Пишем load balancer на Spring

Sergey Morenets

October, 13th – 14th 2018

DEVELOPER	12 YEARS
TRAINER	4 YEARS
WRITER	3 BOOKS

FOUNDER

ITSimulator

IT DISCOVERY

IT DISCOVERY

SPEAKER

JPRIME

JAVA DAY MINSK 2013

polish jug

Dev(Talks):

DEVOXX™ POLAND

JEE Conf

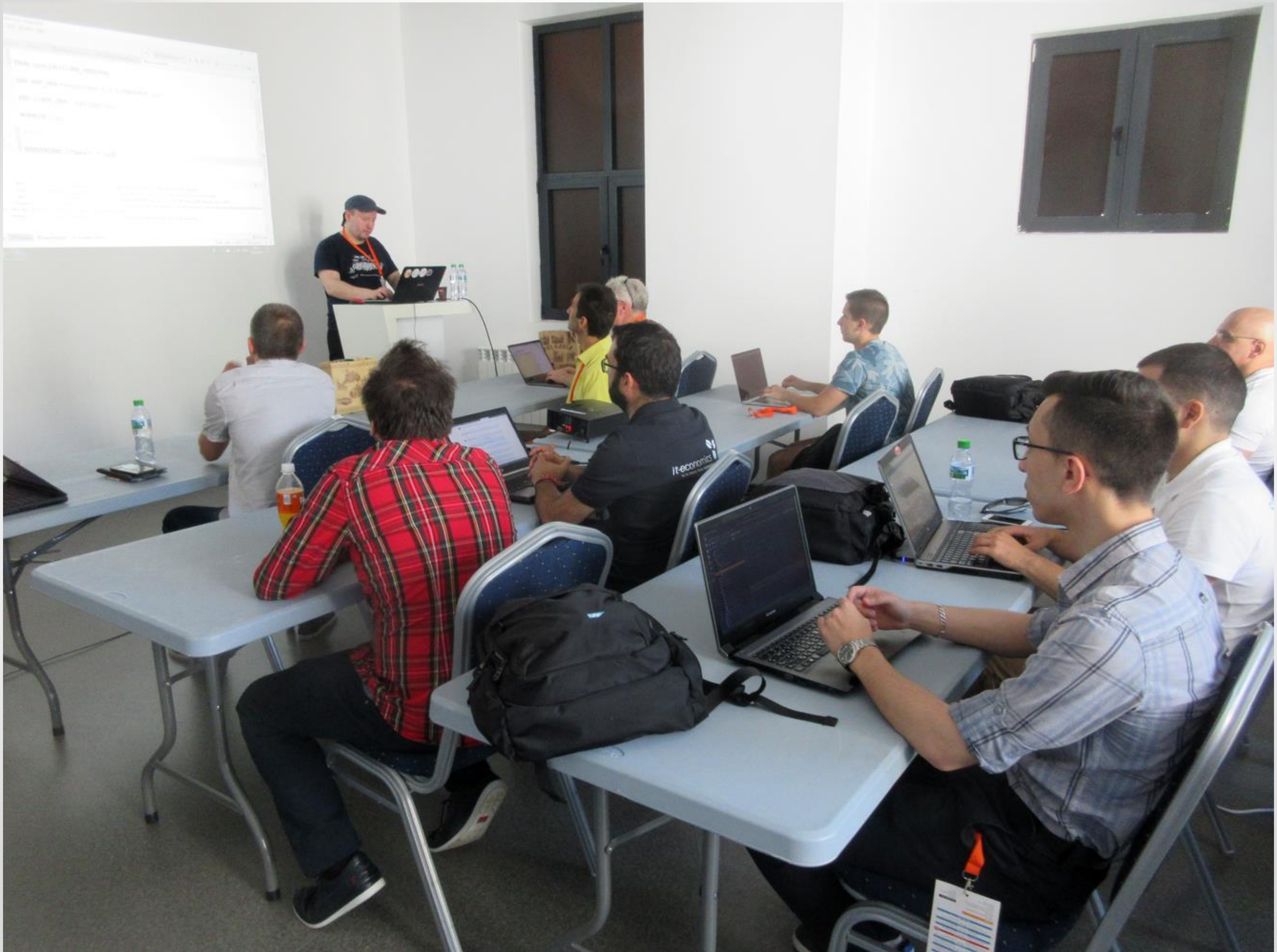
JAVA DAY 2015

IT talk

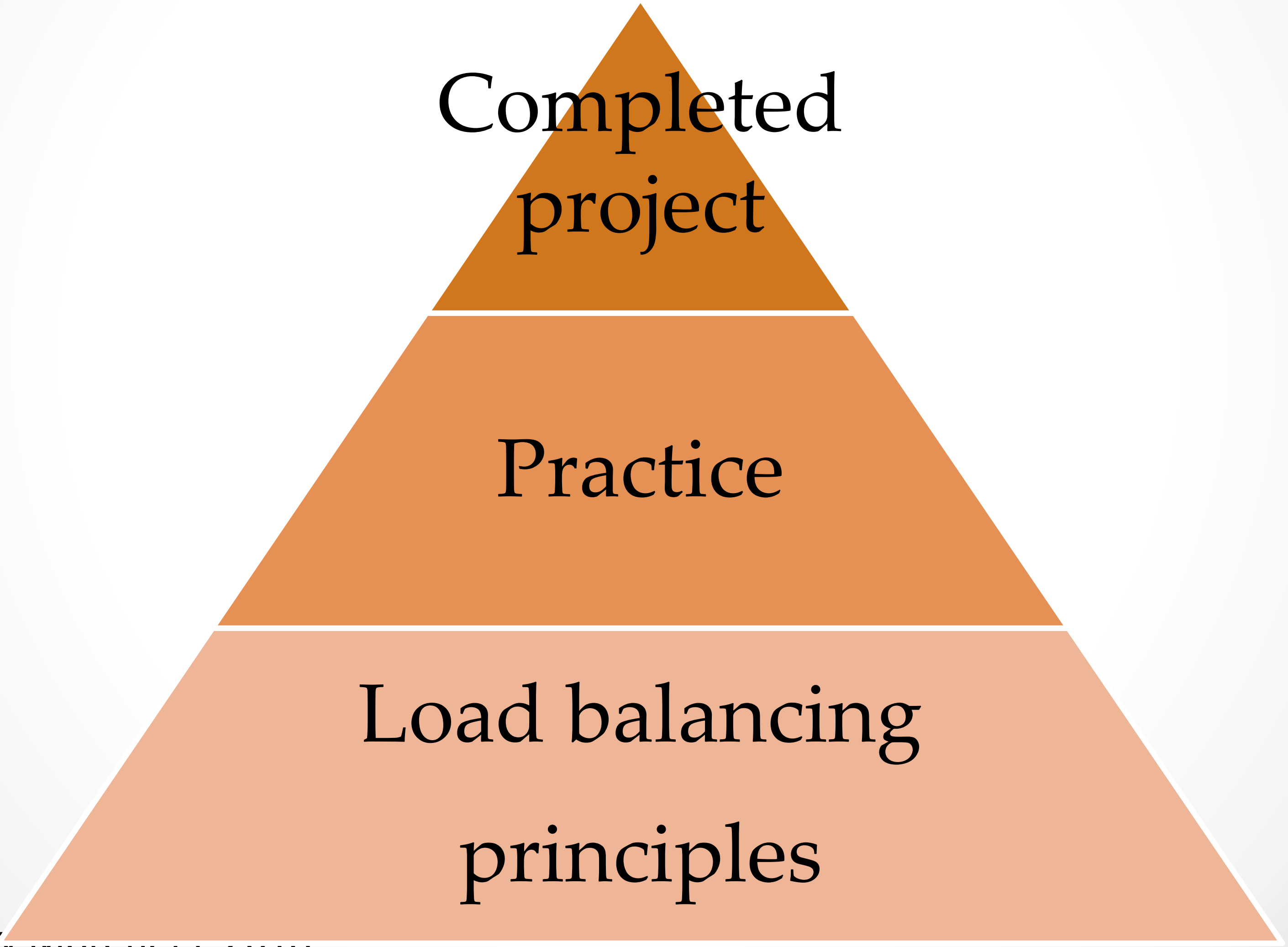
Java frameworks day

GEEKS LAB

@sergey-morenets
DEVOXX POLAND

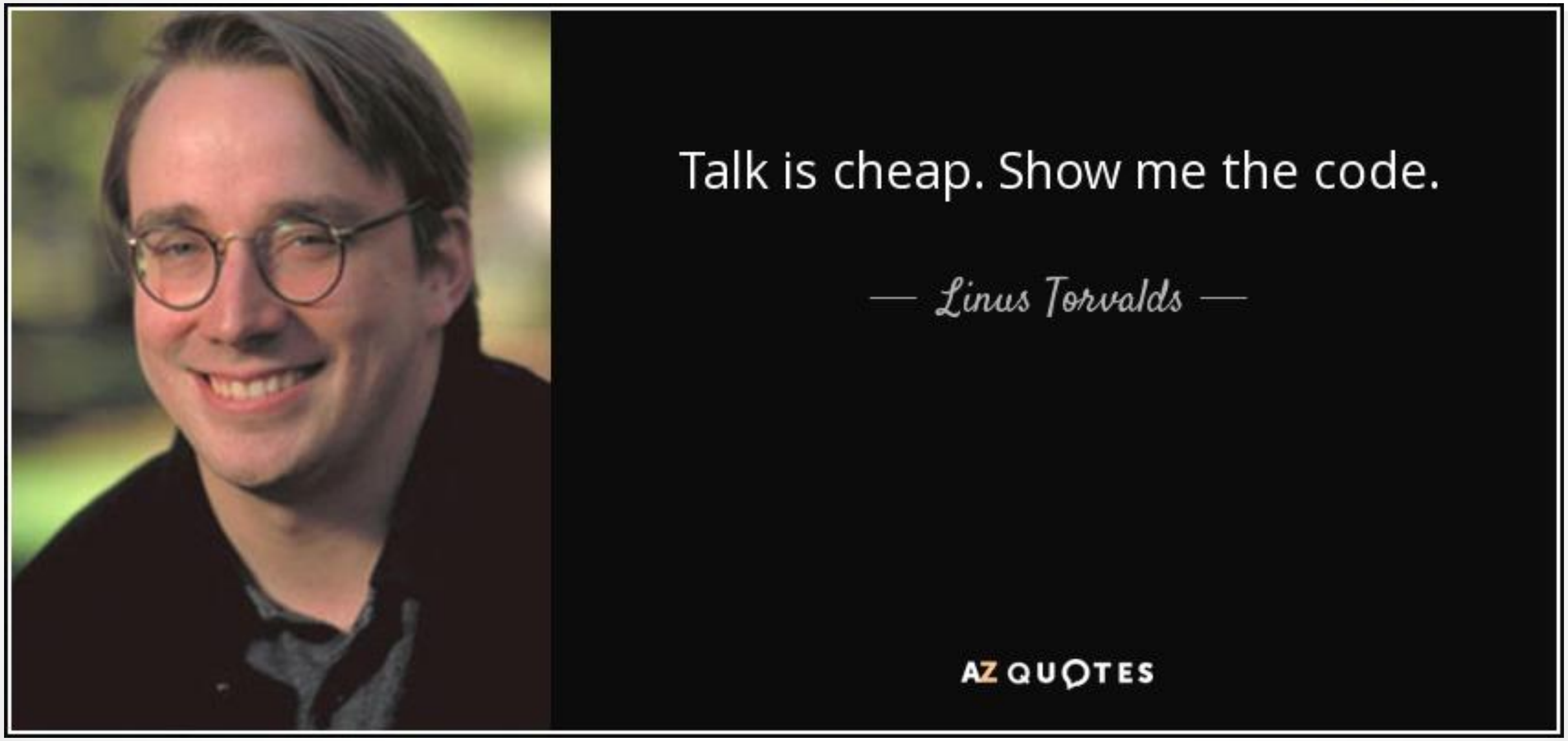

it-economics

Goals



Agenda

- ✓ REST and REST services
- ✓ Spring and Spring Boot usage
- ✓ Client-side and server-side load balancers
- ✓ Algorithms and principles
- ✓ Functional testing
- ✓ Monitoring using Spring Boot Actuator
- ✓ Error handling
- ✓ Configuration and customization
- ✓ Performance and security testing
- ✓ Reactive programming


Talk is cheap. Show me the code.

— Linus Torvalds —

Task 1. Reviewing project

1. Import **load-balancer-rest** project into your IDE (you should import it as Maven or Gradle project) and open **book-service** sub-project.
2. Review project structure, dependencies and REST services.
3. Run **BookApplication** and try to invoke REST services using Postman application.
4. Review application logs and Spring configuration

Load balancing

- Sergey Morenets, 2018

Load balancing

- ✓ Distributing incoming network traffic through group of back-end servers (server pool)
- ✓ Client request router
- ✓ Allows horizontal scaling, optimize resource use, maximizes speed and capacity utilization
- ✓ Automatic configuration and error handling
- ✓ Health checking and failovers
- ✓ DDos attack protection
- ✓ Client authentication and firewall
- ✓ Caching

Load balancers

NGINX

HAPROXY

Amazon EC2

CLOUDFLARE®

Load balancers

Apache Camel

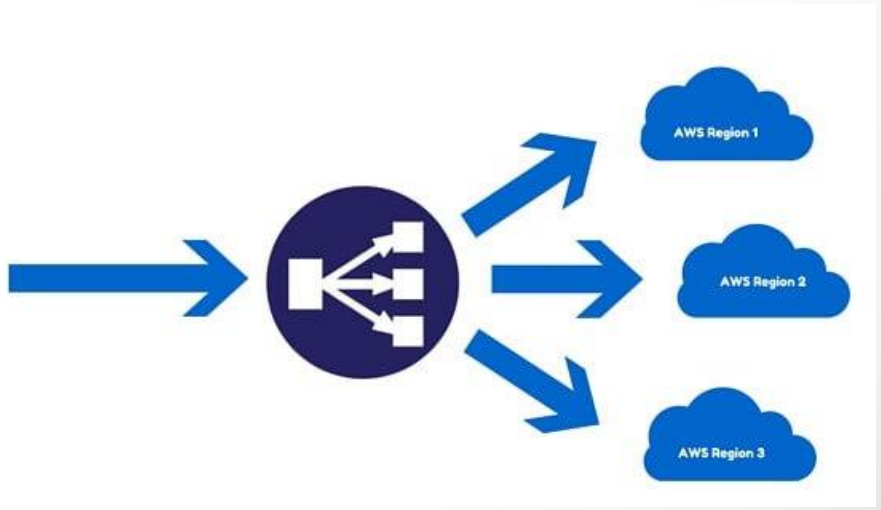
NETFLIX

Load balancer workflow

- ✓ Configure back-end servers (static or dynamic)
- ✓ Health checks and monitoring
- ✓ Choose back-end server to route request
- ✓ Session persistence
- ✓ Error handling
- ✓ Elastic load balancing

Load balancing algorithms

- ✓ Round Robin
- ✓ Least utilization
- ✓ Geographic location



Task 2. Order service

1. Review **order-service** sub-project and its dependencies.
2. Review **OrderController** class and complete its **makeOrder** REST service. This service should query **book-service** and gets book by id.
3. Run both services and make sure they communicate properly. What will happen if book-service shuts down?
4. Write integration (or unit-test) for OrderController.

Mapping annotations

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

Spring configuration

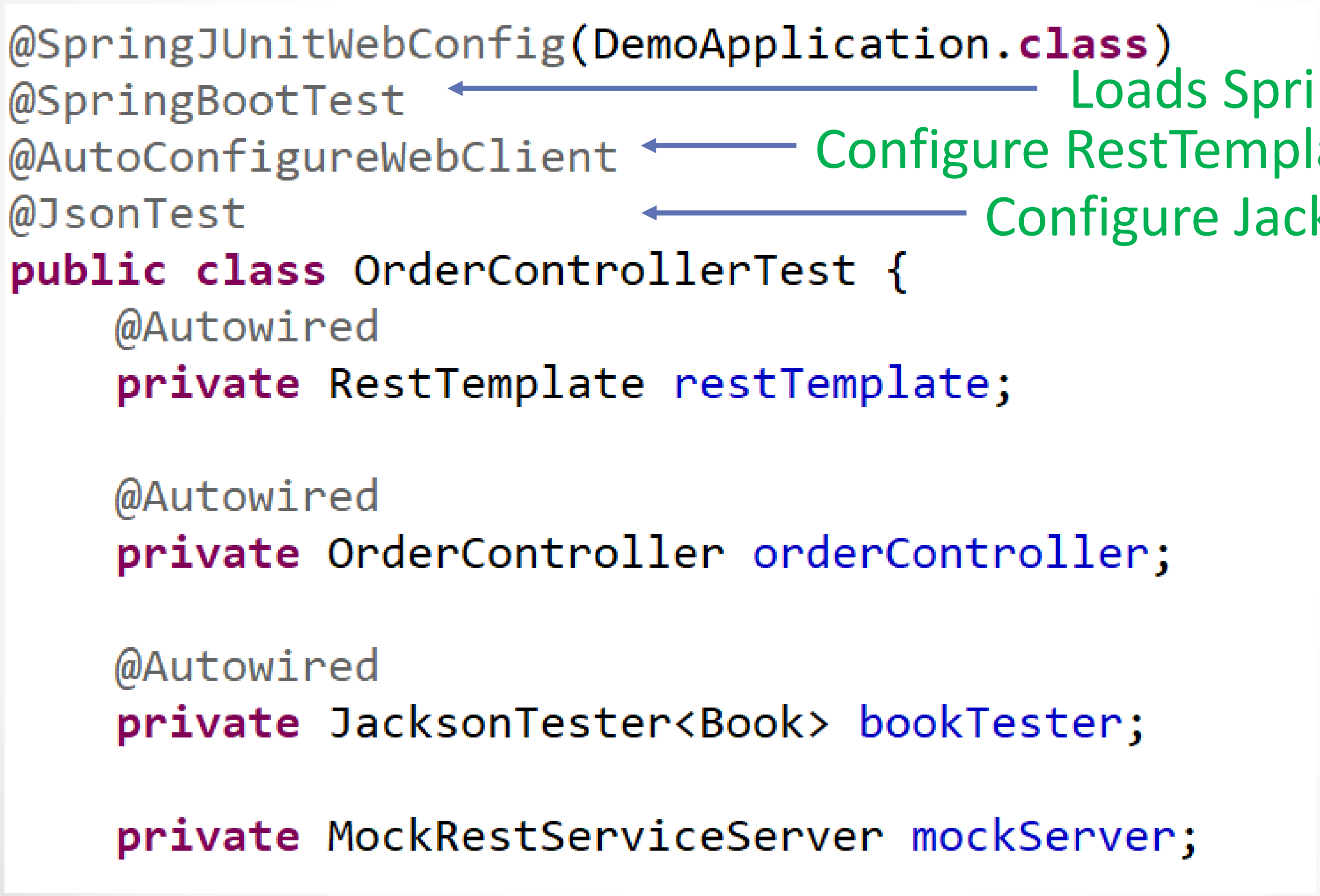
```
@Autowired
private RestTemplate restTemplate;
```

Spring bean

RestTemplateBuilder

Method	Description
rootUri()	Applies root URI for each request
Interceptors()	Applies client request interceptors
errorHandler()	Specifies response error handler
basicAuthorization()	Specifiies username/password for HTTP basis authorization
customizers()	Sets customizers to change output RestTemplate
setConnectionTimeout()	Change connection timeout
setReadTimeout()	Changes read timeout

Testing REST services. Configuration



```
@SpringJUnitWebConfig(DemoApplication.class)
@SpringBootTest
@AutoConfigureWebClient
@JsonTest
public class OrderControllerTest {
    @Autowired
    private RestTemplate restTemplate;

    @Autowired
    private OrderController orderController;

    @Autowired
    private JacksonTester<Book> bookTester;

    private MockRestServiceServer mockServer;
```

Loads Spring context

Configure RestTemplateBuilder

Configure JacksonTester

Testing REST services. Tests

Configure MockServer

```
@BeforeEach
public void setUp() {
    mockServer = MockRestServiceServer.createServer(restTemplate);
}

@Test
public void makeOrder_validBookId_orderCreated() throws IOException
    Book book = new Book();
    book.setId(1);
    mockServer.expect(requestTo("http://localhost:8081/books"))
        .andRespond(withSuccess(bookTester.write(book).getJson(),
                MediaType.APPLICATION_JSON_UTF8));

    Order order = orderController.makeOrder(1);

    mockServer.verify();
    assertEquals(order.getBookId(), 1);
}
```

Mock response

Task 3. Order service performance

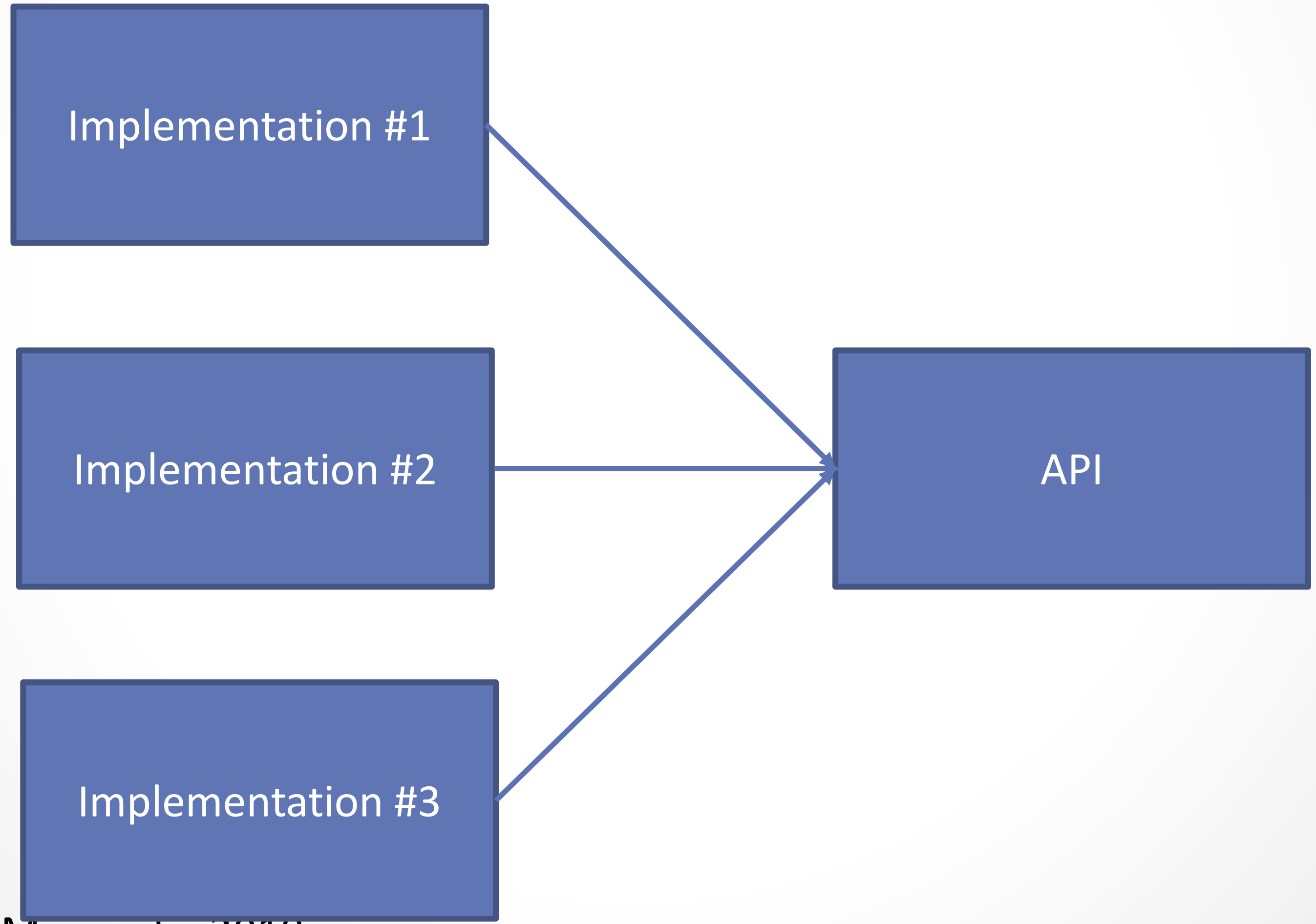
1. Download and install Apache JMeter 5.0 (and later), for example, into c:/JMeter folder.
2. Run JMeter executable file in c:/Jmeter/bin folder.
3. Click on **Test Plan** node and add new Thread Group. Specify number of threads (users), for example, 10, then ramp-up time and loop count (10 or 20 is recommended).
4. Click on **Thread Group** node and add new Sampler -> HTTP request

Client load balancer

vs

Server load balancer

Client-side load balancer



Task 4. Starting balancer component

1. Review client-balancer sub-project.
2. Add Spring Boot startup configuration
3. Implement basic interfaces and classes that will be part of load-balancer API
4. Add static configuration that will store list of back-end servers.

Task 5. Link order-service and balancer

1. Include client-balancer dependency to the order-service project.
2. Add configuration property in order-service that will store back-end server-list
3. Create implementation of the load balancer API that randomly chooses target server and declare Spring bean of this implementation.

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

Enable auto-configuration

```java
@Component
public class CustomService {

}
```

```java
public class CustomService {

}
```

```java
@Configuration
public class CustomServiceAutoConfiguration {

    @Bean
    @ConditionalOnMissingBean
    public CustomService customService() {
        return new CustomService();
    }
}
```

Task 6. Validation and auto-configuration

1. Add validation annotations to verify that configuration data are valid
2. Enable auto-configuration for the default implementation of the client-side load balancer

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

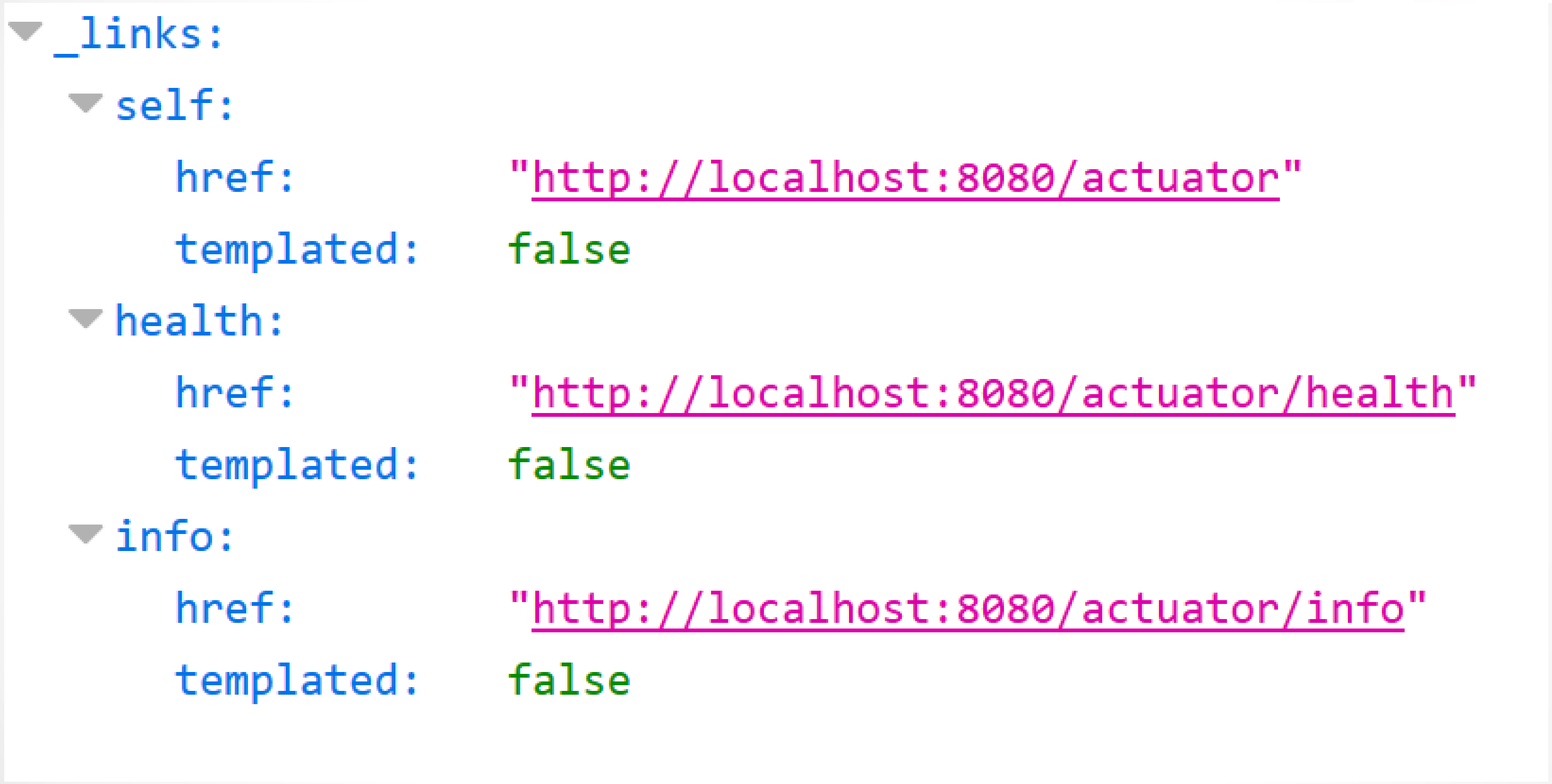
Spring Boot Actuator

- ✓ 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

Spring Boot Actuator. Maven

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

Spring Boot Actuator. Endpoints



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

https://localhost:8080/actuator

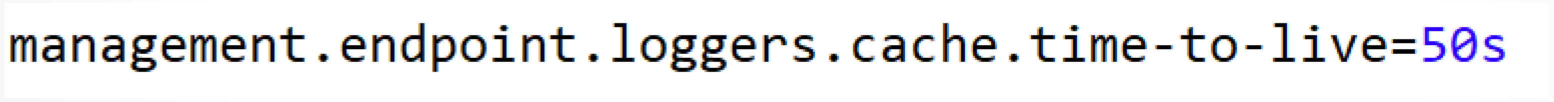
Spring Boot Actuator. Endpoints

Endpoint	Description
/actuator/beans	Displays list of Spring beans in the application
/actuator/metrics	Shows application metrics
/actuator/env	Exposes environment variables
/actuator/loggers	Allows to read/change logger settings
/actuator/health	Shows application health information
/actuator/mappings	Display a list of @RequestMapping info
/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=*
```

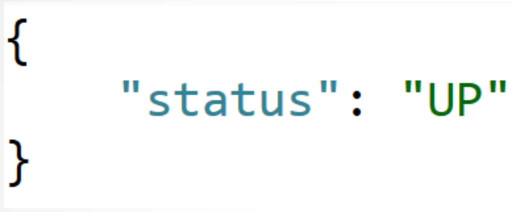
Enable all endpoints



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

application.properties

Health information



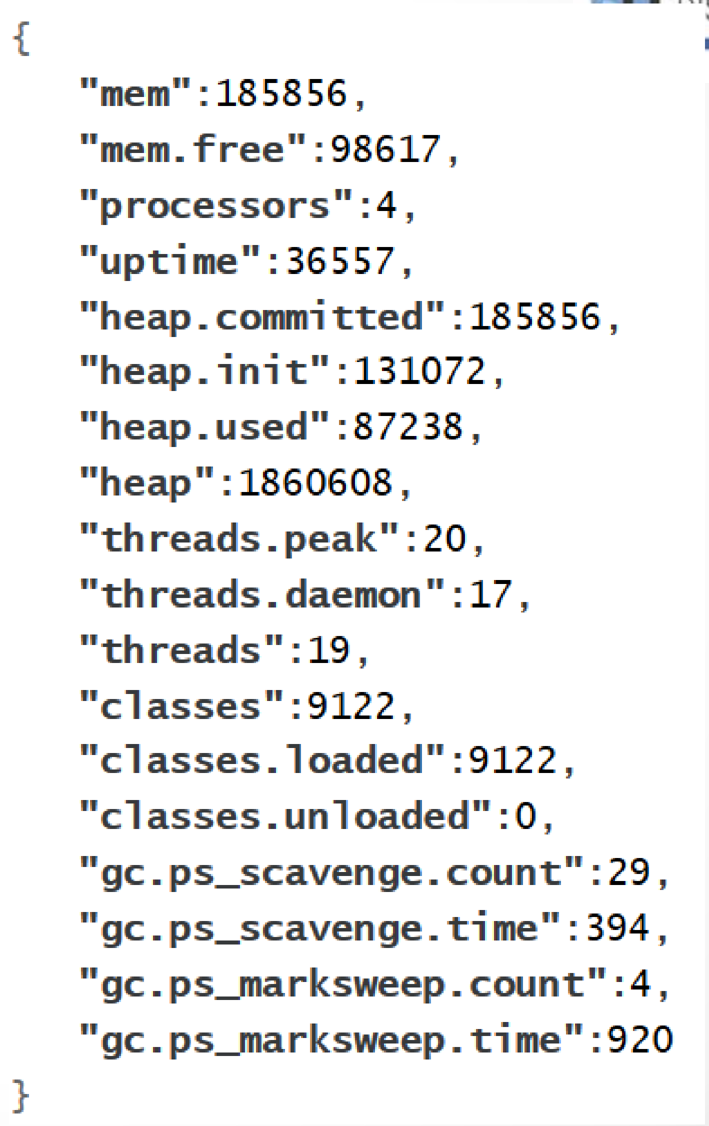
```
{
    "status": "UP"
}
```

/actuator/health

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

Metrics. Spring Boot 1.5

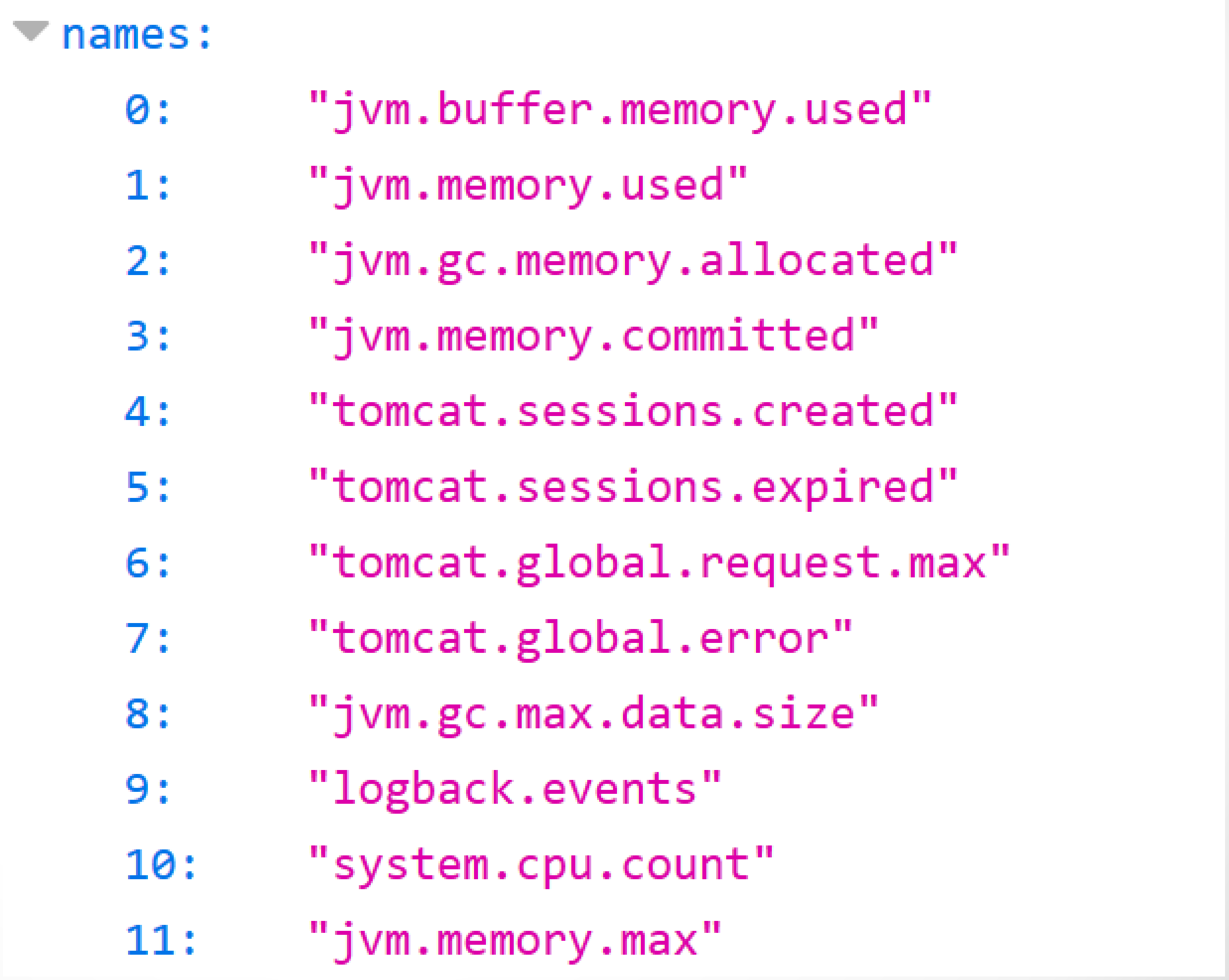
/application/metrics



```
{
    "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
}
```

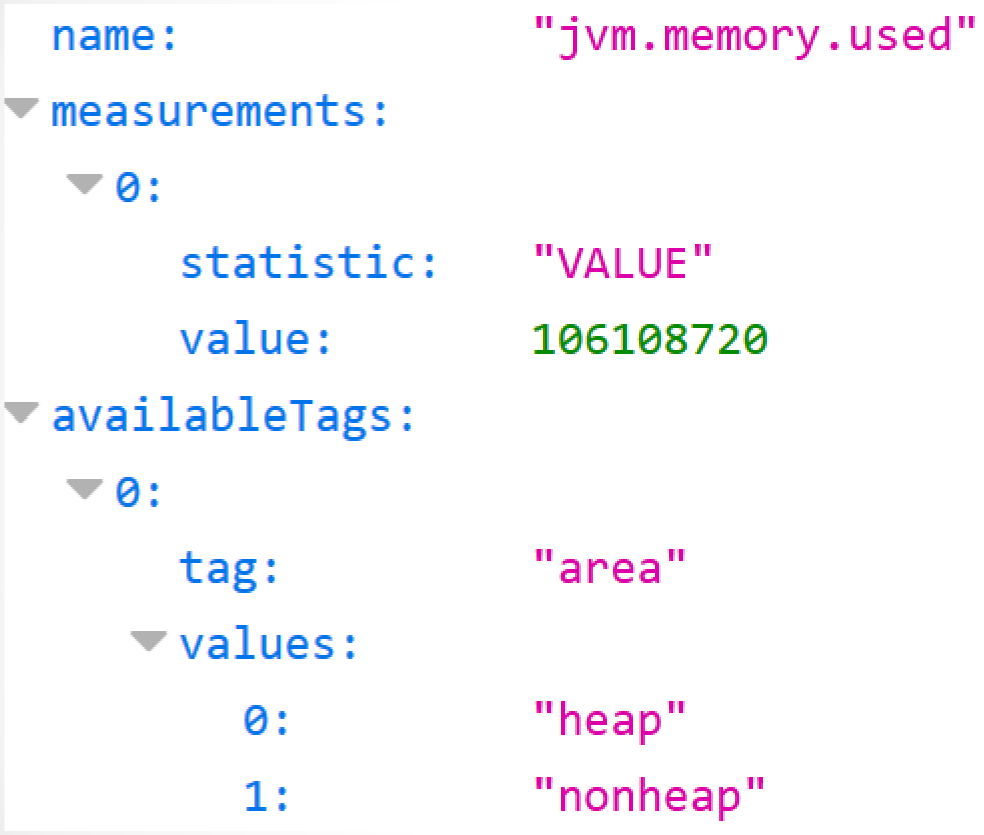
Metrics. Spring Boot 2.0

/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

IT DISCOVERY

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

WAVEFRONT

New Relic

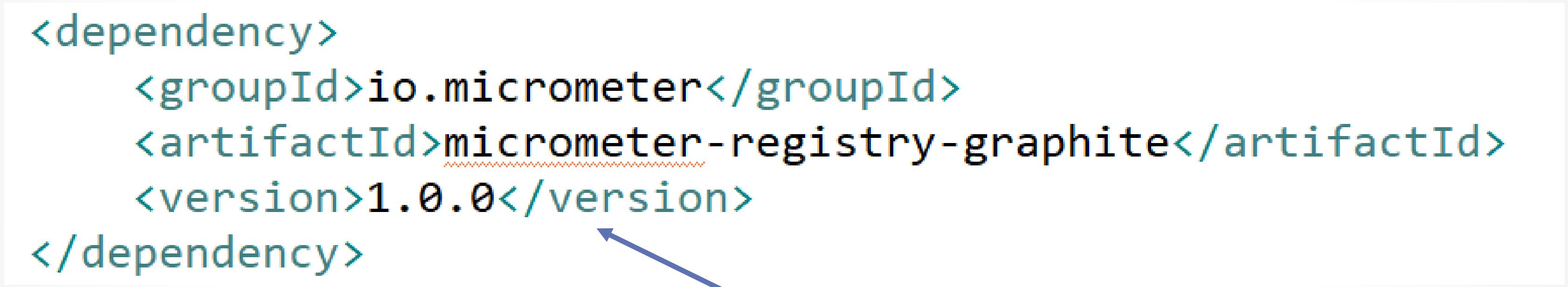
InfluxDB

graphite

Graphite

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

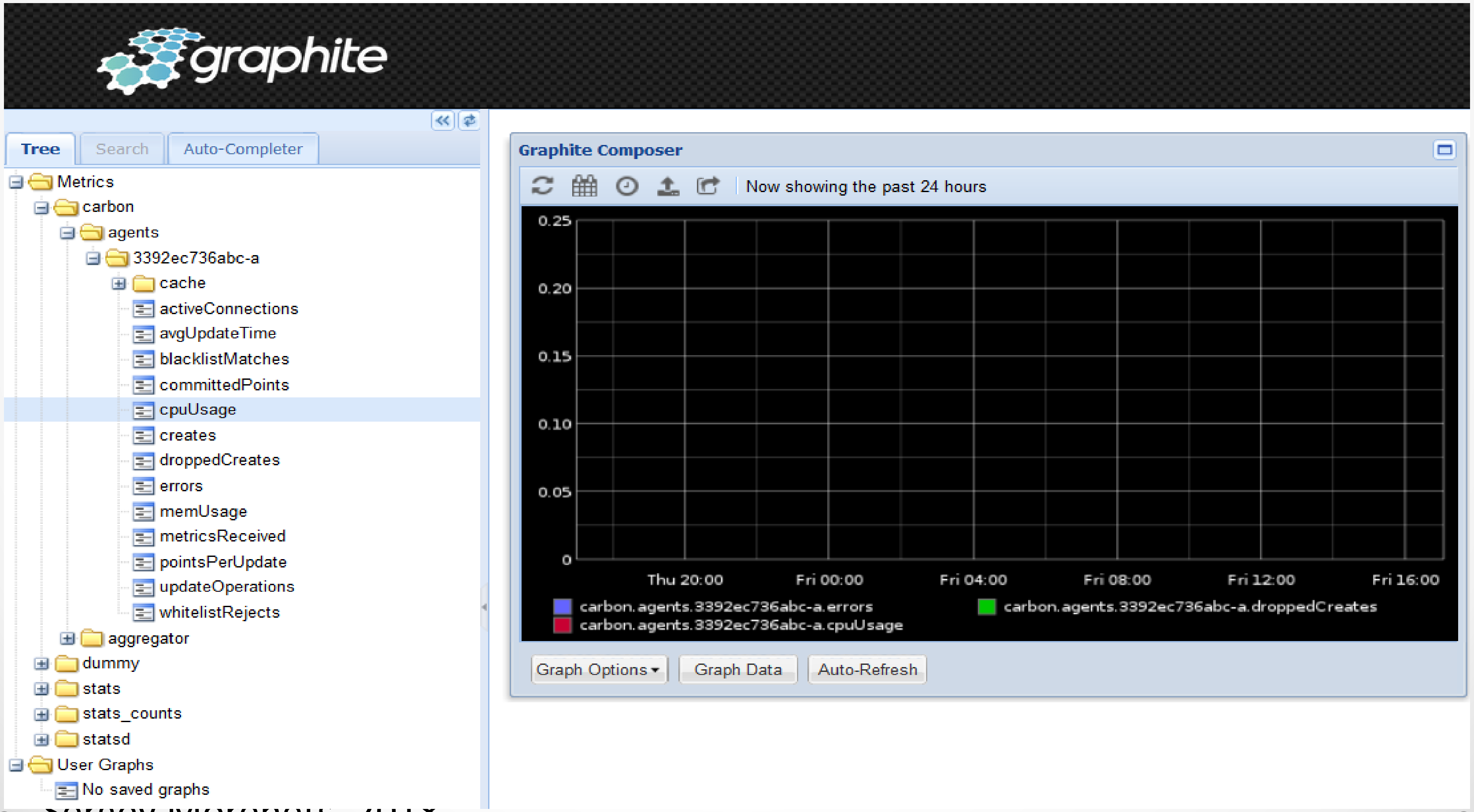
Micrometer and Graphite



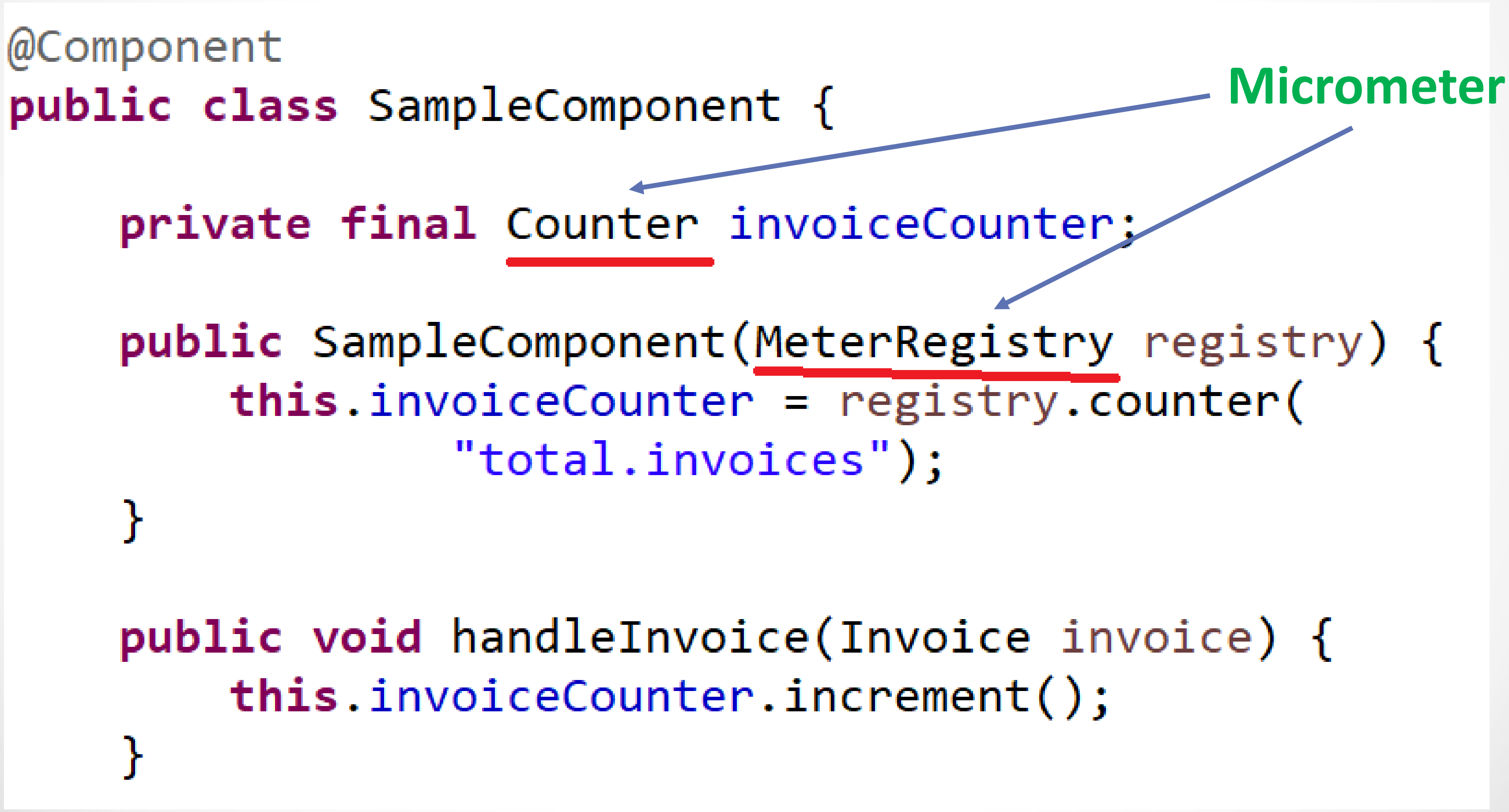
```
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-graphite</artifactId>
    <version>1.0.0</version>
</dependency>
```

Use default settings to report metrics to Graphite server

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();
    }
```

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"
    }
}
```

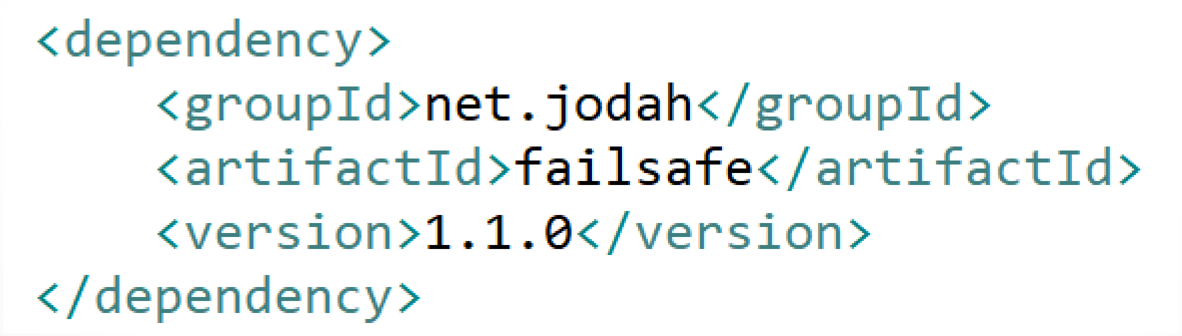
/actuator/info

Task 7. Load balancer metrics

1. Add new metrics to order-service project that represent statistics of load balancer activities.
2. Add Micrometer provider dependency to order-service project, for example, for Graphite
3. Start Graphite using Docker image or standalone installation:
4. Call REST services of the order-service project and make sure new metrics are displayed in the Graphite dashboard: http://localhost/dashboard

Failsafe library

- ✓ Lightweight zero-dependency library for handling failures
- ✓ Supports retries, circuit breakers, fallbacks, event listeners asynchronous execution, execution tracking



```
<dependency>
    <groupId>net.jodah</groupId>
    <artifactId>failsafe</artifactId>
    <version>1.1.0</version>
</dependency>
```

Failsafe library. Examples

```
RetryPolicy retryPolicy = new RetryPolicy()
        .retryOn(ConnectException.class)
        .withDelay(1, TimeUnit.SECONDS)
        .withMaxRetries(3);
Failsafe.with(retryPolicy).run(() ->
    System.out.println("Test"));
```

```
ScheduledExecutorService executor =
        Executors.newScheduledThreadPool(2);

Failsafe.with(retryPolicy)
  .with(executor).onSuccess(System.out::println)
  .onFailure(System.err::println).get(() -> "result");
```

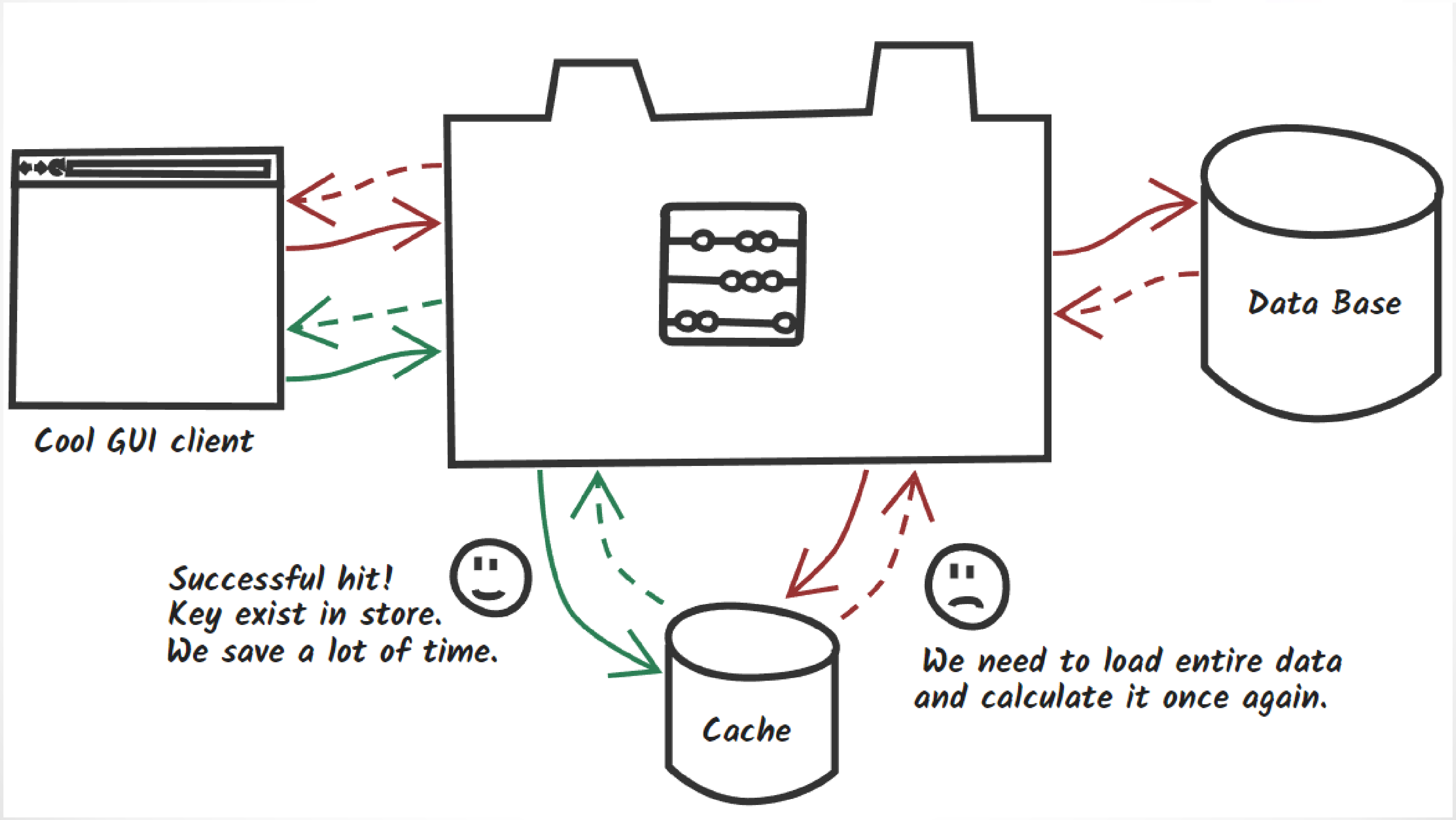
RetryPolicy class API

Annotation	Description
abortIf()	Abort execution if specified predicate matches the result
abortOn()	Abort execution if specified exception occurs
retryIf()	Retries if predicate matches the result
retryOn()	Retries attempt if specified exception occurs
withBackoff()	Sets backoff for the next retry
withDelay()	Sets delay between retries
withMaxDuration()	Specifies max duration for the overall retries
withMaxRetries()	Specifies maximum number of retries

Task 8. Health check and retry policy

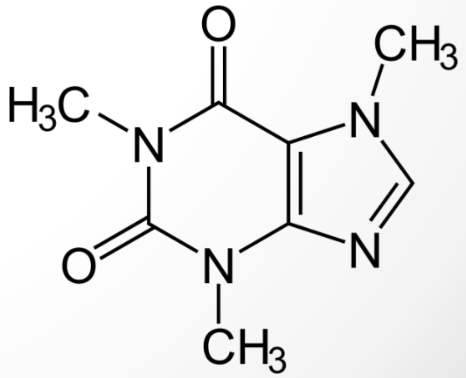
1. Create RetryPolicy interface that will define how load balancer act if request to the server fails. What should be default retry policy?
2. Create several implementations of RetryPolicy and provide configuration properties for them.
3. Implement heath checks by sending regular requests to the **/status** endpoint of configured servers. Add configuration properties so that you can customize endpoint value

Caching

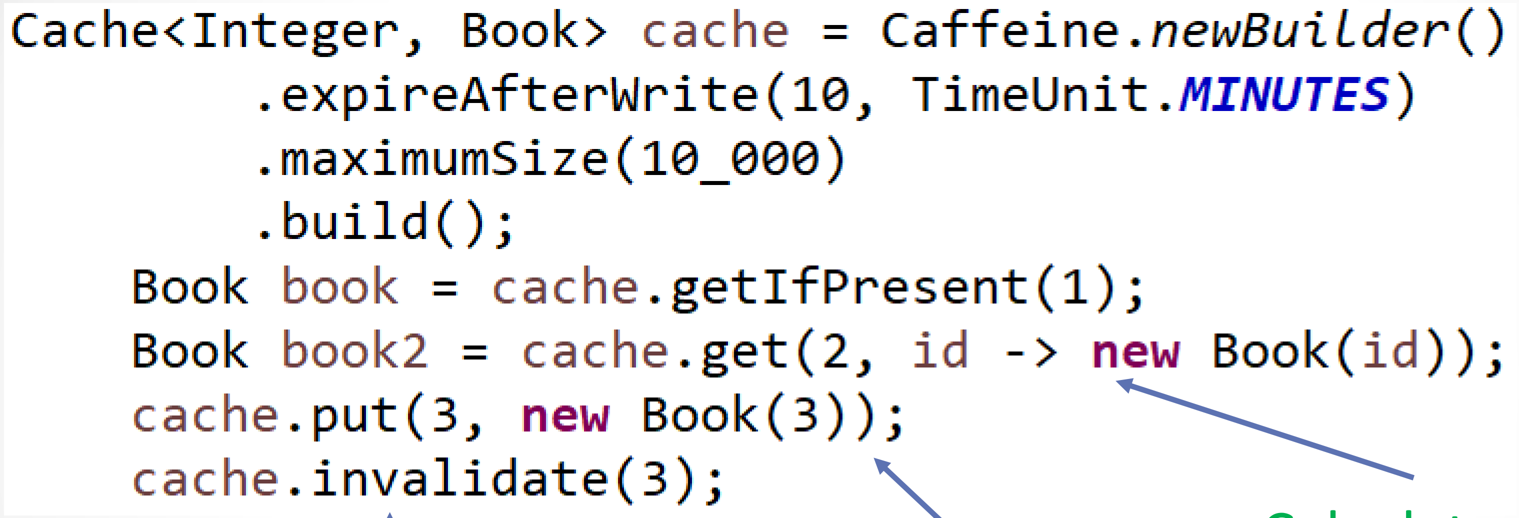

Cool GUI client
Data Base
Successful hit!
Key exist in store.
We save a lot of time.
Cache
We need to load entire data
and calculate it once again.

Caffeine

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



Caffeine. Examples



```java
Cache<Integer, Book> cache = Caffeine.newBuilder()
        .expireAfterWrite(10, TimeUnit.MINUTES)
        .maximumSize(10_000)
        .build();
    Book book = cache.getIfPresent(1);
    Book book2 = cache.get(2, id -> new Book(id));
    cache.put(3, new Book(3));
    cache.invalidate(3);
```

Calculate value if key is absent

Removes key from cache

Insert or update key

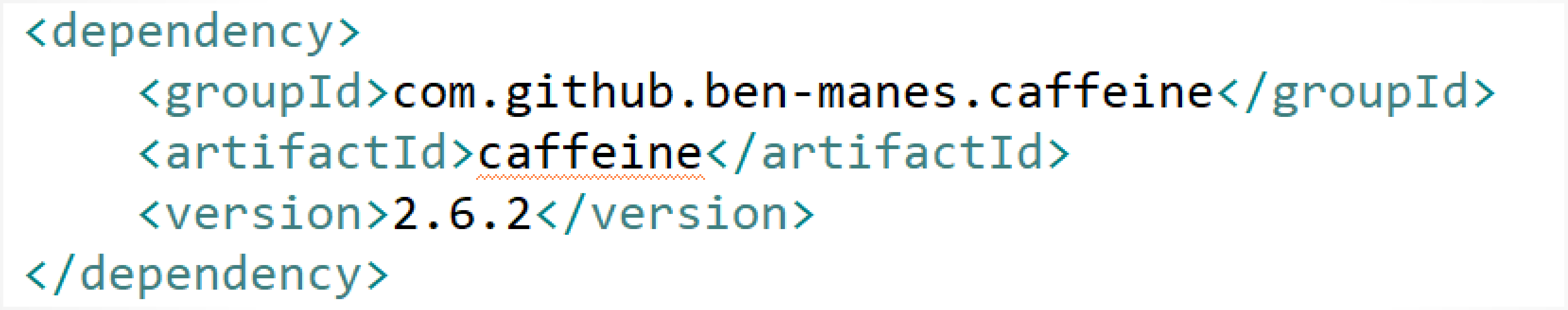
Caffeine API

Annotation	Description
initialCapacity()	Setups initial capacity of the cache
maximumSize()	Maximum amount of entries in the cache
expiresAfterWrite()	Specifies duration after the last insert(update) when entry should removed
expiresAfterAccess()	Specifies duration after the last read(write) when entry should removed from cache
removalListener()	Specifies listener after entry removal
recordStats()	Enables accumulation of statistics

Cache API

Annotation	Description
getIfPresent()	Returns key value of null
get()	Returns of compute cache value
put()	Inserts or updates key value
invalidate()	Removes cache entry
invalidateAll()	Discards all the entries in the cache
stats()	Returns current snapshot of the cache statistics
cleanUp()	Performs pending maintenance operations

Caffeine. Maven dependencies



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

Task 9. Caching

1. Add Caffeine dependency to the **order-service** sub-project:
2. Update load-balancer component to cache responses of the book-service. Assume that books objects are immutable and never change.
3. Add configuration properties that store cache properties: maximum and initial size, expiration.
4. Run integration tests and verify that orderController works properly.

Server load balancer

Sergey Morenets, 2018

Task 10. Starting server balancer

1. Review server-balancer sub-project.
2. Import client-balancer library into server-balancer sub-project.
3. Add necessary endpoints for server-balancer project
4. Update order-service project so that it uses server-side load balancing.

Caching

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

Caching. Maven dependencies

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

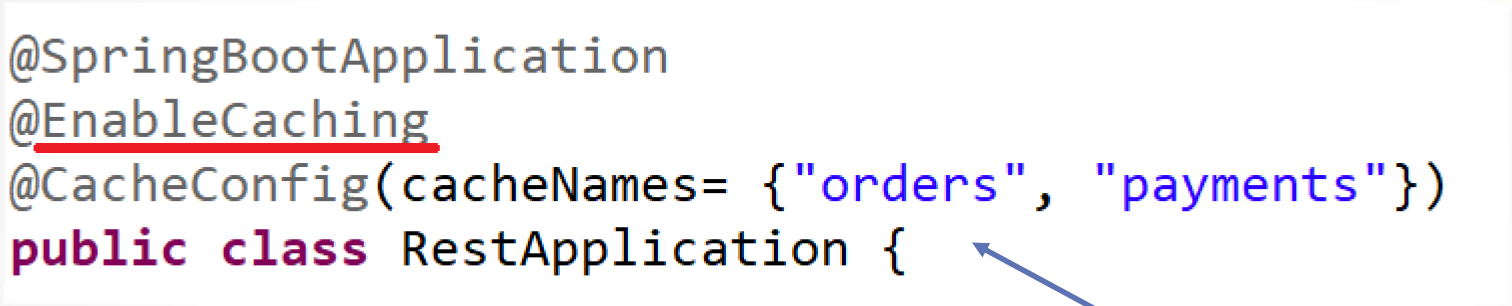
Generic provider

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

Spring annotations

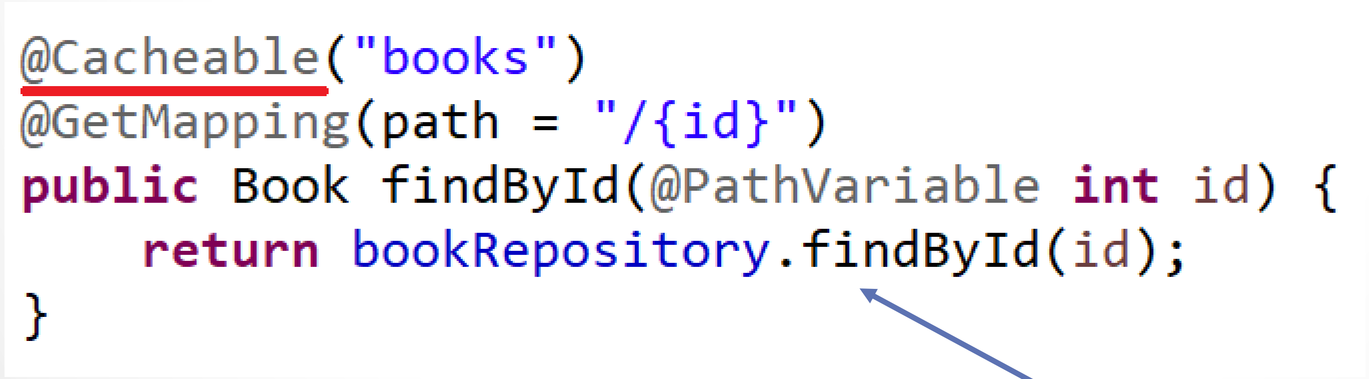
Name	Description
@EnableCaching	Enables annotation-driven cache management
@CacheConfig	Allows to config cache parameters
@Cacheable	Indicate that method(or all methods) result should be cached depending on the method arguments
@CachePut	Indicate that method result should be put into cache whereas method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

Enable caching



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

Optionally specify cache names

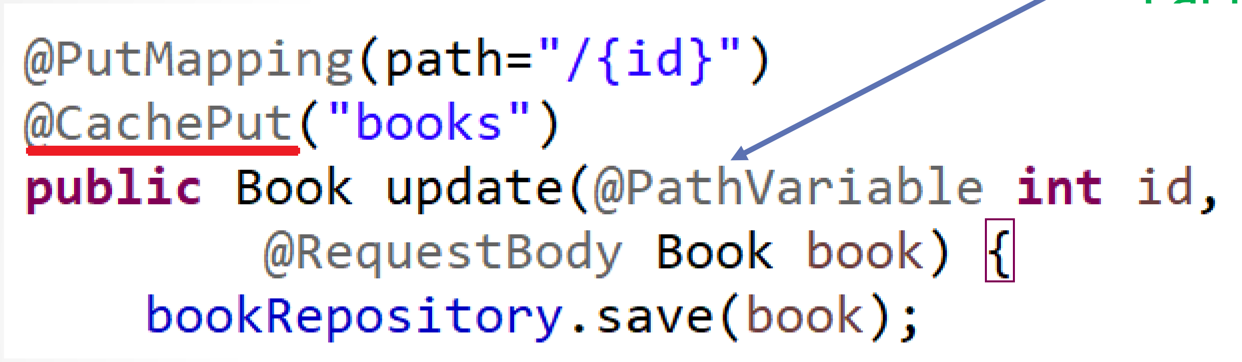


```
@Cacheable("books")
@GetMapping(path = "/{id}")
public Book findById(@PathVariable int id) {
    return bookRepository.findById(id);
}
```

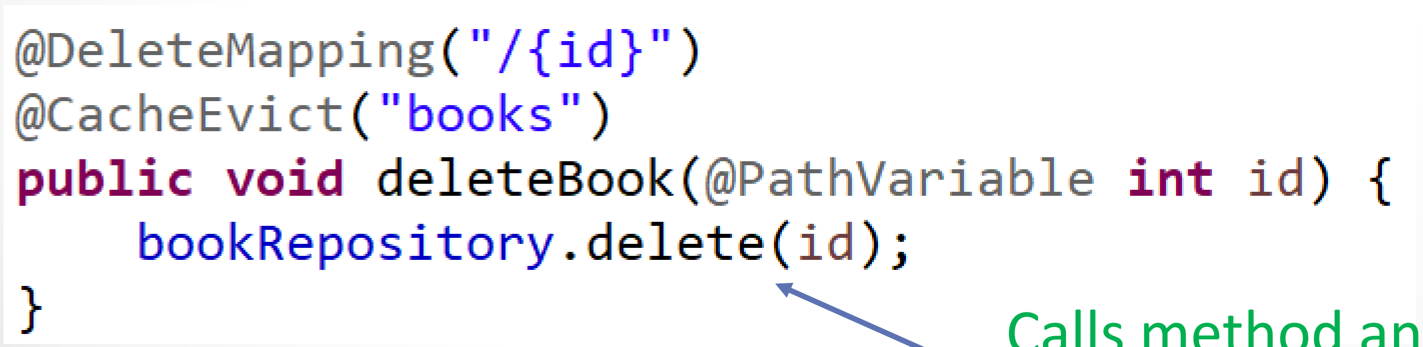
Cache returned value

Update cache

Calls method and cache returned value



```
@PutMapping(path="/{id}")
@CachePut("books")
public Book update(@PathVariable int id,
        @RequestBody Book book) {
    bookRepository.save(book);
```



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

Calls method and remove cache entry

Advanced caching

```
@GetMapping("/{id}")
@Cacheable(value = "books", key = "#id")
public Book findById(@PathVariable String id) {
    return bookService.findBook(id);
}
```

Specify cache key

```
@PutMapping("/{id}")
@CachePut(value = "books", key = "#id")
public Book update(@PathVariable int id,
        @RequestBody Book book) {
```

```
spring.cache.type=caffeine
```

Specify exact cache provider

Task 11. Spring Cache

1. Add **@EnableCaching** annotation to the **LoadBalancerApplication** class.
2. Add Cache dependency
3. Add **@Cacheable** annotation to the **GET** REST-services. How does it affect its behavior?
4. Add **@CachePut** annotation on POST/PUT REST services and **@CacheEvict** annotation on DELETE REST services. Verify its behavior.

Authentication

Из диалога двух программистов:

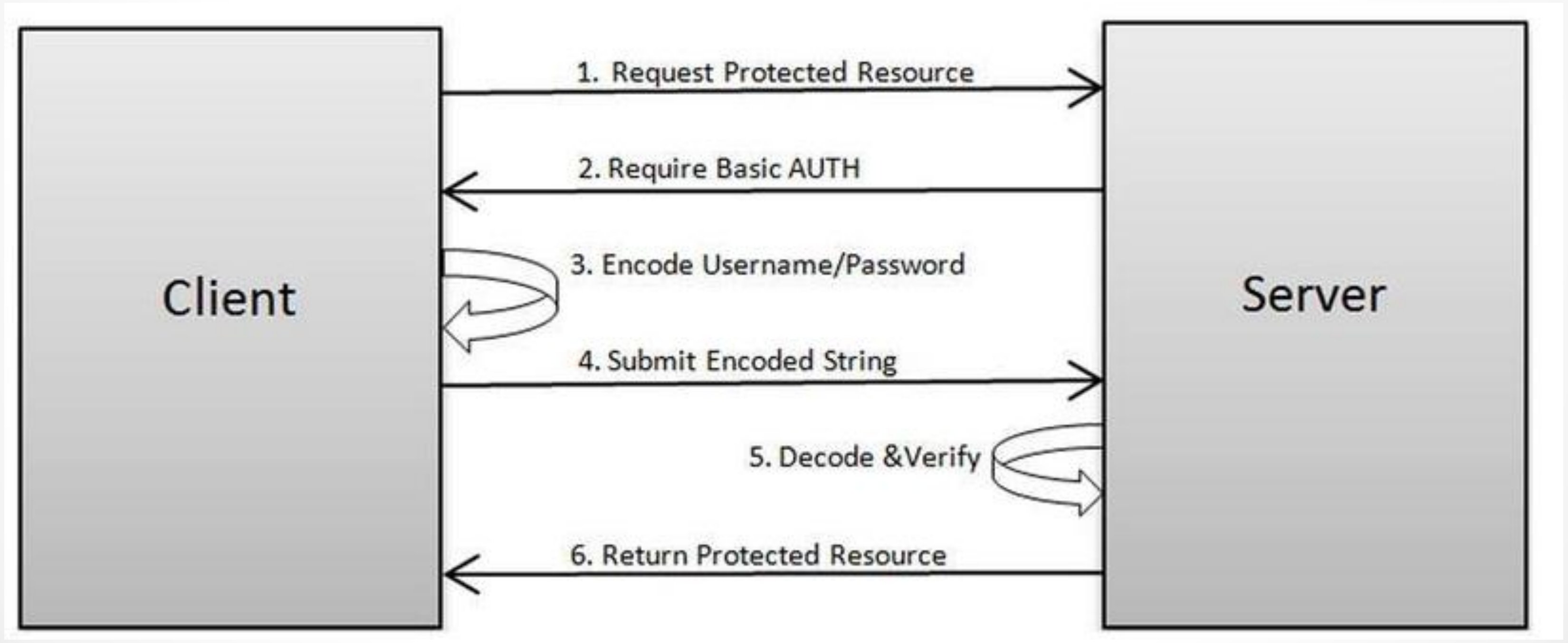
— Кажется, у нас дыра в безопасности!

— Слава Богу, хоть что-то у нас в безопасности…

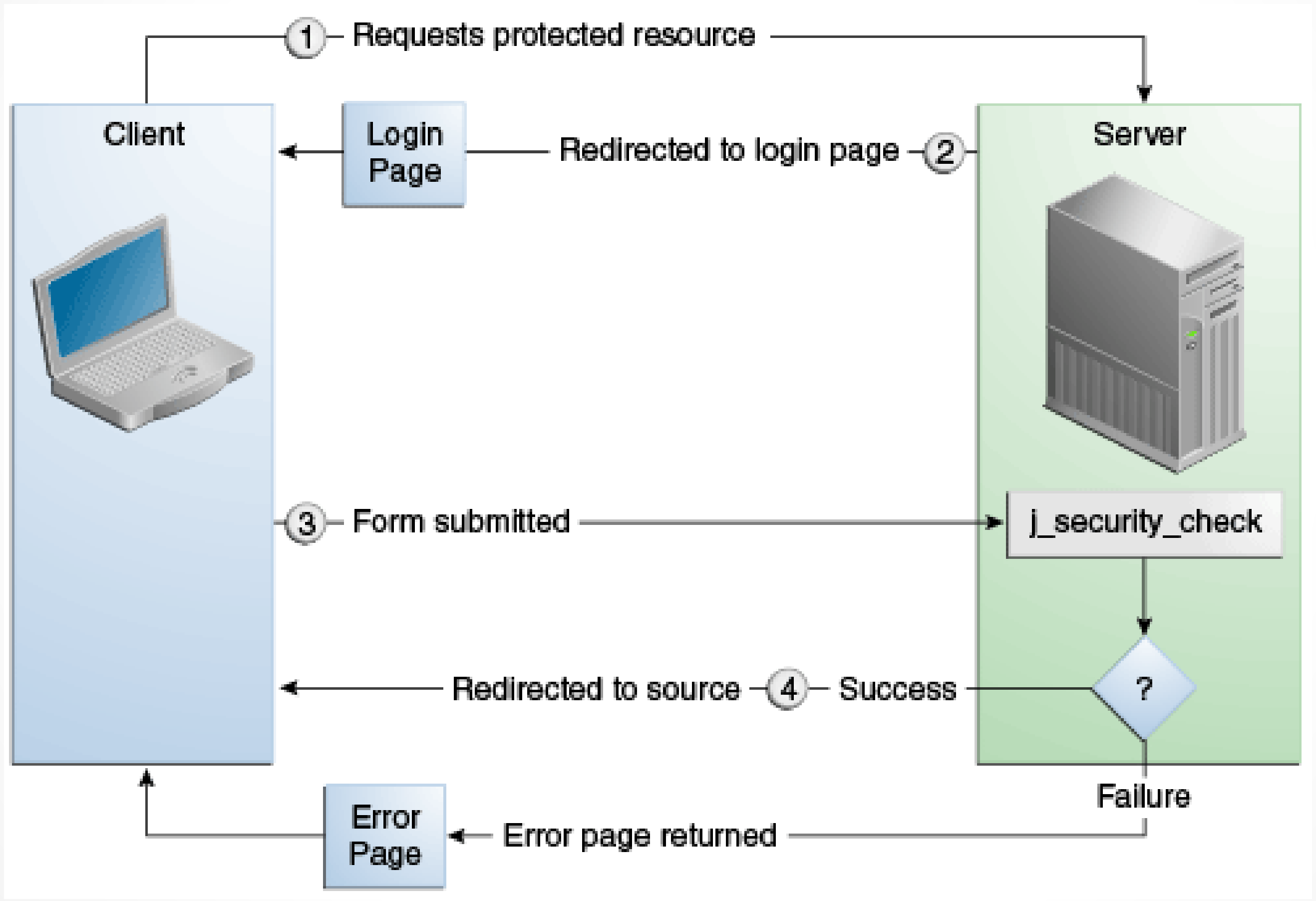
Authentication

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

Basic authentication


Client
Server
1. Request Protected Resource
2. Require Basic AUTH
3. Encode Username/Password
4. Submit Encoded String
5. Decode &Verify
6. Return Protected Resource

Form-based authentication



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

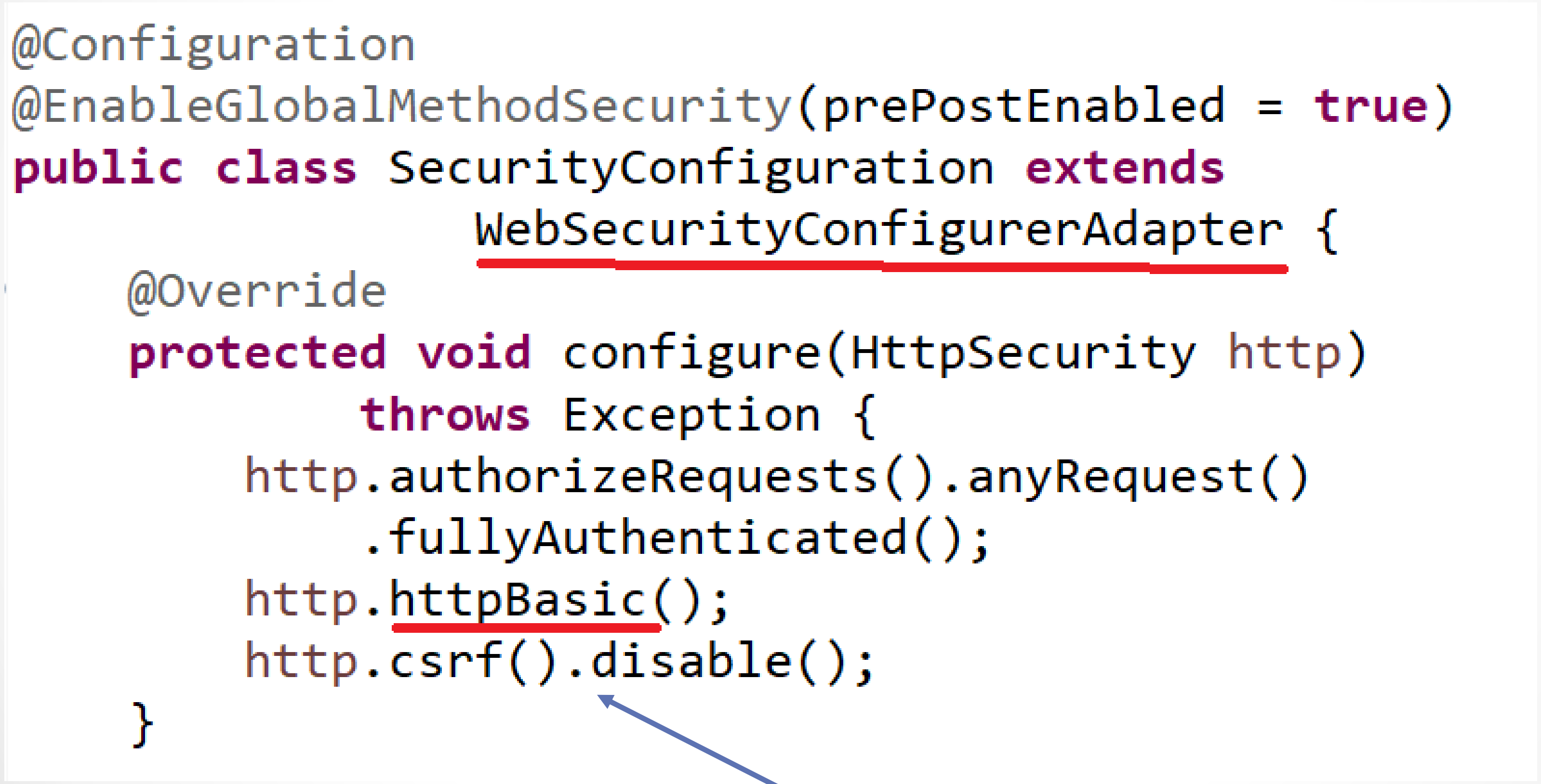
Spring Security. Maven

```
<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

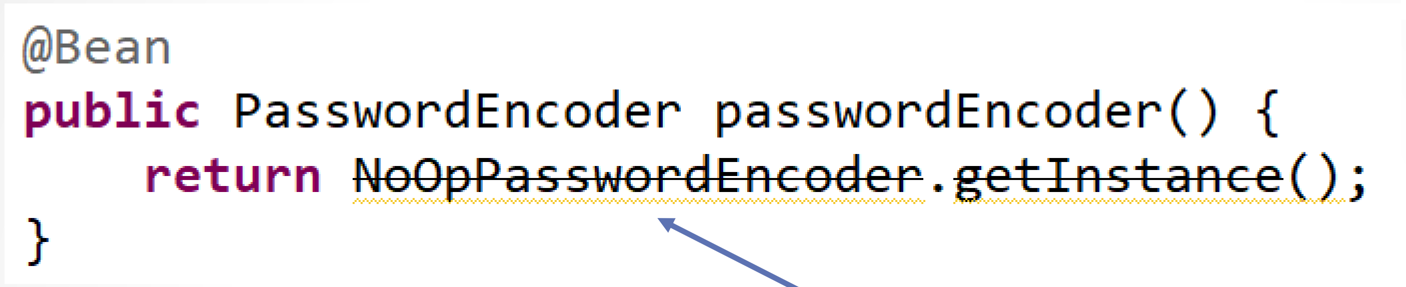
Spring Security



```
@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();
    }
```

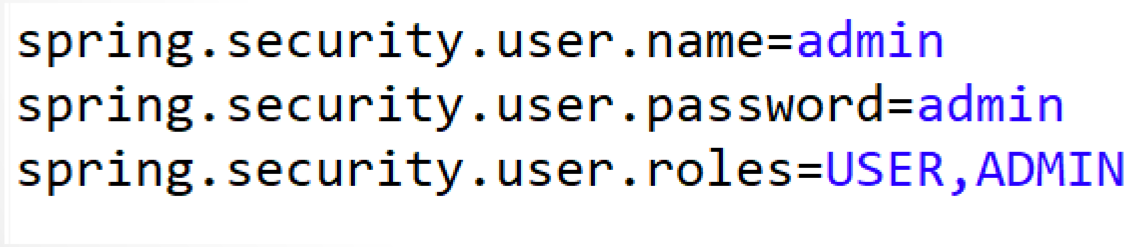
Enable POST/PUT requests

Spring Security. Users



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

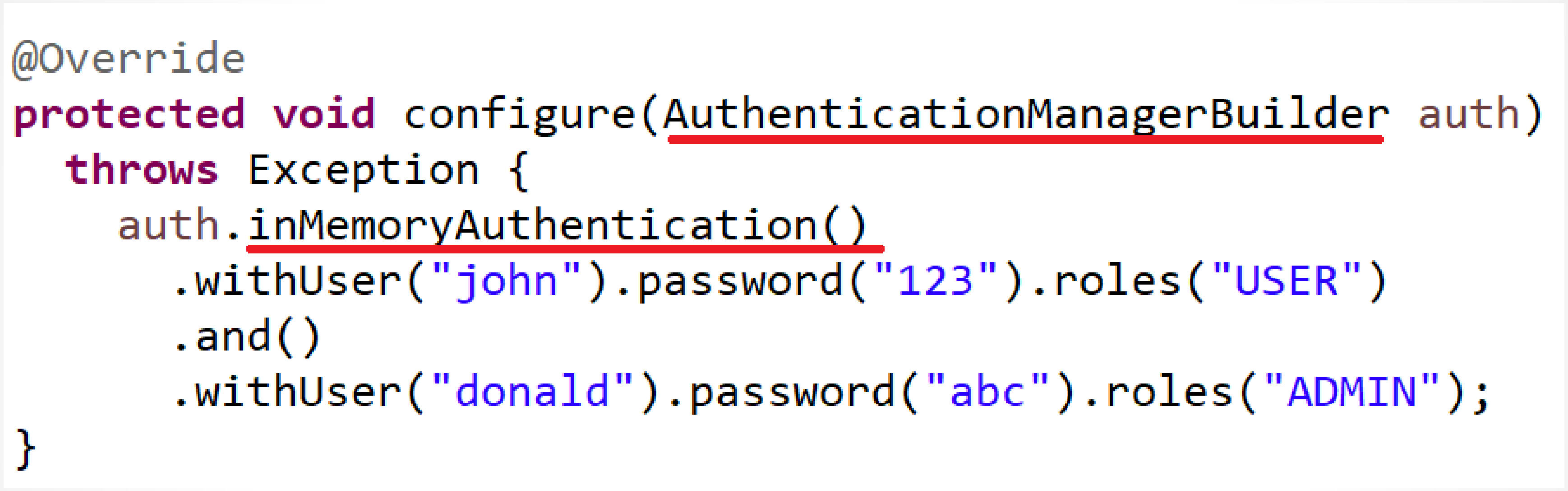
Can be BCrypt, SCrypt, MD4



```
spring.security.user.name=admin
spring.security.user.password=admin
spring.security.user.roles=USER,ADMIN
```

application.properties

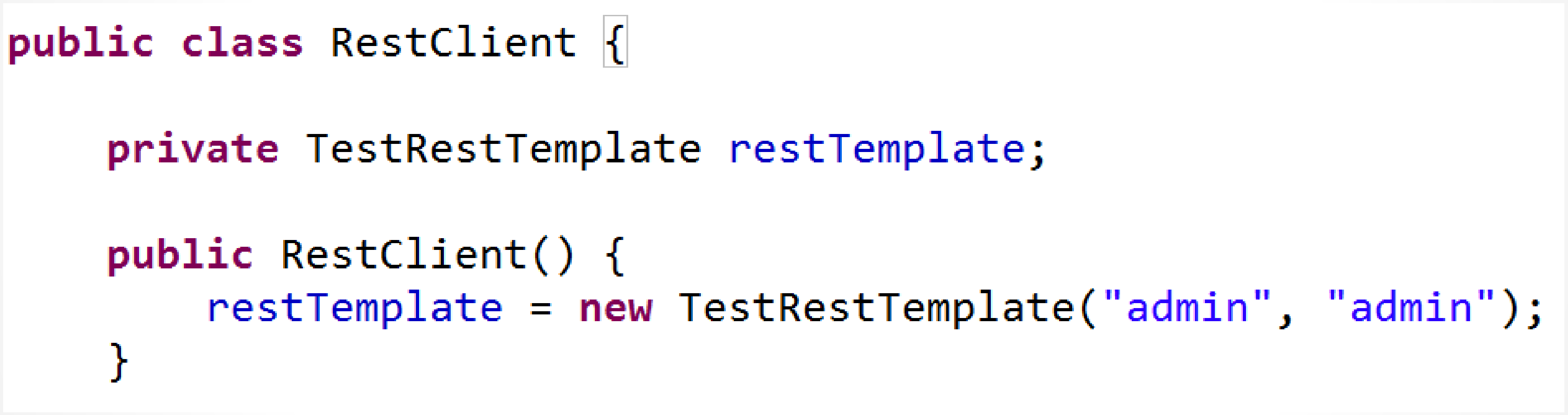
Spring Security. Users



```
@Override
protected void configure(AuthenticationManagerBuilder auth)
  throws Exception {
    auth.inMemoryAuthentication()
      .withUser("john").password("123").roles("USER")
      .and()
      .withUser("donald").password("abc").roles("ADMIN");
}
```

class SecurityConfiguration

Rest Client. Supply credentials



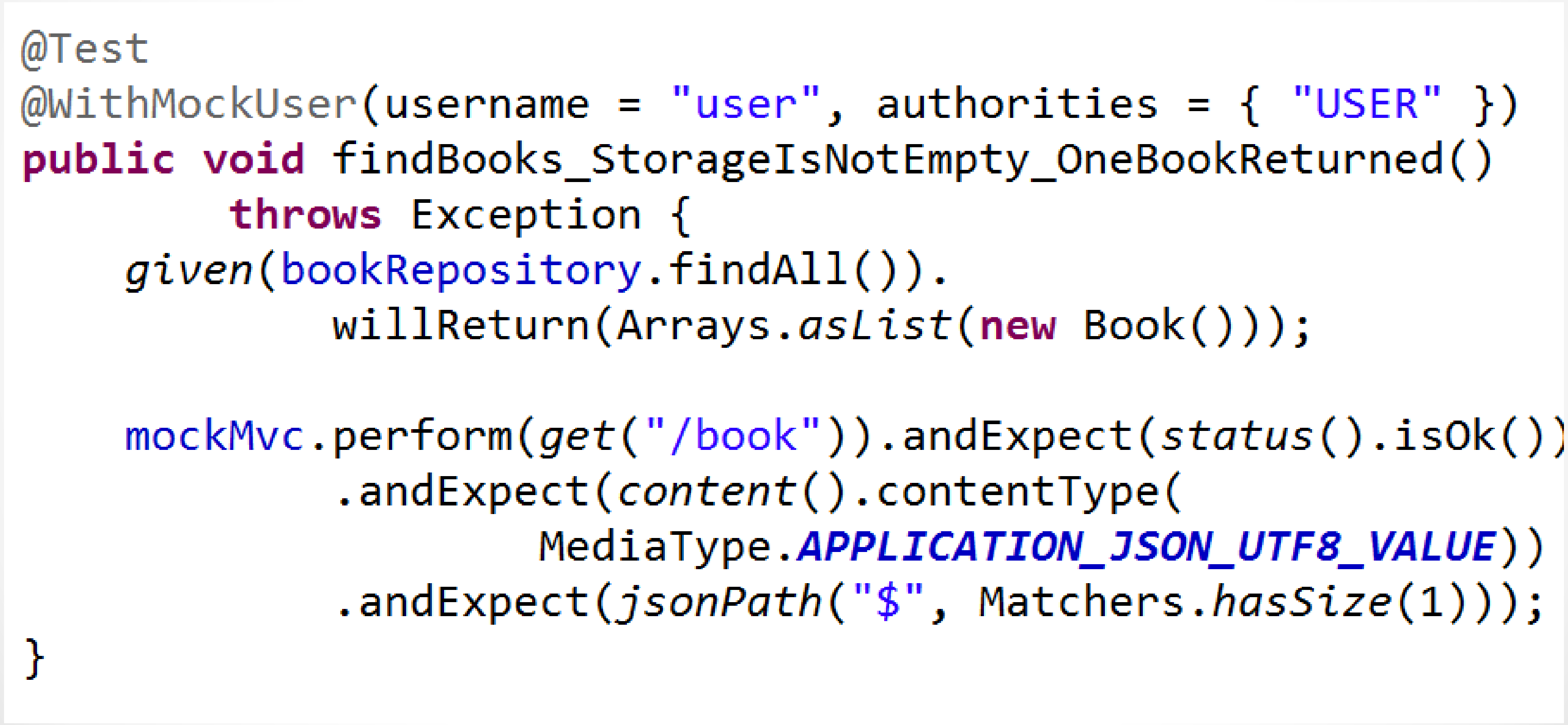
```
public class RestClient {

    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 12. Spring Security

1. Add Spring Security dependency to server-balancer project
2. Add Spring Security Test dependencies
3. Add **Spring Security** configuration:

   3.1 Add user(s) with different roles

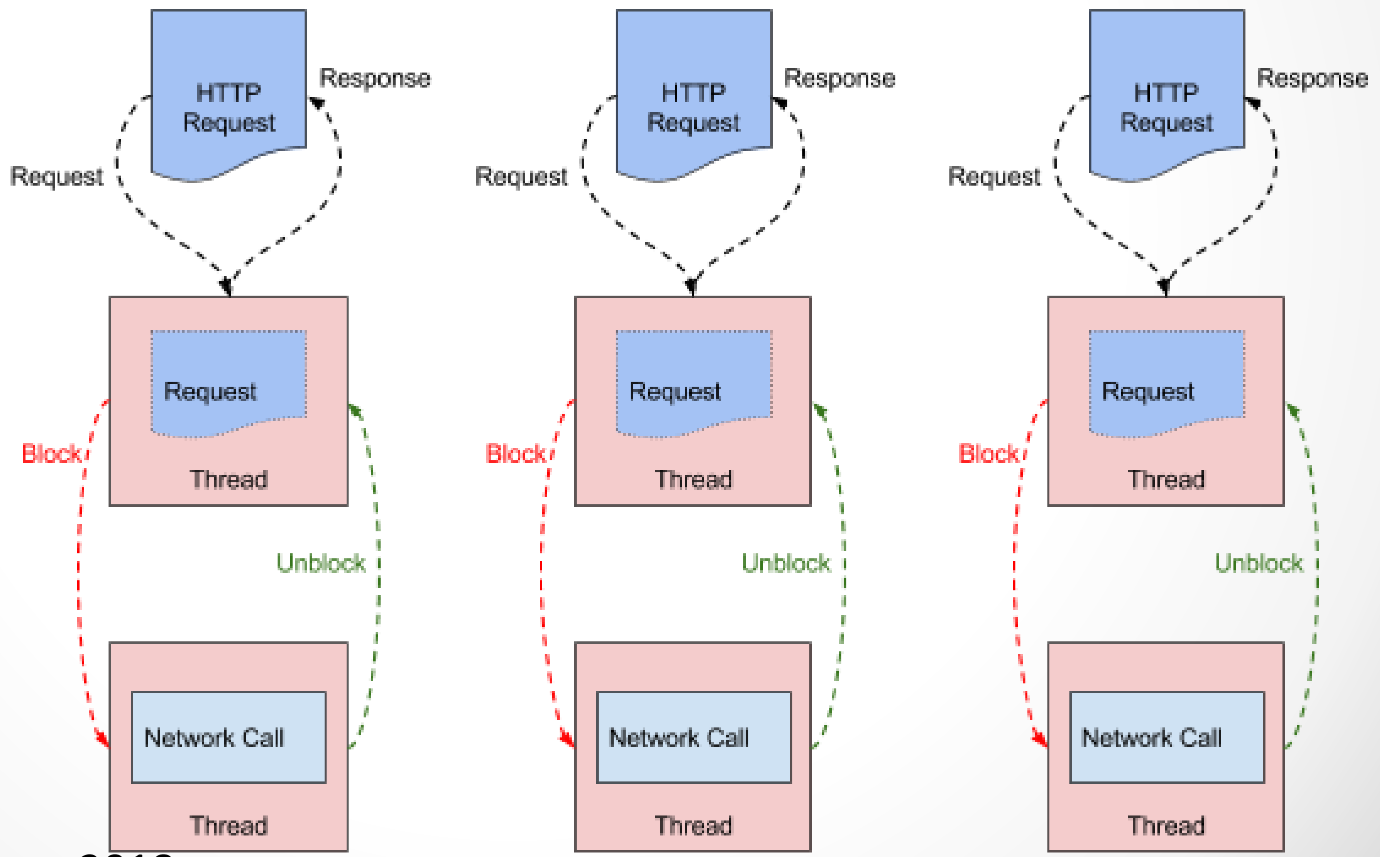
   3.2 Add HTTP **basic authentication**

4. Update order-service project and load-balancer component so that it sends basic authentication credentials.

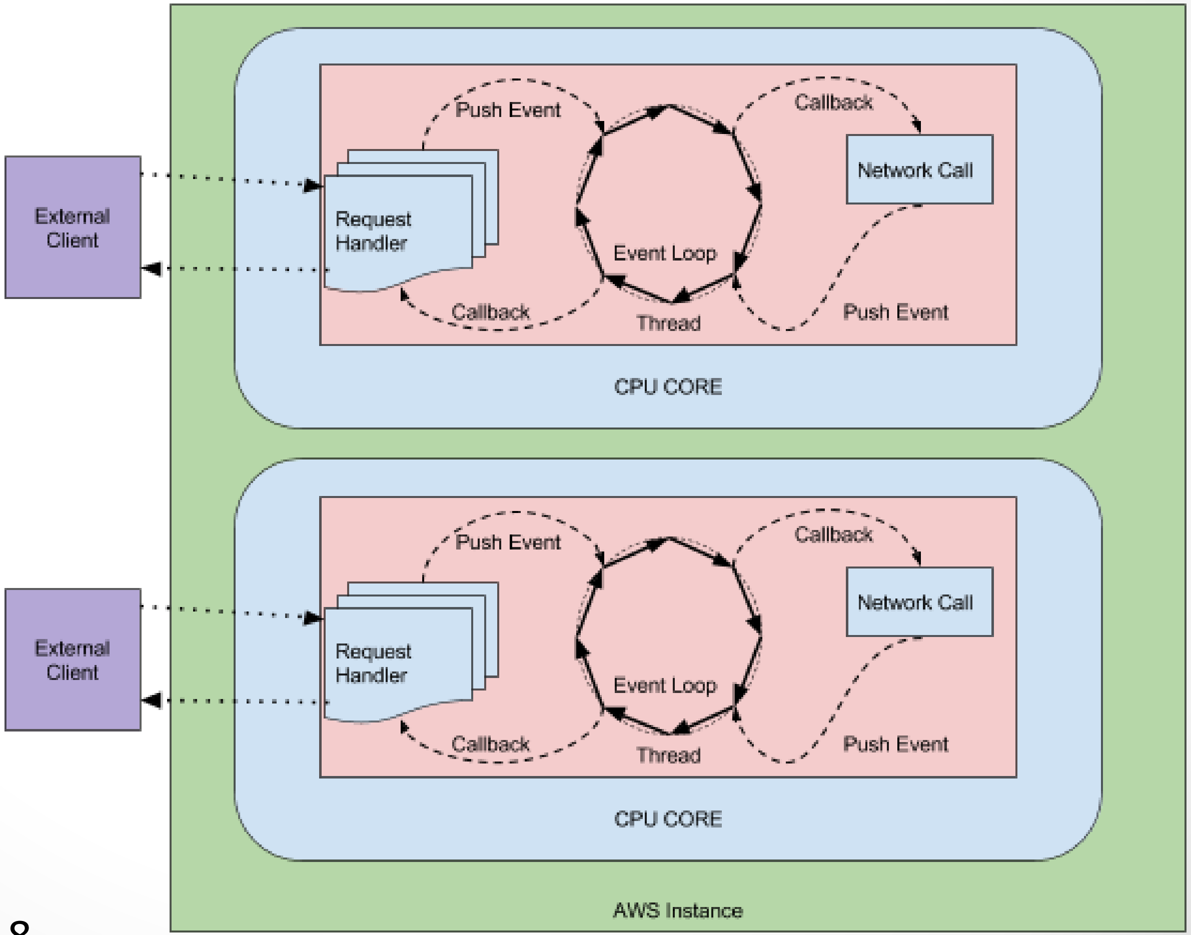
Think Reactive

Big Data... Big Returns...

Blocking I/O


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

Non-blocking I/O

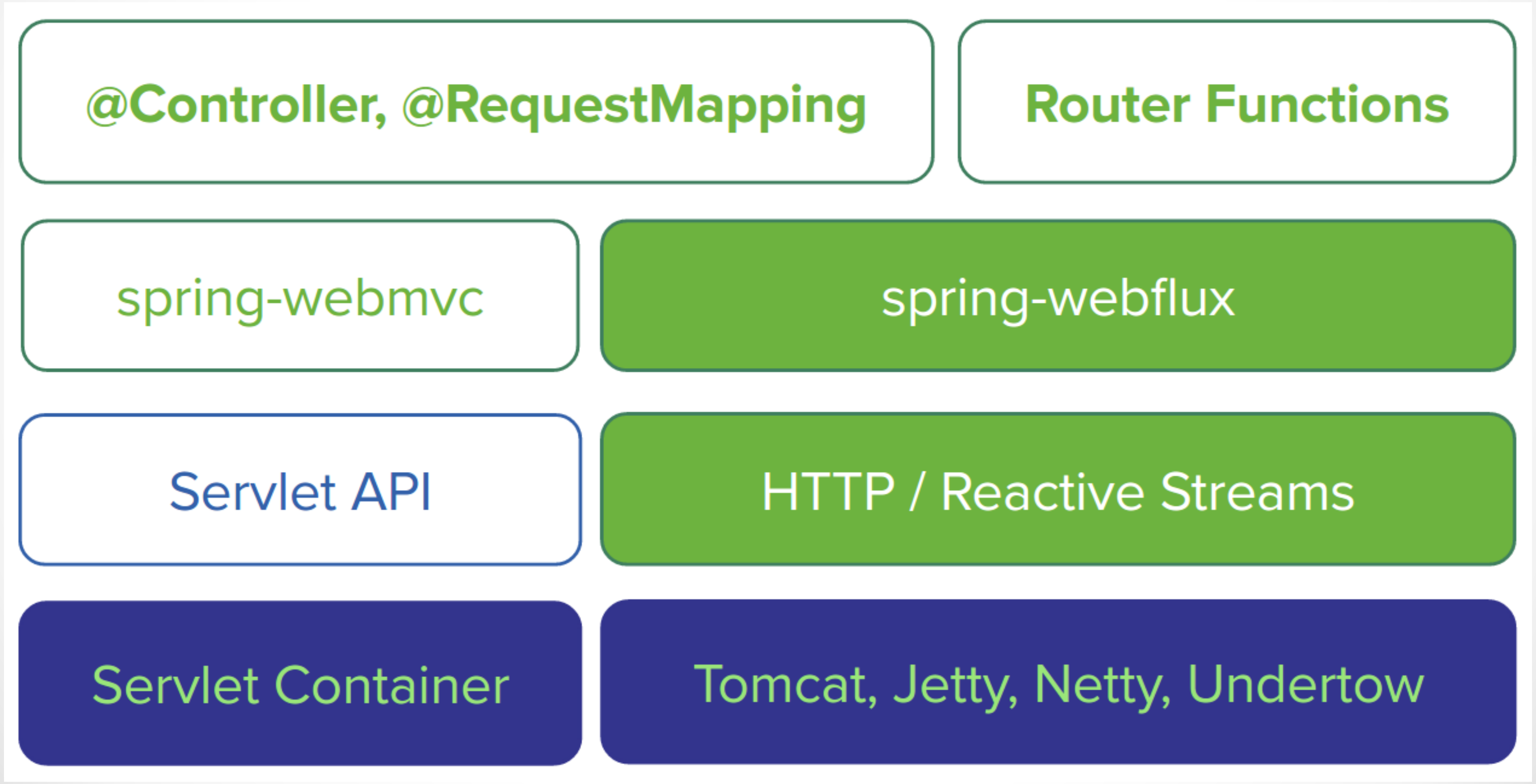


- Sergey Morenets, 2018

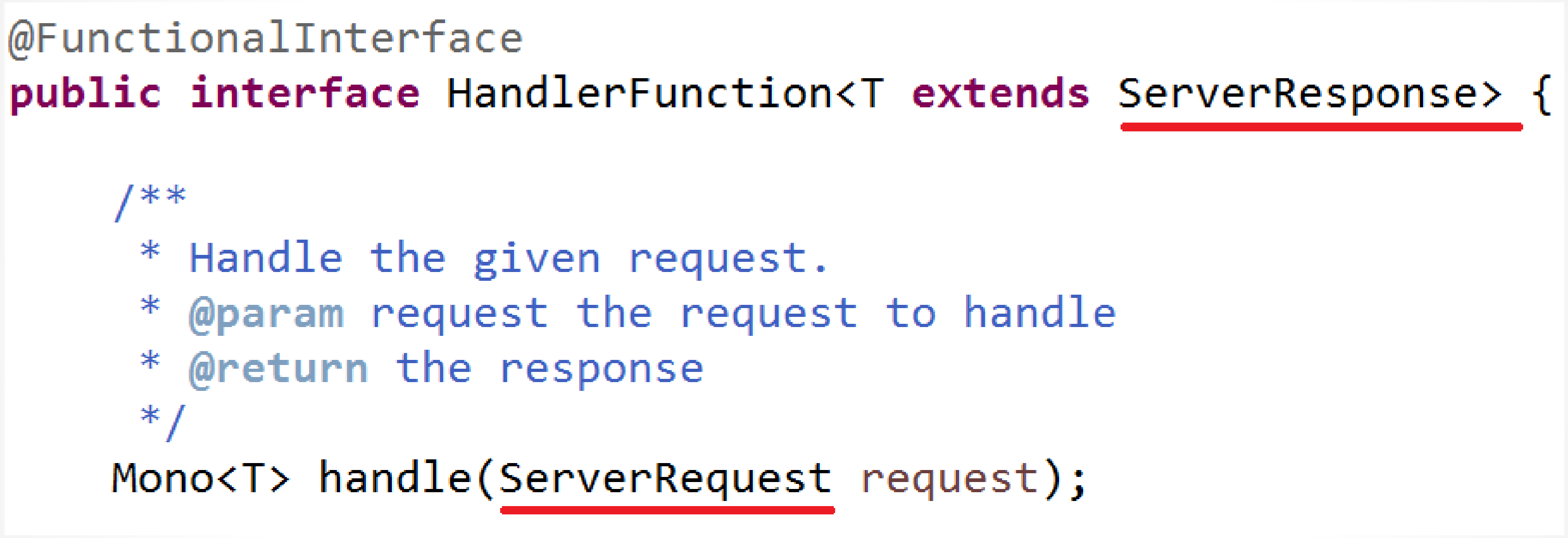
Spring 5 WebFlux

- ✓ 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**

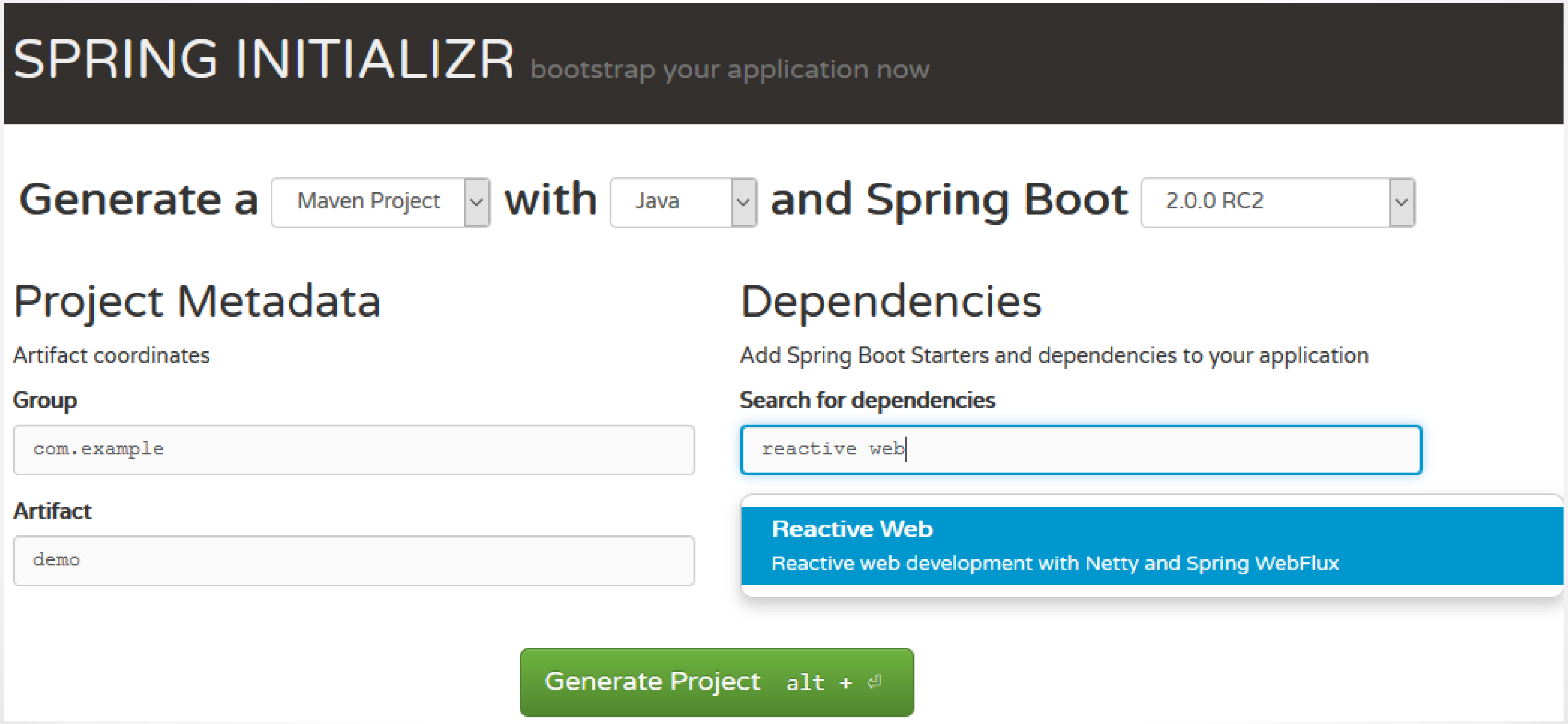
Spring Web Flux



Spring Web Flux



```java
@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);
```


SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Java and Spring Boot 2.0.0 RC2

Project Metadata

Artifact coordinates

Group

com.example

Artifact

demo

Dependencies

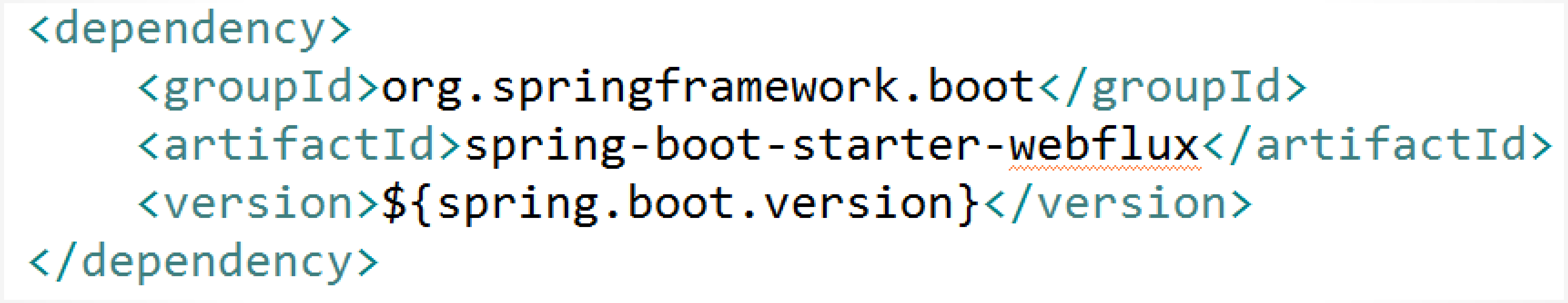
Add Spring Boot Starters and dependencies to your application

Search for dependencies

reactive web

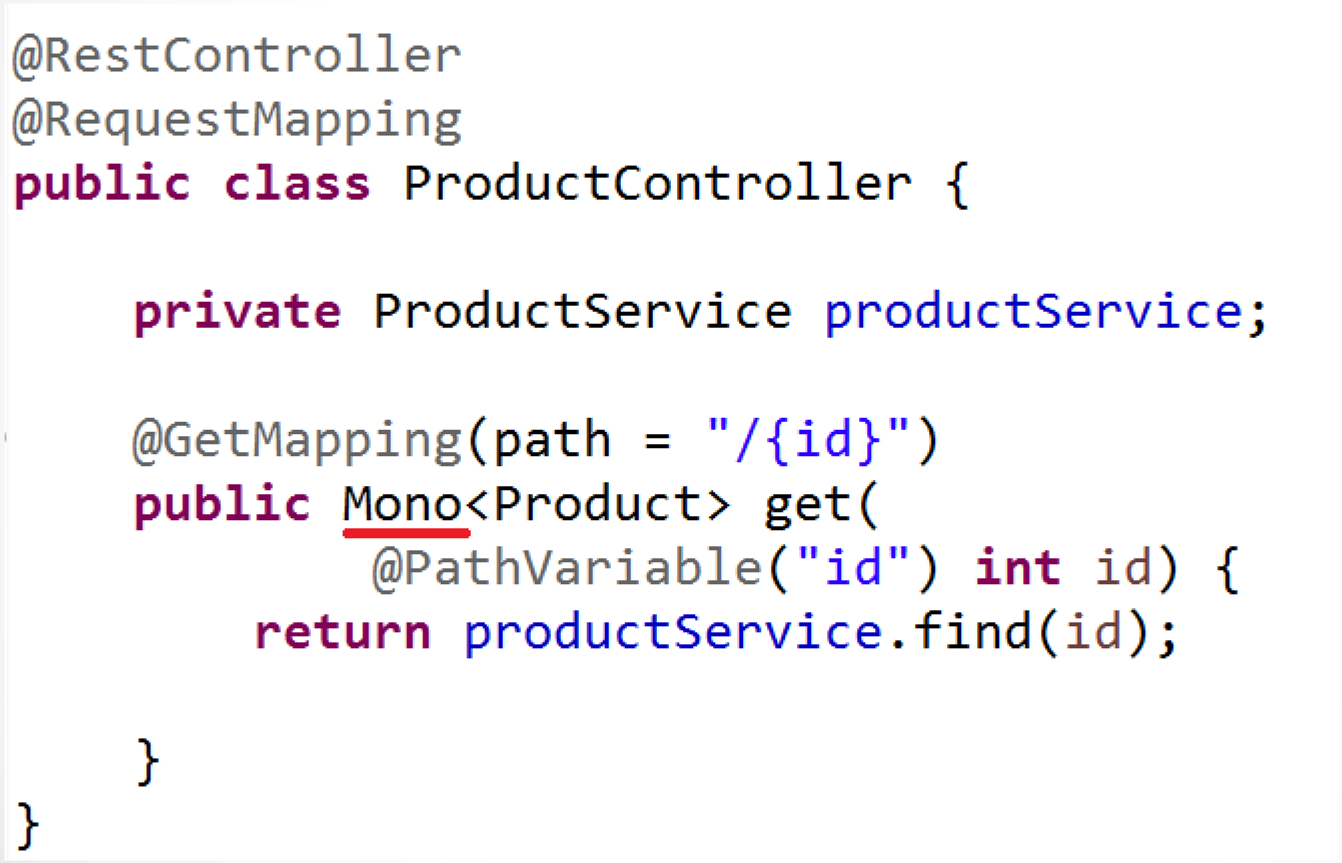
Reactive Web
Reactive web development with Netty and Spring WebFlux

Spring Web Flux. Maven



```xml
<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);

    }
}
```

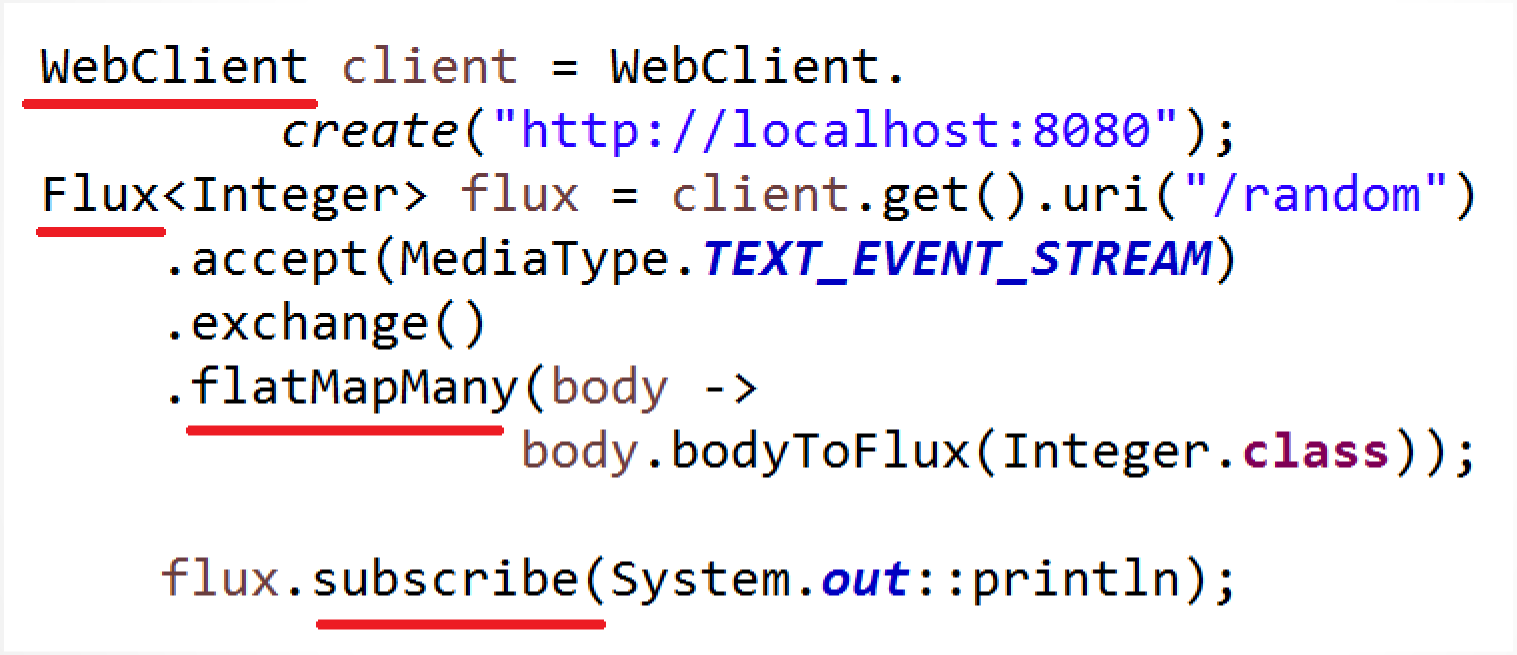
REST controller. Flux

```
@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

```java
@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

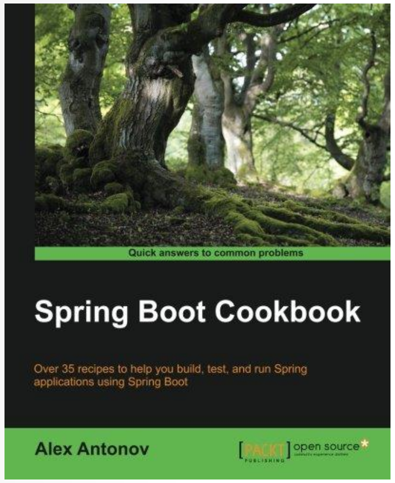


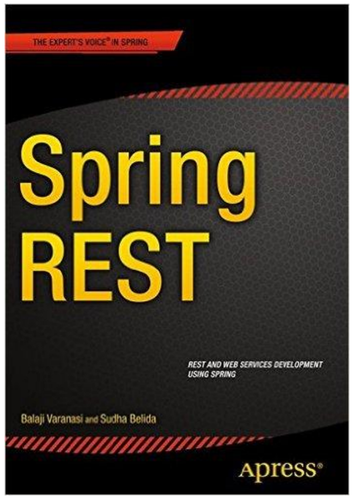
```
WebClient client = WebClient.
            create("http://localhost:8080");
Flux<Integer> flux = client.get().uri("/random")
      .accept(MediaType.TEXT_EVENT_STREAM)
      .exchange()
      .flatMapMany(body ->
                  body.bodyToFlux(Integer.class));

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

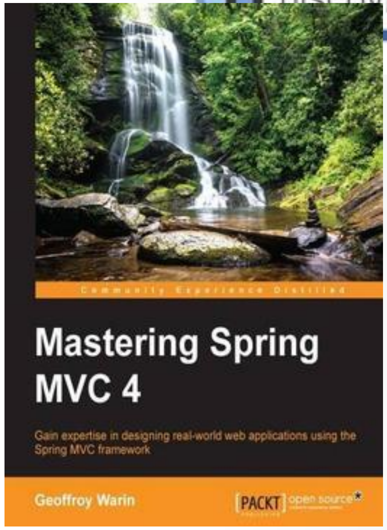
Task 13. Spring Web Flux

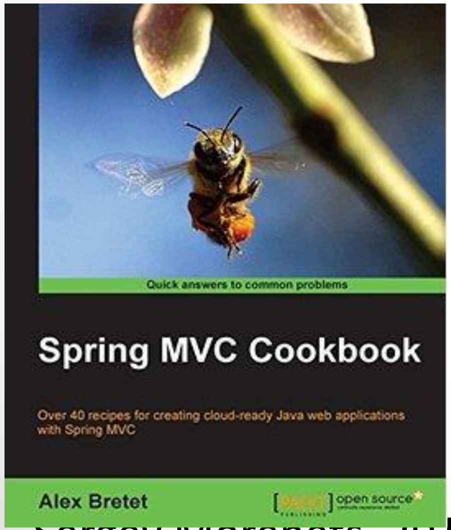
1. Add Sping WebFlux dependency to pom.xml **in order-service/server-balancer** projects**:**
2. Update REST services in book-service or server-balancer applications so that uses **Spring Web Flux** API
3. Update order-service application and balancer component and switch from RestTemplate to **WebClient** as REST client.

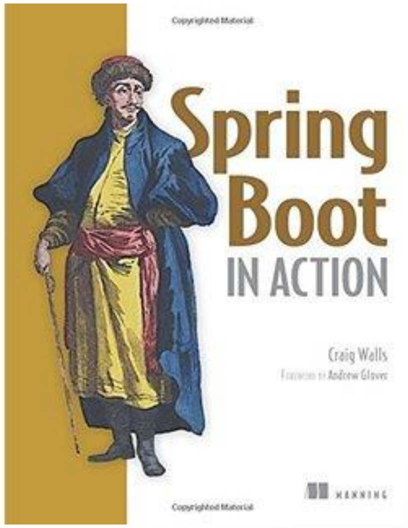
Books


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


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


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


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


Copyrighted Material
Spring Boot IN ACTION
Craig Walls
Foreword by Andrew Glover
MANNING
Copyrighted Material

Spring Security
Eugen Paraschiv
[PACKT] VIDEO

✓ Sergey Morenets, sergey.morenets@gmail.com