REST services. Spring

Sergey Morenets

March, 3rd 2018

DEVELOPER 12 YEARS

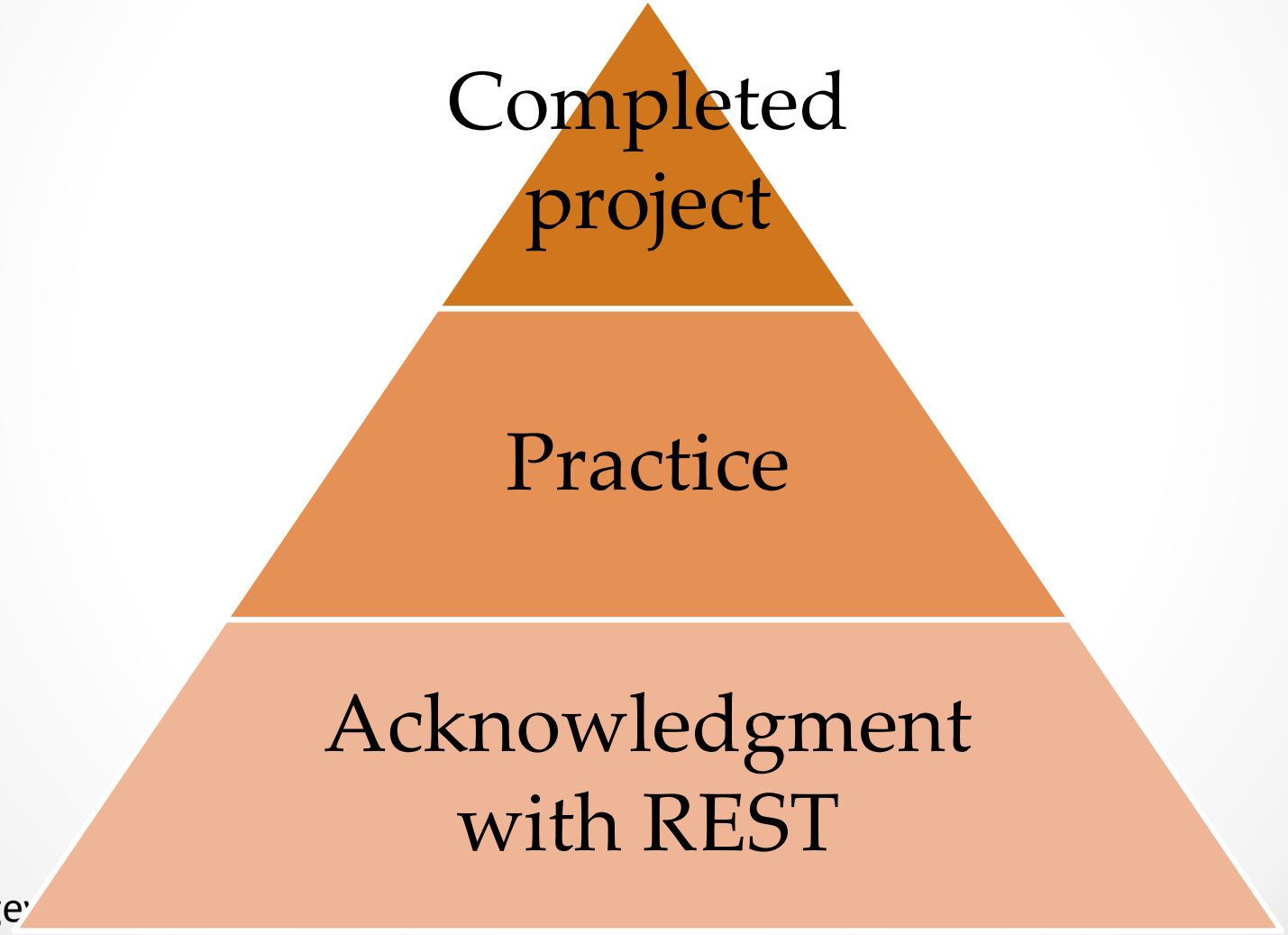
TRAINER 4 YEARS

WRITER 3 BOOKS

FOUNDER

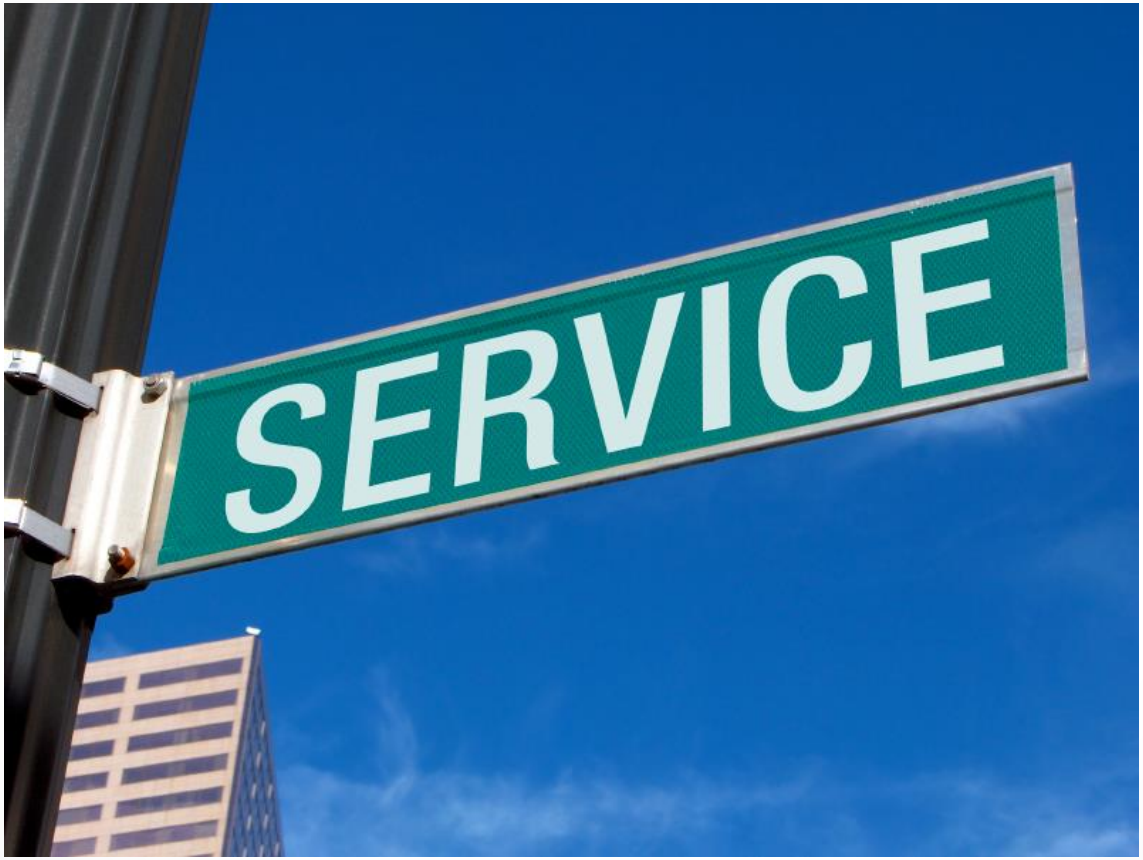
SPEAKER

Goals



Agenda

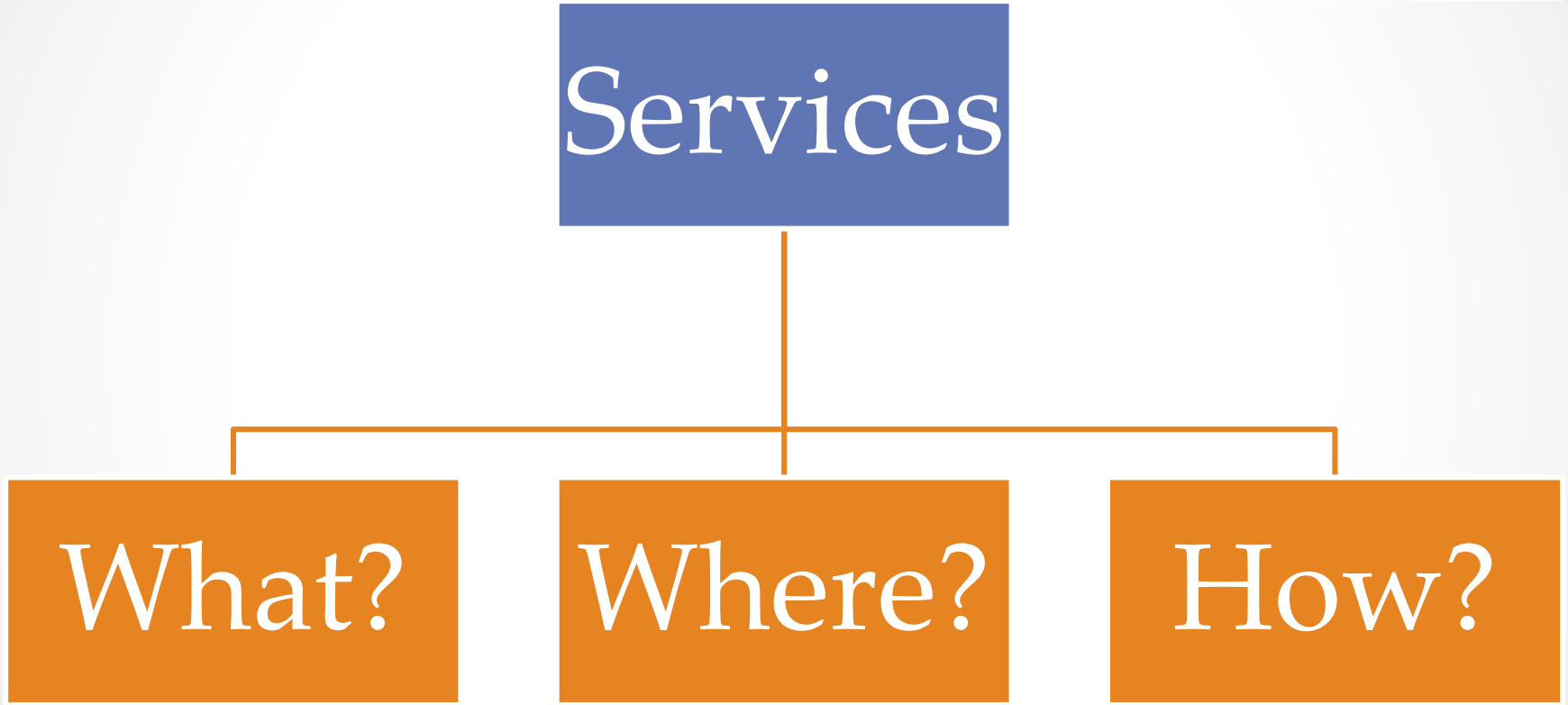
- ✓ REST and REST services
- ✓ REST over HTTP
- ✓ Spring 5/Spring Boot 2 usage
- ✓ REST controllers
- ✓ Service testing
- ✓ Exception handling
- ✓ HATEOAS
- ✓ Authentication
- ✓ Scaling
- ✓ Security/performance testing

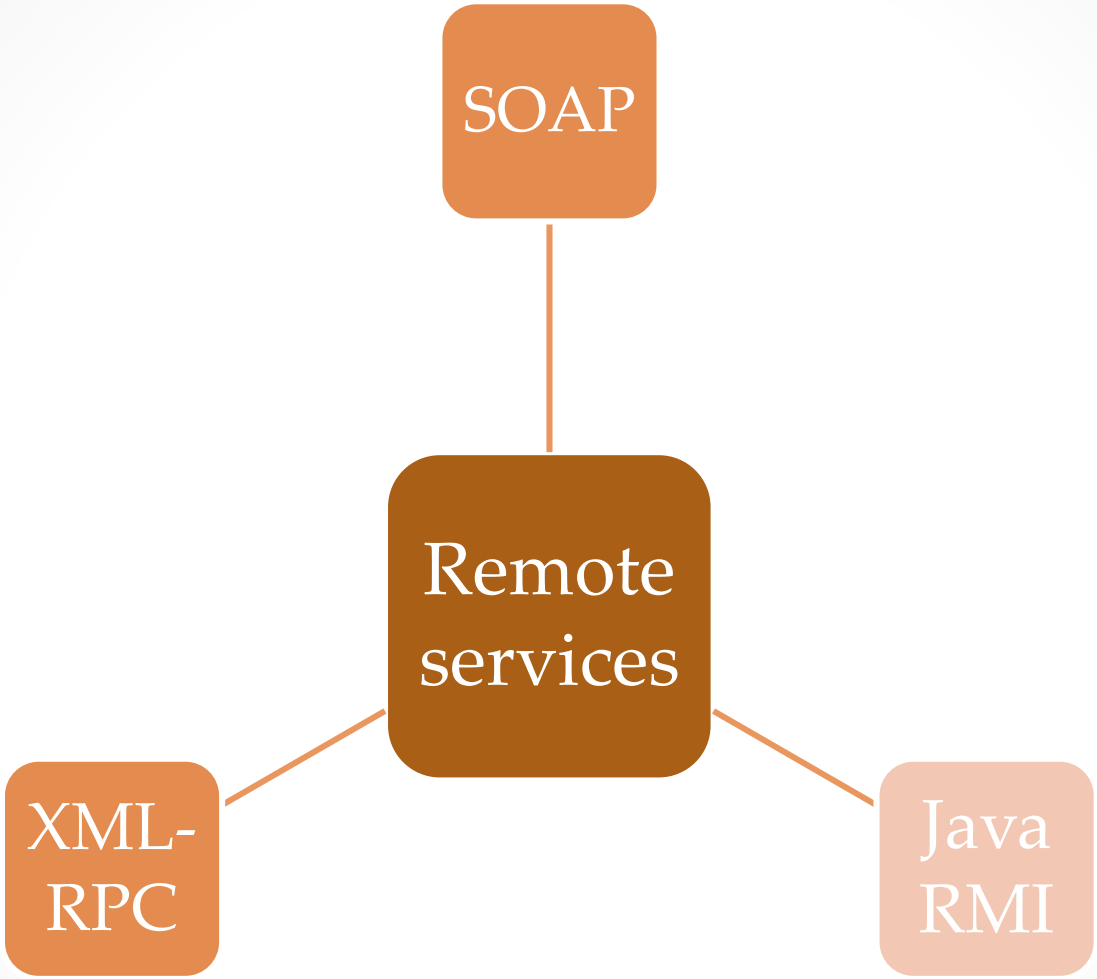


Sergey Morenets, 2018

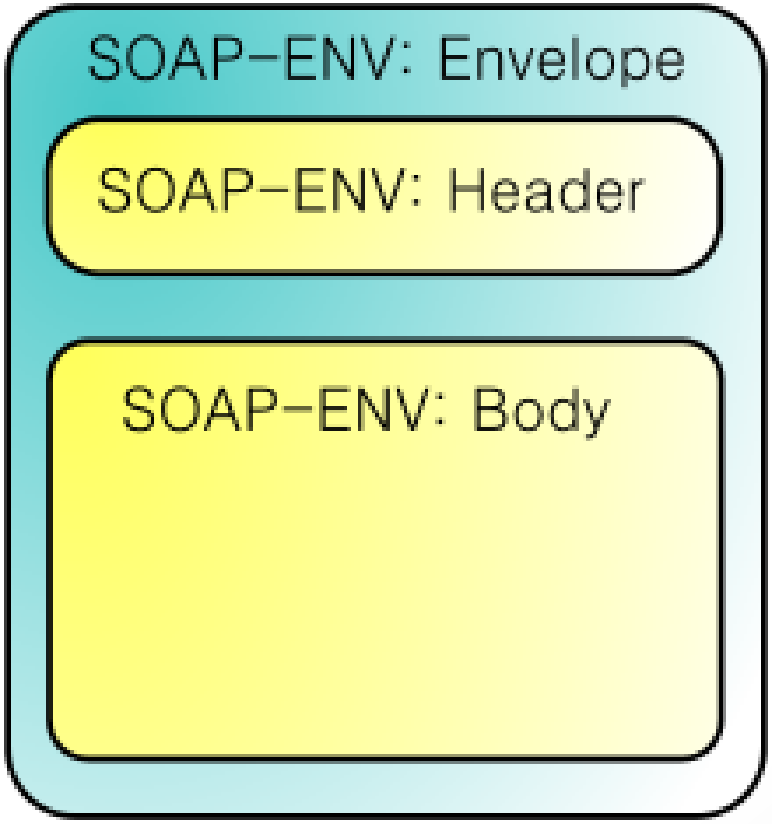
Service

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

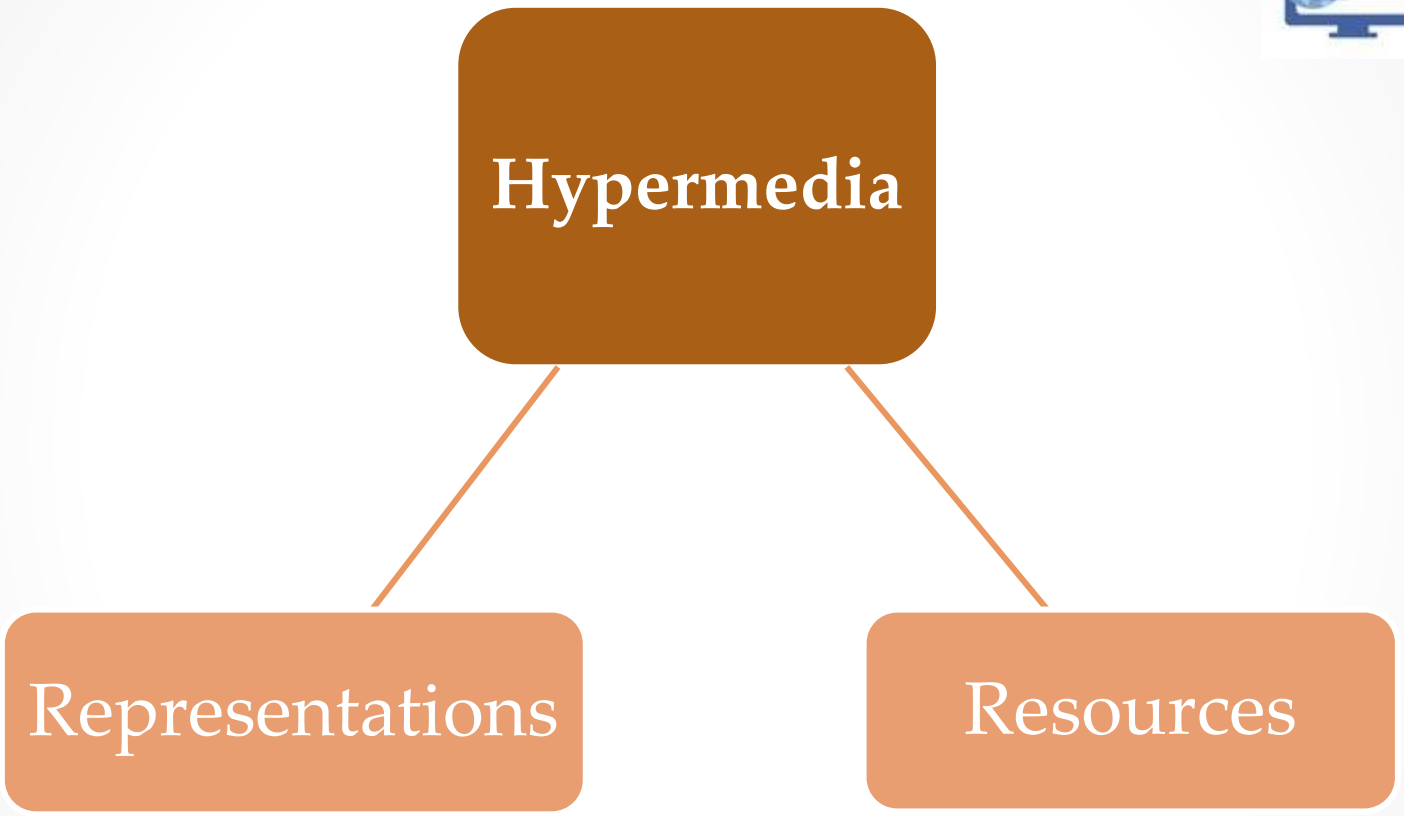
Sergey Morenets, 2018

Constraints

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

REST is not

- ✓ Protocol
- ✓ File format
- ✓ Development framework


Hypermedia
Representations
Resources

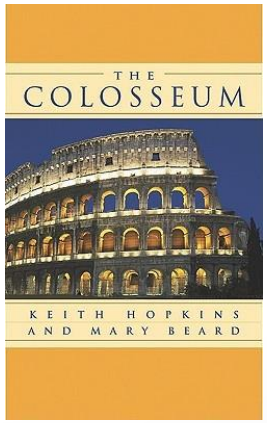
Sergey Morenets, 2018

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"
}
```

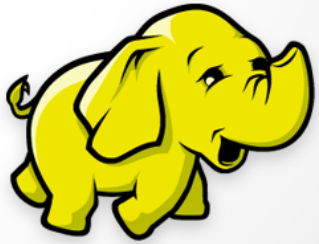
```
<document>
      <text>helloWorld</text>
</document>
```

Representations

Apache Thrift ™

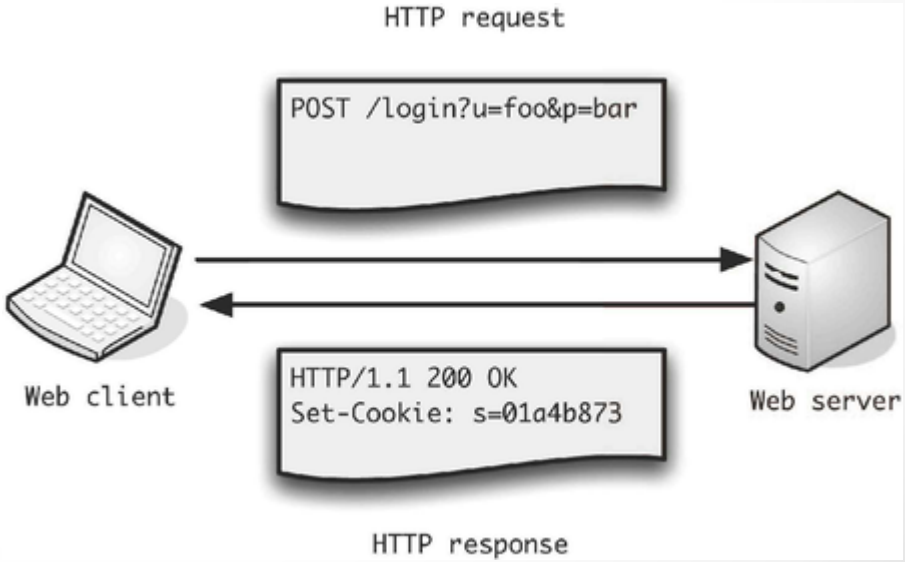
Protocol Buffers

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

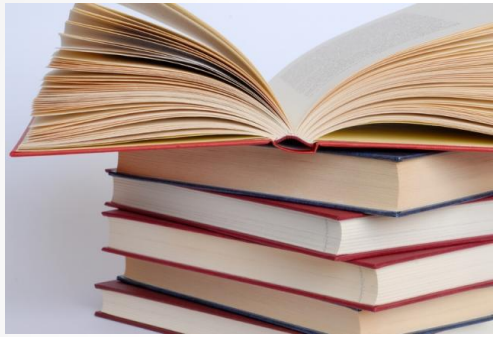
Goals

- ✓ 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

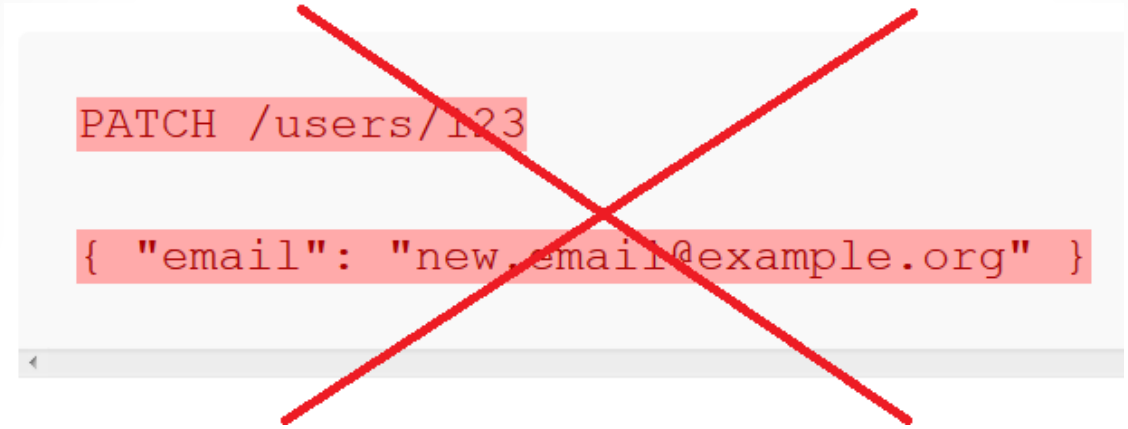
PUT /books

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

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

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

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

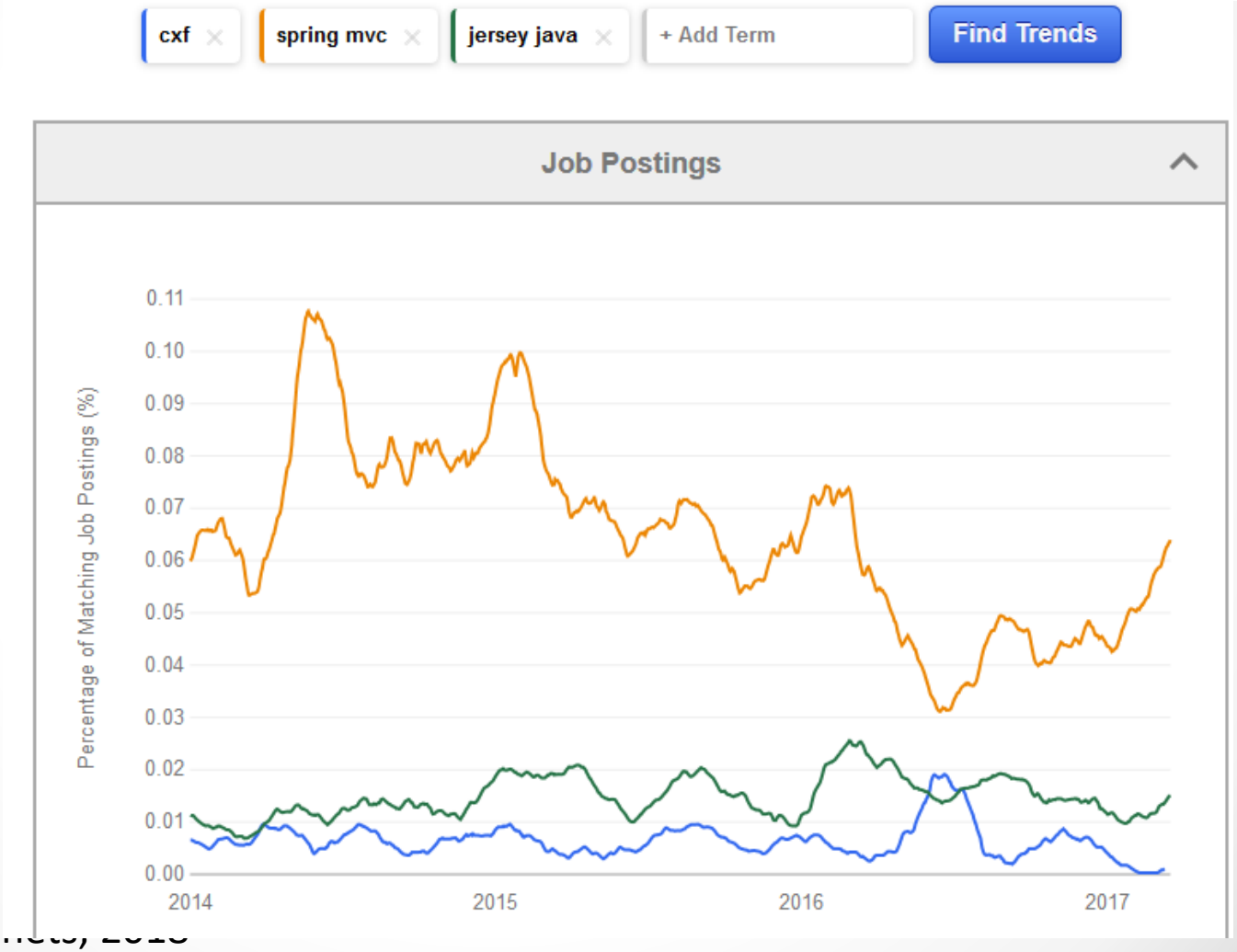

```
{
  "links": [
  {
    "href": "https://api.paypal.com/v1/payments/payment/PAY-1B569607296042 35TKQQIYVY
",
    "rel": "self",
    "method": "GET"
  },
  {
    "href": "https://api.paypal.com/v1/payments/payment/PAY-1B569607296042 35TKQQIYVY
/execute",
    "rel": "execute",
    "method": "POST"
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

Implementations

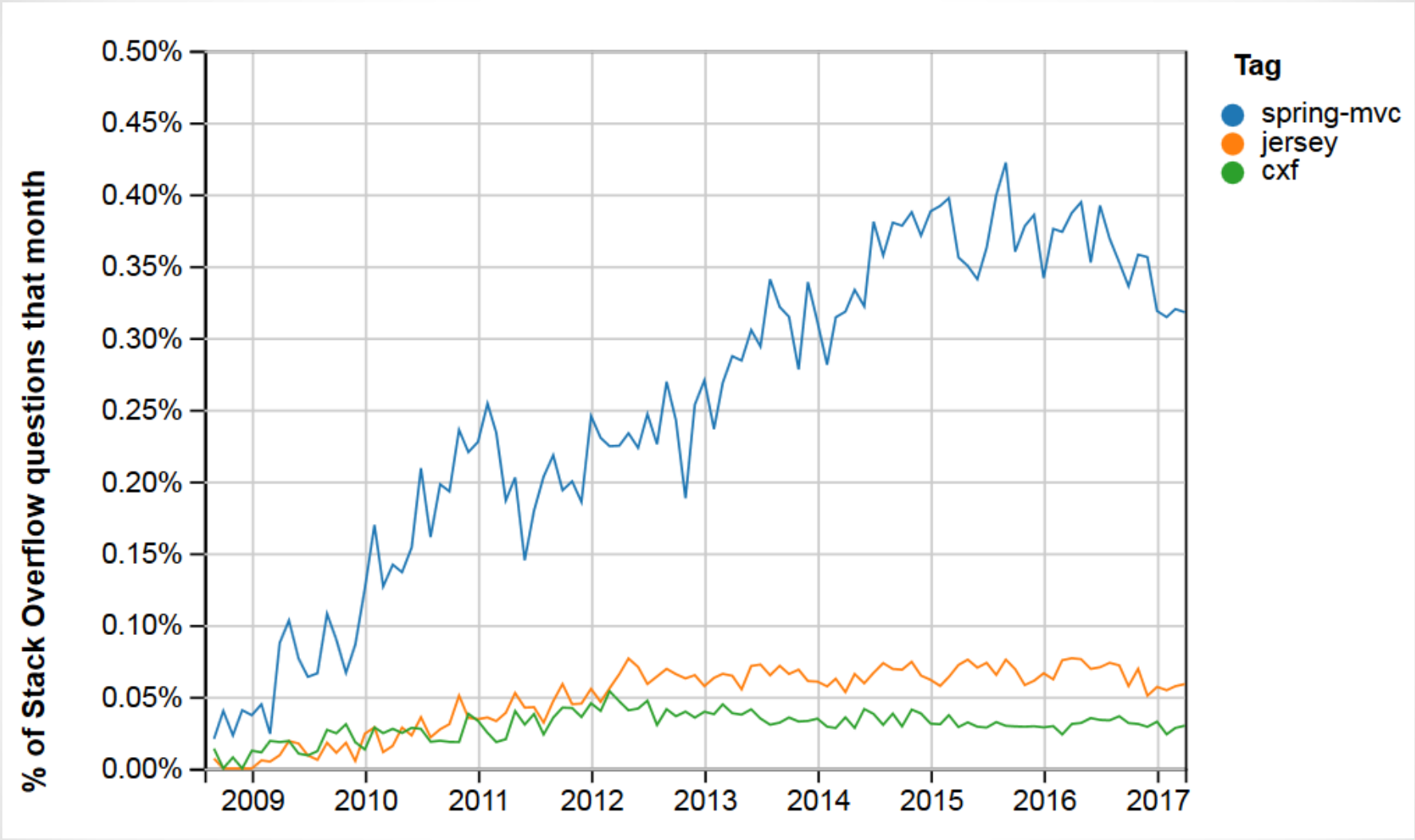
- ✓ Spring MVC
- ✓ **JAX-RS** implementations:
- ✓ *Apache CXF*
- ✓ *Jersey*
- ✓ *RestEasy*
- ✓ *Restlet*

- ✓ Ratpack
- ✓ Spark

Popularity



- Sergey More...

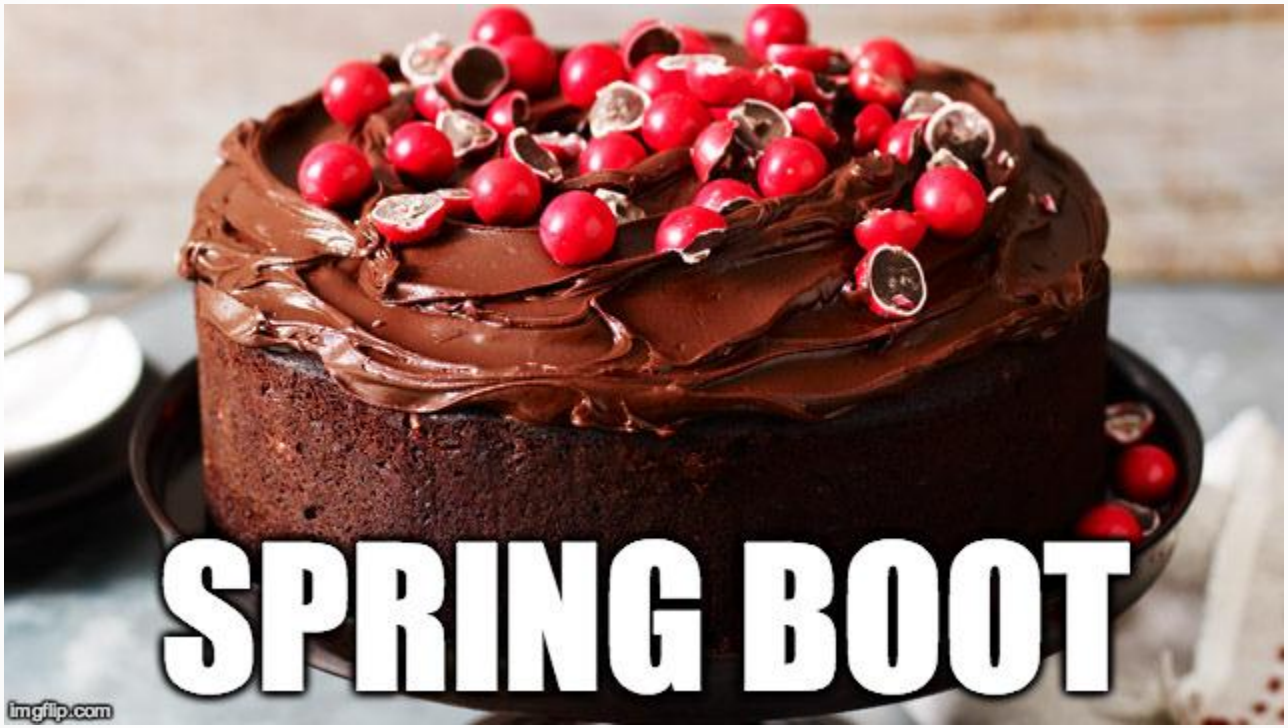


Sergey Morenets, 2018

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

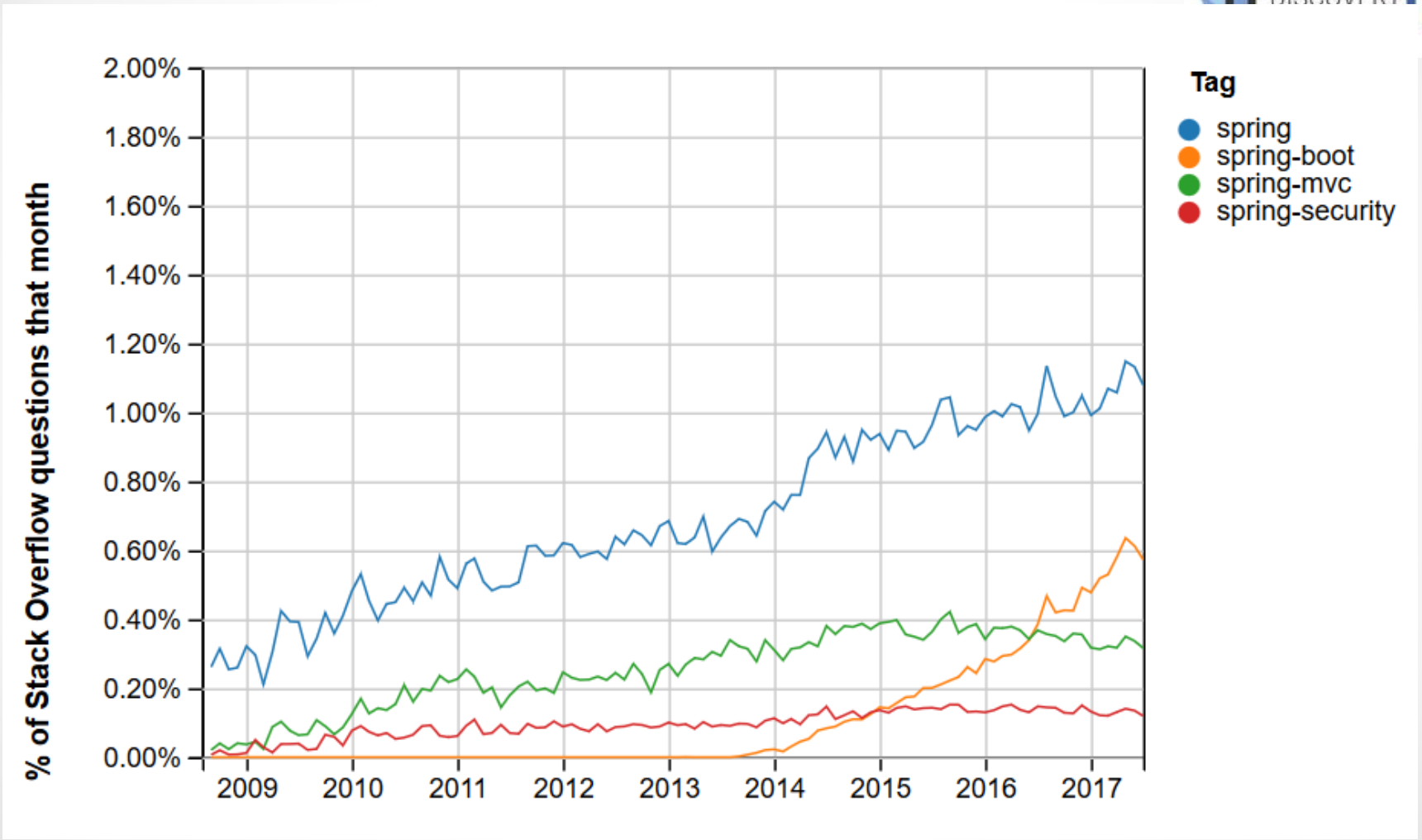
Sergey Morenets, 2018



Sergey Morenets, 2018

Spring Boot

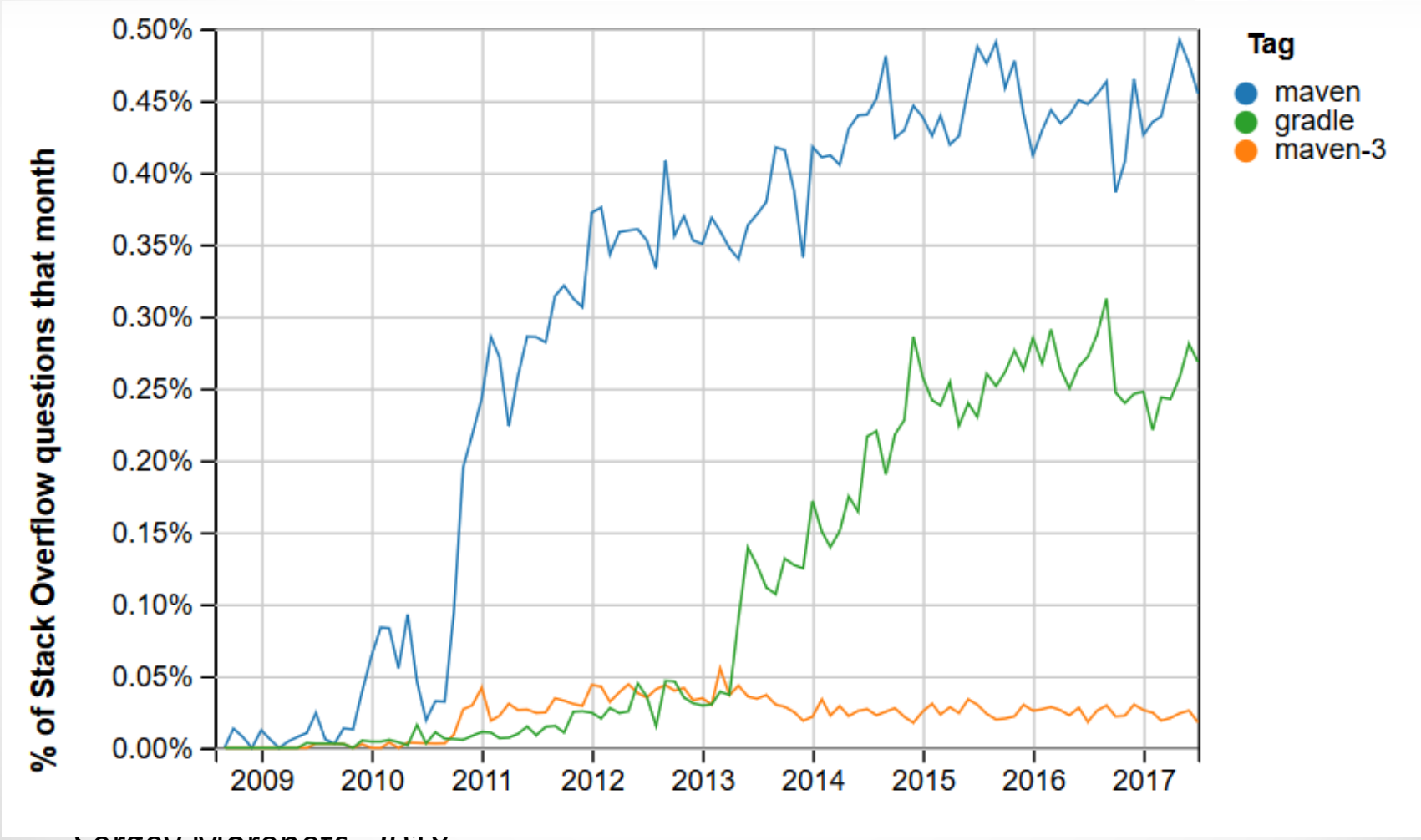
- Stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly
- Automatically Spring configuration
- Convention-over-configuration
- Absolutely no code generation and no requirement for XML configuration
- Focus on business features and less on infrastructure



Sergey Morenets, 2018

Build management

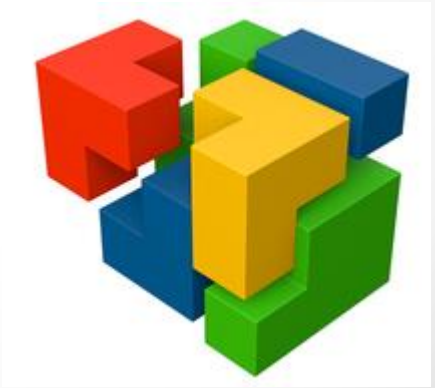
Maven vs Gradle



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

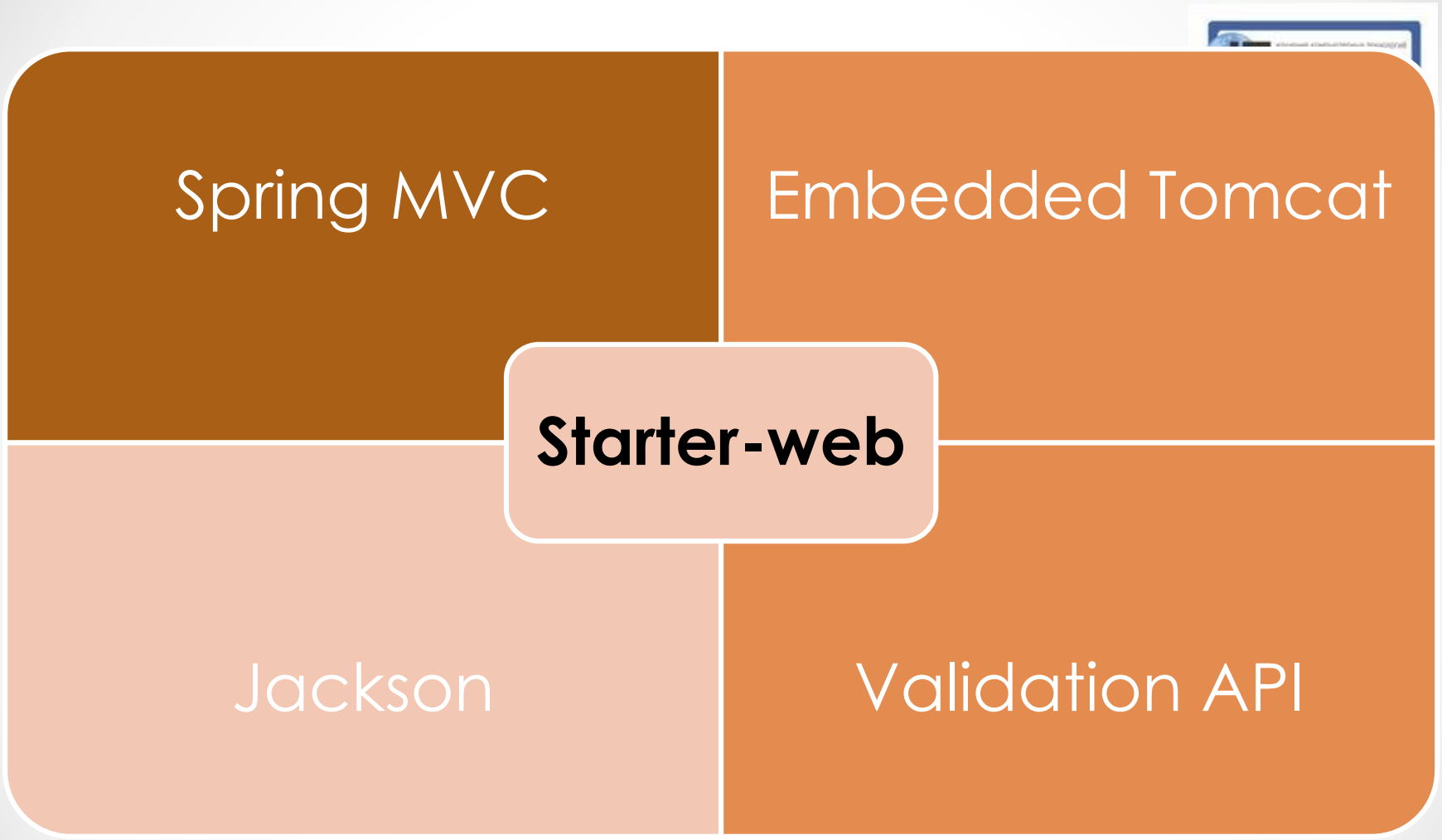
Build dependencies


```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}")
```

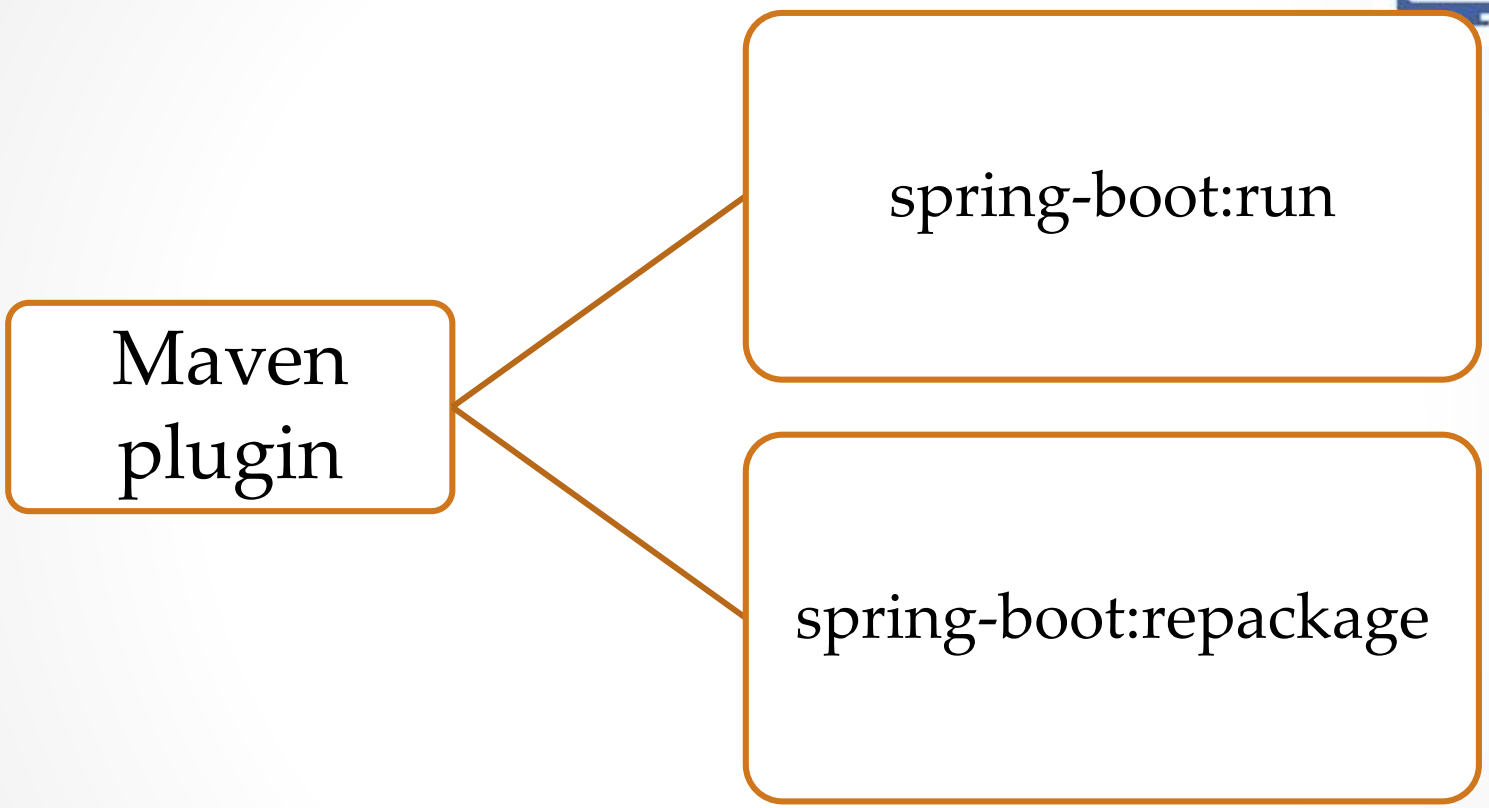
Gradle


Spring MVC
Embedded Tomcat
Starter-web
Jackson
Validation API


```java
@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

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>2.0.0.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>
```

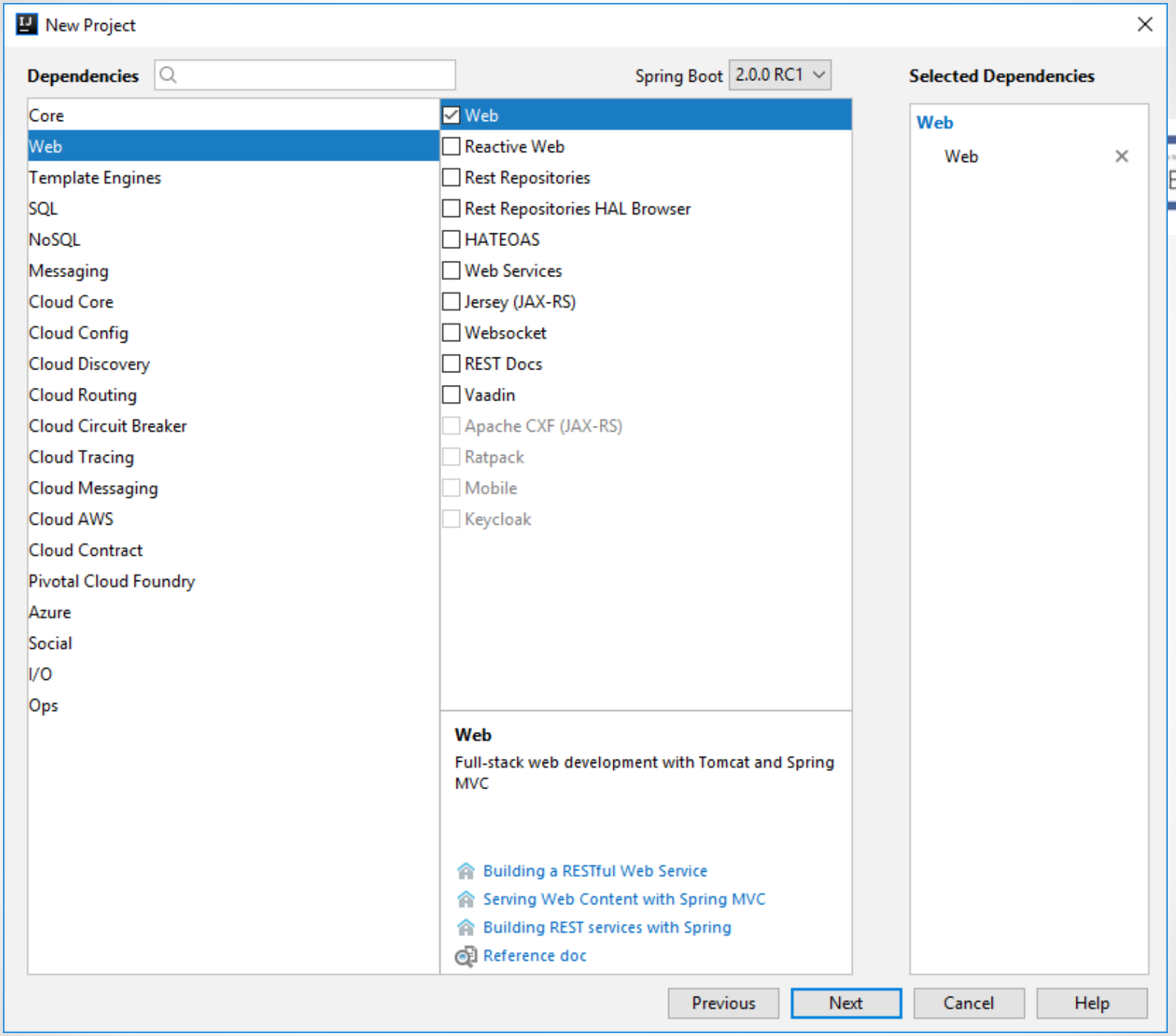
spring-boot-dependencies.pom.xml

Dependency management. Gradle

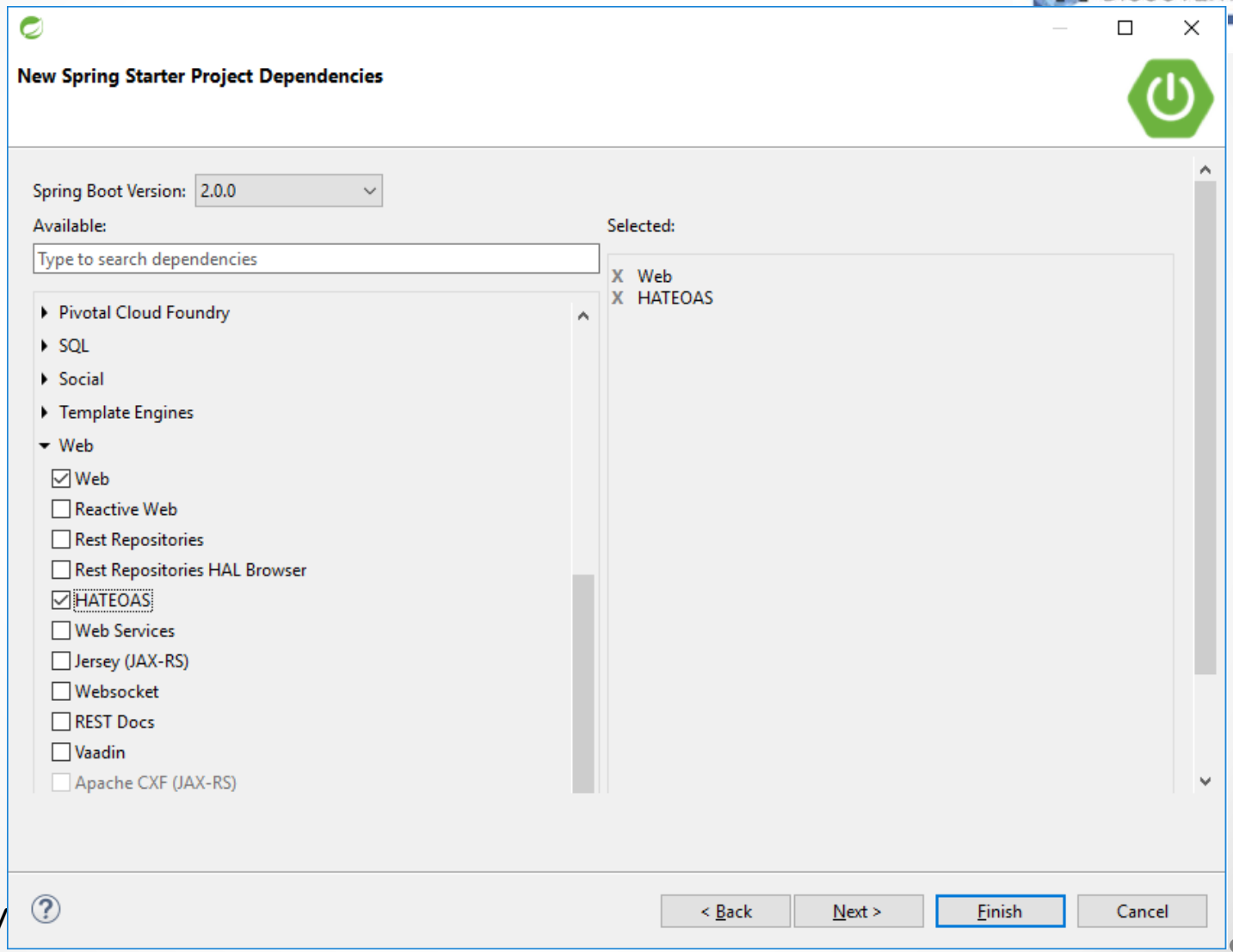

```
plugins {
    id 'org.springframework.boot' version '2.0.0.RELEASE'
}
```

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

IDE

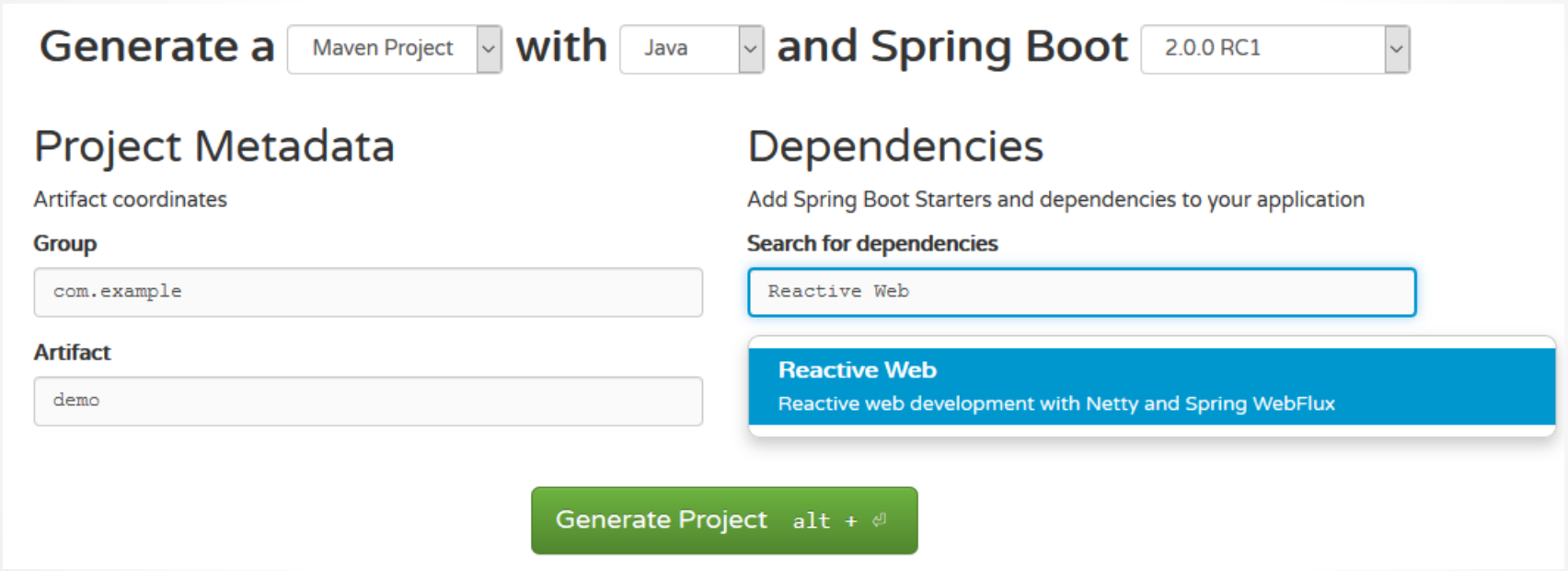

New Project
Dependencies
Spring Boot
2.0.0 RC1
Selected Dependencies
Core
Web
Template Engines
SQL
NoSQL
Messaging
Cloud Core
Cloud Config
Cloud Discovery
Cloud Routing
Cloud Circuit Breaker
Cloud Tracing
Cloud Messaging
Cloud AWS
Cloud Contract
Pivotal Cloud Foundry
Azure
Social
I/O
Ops
Web
Reactive Web
Rest Repositories
Rest Repositories HAL Browser
HATEOAS
Web Services
Jersey (JAX-RS)
Websocket
REST Docs
Vaadin
Apache CXF (JAX-RS)
Ratpack
Mobile
Keycloak
Web
Web
Full-stack web development with Tomcat and Spring MVC
Building a RESTful Web Service
Serving Web Content with Spring MVC
Building REST services with Spring
Reference doc
Previous
Next
Cancel
Help

Spring Boot Starter. STS



- Sergey M

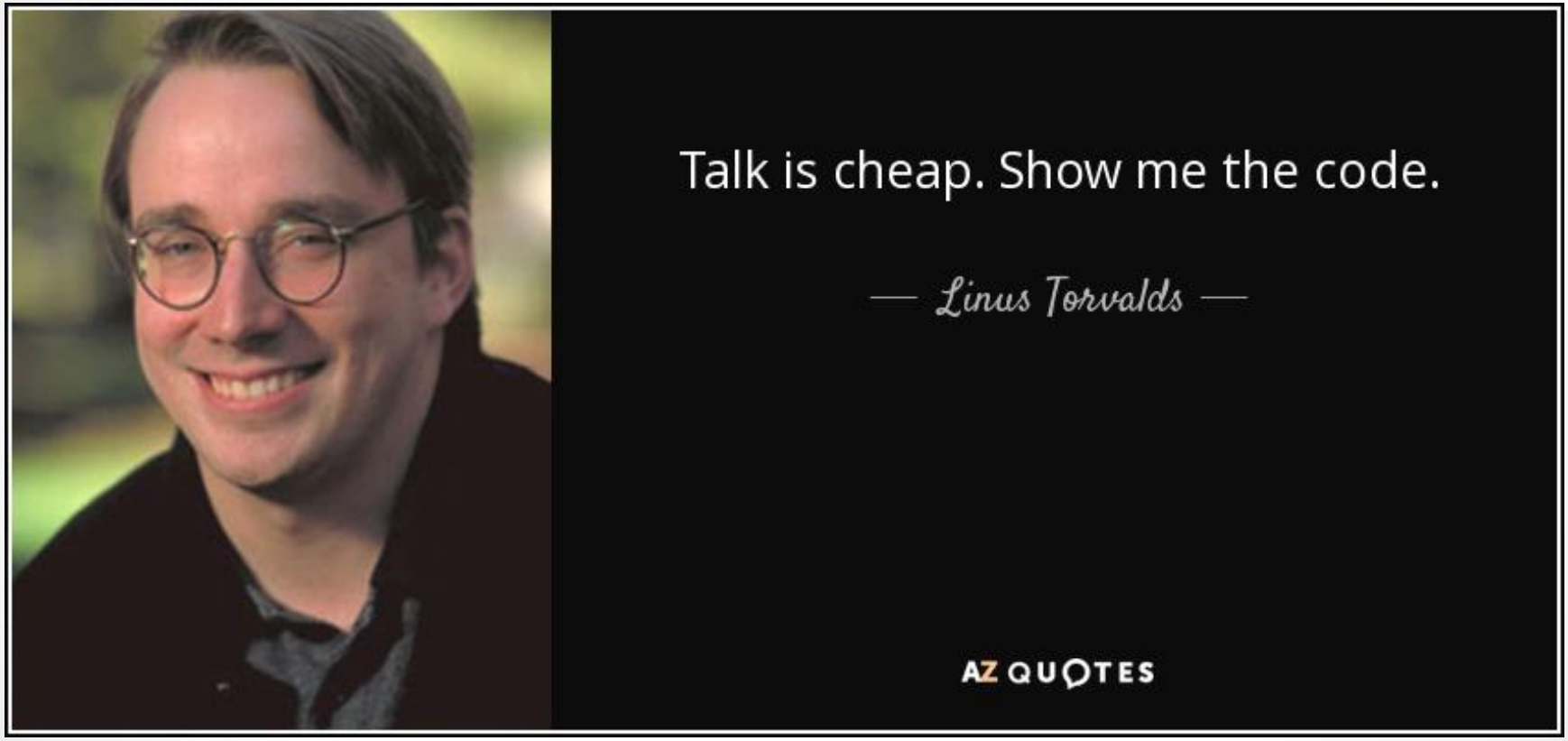
Spring Initializr



https://start.spring.io/

Task 2

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

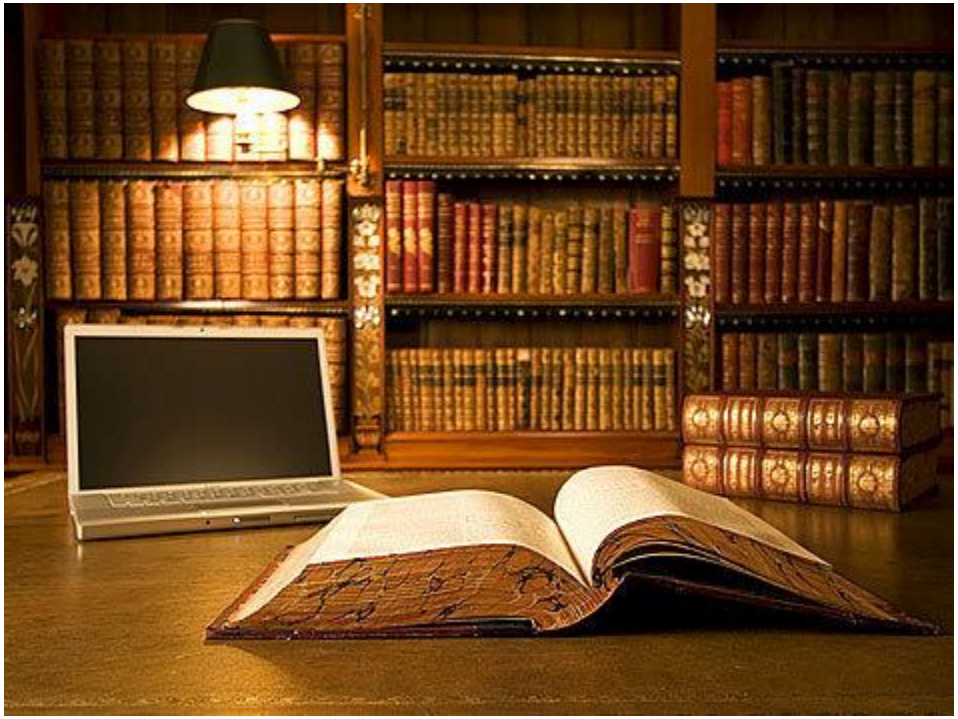


Sergey Morenets, 2018

Task 3

1. Import Task **#3** project into your IDE
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

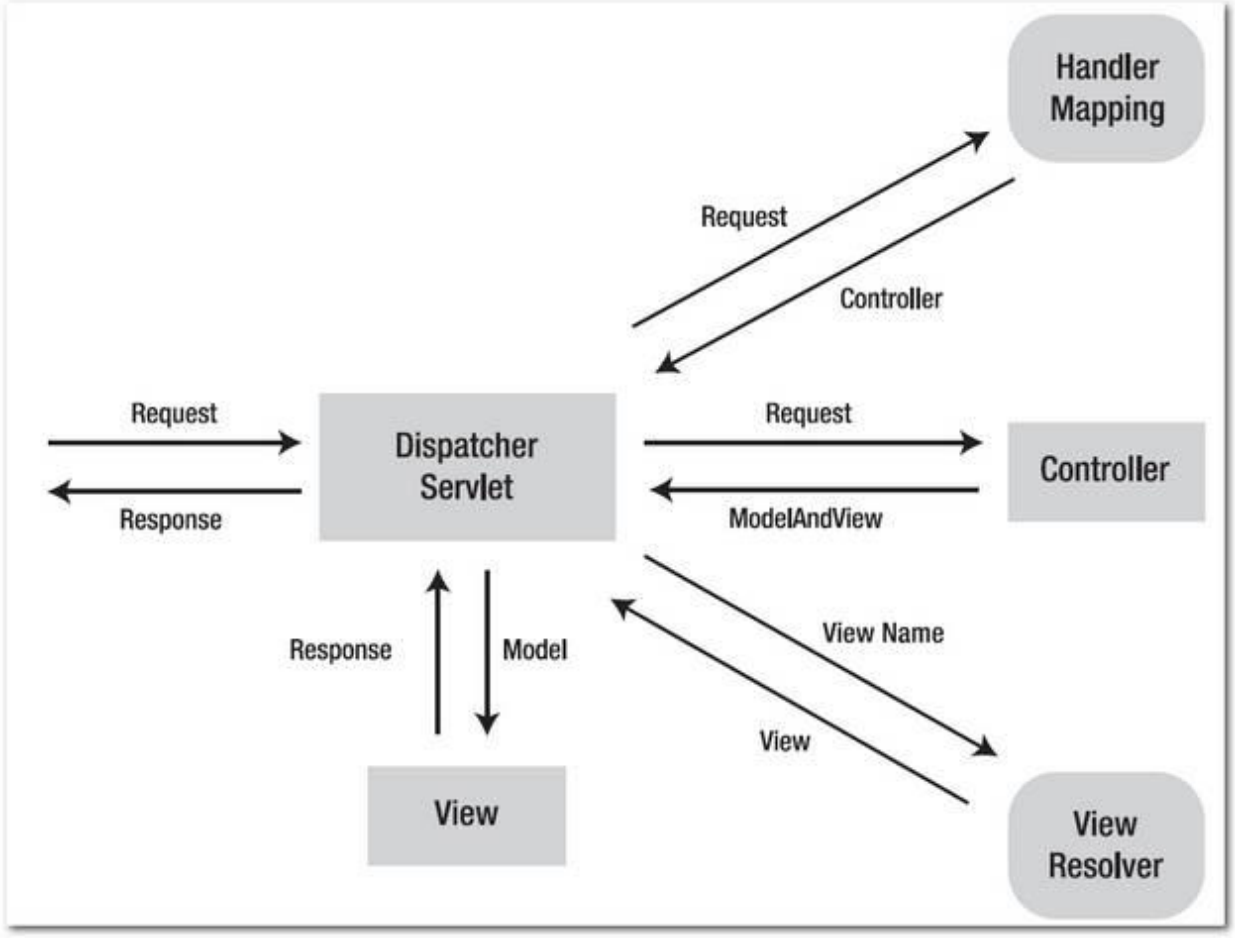


Sergey Morenets, 2018

Business domain


```
public class Book {
    private int id;

    private String author;

    private String name;

    private int year;
```

Spring MVC



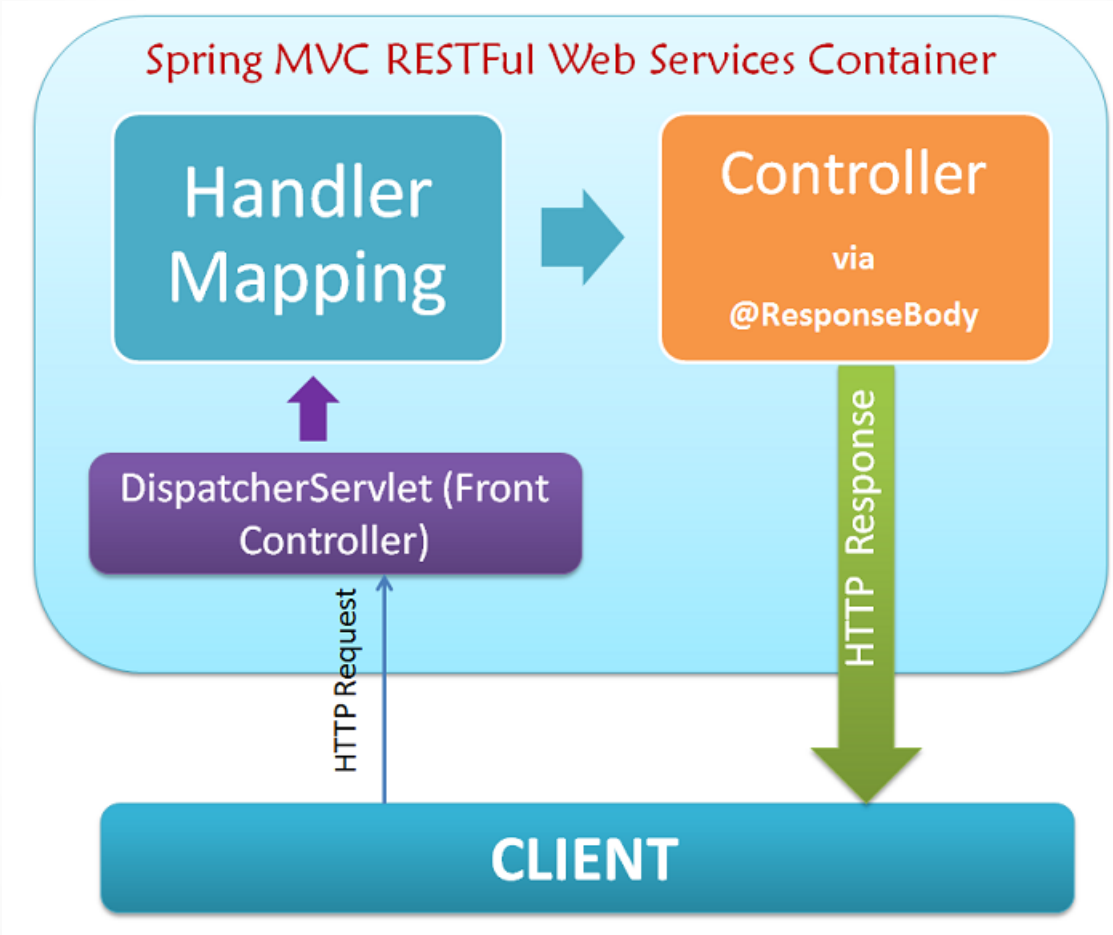
Sergey Morenets, 2018

Spring MVC


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

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

}
```

Spring MVC


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



Sergey Morenets, 2018

Spring MVC. Controllers


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

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

}
```

Spring MVC


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

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

}
```

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

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


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

Task 5. Response format

1. Write **GET service** that return sample book instance in JSON format
2. Check in Postman that service works correctly
3. Modify service to produces data in **JSON/XML** format
4. Configure request in Postman to receive book

in both formats

Task 6

Sergey Morenets, 2018

@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;
}
```

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

Requires validation provider

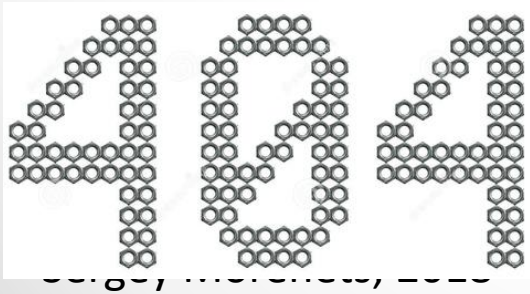
Task 7. REST services and CRUD operations


```
@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

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


```
    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


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

    private static final long serialVersionUID =
            3591666710943050205L;

}
```

Error handling


```
@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 8. 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

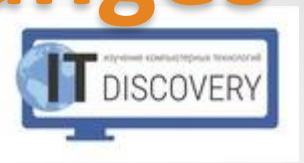
Actuator

- 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/status	
/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.
ElasticsearchHealthIndicator	Checks that Elasticsearch cluster 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

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();
    }
```

Health


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

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 9. 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

Sergey Morenets, 2018

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"
```

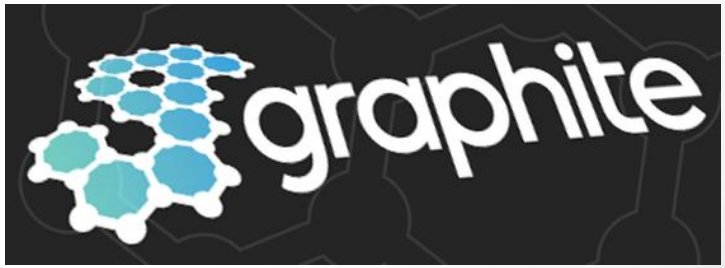
Metrics. Spring Boot 2.0


```
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:



Micrometer and Graphite


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

Graphite Dashboard

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

Micrometer

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

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

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 endpoints


```
@Component
@Endpoint(id = "books")
public class BookEndpoint {

    private final BookService bookService;

    public BookEndpoint(BookService bookService) {
        this.bookService = bookService;
    }

    @ReadOperation
    public List<Book> books() {
        return bookService.findBooks();
    }
```

/actuator/books

Custom endpoints

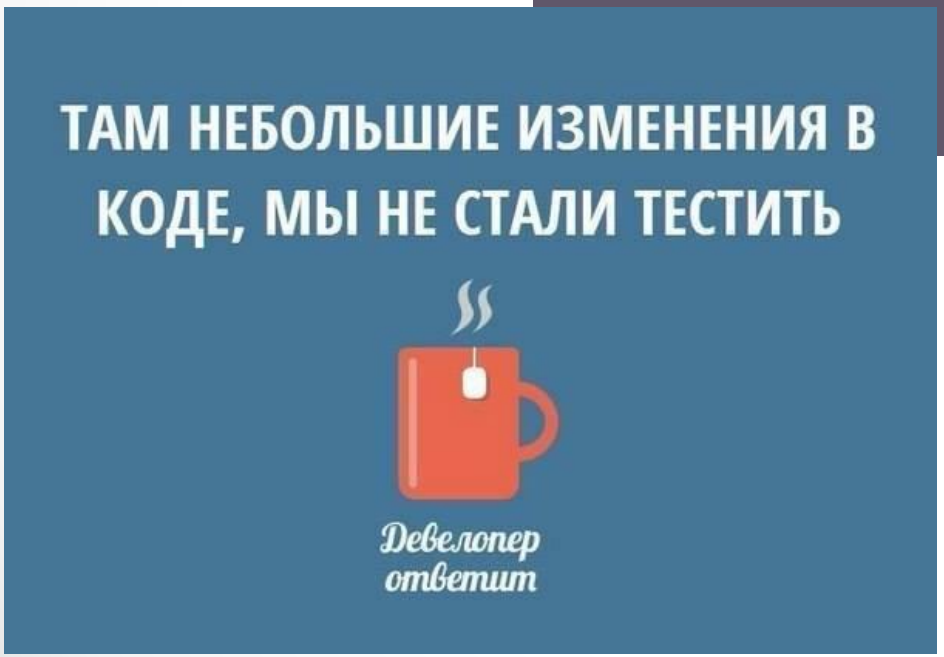

```
@ReadOperation
public Book feature(@Selector String id) {
    return bookService.findBook(id);
}

@WriteOperation
public void saveBook(@Selector String name, Book book) {
}

@DeleteOperation
public void deleteBook(@Selector String name) {
}
```

@POST → **@WriteOperation**

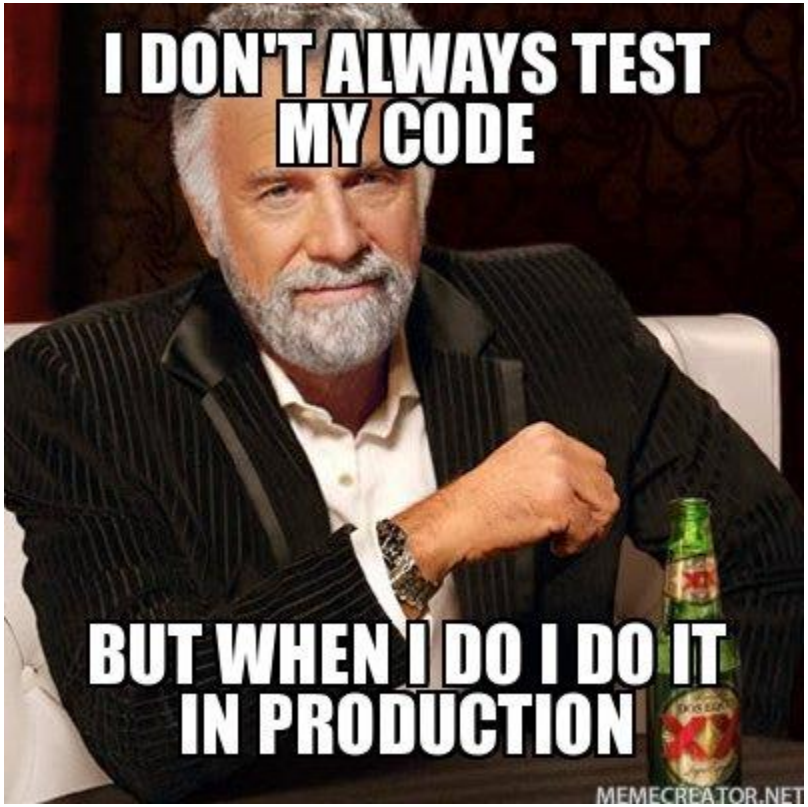
@DELETE → **@DeleteOperation**

Task 10. Metrics and endpoints

1. Add new endpoint */actuator/books* using **@Endpoint** annotation.
2. Open URL */actuator/metrics* and review existing metrics
3. Add **@Timed** annotation to some of your REST-services and verify that new metric is created
4. Create two metrics: number of saved books and number of deleted books using **MeterRegistry** bean



Sergey Morenets, 2018

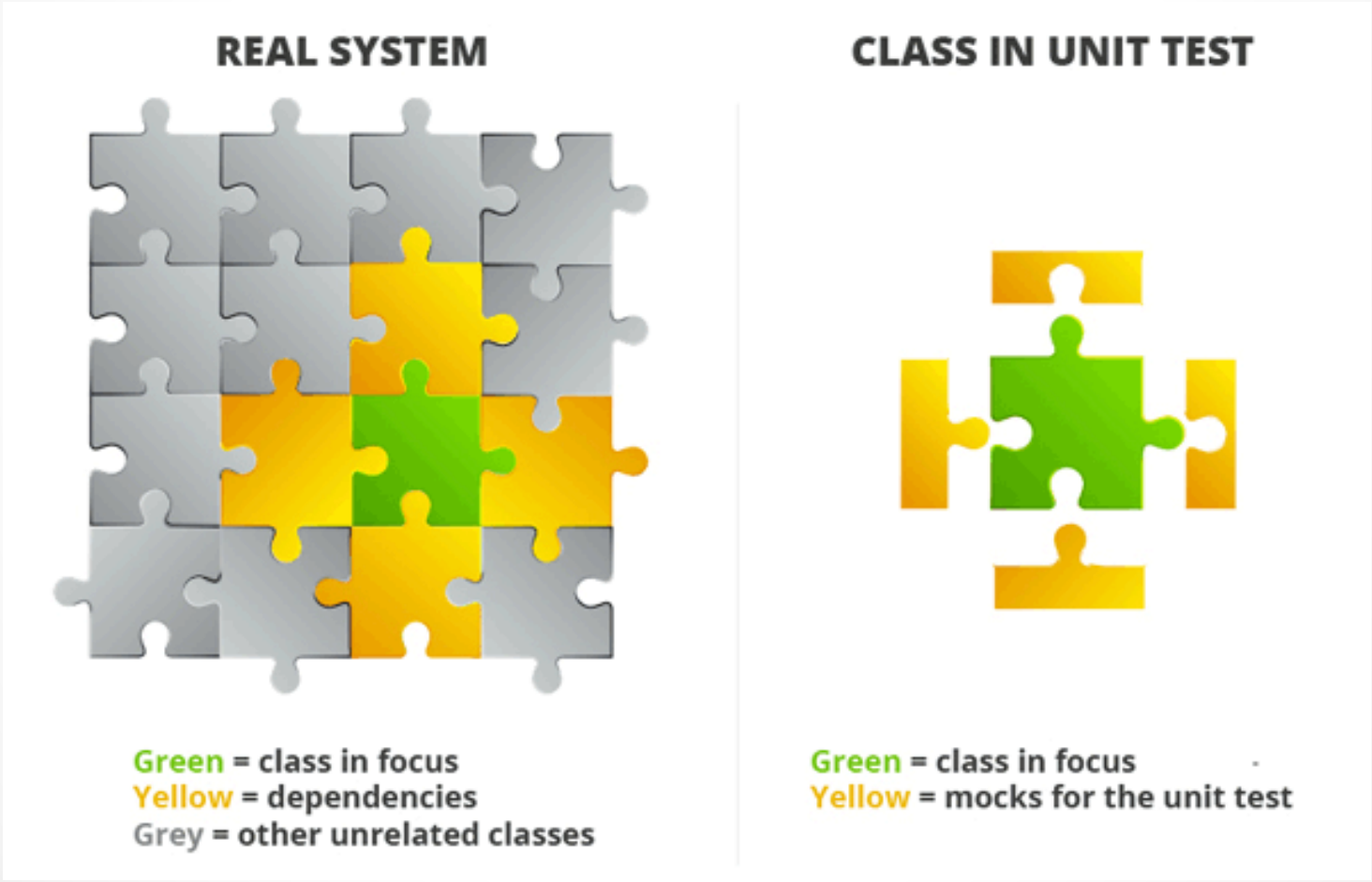


Sergey Morenets, 2018

Spring MVC Test framework

- ✓ Part of **TestContext** framework
- ✓ Based on mock conception
- ✓ High-level abstractions over mocked requests/responses

Mocks



Sergey Morenets, 2018

JUnit 5

- ✓ Separation of concerns
- ✓ API improvements, test extension model
- ✓ Multiple runners support
- ✓ Java 8 - based

Sergey Morenets, 2018

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 Framework. Maven


```
<plugin>
    <artifactId>maven-surefire-plugin</artifactId>
    <dependencies>
        <dependency>
            <groupId>org.junit.platform</groupId>
            <artifactId>junit-platform-surefire-provider</artifactId>
            <version>1.1.0</version>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.1.0</version>
        </dependency>
    </dependencies>
</plugin>
```

Spring MVC Test. JUnit 4


```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration(classes=RestApplication.class)
public class BookControllerTest {

}
```

Spring MVC Test. Junit 4


```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration(classes=RestApplication.class)
public class BookControllerTest {
    @Autowired
    private WebApplicationContext applicationContext;

    private MockMvc mockMvc;

    @Before
    public void setup() {
        mockMvc = MockMvcBuilders.webAppContextSetup(
                applicationContext).build();
    }
```

Spring MVC Test framework


```
@Test
public void findBooks_StorageIsEmpty_NoBooksAreReturned()
        throws Exception  {
    mockMvc.perform(MockMvcRequestBuilders.get("/book"))
      .andExpect(MockMvcResultMatchers.status().isOk())
      .andExpect(MockMvcResultMatchers.content().
              contentType(MediaType.APPLICATION_JSON_UTF8_VALUE))
      .andExpect(MockMvcResultMatchers.jsonPath("$", Matchers.hasSize(0)));
}
```

Spring MVC Test framework

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

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

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

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

Sergey Morenets, 2018

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'
```

Sergey Morenets, 2018

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"]
}
```

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
        }
    }
]
```

Testing POST method


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

@Test
public void saveBook_validBook_BookIsReturned()
        throws Exception  {
    Book book = new Book();
    book.setName("Java 8");

    mockMvc.perform(post("/book").contentType(
            MediaType.APPLICATION_JSON_UTF8_VALUE)
            .content(MAPPER.writeValueAsString(book)))
      .andExpect(status().isOk())
      .andExpect(content().
          contentType(MediaType.APPLICATION_JSON_UTF8_VALUE))
      .andExpect(jsonPath("$.name", Matchers.equalTo("Java 8")));
}
```

Testing improvements

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

@SpringRunner,

@AutoConfigureMockMvc

@MockBean

@SpringBootTest

@WebMvcTest

@JsonTest

spring boot

Mockito


```
interface Execute {
    String execute();

    void process();
}
```

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

Testing improvements

Junit 4

```
@RunWith(SpringRunner.class)
@AutoConfigureMockMvc
@SpringBootTest(classes=RestApplication.class)
public class BookControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private BookRepository bookRepository;
```

Testing improvements


```
@Test
public void findBooks_StorageIsNotEmpty_OneBookReturned()
        throws Exception {
    given(bookRepository.findAll()).
            willReturn(Arrays.asList(new Book()));

    mockMvc.perform(get("/book")).andExpect(status().isOk())
            .andExpect(content().contentType(
                    MediaType.APPLICATION_JSON_UTF8_VALUE))
            .andExpect(jsonPath("$", Matchers.hasSize(1)));
}
```

Testing improvements

Junit 5

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

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private BookRepository bookRepository;
```

Task 11. Writing tests (Junit 4/5)

1. Review **BookControllerTest** class. Add integration tests (using JUnit 4) for your REST service operations
2. Verify **status codes** and returned response in the unit-tests
3. Update unit-tests for the controller using Junit 5 annotations. Apply new Spring Boot annotations: **@SpringJUnitWebConfig** and **@AutoConfigureMockMvc**
4. Add mocking for the repository using **@MockBean** annotation

REST support on client side

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

RestTemplate. Find


```
public class RestClient {

    private final RestTemplate restTemplate =
            new RestTemplate();

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

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

RestTemplate. Save & update


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

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

RestTemplate. Customization

Autowired

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

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

Task 12. 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

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>
```

Spring HATEOAS


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

```
@RequestMapping
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());
        resources.add(resource);
    }

    return resources;
}
```

Spring HATEOAS


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

Spring HATEOAS

```
@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


```
[ ⊖
   { ⊖
      "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"
         }
      ]
   },
```

Task 13. 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

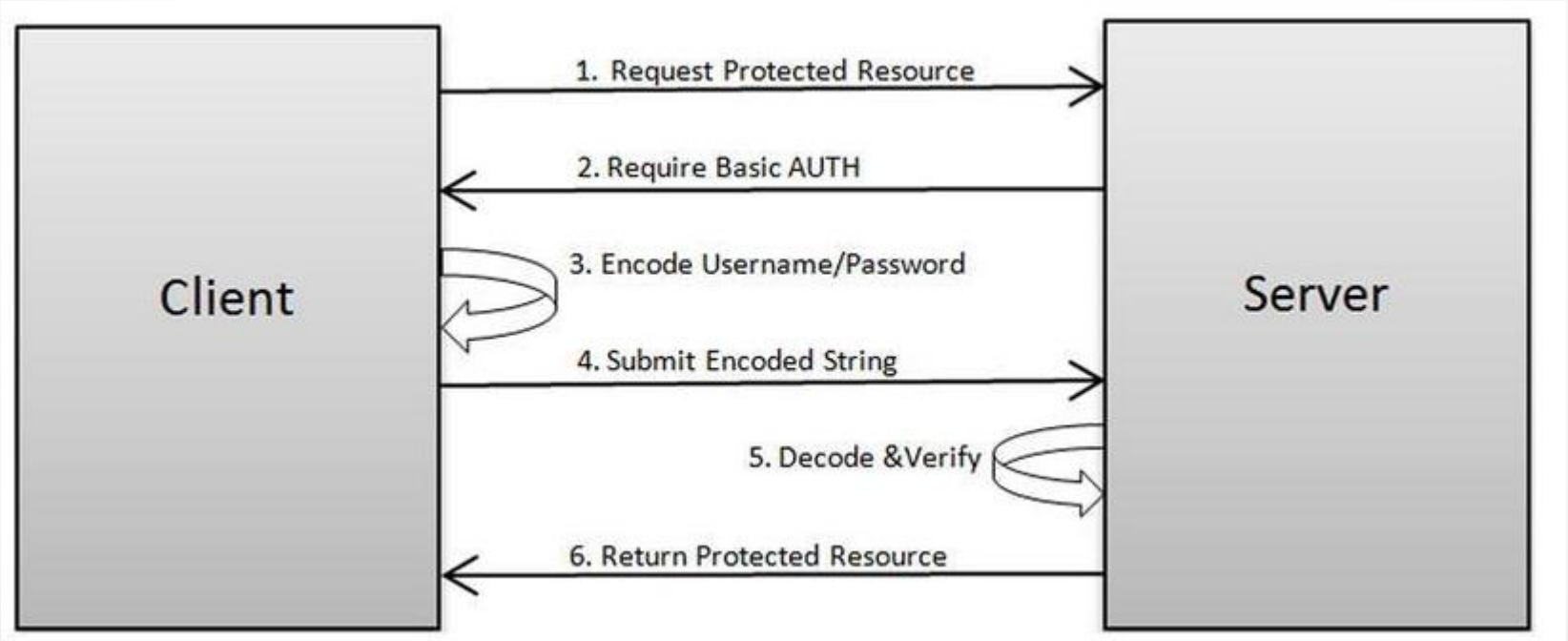
Authentication

Из диалога двух программистов:
— Кажется, у нас дыра в безопасности!
— Слава Богу, хоть что-то у нас в безопасности...

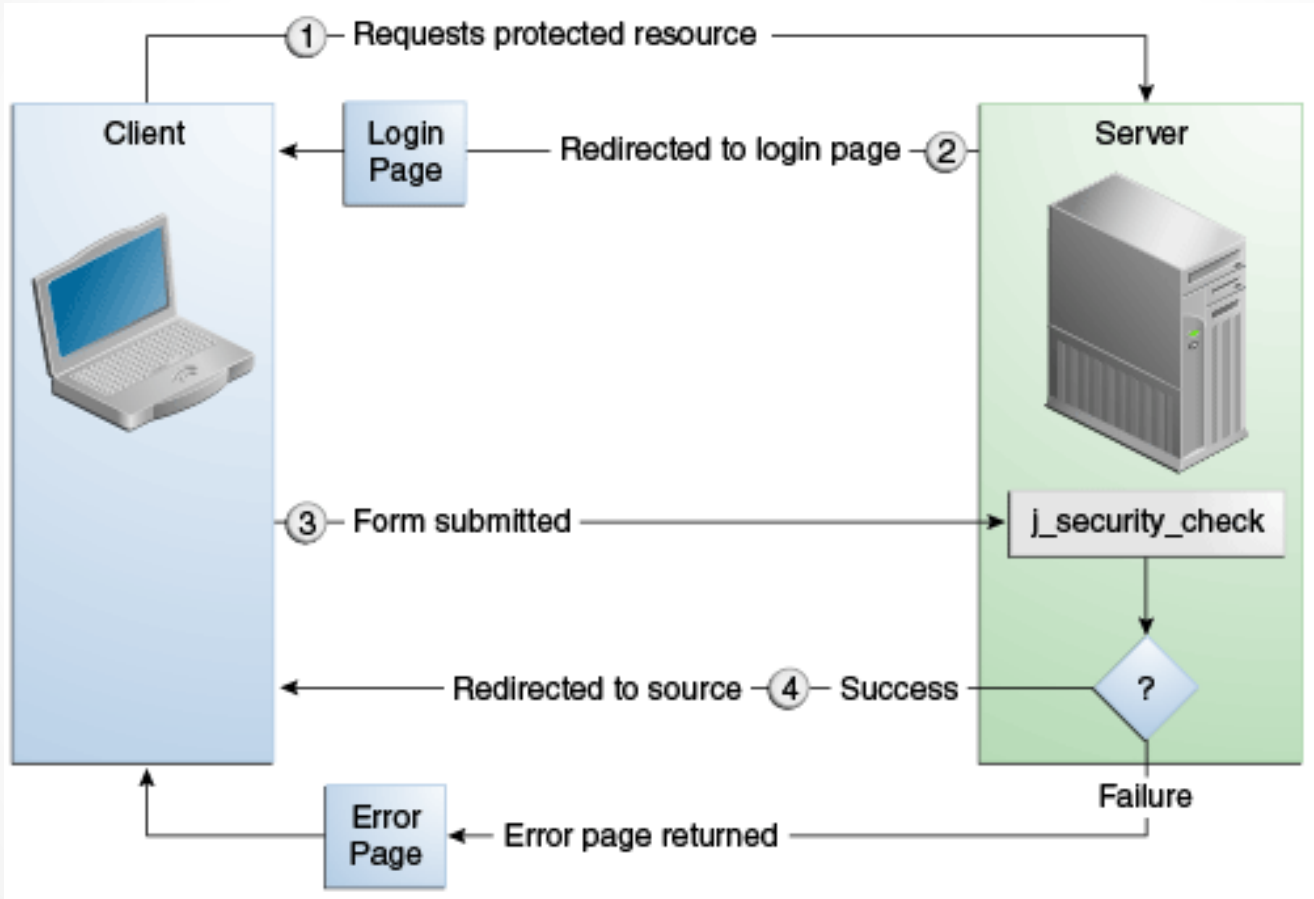
Authentication

- ✓ Basic
- ✓ Digest
- ✓ Token (JWT)
- ✓ Digital Signature
- ✓ Certificate
- ✓ OAuth 1.0/2.0

Basic authentication



Form-based authentication



Sergey Morenets, 2018

Spring Security

- Formerly **Acegi Security**
- Support for authentication and authorization
- Integration with **Spring MVC(REST)**
- Attacks protections (CSRF, session fixation)
- Basic, Digest, CAS, OpenID, JAAS and LDAP authentication
- Integration testing
- Based on filter chain

Spring Security. Maven


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

Configuration steps

- ✓ Declare protected resources
- ✓ Setup authentication
- ✓ Setup authorization (optional)
- ✓ Manage user storage

Spring Security


```
@Configuration
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfiguration extends
                WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http)
            throws Exception {
        http.authorizeRequests().anyRequest()
            .fullyAuthenticated();
        http.httpBasic();
        http.csrf().disable();
    }
```

Spring Security


```
@Bean
public PasswordEncoder passwordEncoder() {
    return NoOpPasswordEncoder.getInstance();
}
```

Can be BCrypt, SCrypt, MD4

```
spring.security.user.name=admin
spring.security.user.password=admin
```

application.properties

Important types

Name	Description
Principal	Any logged user with login id
Authentication	Token for authentication request, including authorities, credentials and principal
SecurityContext	Authentication information linked to current execution thread
SecurityContextHolder	Allows to get/update security context for current execution thread
AuthenticationProvider	Implementation that can authorize user based on current **Authentication** object

Rest Client


```
public class RestClient {

    private TestRestTemplate restTemplate;

    public RestClient() {
        restTemplate = new TestRestTemplate("admin", "admin");
    }
```

Unit-testing


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

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <version>${spring.security.version}</version>
    <scope>test</scope>
</dependency>
```

Unit-testing


```
@Test
@WithMockUser(username = "user", authorities = { "USER" })
public void findBooks_StorageIsNotEmpty_OneBookReturned()
        throws Exception {
    given(bookRepository.findAll()).
            willReturn(Arrays.asList(new Book()));

    mockMvc.perform(get("/book")).andExpect(status().isOk())
            .andExpect(content().contentType(
                    MediaType.APPLICATION_JSON_UTF8_VALUE))
            .andExpect(jsonPath("$", Matchers.hasSize(1)));
}
```

Task 14. Spring Security

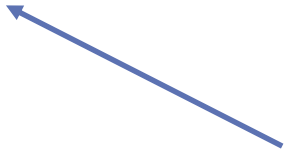
1. Add **Spring Security** configuration
2. Update RestClient
3. Verify that services require **authentication** via RestClient and POSTMAN
4. Update unit-tests using **@WithMockUser** annotation so that all the tests pass.

Security for Actuator

- ✓ Mostly local usage
- ✓ Read-only access
- ✓ Integration with Spring Security

Security for Actuator

```
management.endpoint.shutdown.enabled=true

management.endpoints.enabled-by-default=true

management.endpoints.web.base-path=/management
management.endpoints.web.path-mapping.shutdown=stop

management.server.port=9000
management.server.address=127.0.0.1
```



Disable remote connections

Task 15. Security and actuator

1. Create **application.properties** file in **src/main/resources** folder
2. Change actuator port and context path
3. Disable **/health** and **/info** endpoints and check Spring Boot behavior if you hit these endpoints
4. Disable all actuator endpoints
5. Change path mapping for some endpoints

Caching

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

Spring annotations

Name	Description
@EnableCaching	Enables annotation-driven cache management
@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
public class Application {
```

```
@RestController
public class GreetingController {

    @GetMapping("/greet")
    @Cacheable("greeting")
    public String greet() {
        return "Hello";
    }
```

Update cache


```
@GetMapping("/greet-nocache")
@CachePut("greeting")
public String greetNoCache() {
    return "Hello";
}
```

Calls method and update cache

```
@GetMapping("/greet-clear")
@CacheEvict(cacheNames="greeting",allEntries=true)
public void clearCache() {
}
```

Advanced caching

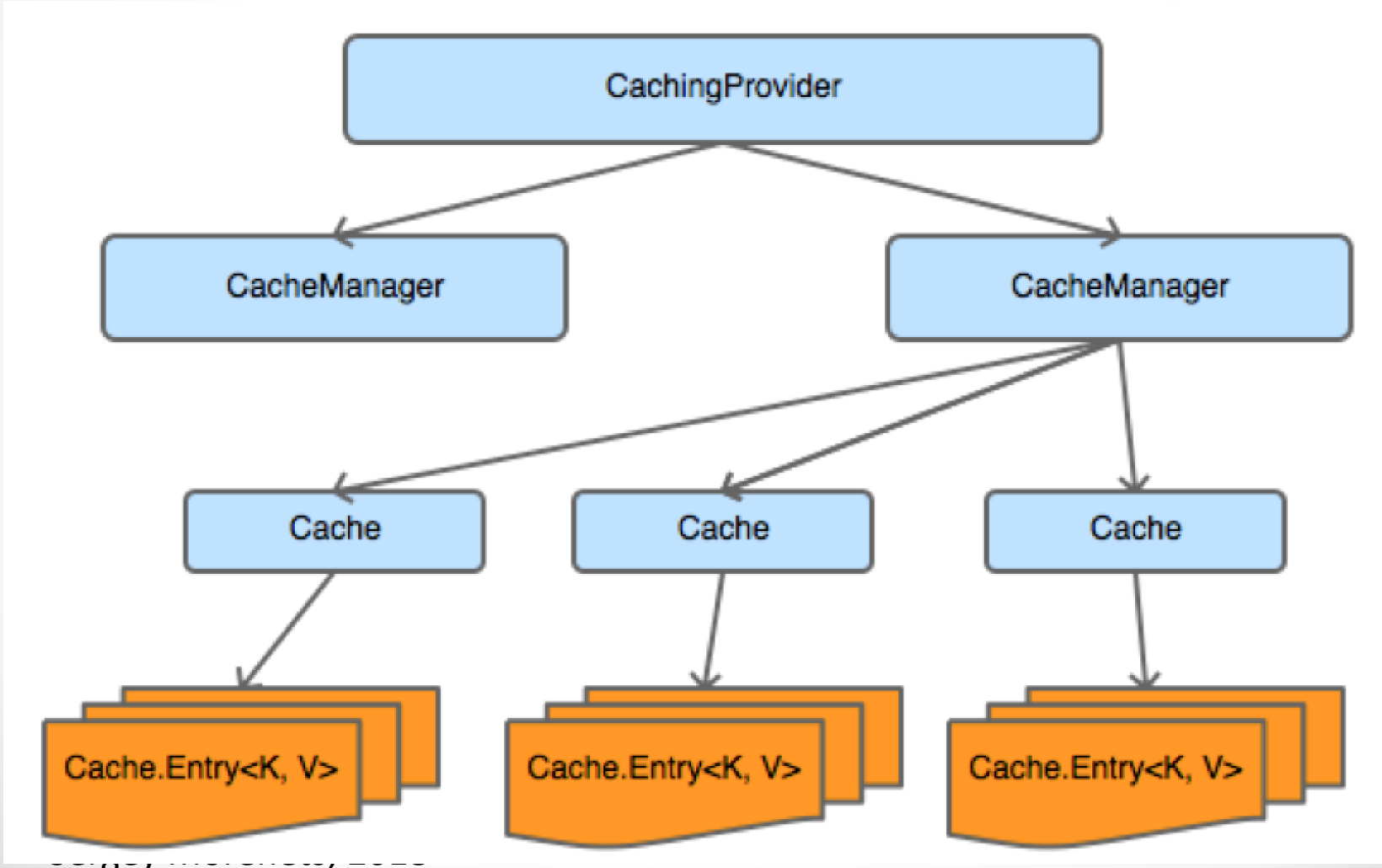

```
@GetMapping(path = "/{id}")
@Cacheable(cacheNames= {"books", "storage"},
        condition="#id!='1'")
public Book findById(@PathVariable int id) {
    return bookRepository.findById(id);
}
```

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

JSR-107

JCache API


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

JCache annotations


```
@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;
}
```

JCache configuration


```
spring.cache.jcache.config=classpath:jcache.xml
```

```
<config xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns='http://www.ehcache.org/v3'
    xsi:schemaLocation="
        http://www.ehcache.org/v3
        http://www.ehcache.org/schema/ehcache-core-3.0.xsd">

    <cache alias="books">
        <key-type>org.springframework.cache.interceptor.SimpleKey</key-type
        <value-type>java.util.Collections$SingletonList</value-type>
        <heap unit="entries">200</heap>
        <heap-store-settings>
            <max-object-size>2000</max-object-size>
        </heap-store-settings>
    </cache>
</config>
```

jcache.xml

Task 16. Caching

1. Add **@EnableCaching** annotation to the **RestApplication** class.
2. Add **@Cacheable** annotation to the **GET** REST-services.
3. Add new REST service that would clear the cache and put **@CacheEvict** annotation on it. Verify its behavior.
4. Add **JCache** dependency.
5. Replace **@Cacheable** with **@CacheResult** and **@CacheEvict** with **@CacheRemove**.

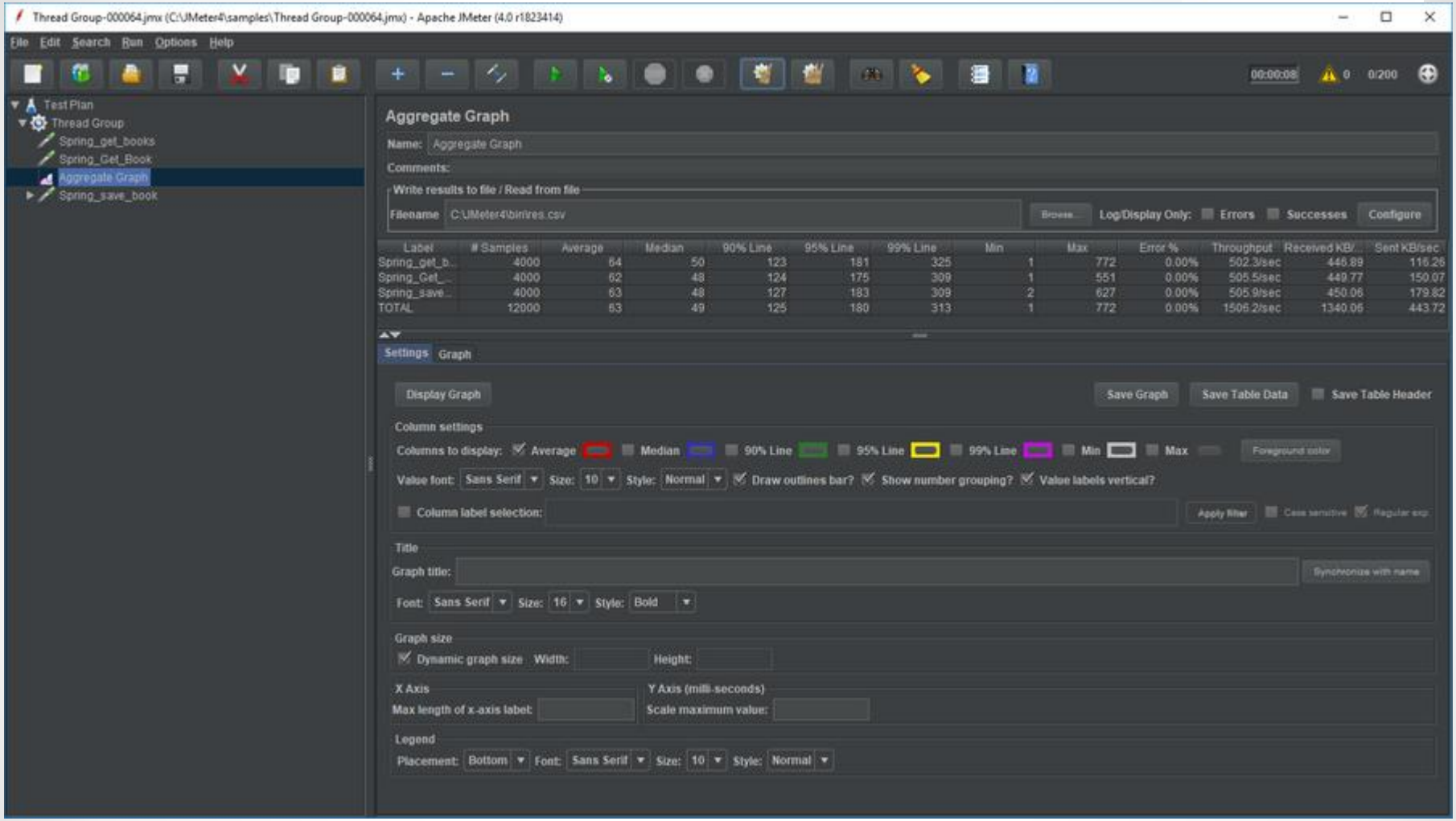
Performance/load testing

- ✓ 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

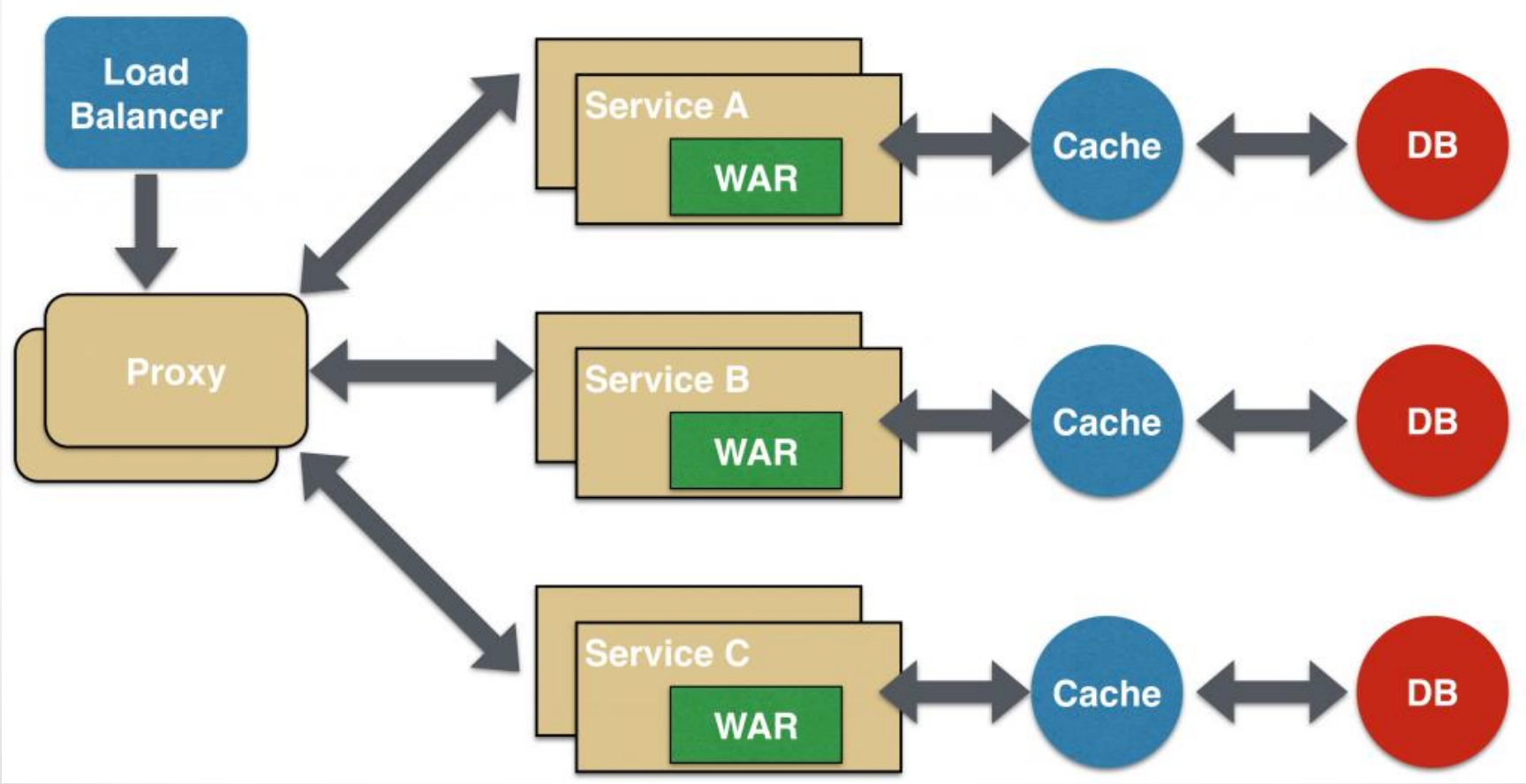
Work use-case

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

Task 17. 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).

Scaling


Load Balancer
Proxy
Service A
WAR
Service B
WAR
Service C
WAR
Cache
Cache
Cache
DB
DB
DB

Redis

- Created in 2009 by Salvatore Sanfillipo as **Re**mote **D**isctionary **S**erver
- Open-source
- Super-lightweight and easy to use
- In-memory **data structure** store
- 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

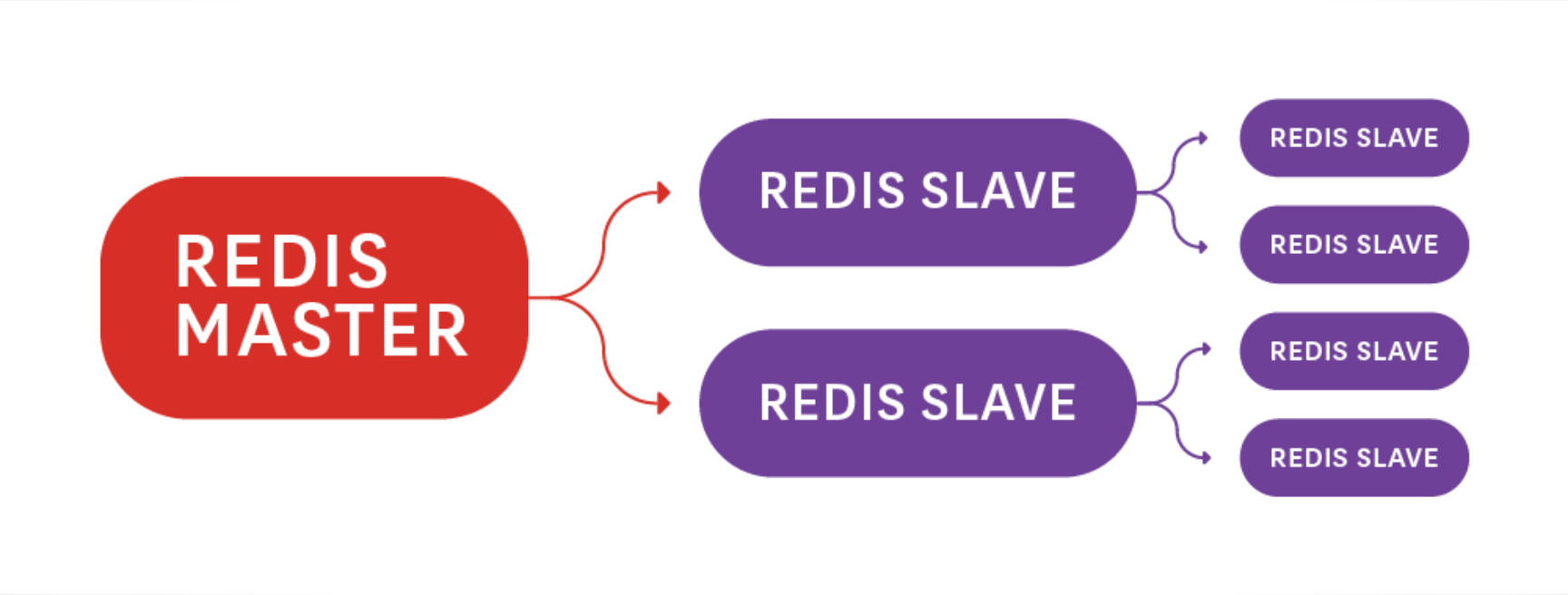
redis

Redis. Popularity

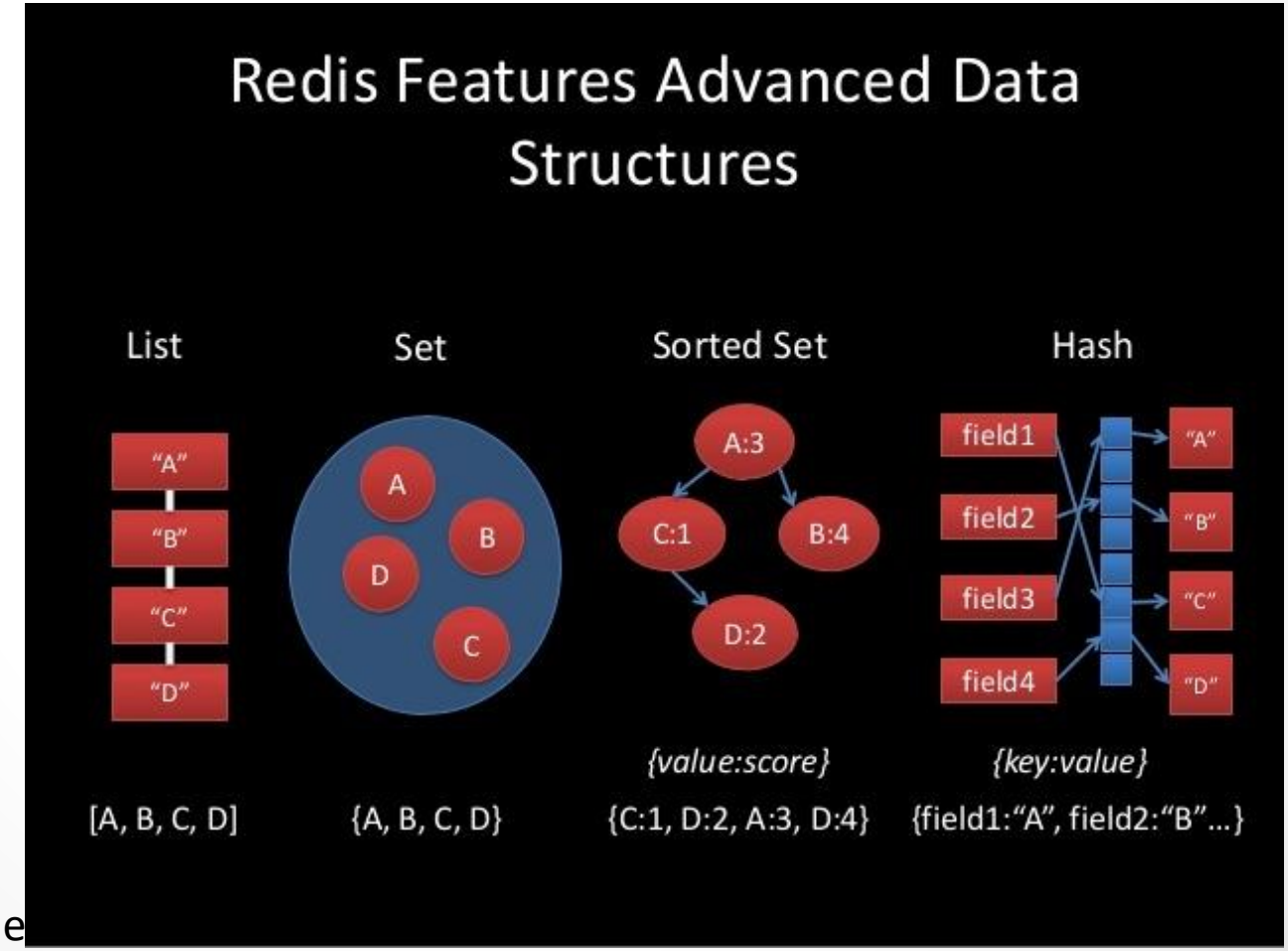
- Competes with **Memcached**(multi-threaded vs single-threaded)
- Sometimes called "**Memcached on steroids**"
- Supports up to **512M** for key/value size (Memcached supports 250 bytes)
- #1 key-value database
- #4 NoSQL database

redis

Redis replication



Redis data types



Sergey Morene

Task 18. 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.

Redis dependencies


```
<dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-data-redis</artifactId>
    <version>${spring.data.redis.version}</version>
</dependency>
<dependency>
    <groupId>io.lettuce</groupId>
    <artifactId>lettuce-core</artifactId>
    <version>5.0.2.RELEASE</version>
    <optional>true</optional>
</dependency>
```

Spring Security configuration


```
@Configuration
public class SessionConfig {

    @Bean
    public LettuceConnectionFactory connectionFactory() {
        return new LettuceConnectionFactory();
    }

}
```

Redis usage

- **HttpSession** was previously used in the application server
- Now **SecurityContext** is persisted in Redis
- Each new **HttpSession** has identifier supplied by browser

Task 19. Spring Security and Redis

1. Add new project dependencies:
2. Add new **Spring Security** configuration class
3. Clear all the keys in Redis server using **FLUSHALL** command.
4. Review **Redis monitor** log and newly added keys/values.
5. Review HTTP request headers in browser. How does request supply authentication info?
6. **Restart** Spring Boot application. Try to call REST service again.

Authorization

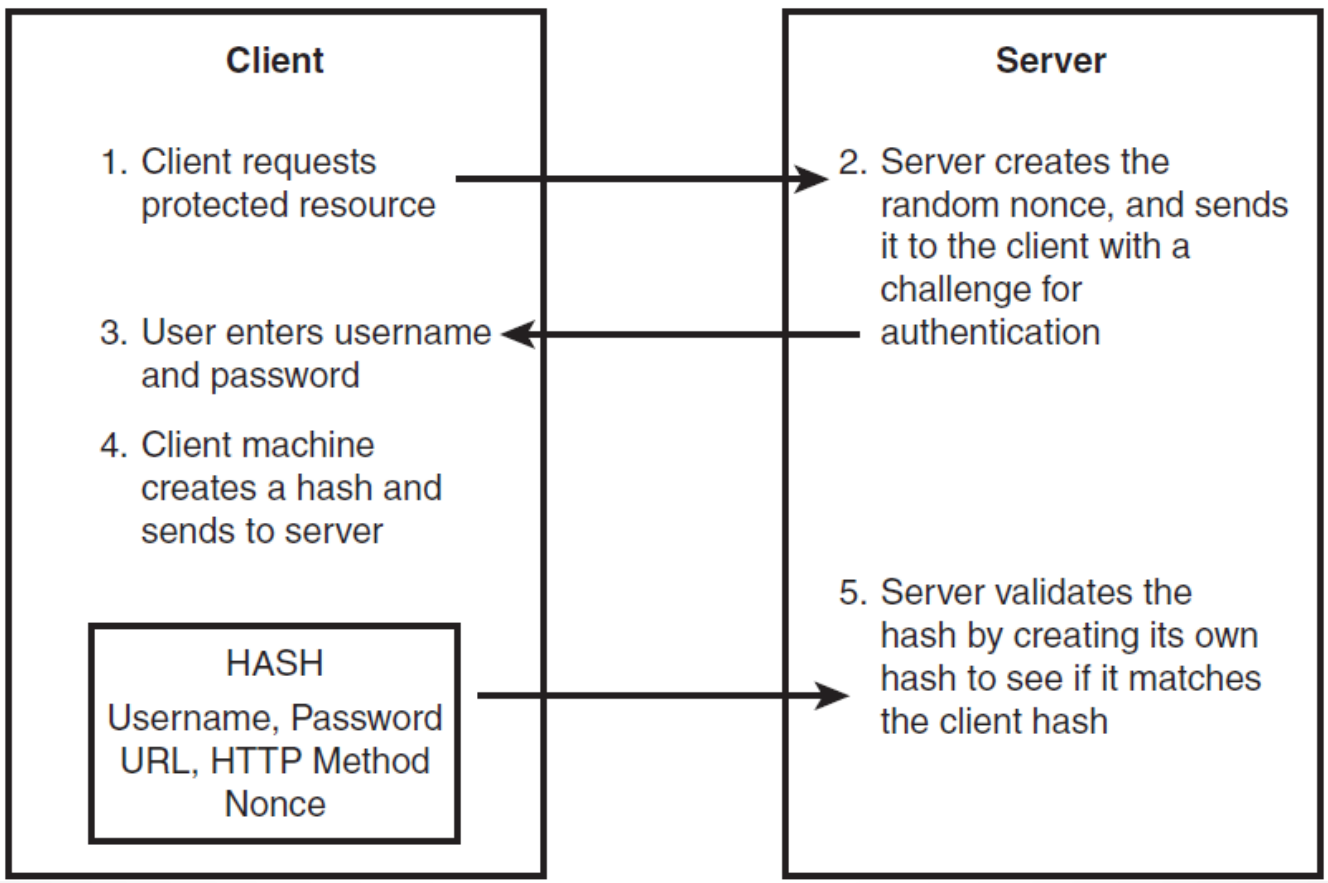
Sergey Morenets, 2018

Authorization

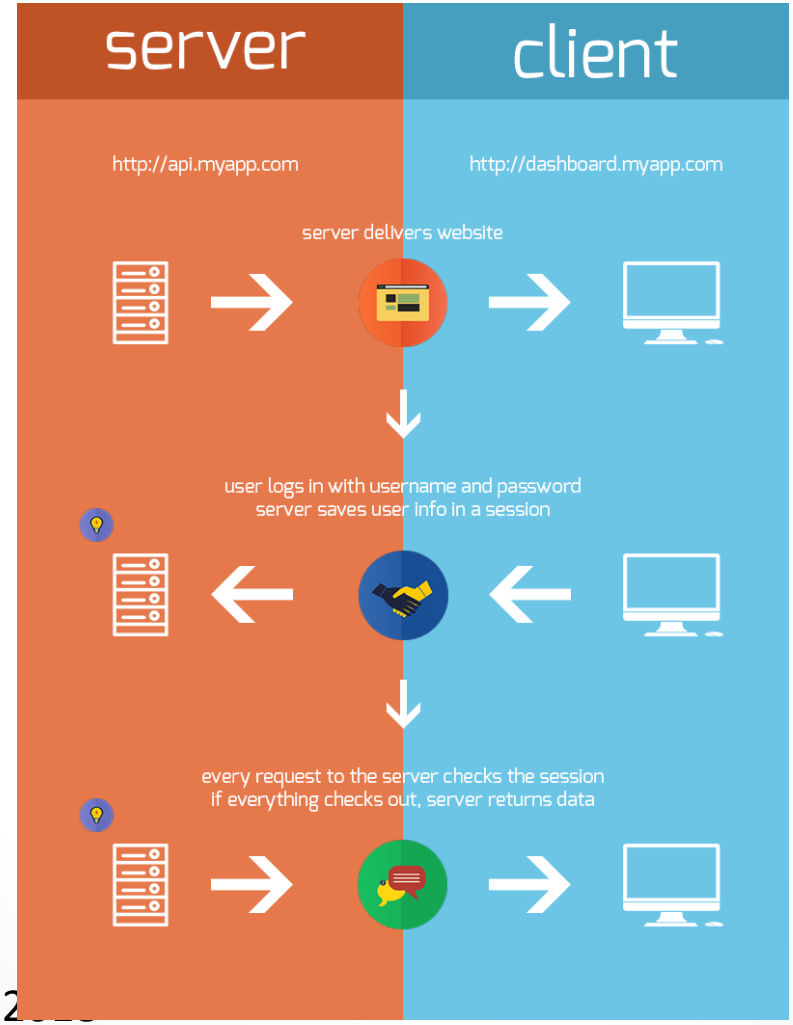

```
@RestController
@RequestMapping("/book")
public class BookController {
    @Autowired
    private BookRepository bookRepository;

    @RequestMapping
    @Secured("ADMIN")
    public List<Resource<Book>> findBooks() {
        List<Book> books = bookRepository.findAll();
```

Task 20. Googling

Digest authentication



Token-based authentication



OAuth 2.0

- Security protocol(or framework)
- Released in 2012
- Not backward compatible with OAuth 1.0
- Security access to HTTP service is requested by a client
- Access is granted by trusted third-party application
- Included in Spring Security 5

OAuth 2.0. Roles

- **Resource owner**

  You

- **Resource server**

  server with protected data

- **Client**

  application requested access to the resource server

- **Authorization server**

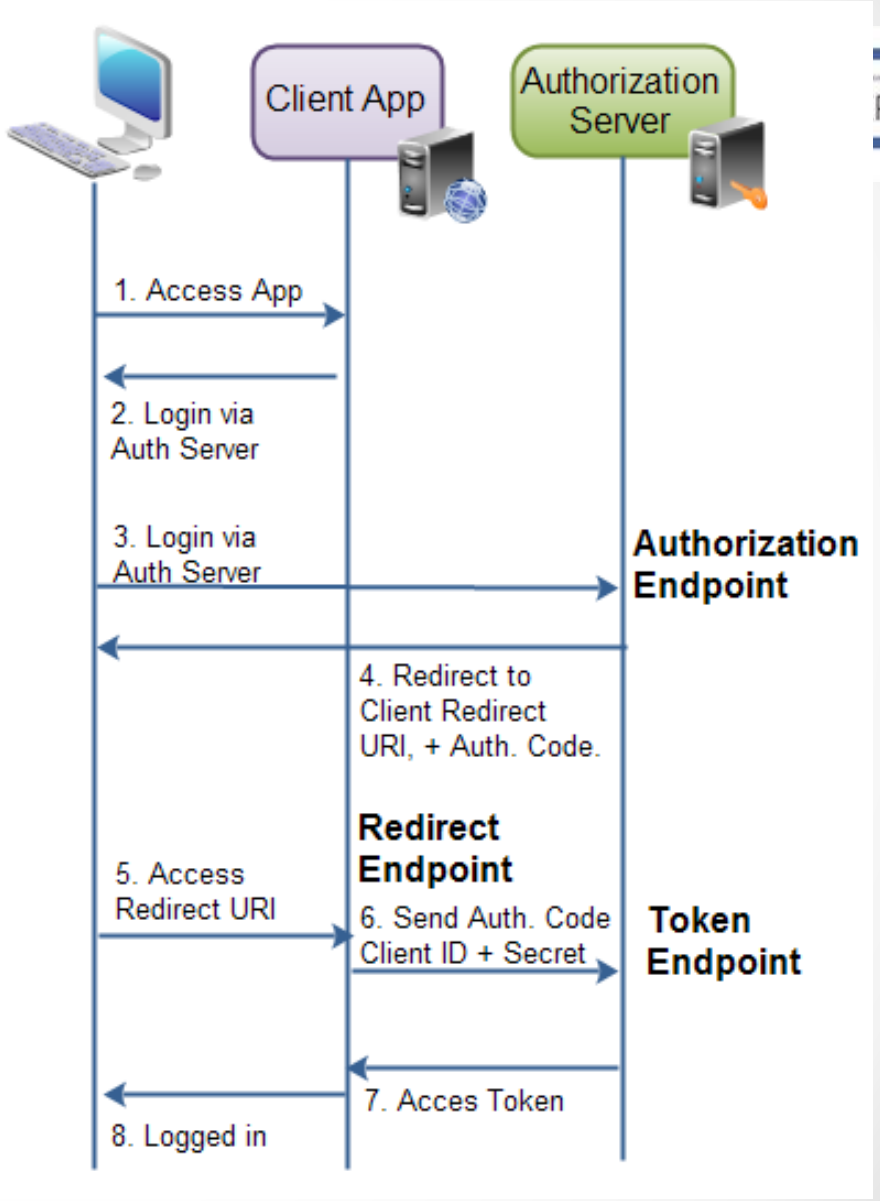
  server who grants access and sends access token to the client

  client sends token to the resource server

Access token

- Sent by client to the resource server inside request header
- Limited lifetime defined by authorization server
- Must be kept confidential
- Scope might be needed to specify your access rights

OAuth 2.0



Sergey Morenets, 2018

Spring OAuth 2.0 Client. Maven


```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-client</artifactId>
    <version>${spring.security.version}</version>
</dependency>
```

Spring OAuth 2.0 configuration


```
@Configuration
public class SecurityConfiguration extends
                        WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http)
            throws Exception {
        http.authorizeRequests()
          .anyRequest()
          .authenticated()
          .and()
          .oauth2Login()
          .defaultSuccessUrl("/success.html");
    }
```

Spring OAuth 2.0 configuration


```yaml
spring:
  security:
    oauth2:
      client:
        registration:
          facebook:
            clientId: 233668646673605
            clientSecret: 33b17e044ee6a4fa383f46ec6e28ea1d
          github:
            clientId: bd1c0a783ccdd1c9b9e4
            clientSecret: 1a9030fbca47a5b2c28e92f19050bb77824b5ad1
```

application.yml

OAuth 2.0 login page

Login with OAuth 2.0

GitHub

Facebook

http://localhost:8080/login

Task 21. OAuth 2.0 and custom servers

1. Review **rest-task21** module and its dependencies.
2. Run application and open http://localhost:8080/ URL. Try to login using Github/Facebook.
3. Review **AuthServerApplication** class.

Spring OAuth 2.0


```
<dependency>
    <groupId>org.springframework.security.oauth</groupId>
    <artifactId>spring-security-oauth2</artifactId>
    <version>${springsecurityoauth2.version}</version>
    <exclusions>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-core</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

Authorization server. Configuration


```
@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfiguration
            extends AuthorizationServerConfigurerAdapter {

    @Override
    public void configure(ClientDetailsServiceConfigurer clients)
            throws Exception {
        clients.inMemory().withClient("clientId").secret("secret")
            .authorizedGrantTypes("client_credentials")
            .scopes("read", "write");
    }
}
```

Authorization server. Endpoints

Endpoint	Description
/oauth/token	Generates access token
/oauth/check_token	Used by resource server to decode access token received by auth server
/oauth/confirm_access	Posted by resource user to grant access
/oauth/error	Used by auth server to display an error
/oauth/authorize	Used by resource server to generate authorization code

Access token

http://localhost:8080/oauth/token?grant_type=client_credentials


```
{
    "access_token": "2d56d008-78bd-44ab-8c0a-a317f3e66a04",
    "token_type": "bearer",
    "expires_in": 43199,
    "scope": "read write"
}
```

Authorization scopes. Rest controller


```
@GetMapping
@PreAuthorize("#oauth2.hasScope('read')")
public List<Resource<Book>> findBooks() {
    List<Book> books = bookRepository.findAll();
```

Resource server configuration

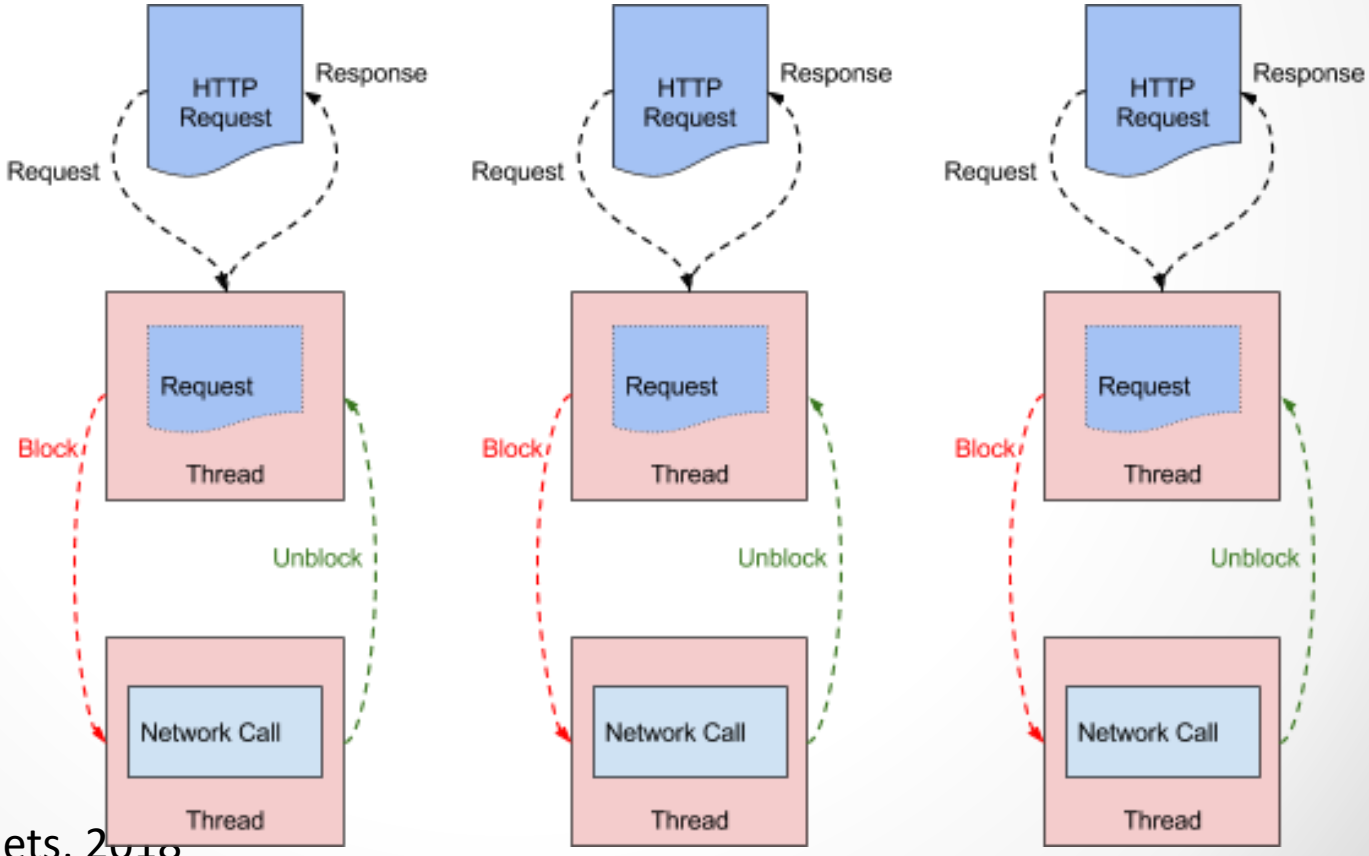

```
@Configuration
@EnableResourceServer
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class ResourceServerConfiguration
  extends GlobalMethodSecurityConfiguration {

    @Override
    protected MethodSecurityExpressionHandler
            createExpressionHandler() {
        return new OAuth2MethodSecurityExpressionHandler();
    }
}
```

Task 22. Spring Security OAuth 2.0

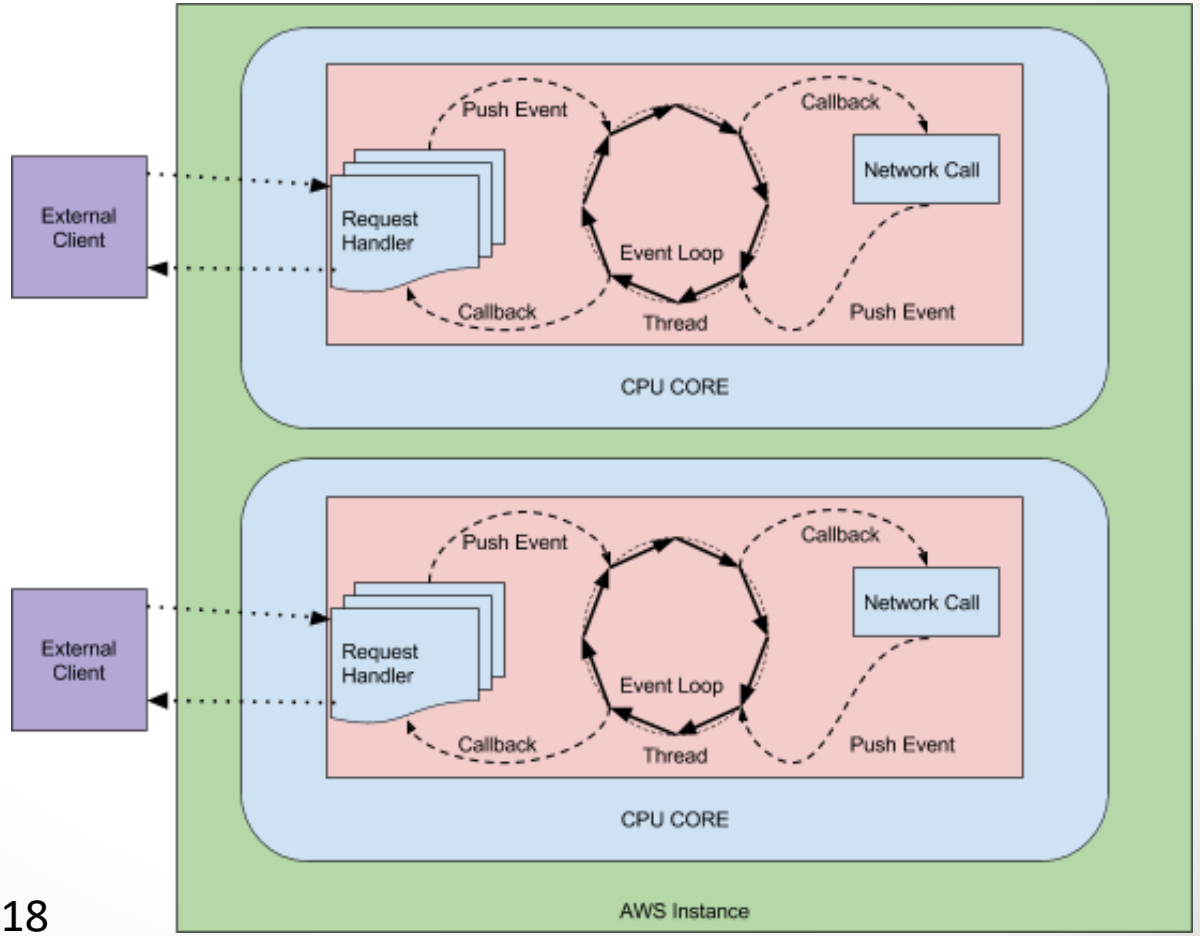
1. Review **rest-task22** module and its dependencies.
2. Copy BookController from rest-task10, run application and open http://localhost:8080/book URL.
3. Open Postman and create new requests with following parameters.
4. What is the server response? How will server handle your response if you remove “Authorization” header?

Sergey Morenets, 2018

Blocking I/O


HTTP Request
Response
Request
Request
Block
Thread
Unblock
Network Call
Thread
HTTP Request
Response
Request
Request
Block
Thread
Unblock
Network Call
Thread
HTTP Request
Response
Request
Request
Block
Thread
Unblock
Network Call
Thread

Non-blocking I/O

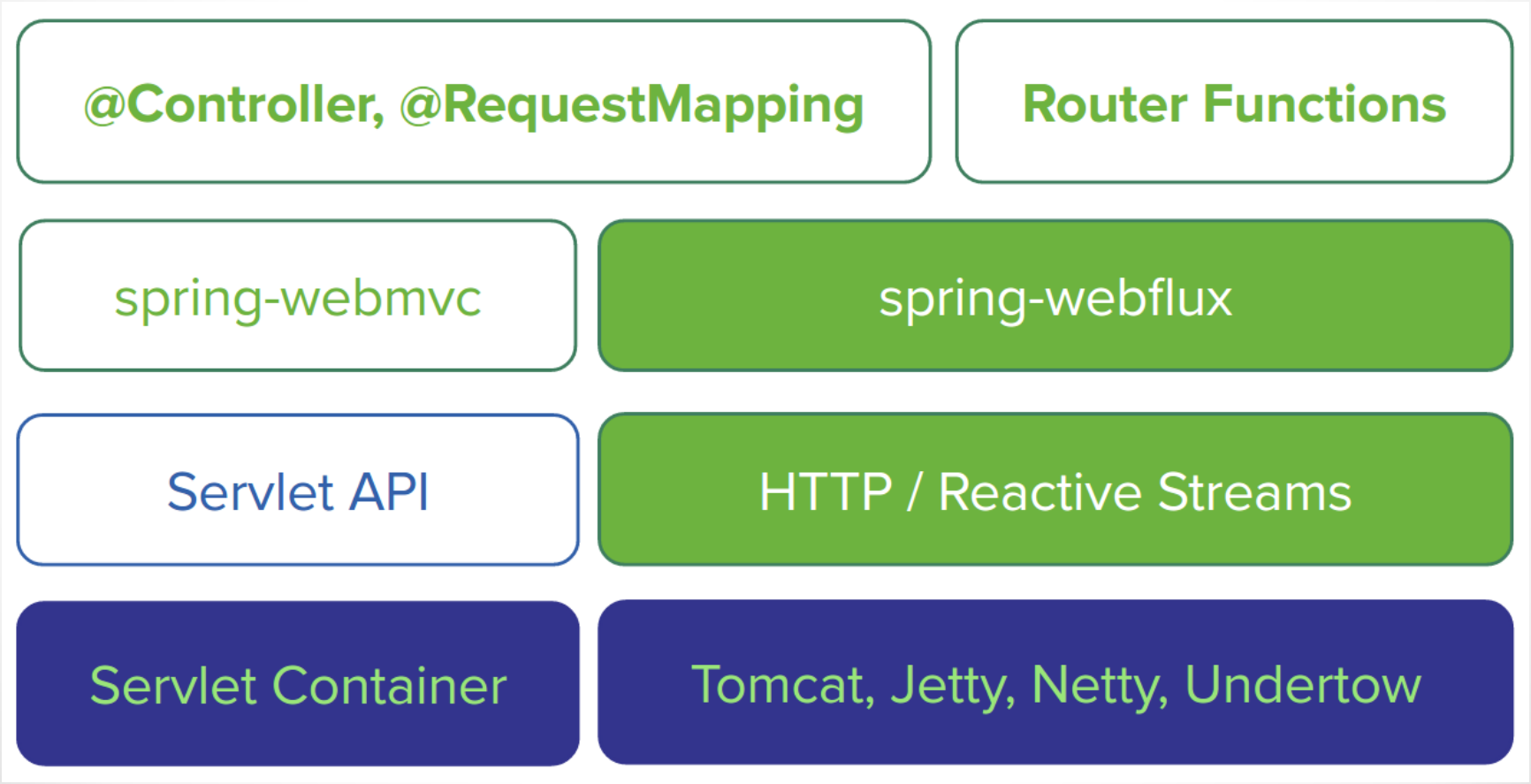


Sergey Morenets, 2018

Spring 5 WebFlux

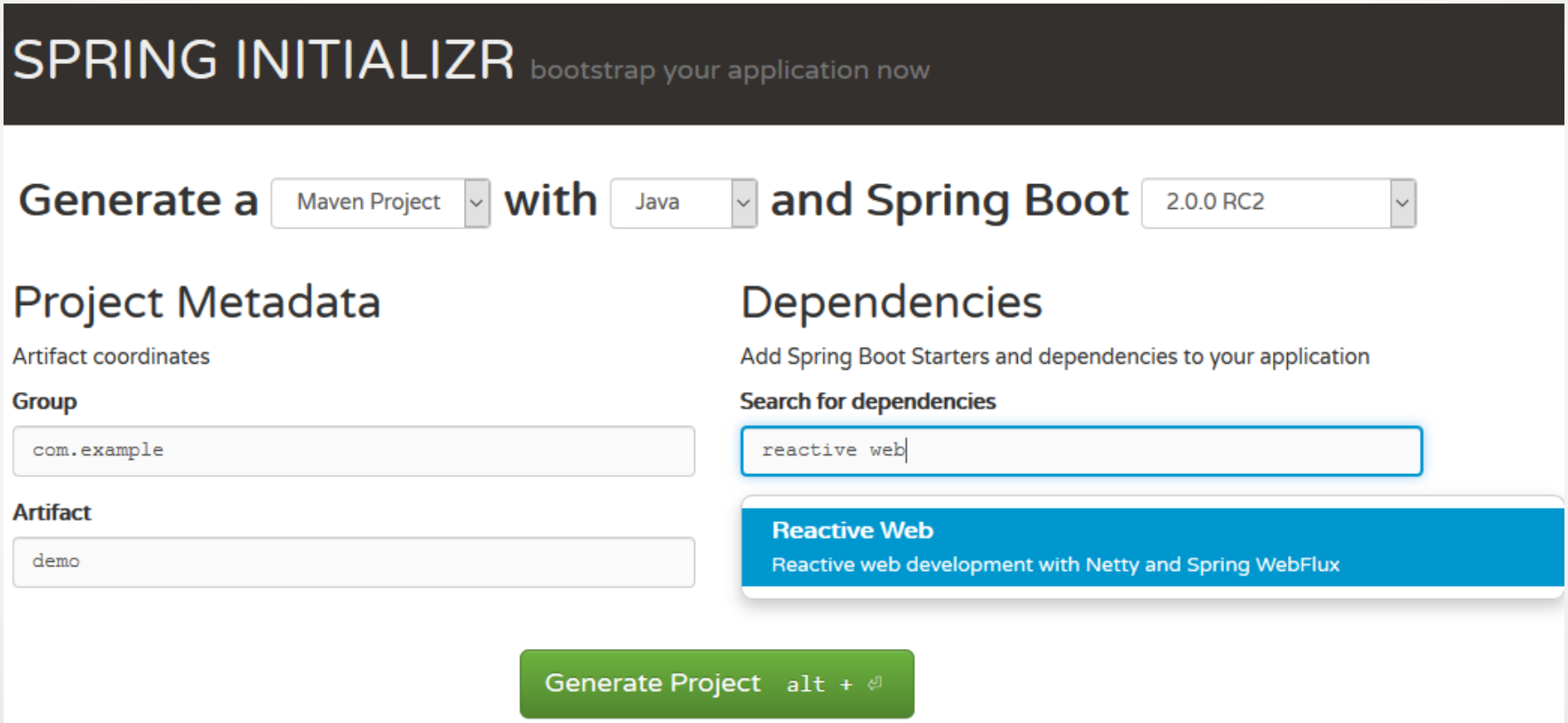
- ✓ New **spring-web-flux** module adapting **Reactive Streams** specification
- ✓ Based on **Project Reactor**
- ✓ Reusing Spring MVC programming model but in non-blocking mode
- ✓ Based on non-blocking Servlet API or native SPI connectors(Netty, Undertow)
- ✓ Supported by Tomcat/Jetty/Netty/Undertow
- ✓ Reactive client **WebClient**
- ✓ Support unit-testing with **WebTestClient**

Spring Web Flux



Spring Web Flux


```
@FunctionalInterface
public interface HandlerFunction<T extends ServerResponse> {

    /**
     * Handle the given request.
     * @param request the request to handle
     * @return the response
     */
    Mono<T> handle(ServerRequest request);
```



Sergey Morenets, 2018

Spring Web Flux. Maven


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

REST controller. Mono


```
@RestController
@RequestMapping
public class ProductController {

    private ProductService productService;

    @GetMapping(path = "/{id}")
    public Mono<Product> get(
            @PathVariable("id") int id) {
        return productService.find(id);

    }
}
```

REST controller. Flux


```
@RestController
@RequestMapping("/random")
public class RandomController {

    private Random random = new Random();

    @GetMapping
    public Flux<Integer> generate() {
        return Flux.fromStream(Stream.
                generate(random::nextInt)).
                delayElements(Duration.ofSeconds(1));
    }
```

REST controller. Flux


```
@RestController
@RequestMapping("/random")
public class RandomController {

    private Random random = new Random();

    @GetMapping(produces=MediaType.TEXT_EVENT_STREAM_VALUE)
    public Flux<Integer> generate() {
        return Flux.fromStream(Stream.
                generate(random::nextInt)).
                delayElements(Duration.ofSeconds(1));
    }
```

Web client


```
WebClient client = WebClient.
        create("http://localhost:8080");
Flux<Integer> flux = client.get().uri("/random")
    .accept(MediaType.TEXT_EVENT_STREAM)
    .exchange()
    .flatMapMany(body ->
            body.bodyToFlux(Integer.class));

    flux.subscribe(System.out::println);
```

Task #23. Spring Web Flux

1. Add Sping WebFlux dependency to **pom.xml:**
2. Create **REST controller** that returns stream of prime numbers as Flux type. Run Spring Boot application and verify in the browser that prime numbers are displayed on the page
3. Create REST client that uses **WebClient** class and prints received prime numbers to the console.

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
```

Security testing

arachni

web application security scanner framework

GAUNTLT

BE MEAN TO YOUR CODE AND LIKE IT

clair

{} Find Security Bugs

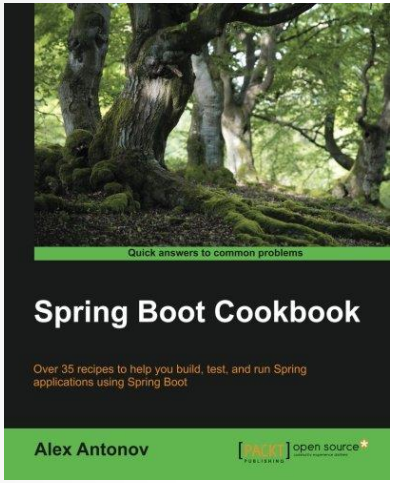
OWASP

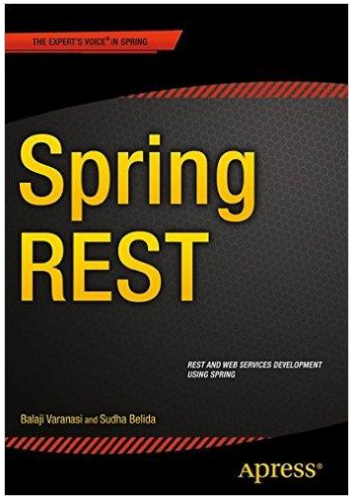
Sergey Morenets, 2018

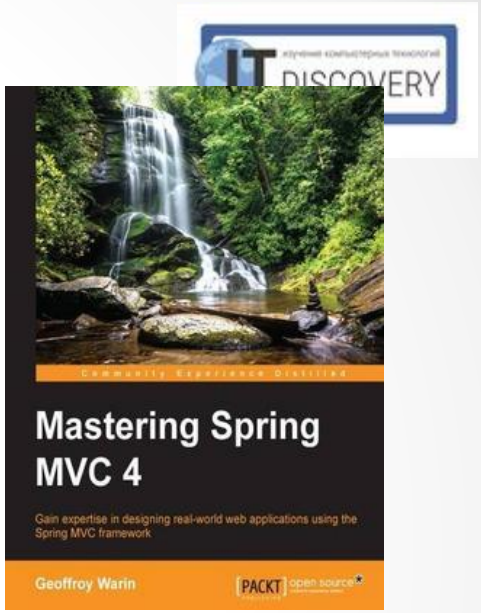
Future

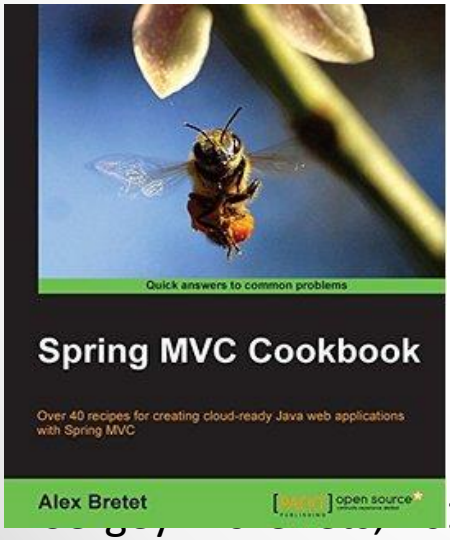
- ✓ Pagination & sorting
- ✓ Documentation
- ✓ Spring Data REST
- ✓ Server-side events

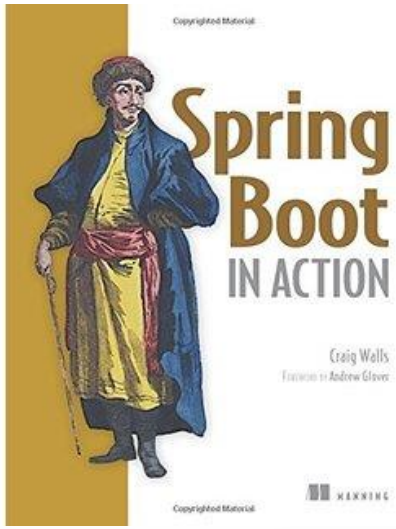
Books


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


IT DISCOVERY
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
MANNING
Copyrighted Material


Spring Security
Eugen Paraschiv
[PACKT] VIDEO

Tutorials

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

✓ Sergey Morenets, sergey.morenets@gmail.com