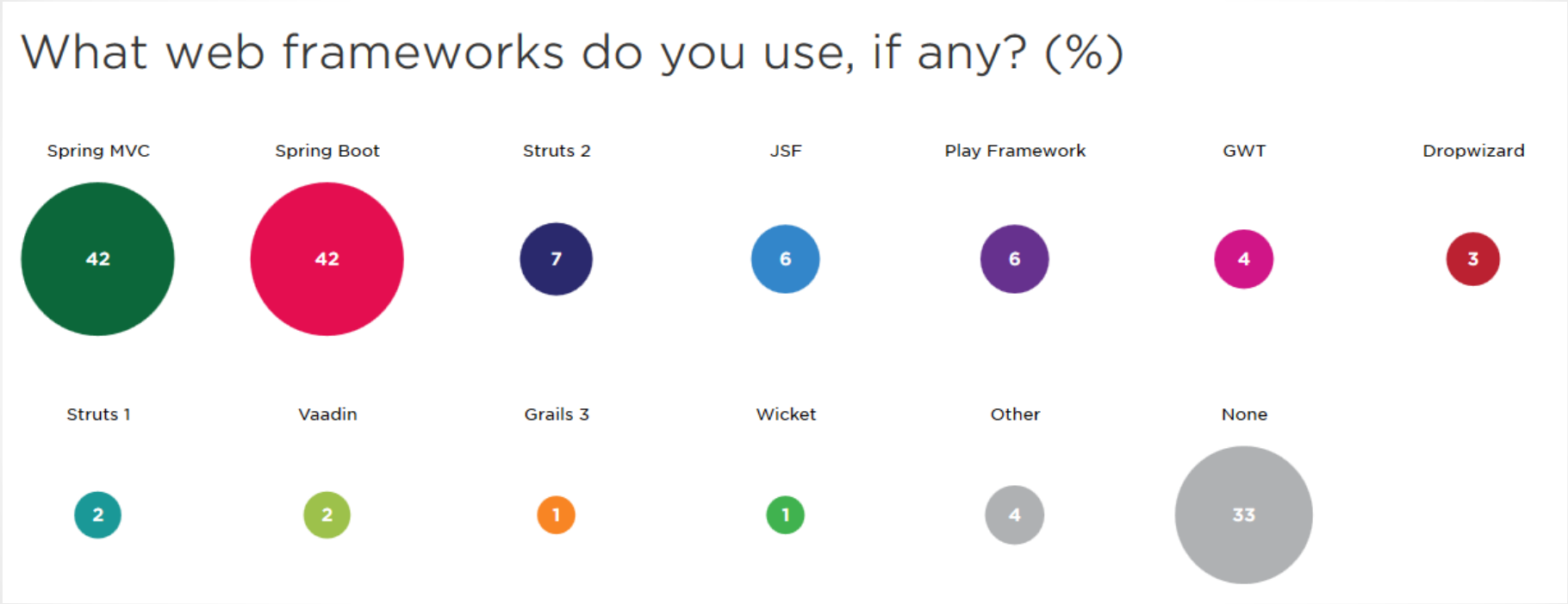
- Sergey Morenets, 2018

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



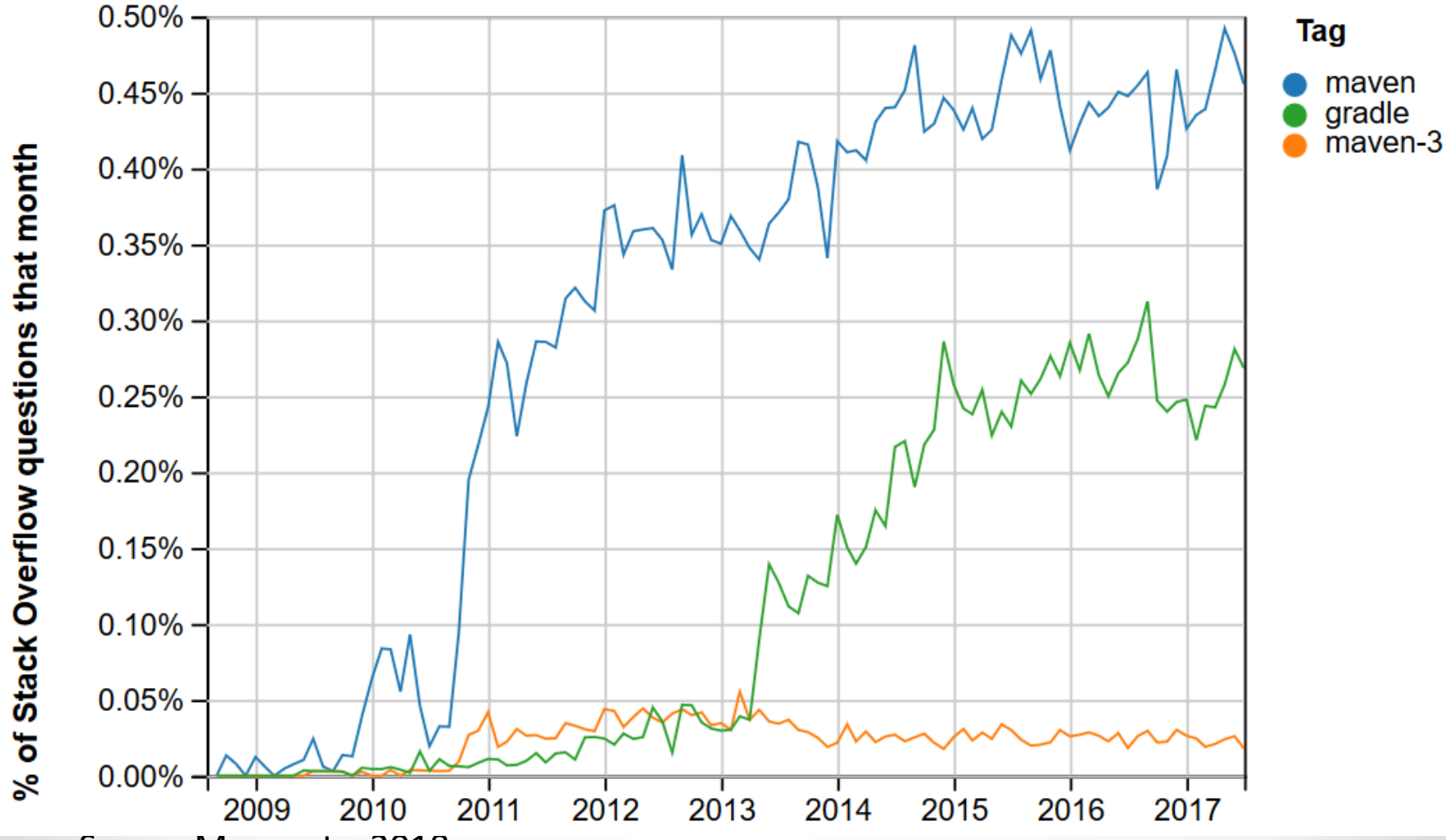
- Sergey Morenets, 2018



- Sergey Morenets, 2018

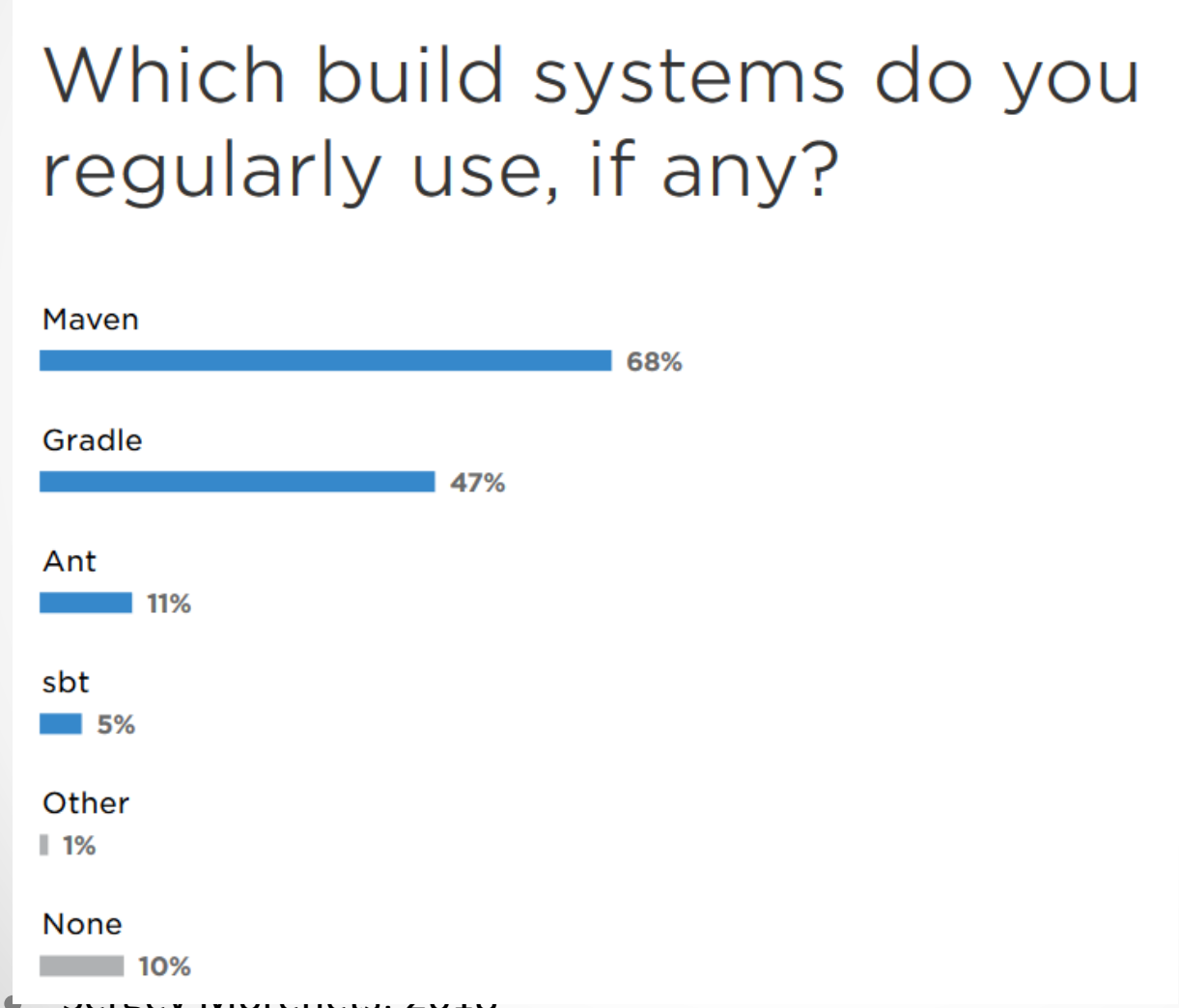
Build management

Maven vs Gradle

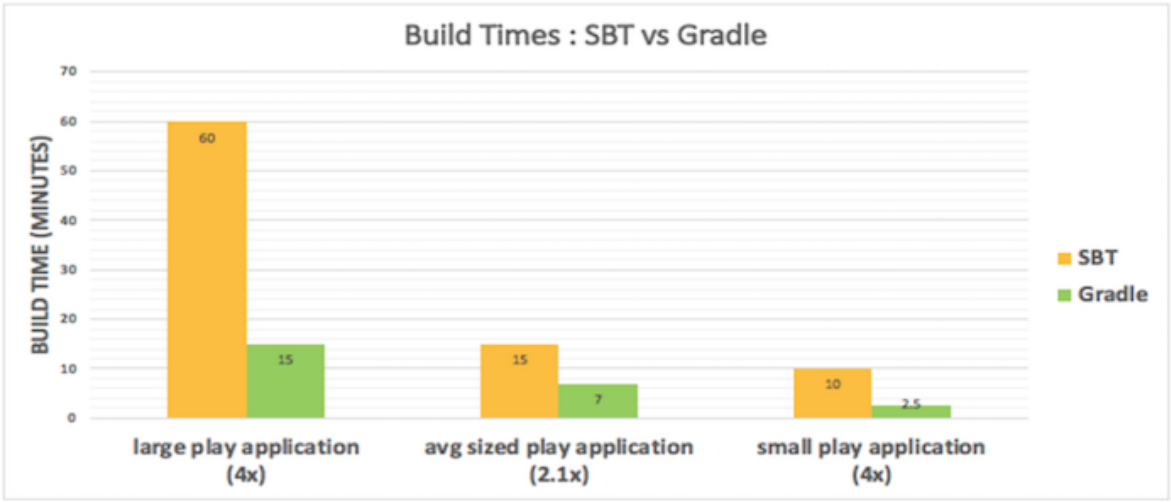


- Sergey Morenets, 2018

Maven vs Gradle



Gradle vs SBT

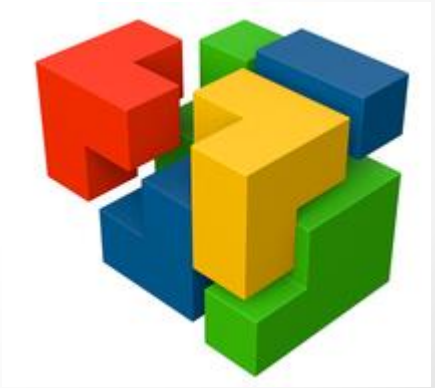


https://engineering.linkedin.com/blog/2018/07/how-we-improved-build-time-by-400-percent

- 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

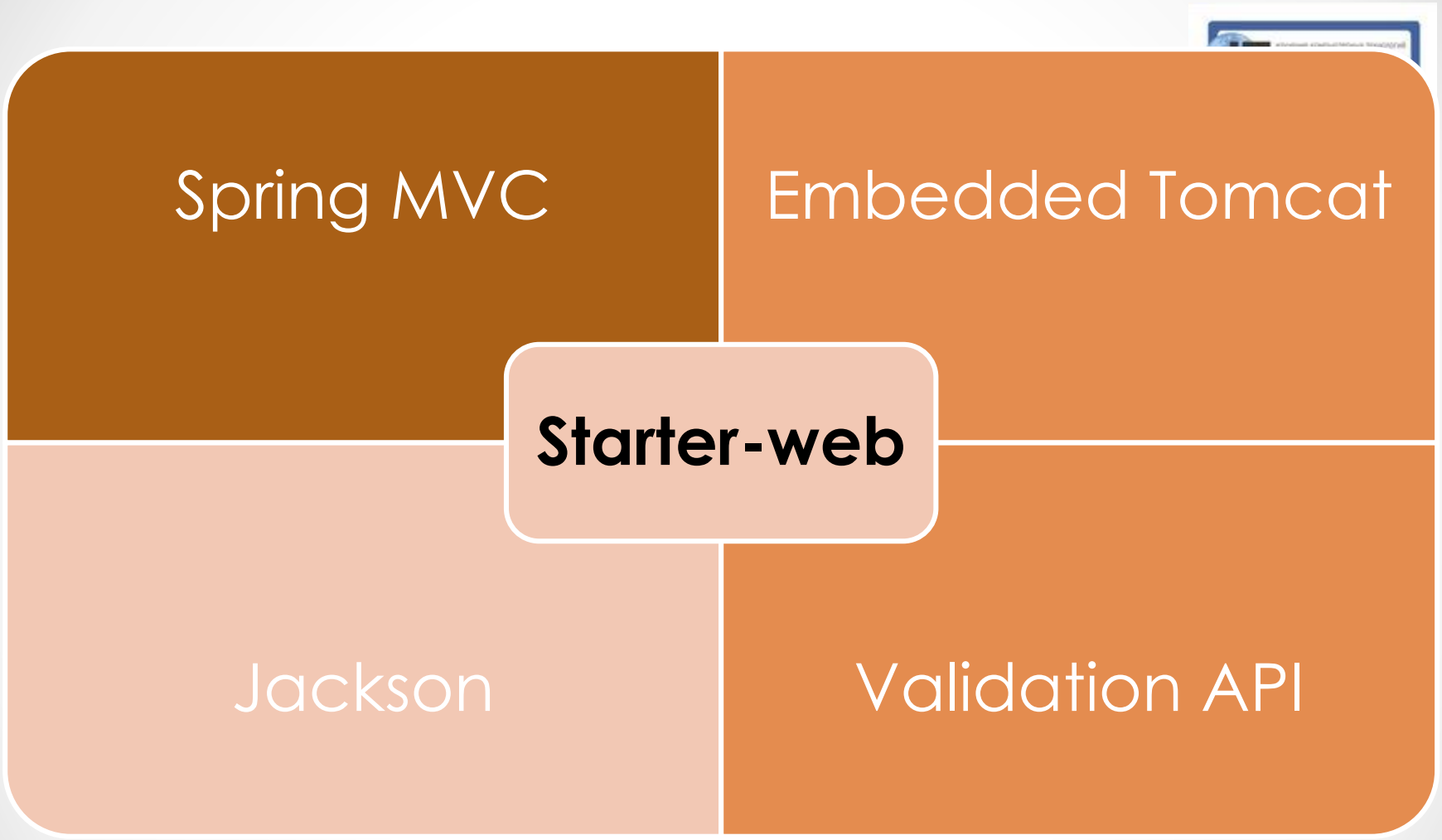
Build dependencies


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

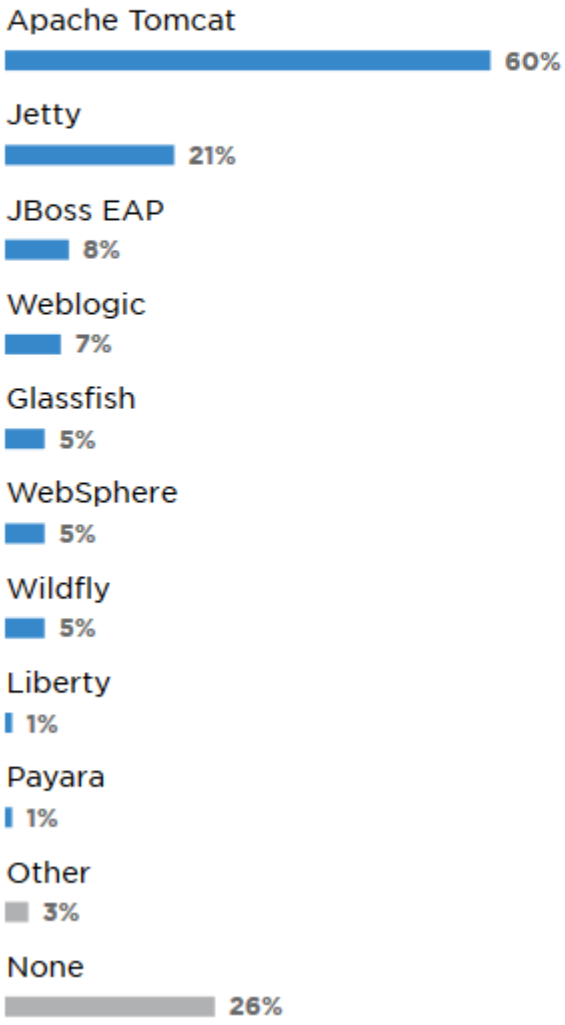
Maven

```
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?

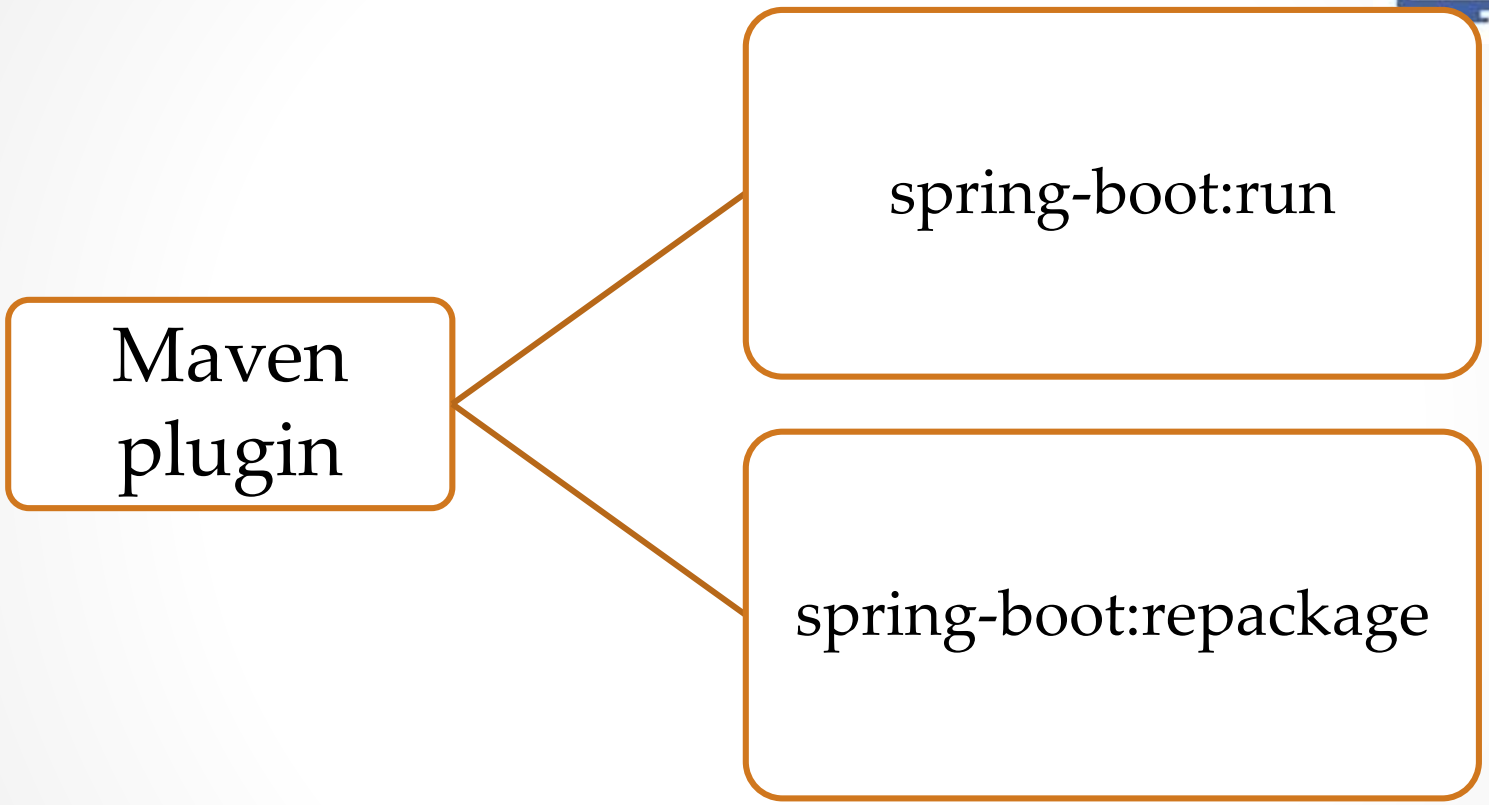


Spring configuration

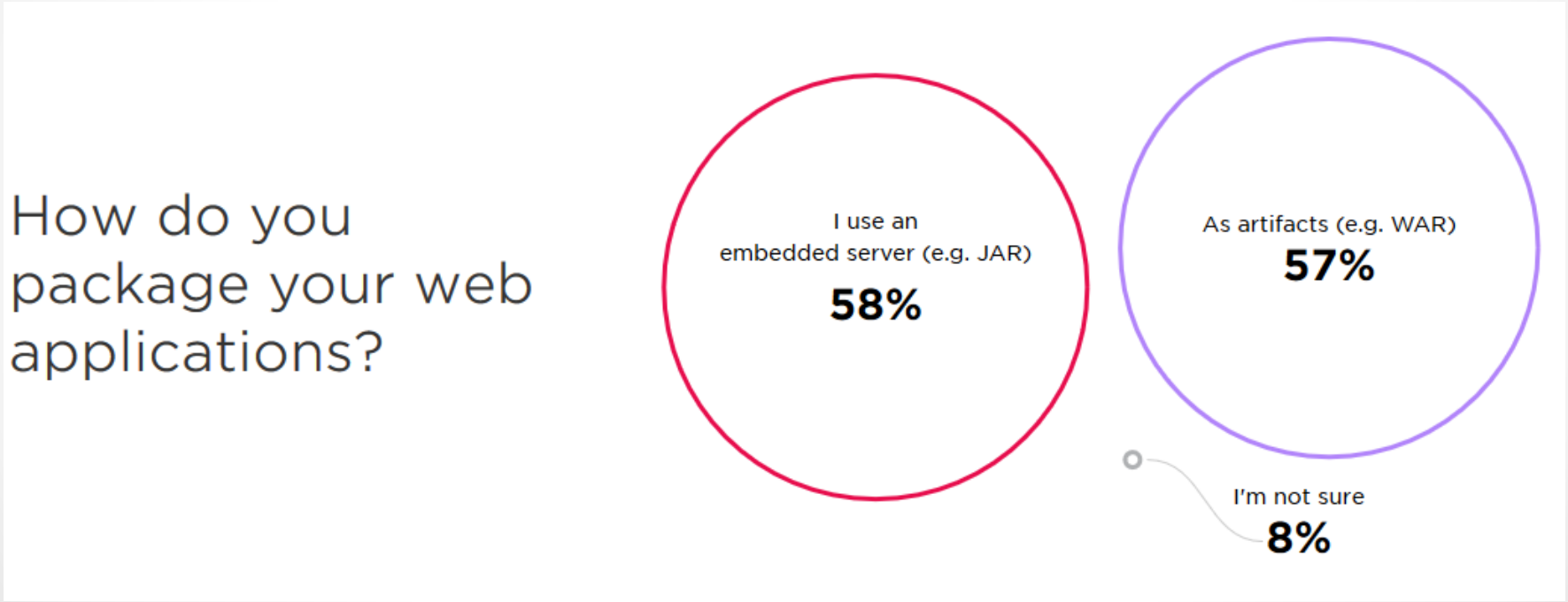

```
@SpringBootApplication
public class RestApplication {
    public static void main(String[] args) {
        SpringApplication.run(
                RestApplication.class, args);
    }
}
```

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



- 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>
```

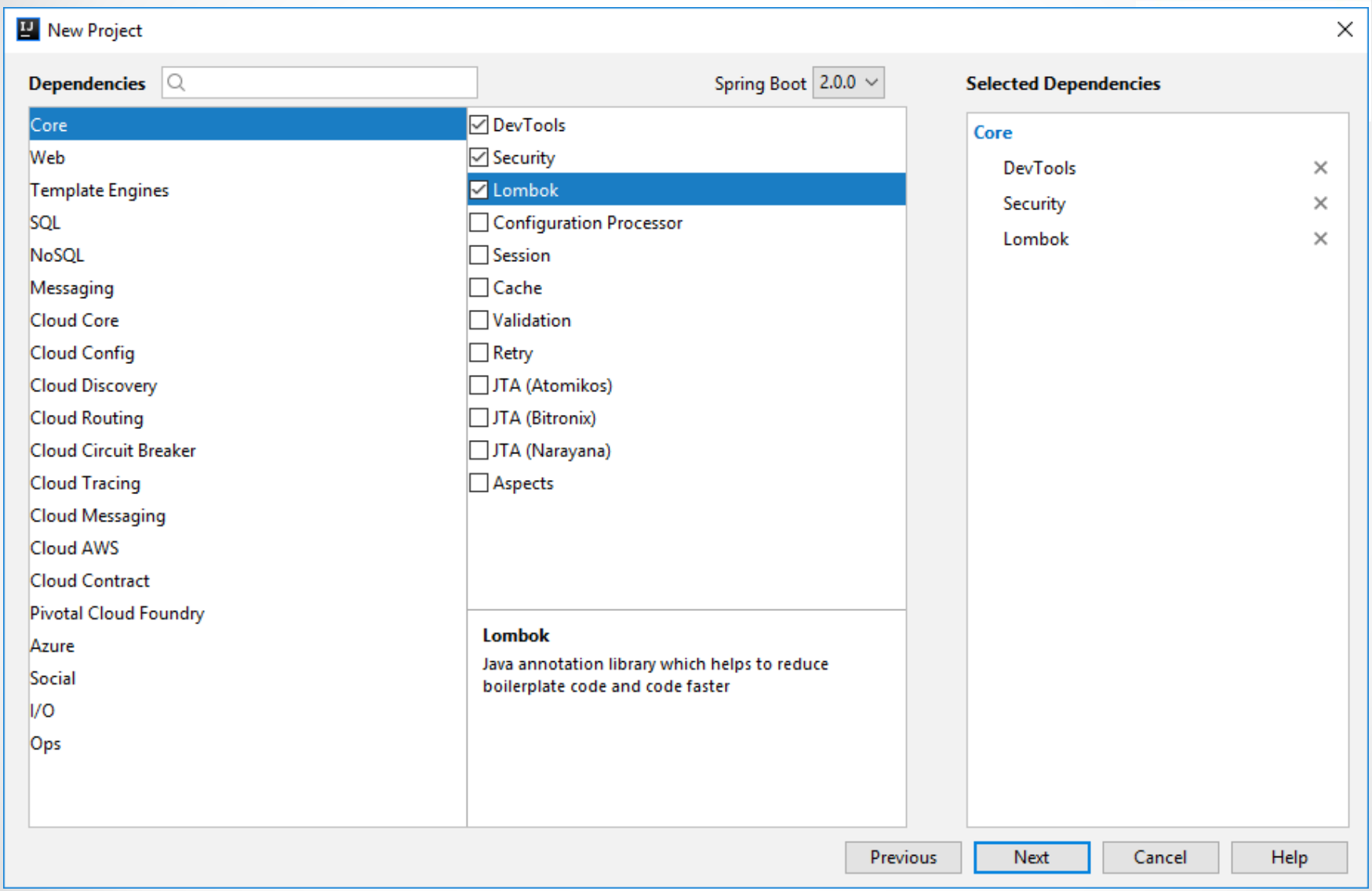
spring-boot-dependencies.pom.xml

Dependency management. Gradle


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

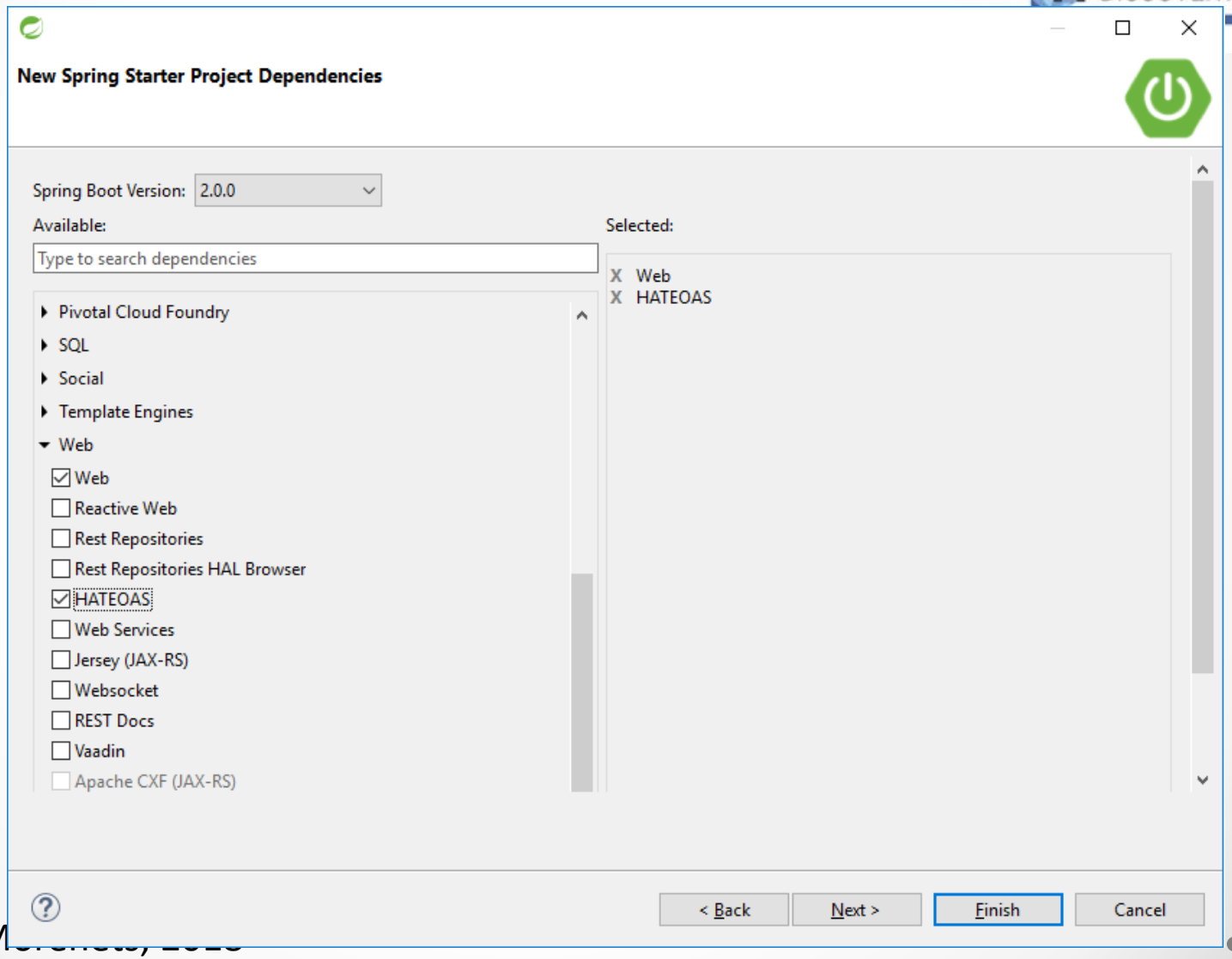
```
dependencies {
    compile 'org.springframework.boot:spring-boot-starter-web'
    compile 'org.apache.commons:commons-lang3'
}
```

IDE



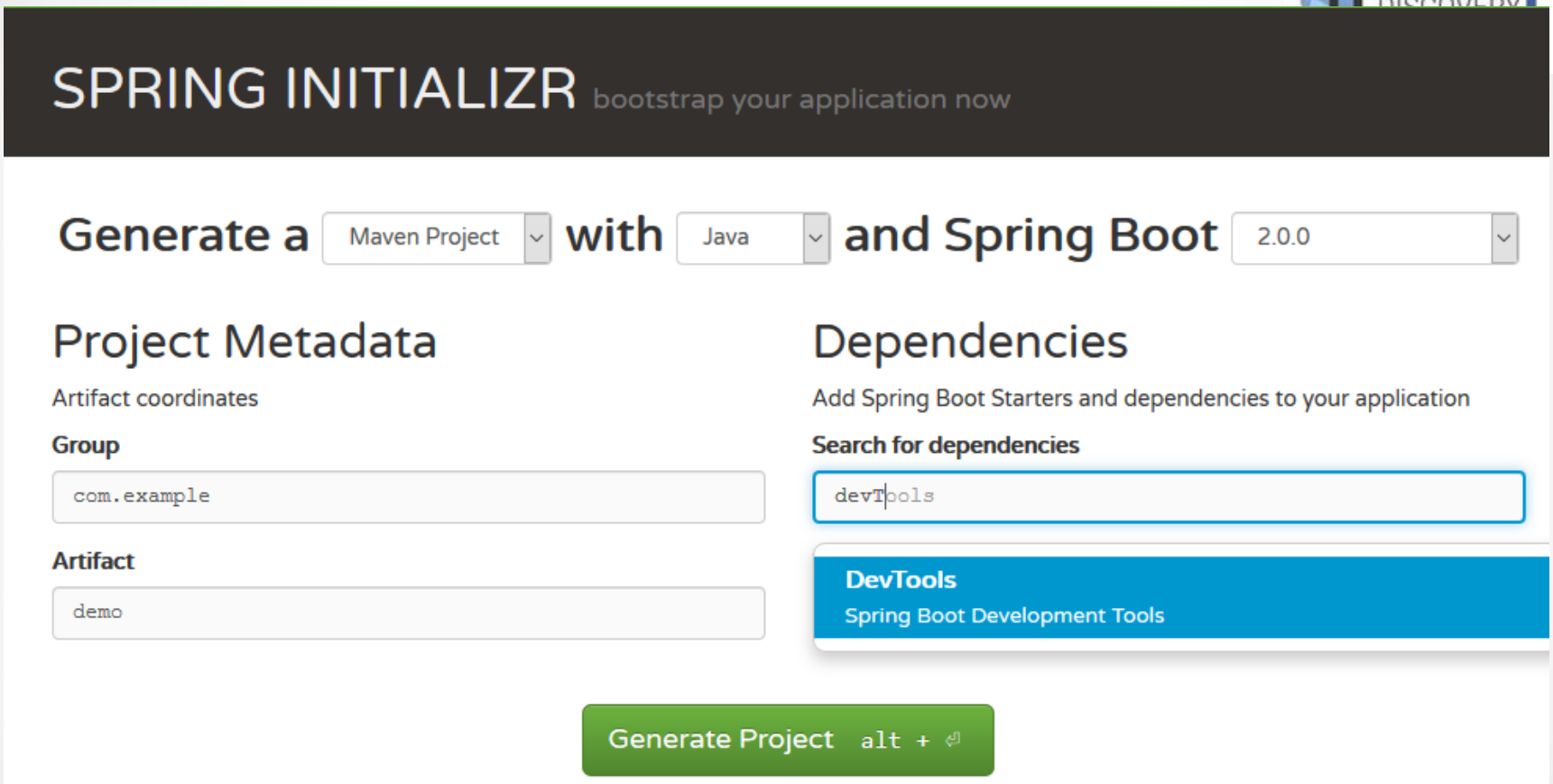
- Sergey Morenets, 2018

Spring Boot Starter. STS



- Sergey M[illegible]

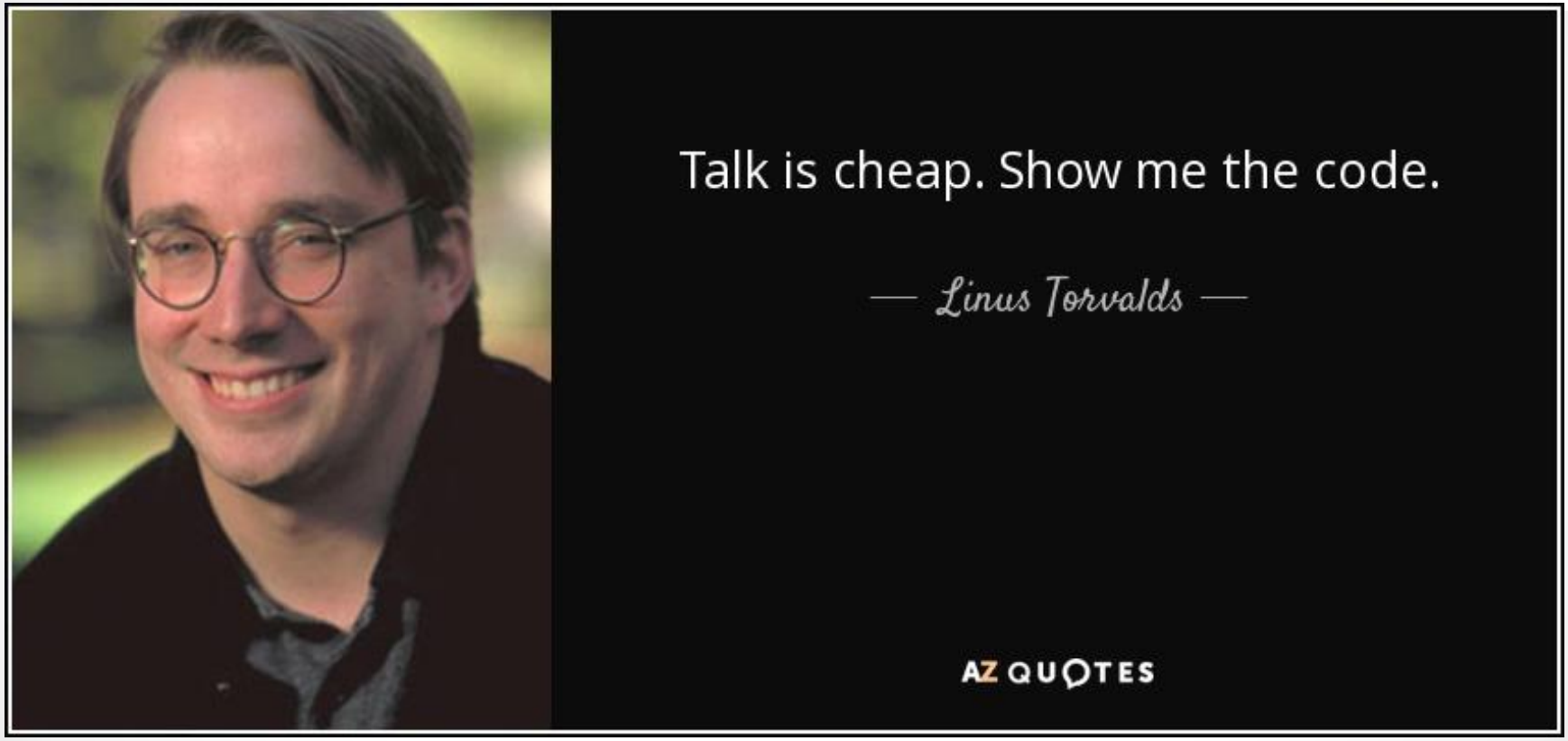
Spring Initializr



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

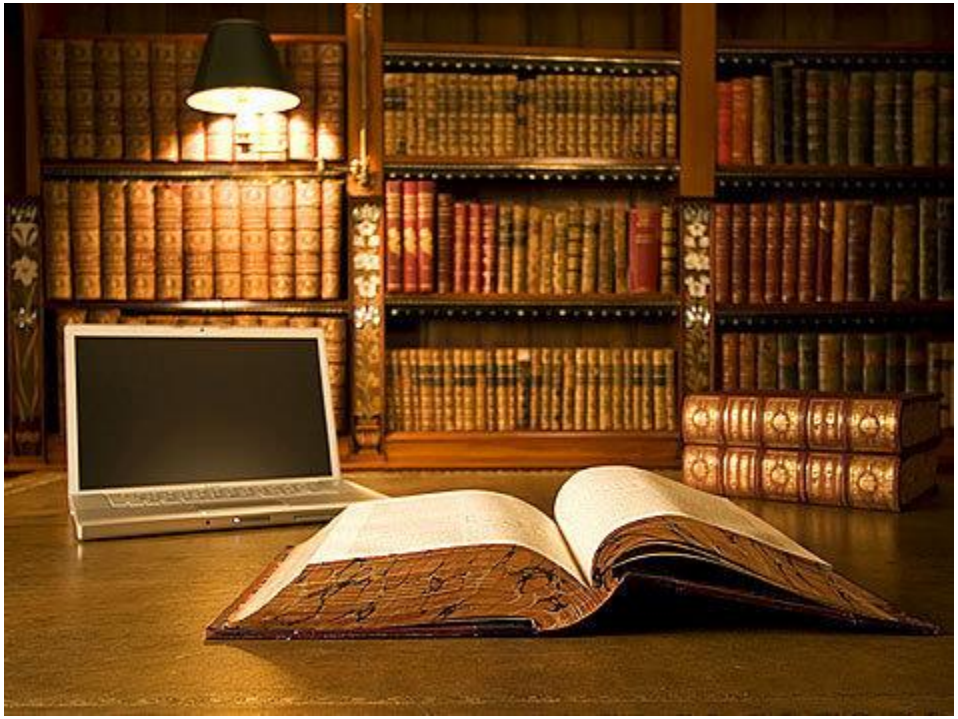


- Sergey Morenets, 2018

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



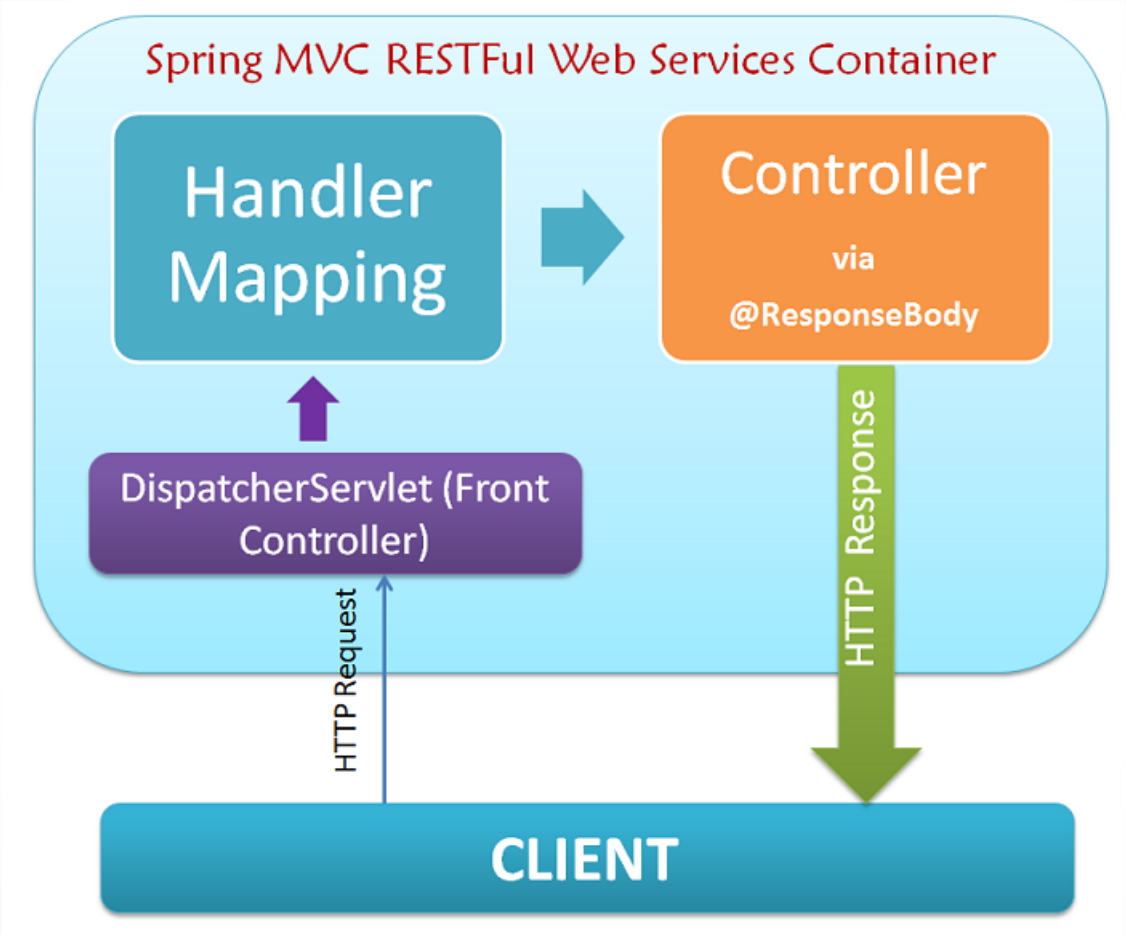
- Sergey Morenets, 2018

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()"}
    ]
  }
}}
```

- Sergey Morenets, 2018

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

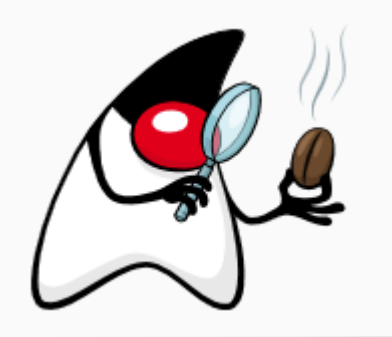

```
@RestController
@RequestMapping("/book")
public class BookController {
    @Autowired
    private BookRepository bookRepository;

}
```

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;
}
```

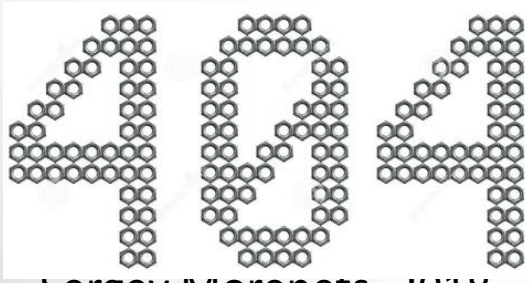
```
@PostMapping
public Book saveBook(@Valid @RequestBody Book book) {
    return book;
}
```

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();
    }
}
```

- Sergey Morenets, 2018

Error handling. Custom exceptions


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

    private static final long serialVersionUID =
            3591666710943050205L;

}
```

- Sergey Morenets, 2018

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);
    }
}
```

- Sergey Morenets, 2018

Task 9. Error handling

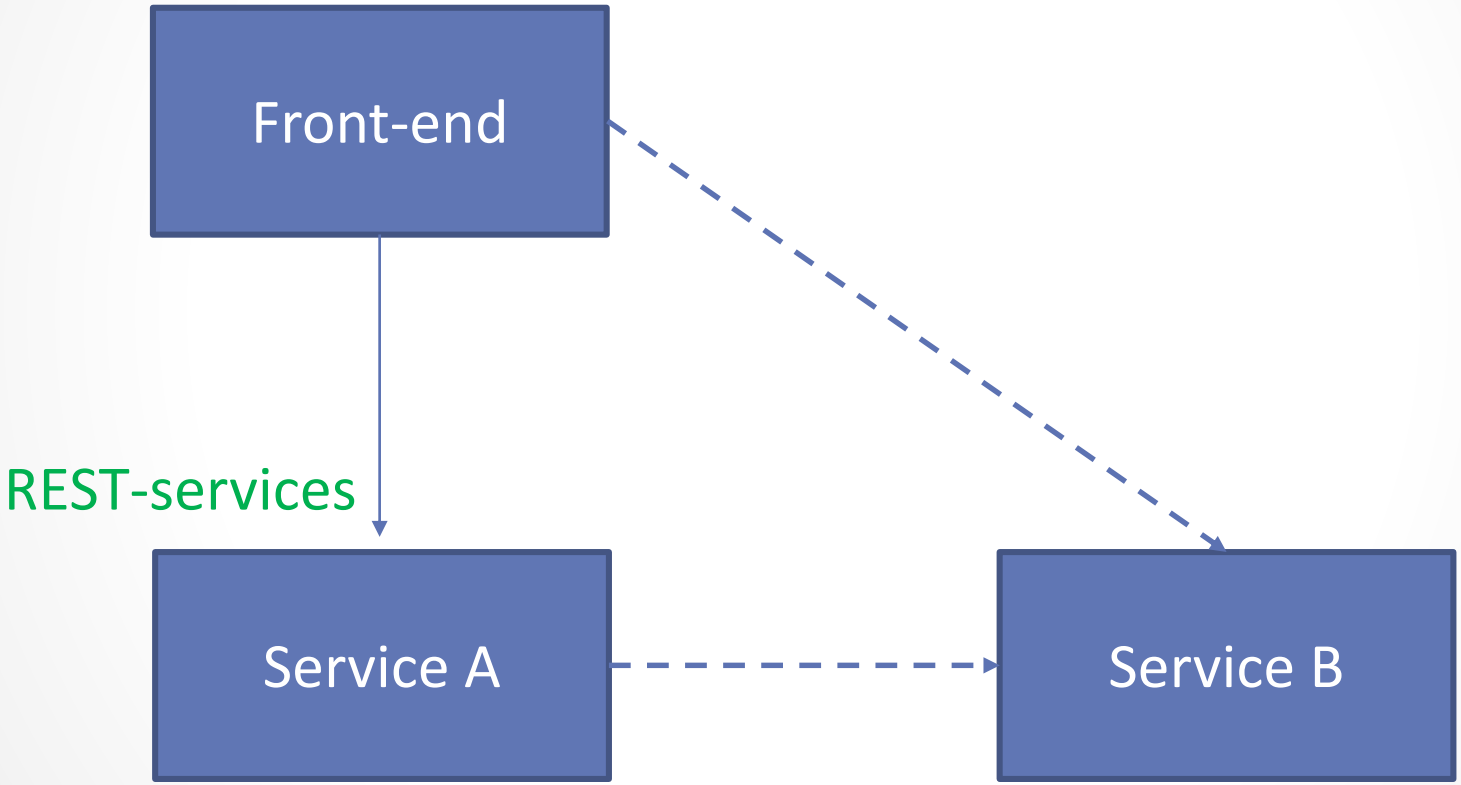
1. Add common error handling in your controller using three approaches:

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

REST support on client side

Sergey Morenets, 2018

Services communication



- Sergey Morenets, 2018

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());
    }
}
```

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());
    }
}
```

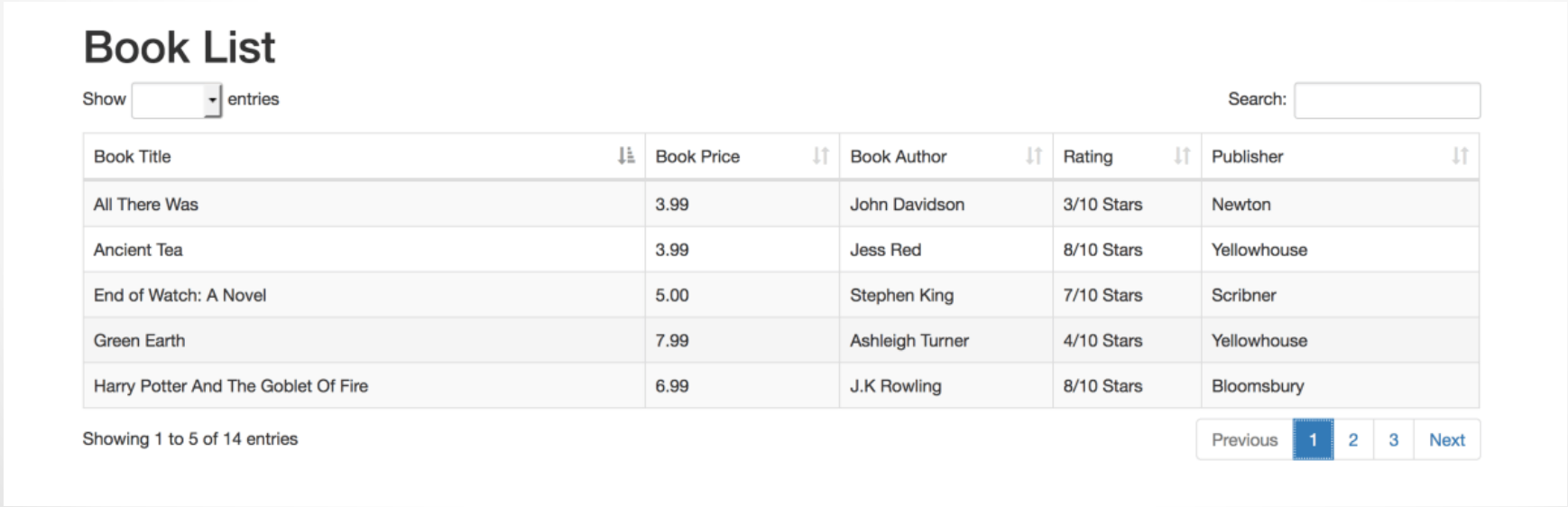
Task 10. RestTemplate

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

✓ Sergey Morenets, sergey.morenets@gmail.com