Advanced REST services with Spring

May, 31st 2020

Sergiy Morenets, 2020

DEVELOPER 16 YEARS

TRAINER 7 YEARS

WRITER 4 BOOKS

FOUNDER

ITSimulator

SPEAKER

JAVA DAY
MINSK 2013

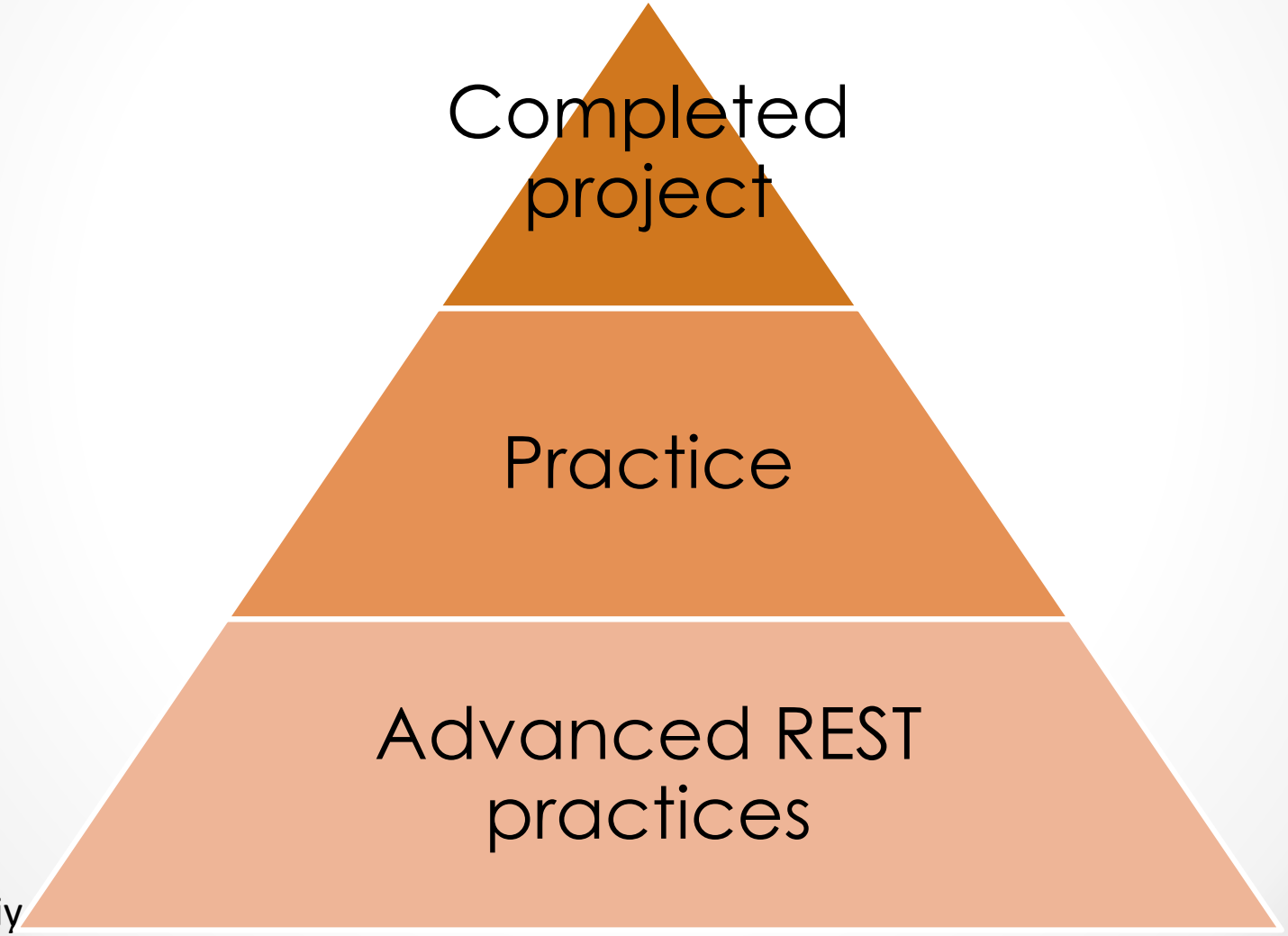
Dev(Talks):

JAVA DAY 2015

Java
frameworks
day

Sergiy Yorenets, 20

Goals



REST-services development with Spring

- ✓ REST and REST services
- ✓ REST over HTTP
- ✓ Spring Framework 5.2/Spring Boot 2.3 usage
- ✓ REST controllers
- ✓ Request input/output customization
- ✓ Validation and pagination
- ✓ REST API testing
- ✓ Exception handling
- ✓ DTO usage
- ✓ HATEOAS/Hypermedia
- ✓ Monitoring

Sergiy Morenets, 2020

Agenda

- ✓ Performance testing
- ✓ Caching
- ✓ Documentation(Spring REST Docs)
- ✓ Specification (Swagger, Springdoc)
- ✓ Scaling
- ✓ Rate limiting (throttling)
- ✓ Versioning
- ✓ Spring Data REST
- ✓ Etags

Task 1. Introduction

1. You should install Maven 3.6 (or later) or Gradle (6.3 or later) on your computer.
2. You should have STS 4.6.x or Intellij IDEA 2020.x installed (with Lombok plugin installed)
3. Import **advanced-rest-training** project into your IDE (you should import it as Maven or Gradle project) and open **rest-task1** sub-project.
4. Review project structure/configuration

Performance/load/stress testing

Sergiy Morenets, 2020

Performance/load testing

- ✓ Server with i7 4Ghz and 16 GB RAM
- ✓ Client with Celeron (4 core) and 8GB RAM

ApacheBench

ARTILLERY.IO

Drill

Gatling

HEY

APACHE JMeter™

k6

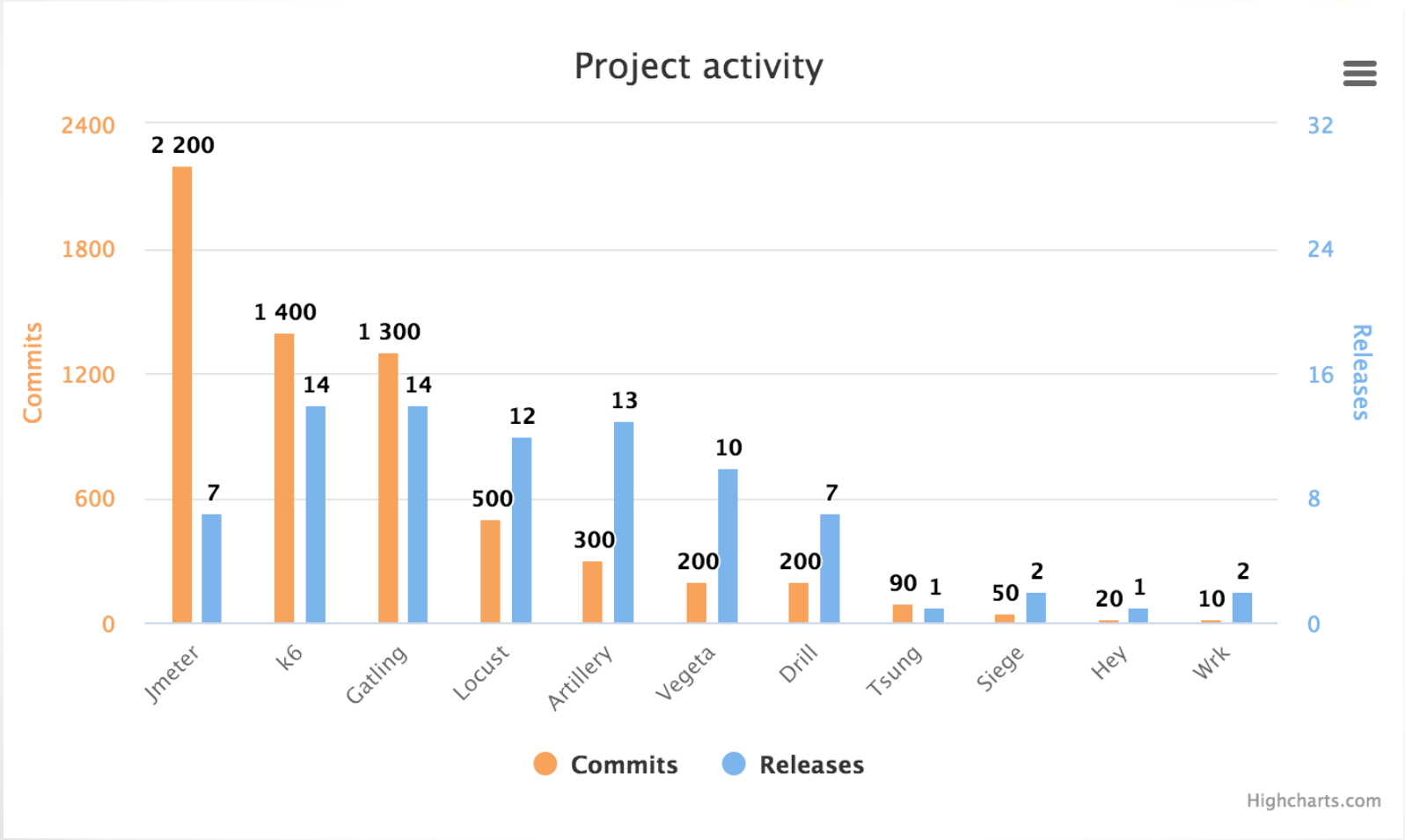
LOCUST

Siege

A distributed Load Testing Tool TSUNG

Wrk

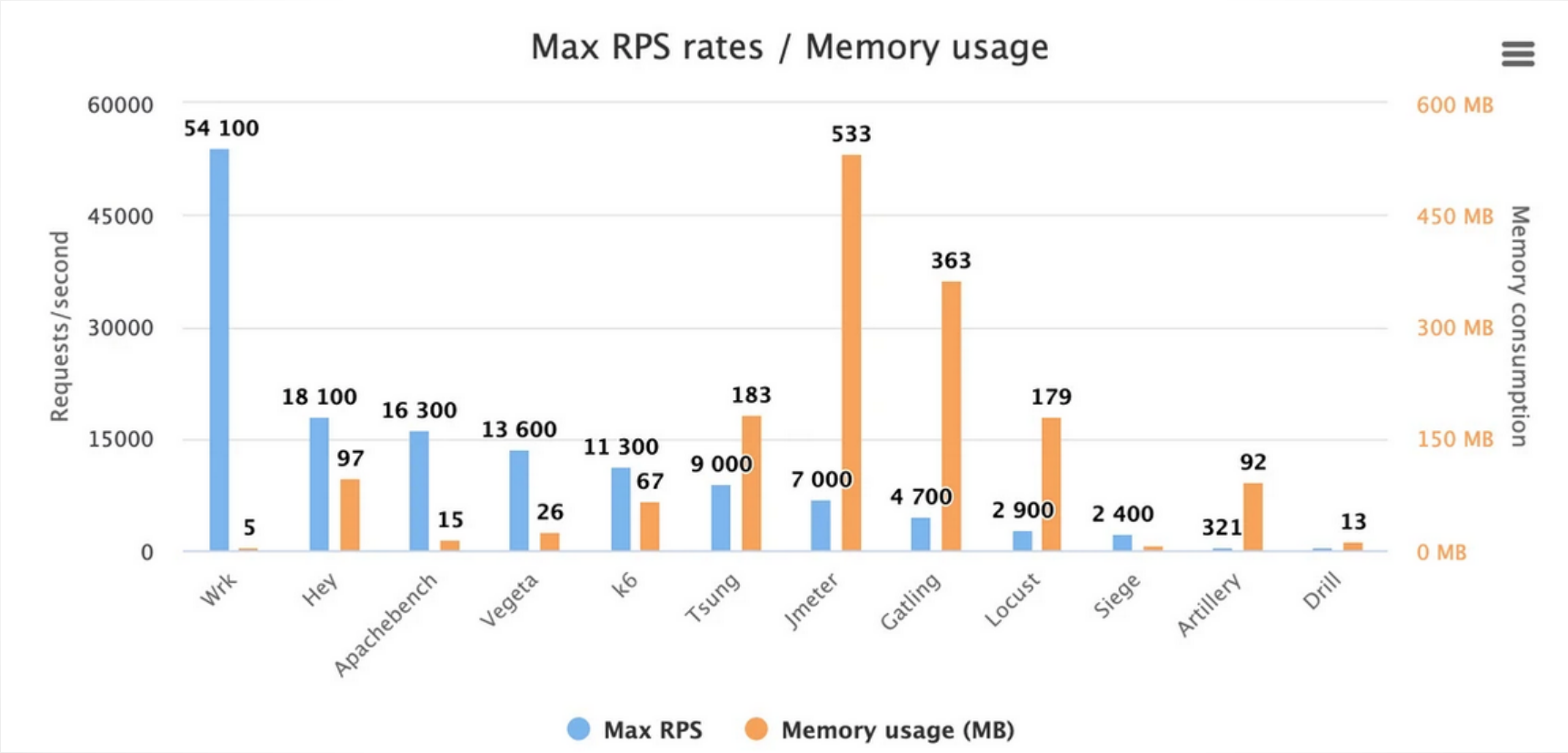
Performance testing. Tools



Sergiy Morenets, 2020

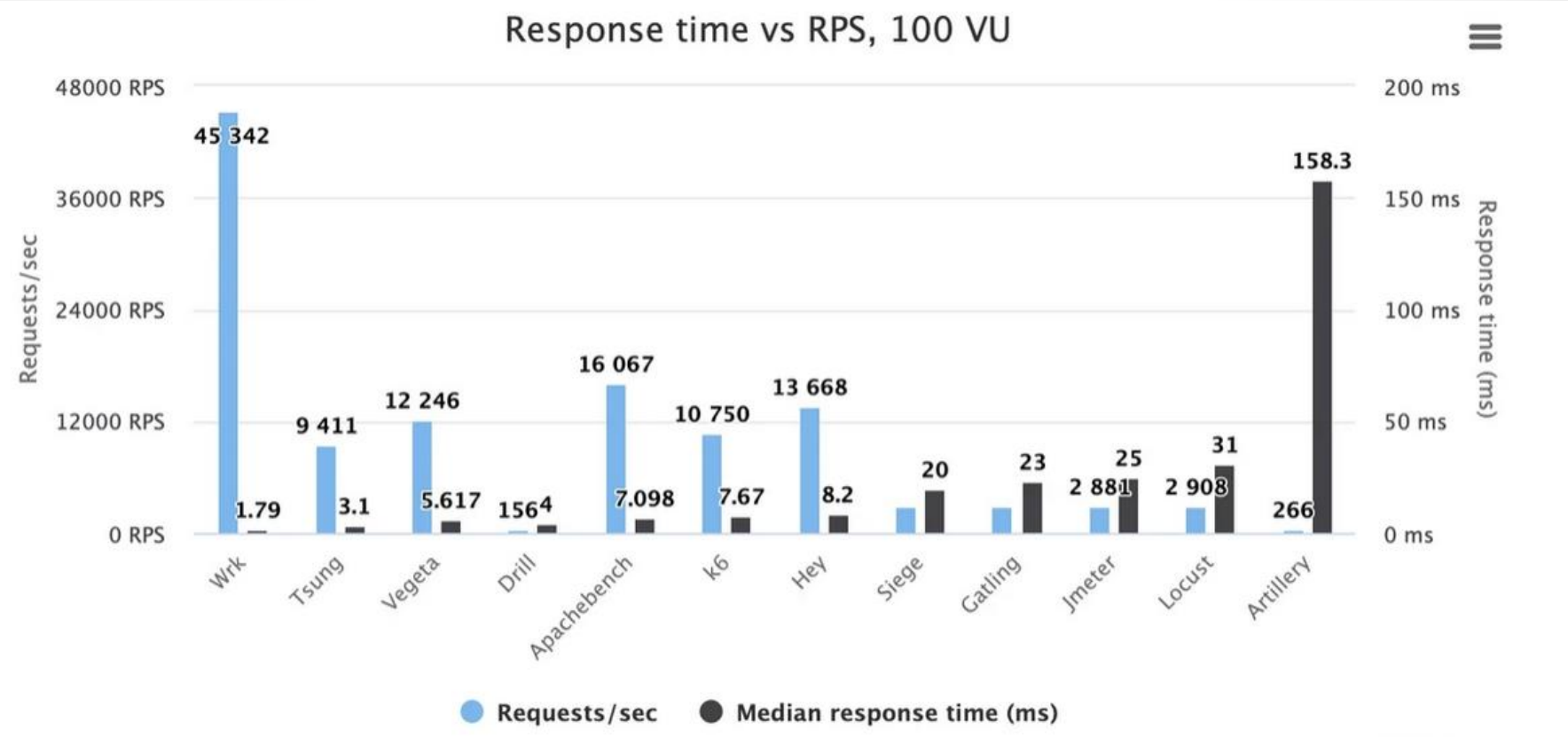
Number of commits since 2017

Performance testing. Tools



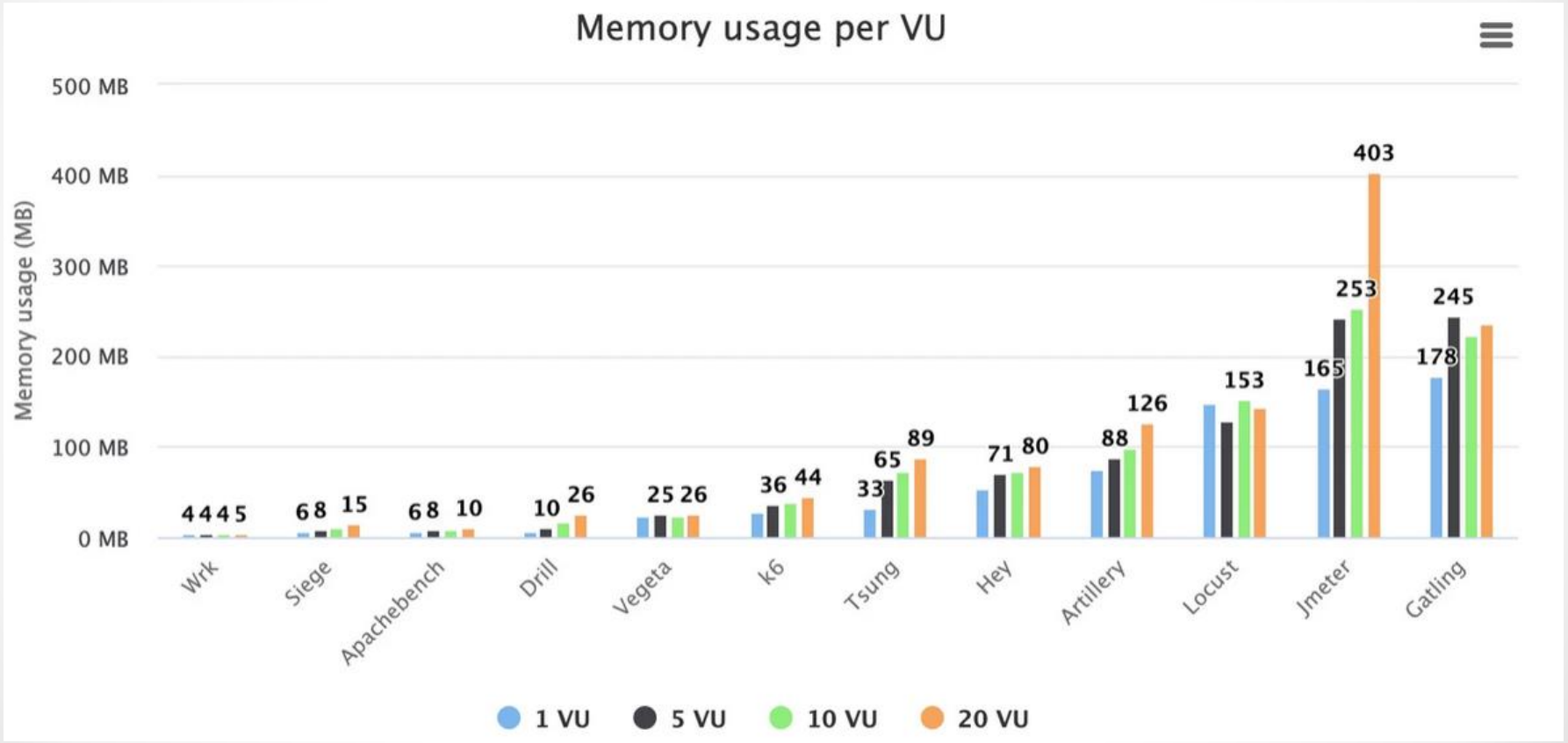
Max requests/seconds vs memory usage

Performance testing. Tools



Max requests/seconds vs media response time

Performance testing. Tools



Sergiy Morenets, 2020

Virtual users vs memory usage

Gatling

- ✓ Created in 2012 for test automation
- ✓ Written in Scala
- ✓ Generates plain text/HTML reports
- ✓ Offers recording tool

```
class GatlingSimulation extends Simulation {

  val vus = Integer.getInteger("vus", 20)
  val duration = Integer.getInteger("duration", 10)

  val scn = scenario("Scenario Name") // A scenario is a chain of re
    .during (duration) {
      exec(http("request_1").get("http://192.168.0.121:8080/"))
        }
        setUp(scn.inject(atOnceUsers(vus)))
```

K6

- Created in 2017 by Load Impact testing service
- Written in Go
- Supports HTTP/2 and Websocket
- Can report to JSON files/Kafka/InfluxDb/Datadog/StatsD
- Can import data from Postman collections

```
import http from 'k6/http';
import {check, sleep} from 'k6';

export default function() {
  const data = {username: 'username', password: 'password'};
  let res = http.post('https://myapi.com/login/', data);
    check(res, { 'success login': (r) => r.status === 200 });
    sleep(0.3);
}
```

K6. Test summary


```
data_received..............: 148 MB 2.5 MB/s
data_sent..................: 1.0 MB 17 kB/s
http_req_blocked...........: avg=1.92ms   min=1µs      med=5µs      max=288.7
http_req_connecting........: avg=1.01ms   min=0s       med=0s       max=166.4
http_req_duration..........: avg=143.14ms min=112.87ms med=136.03ms max=1.18s
http_req_receiving.........: avg=5.53ms   min=49µs     med=2.11ms   max=1.01s
http_req_sending...........: avg=30.01µs  min=7µs      med=24µs     max=1.89m
http_req_tls_handshaking...: avg=0s       min=0s       med=0s       max=0s
http_req_waiting...........: avg=137.57ms min=111.44ms med=132.59ms max=589.4
http_reqs..................: 13491  224.848869/s
iteration_duration.........: avg=445.48ms min=413.05ms med=436.36ms max=1.48s
iterations.................: 13410  223.498876/s
vus........................: 100    min=100 max=100
vus_max....................: 100    min=100 max=100
```

COPY

Sergiy Morenets, 2020

Vegeta

- ✓ Created in 2014
- ✓ Written in Go
- ✓ Supports constant rate and concurrency model
- ✓ Supports HTTP/2
- ✓ Generates graphical time slots
- ✓ No programming/scripting support

Wrk

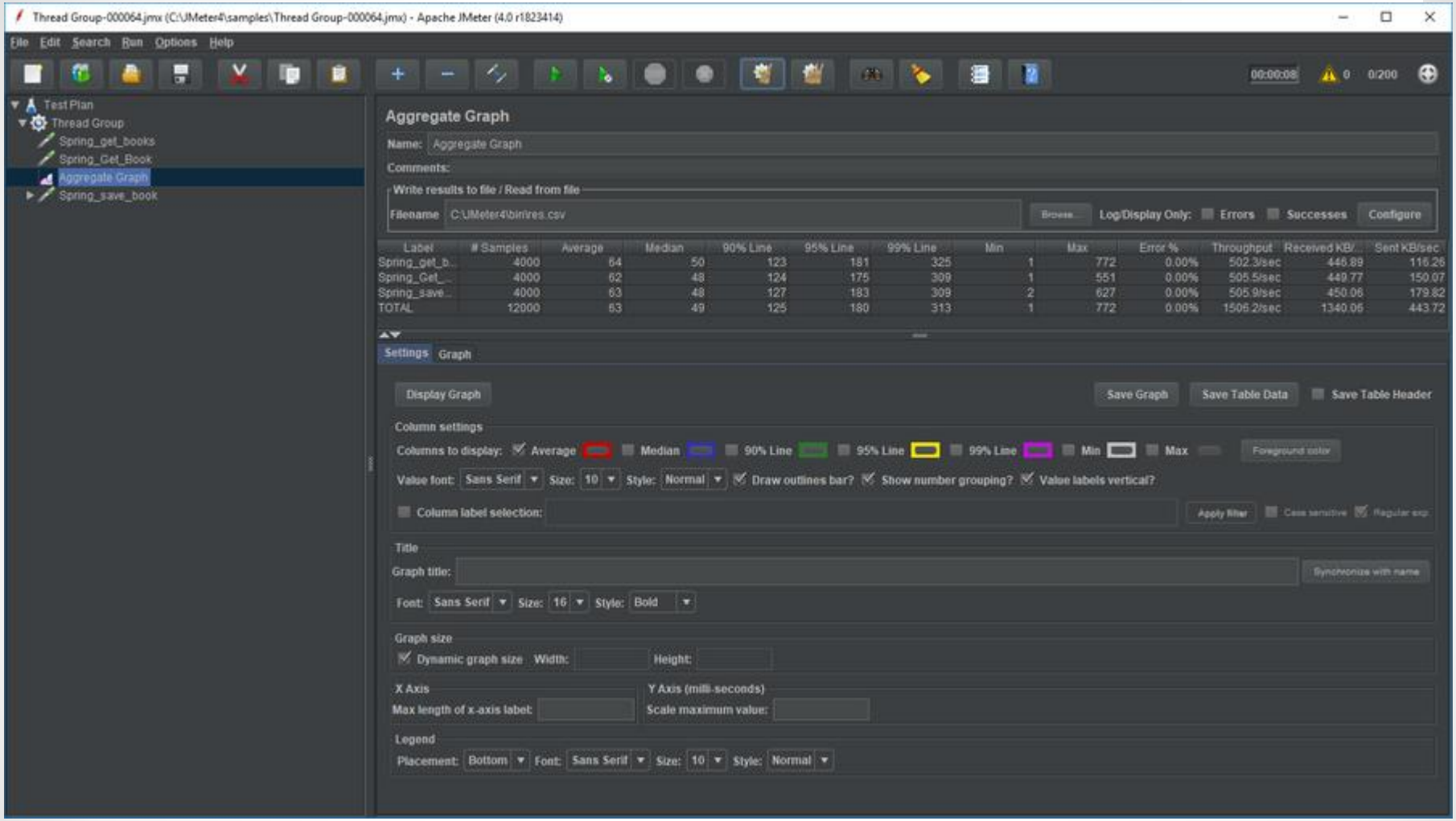
- ✓ Created in 2012
- ✓ Written in C
- ✓ Very fast and reliable solution
- ✓ Has fork Wrk2
- ✓ Offers no scripting and no HTTP/2 support
- ✓ Configuration via Lua scripts

```
wrk.method = "POST"
wrk.body   = "foo=bar&baz=quux"
wrk.headers["Content-Type"] = "application/x-www-form-urlencoded"
```

```
function setup(thread)
   thread:set("id", counter)
   table.insert(threads, thread)
   counter = counter + 1
end
```

JMeter

- First release in 1998 (and 3.0 in 2016)
- Tool for professional testers
- Supports HTTP/JDBC/JMS connections
- Plugin-oriented architecture (with add-ons)
- Supports distributed testing
- Allows GUI or command-line mode (no scripting)
- Used by Blazemeter testing platform



Sergiy Morenets, 2020

Work use-case

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

Reporting

Maximum response time

Minimum response time

Average response time

# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput
4000	41	31	73	95	235	2	773	0.00%	481.9/sec
4000	107	42	384	475	599	1	864	100.00%	488.4/sec
4000	40	30	69	105	238	1	623	0.00%	489.4/sec
12000	63	33	124	257	524	1	864	33.33%	1444.9/sec

Response time that is bigger than 90% faster requests and less than 10% slowest requests

Percentage of requests that responded with error

Parametrization. Request body


```
{
  "title" : "${__machineName()}",
  "id" : 0
}

{
  "title" : test",
  "id" : “${__counter(TRUE, bookCounter)}”
}
```

Jmeter function

Shared flag

Counter name

Benchmarks. JMeter 5.3

- ✓ Ramp-up 5 seconds
- ✓ 5 loops

Users	Avg	Min	Max	Error,%	Req/sec	90%
1	14	3	26	0	67	26
50	4	1	138	0	102	7
250	3	0	182	0	506	5
1000	10	0	394	0	2036	31
2500	383	1	1147	0	4503	669
7500	788	0	4216	0	5599	1738
20000	1600	1	10886	15	5529	3055

Benchmarks. K6 0.26.2

- ✓ Duration 10 seconds
- ✓ No sleep

Users	Avg	Min	Max	Error,%	Req/sec	90%
1	1	0	153	0	1103	1
50	7	1	4070	0	6617	10
250	36	1	4800	0	5691	68
1000	179	1	4700	0	5228	361
2500	268	1	9700	0.09	5193	557
7500	711	2	9040	0.28	4530	1500
20000						

Benchmarks. K6 0.26.2

- ✓ Duration 10 seconds
- ✓ Sleep 1ms between requests

Users	Avg	Min	Max	Error,%	Req/sec	90%
1	2	1	176	0	418	3
50	7	1	3830	0	6716	9
250	41	2	5550	0	5168	68
1000	193	2	5340	0	5000	371
2500	654	3	5040	0.12	3604	1260

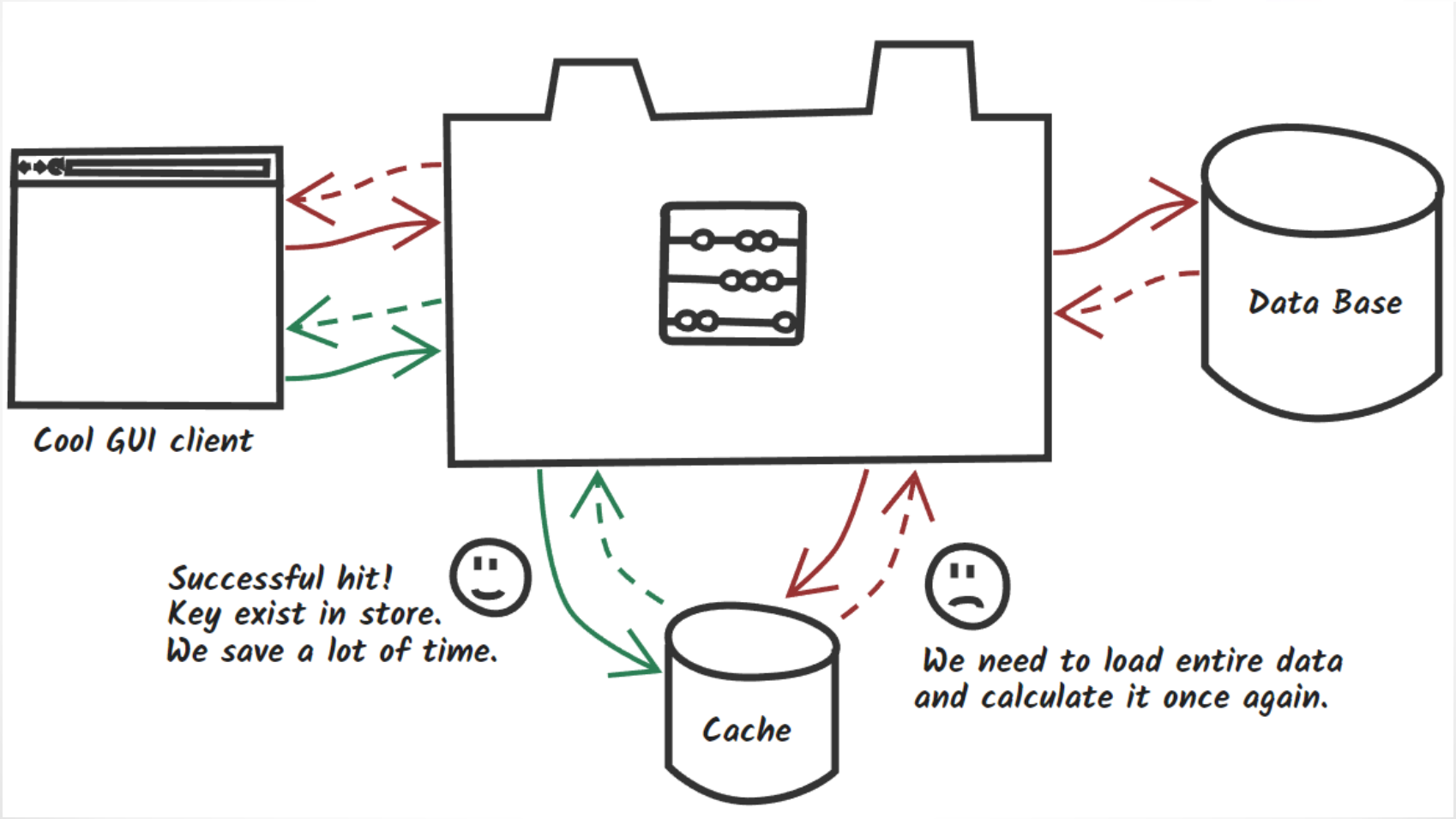
Task 2. Performance testing

1. Download and install Apache JMeter
2. Copy **spring-rest.jmx** from src/main/resources/performance/jmeter into c:/Jmeter/bin folder
3. Run *jmeter -t spring-rest.jmx* command in the **bin** folder. Review JMeter functionality and interface
4. Click on **Test Plan** node and expand Thread Group node. Review its parameters. Review child node[illegible] HTTP request and Aggregated graph.

Caching

- ✓ Internal implementation by Spring (since 3.1)
- ✓ **JCache (JSR-107)** is supported
- ✓ Various 3rd party implementations (EHCache, HazelCast, Infinispan, Couchbase, Redis ,Caffeine)

Caching


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

Caching. Maven dependencies


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

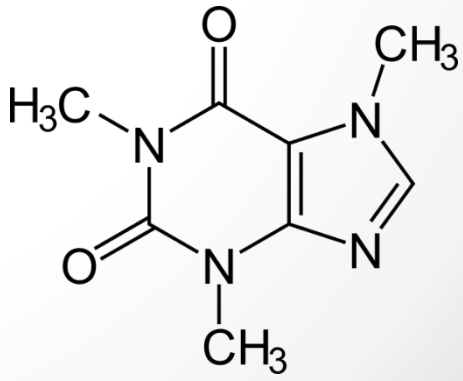
API + default implementation

Cache provider

```
<dependency>
    <groupId>com.github.ben-manes.caffeine</groupId>
    <artifactId>caffeine</artifactId>
    <version>2.8.2</version>
</dependency>
```

Caffeine

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



Caffeine. Examples


```
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

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/evict 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


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

Cache key

```
@PutMapping("{id}")
@CachePut(cacheNames = "books")
public Book update(@PathVariable int id,
                   @RequestBody Book book,
                   HttpServletRequest req) {
```

What is cache key?

```
@PutMapping(path="{id}")
@CachePut(cacheNames = "books", key = "#id")
public BookResource update(@PathVariable int id,
```

Cache key

Advanced caching


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

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

Different response type

```
{   "status": 500,
 "error": "Internal Server Error",
"message": "org.springframework.http.ResponseEntity cannot be cast to
com.example.demo.Book",
 "path": "/book/1" }
```

Cache management

- ✓ Access to cache values
- ✓ Cache statistics
- ✓ Cache configuration

Spring Cache API

```
@Autowired
private CacheManager cacheManager;
```

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

Specify cache provider

```
spring:
    cache:
        cache-names: books
        caffeine:
            spec: expireAfterAccess=1m, maximumSize=1000
```

application.yml

Caching. Statistics

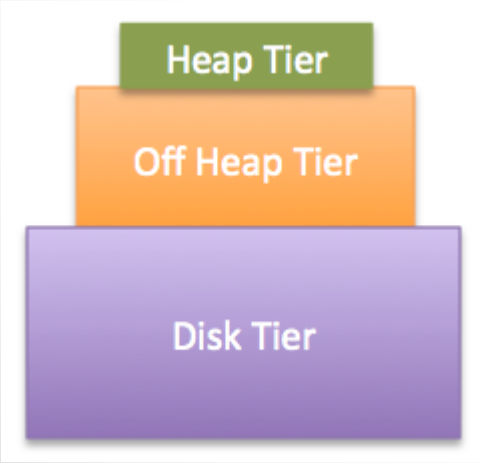
- ✓ Cache usage
- ✓ Hit ratio: hits/(hits + misses
- ✓ Hit rate: hits/seconds
- ✓ Eviction rate
- ✓ Size

Task 2. Caching

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

EHCache 3

- ✓ Declared as Java most wildly used cache
- ✓ Supports heap, off-heap, disk and cluster storage using **resource pools**
- ✓ Integrates with Hibernate/**Spring Cache**/Terracotta
- ✓ Provides programmatic and XML configuration
- ✓ Current version is 3.8.1 (Java 8-based)



EHCACHE

EHCache 3. XML configuration


```
<cache alias="stringCache"> 2
  <key-type>java.lang.String</key-type>
  <value-type>java.lang.String</value-type>
  <heap unit="entries">2000</heap>
</cache>
```

jcache.xml

```
<cache-template name="clientCache">
  <key-type>java.lang.String</key-type>
  <value-type>com.pany.domain.Client</value-type>
  <expiry>
    <ttl unit="minutes">2</ttl>
  </expiry>
  <heap unit="entries">2000</heap>
</cache-template>
```

JCache

- ✓ Standard cache API for Java
- ✓ Implemented by Infinispan, Terracotta, Hazelcust, Oracle Coherence, Apache Ignite and others

EHCACHE

JCache and implementation


```
<dependency>
    <groupId>javax.cache</groupId>
    <artifactId>cache-api</artifactId>
    <version>1.1.0</version>
</dependency>
```

JSR-107

```
 <dependency>
     <groupId>org.ehcache</groupId>
     <artifactId>ehcache</artifactId>
     <version>3.8.1</version>
 </dependency>
```

EHCache as caching provider

JCache annotations


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

```
@GetMapping("/{id}")
@CacheResult(cacheName = "books")
public Book findById(@PathVariable int id) {
    return bookService.findBook(id);
```

Return result and cache it

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

Execute method and update cache

Exception handling


```
@GetMapping("/{id}")
@CacheResult(cacheName = "books")
public Book findById(@PathVariable int id) {
    return bookService.findBook(id);
```

What if exception?

```
@GetMapping("/{id}")
@CacheResult(cacheName = "books",exceptionCacheName = "ex",
        cachedExceptions = Exception.class)
public Book findById(@PathVariable int id) {
```

Cache exceptions

Spring Boot. Cache configuration


```
spring.cache.type=jcache

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

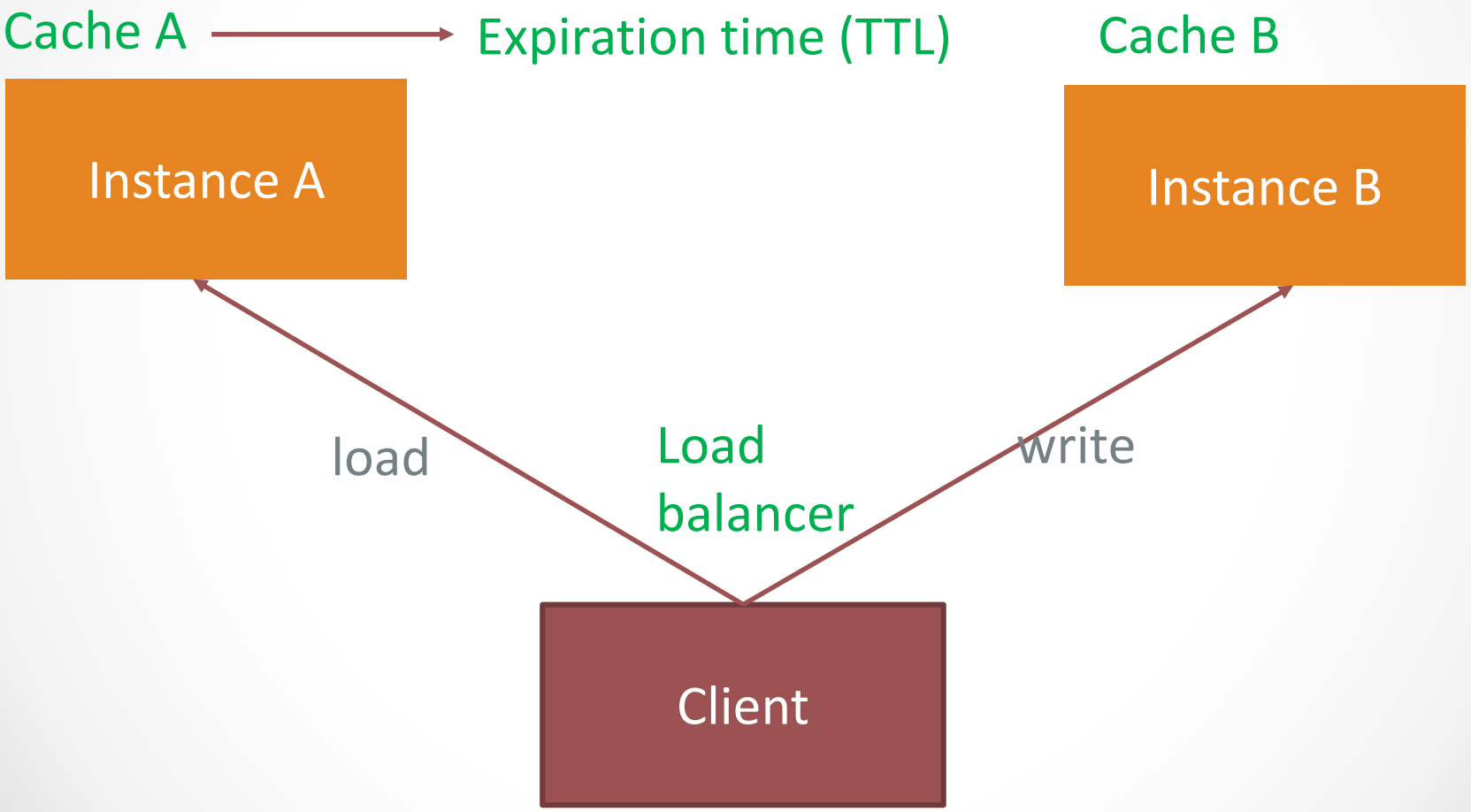
JCache config

src/main/resources/jcache.xml

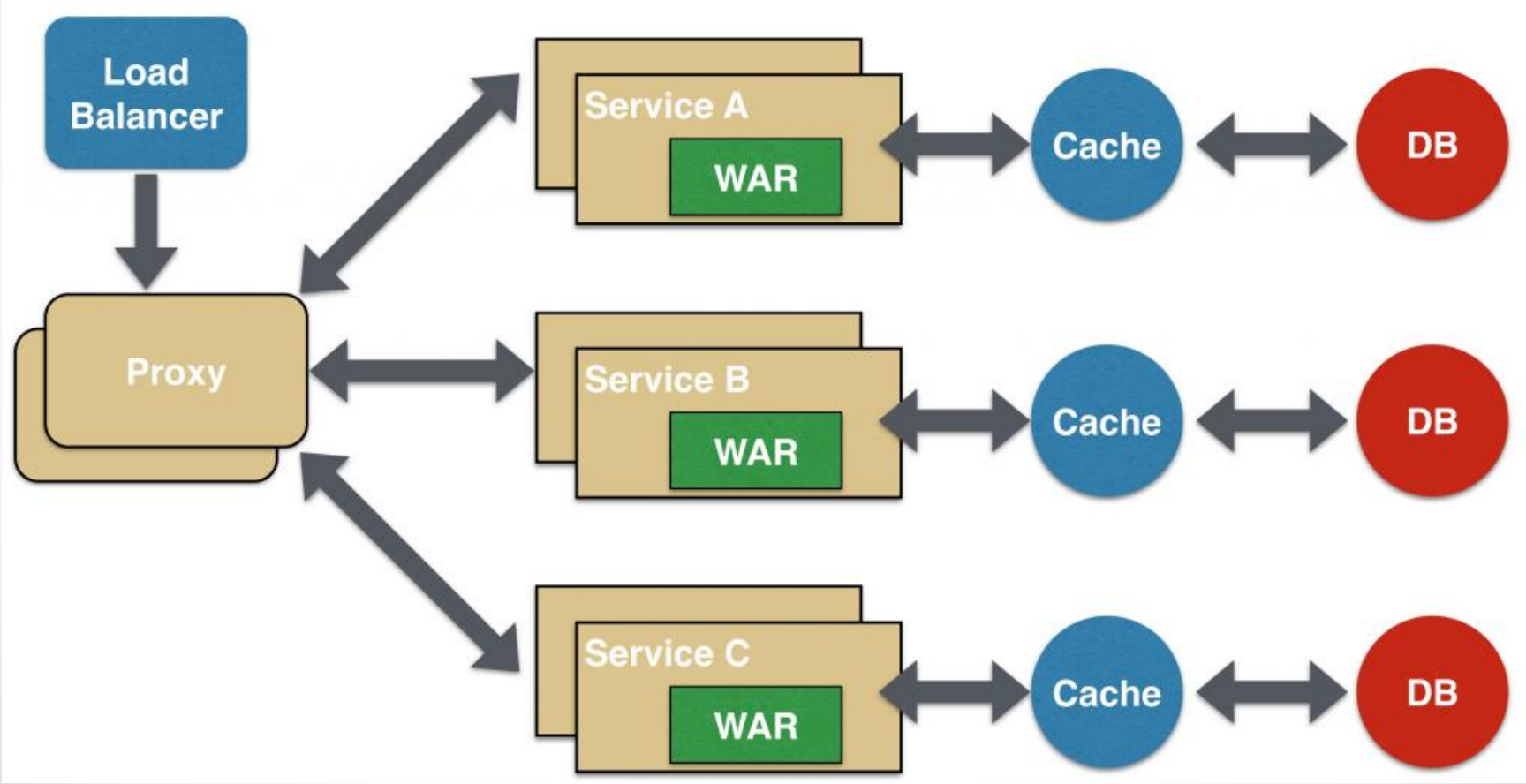
Task 3. JCache and EHCache

1. Add **JCache** and **EHCache** dependencies
2. Replace **@Cacheable** with **@CacheResult** and **@CacheEvict** with @CacheRemove/ @CacheRemoveAll
3. Use **@CacheDefaults** annotation to provide cache name for the entire controller.
4. Review **jcache.xml** configuration file in src/main/resources folder. Where will the cache entries be stored?

Local cache. Drawbacks



Scaling



Sergiy Morenets, 2020

Redis

- Created in 2009 by Salvatore Sanfillipo as **Re**mote **D**isctionary **S**erver
- In-memory database, message broker and cache provider
- Super-lightweight and easy to use
- 6 data types, 180+ commands
- Provides automatic partitioning with **Redis Cluster**
- Supports monitoring and metrics
- Atomic, isolated and consistent
- Eviction algorithms are **LRU** (Last Recently Used) and **LFU** (Last frequently used)

redis

Redis. Popularity

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

redis

Redis. Persistence

- Data is stored in the memory for fast access
- RDB (Redis database file) option makes snapshots at specified intervals
- AOF (Append-only file) option stores every write operation (command) received by server
- You can combine RDB and AOF options or disable persistence
- RDB is perfect for backups and gives maximum performance
- AOF minimize chances of data loss

redis

Redis. High Availability

- **Redis Cluster** automatically split data across multiple servers (partitions)
- Requires Cluster bus for node-to-node communication channel
- **Redis Replication** creates a set of replicate nodes which are copies of master server
- Master periodically sends updates (client commands, keys expired) to the replicas
- Replica can connect to other replicas
- By default replication is asynchronous process

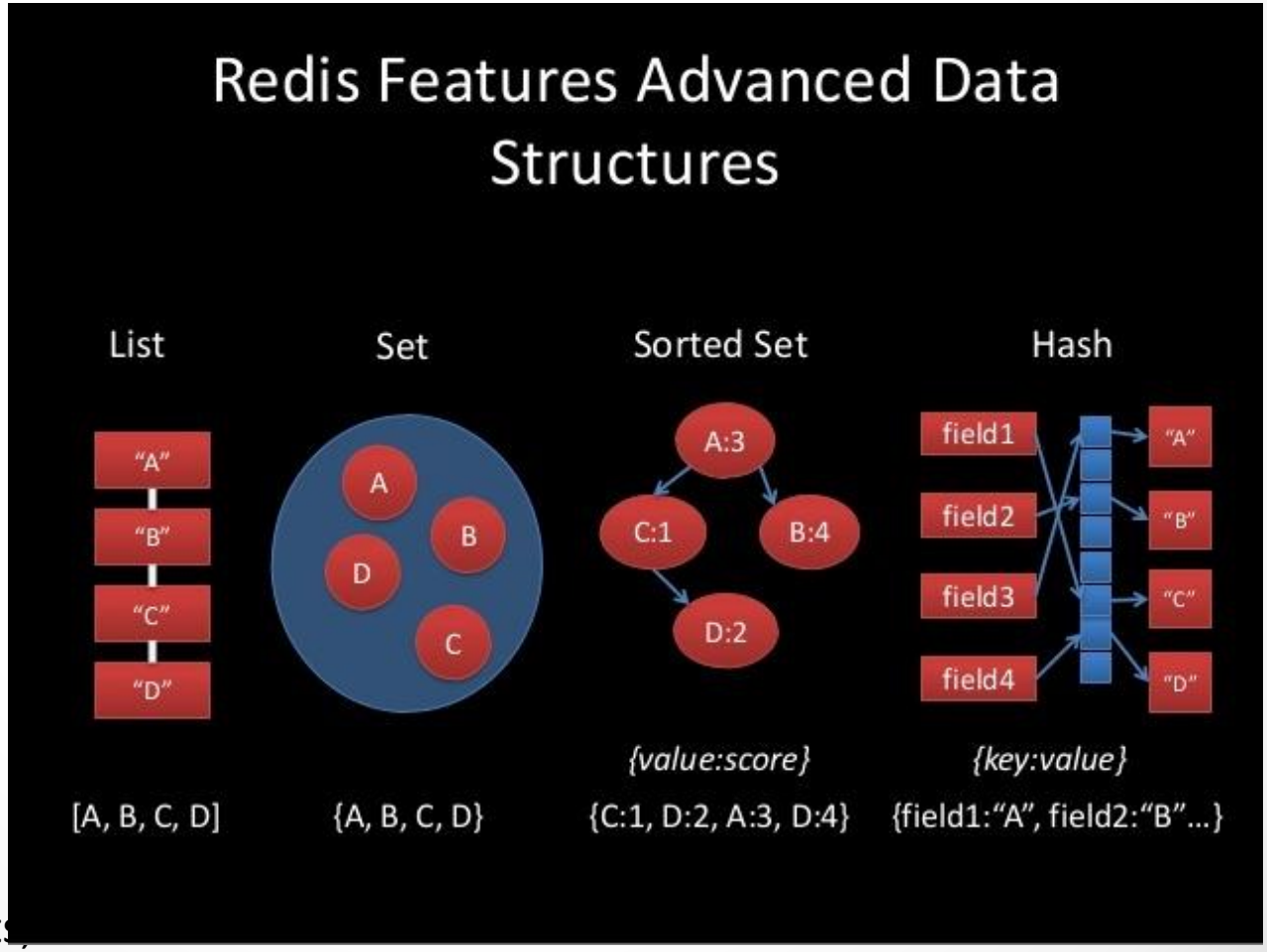
redis

Redis. High Availability

- **Redis Sentinel** is high-availability solution which supports monitoring (health check), automatic failover and configuration provider
- If master server fails then Sentinel promotes some replica server to the master role
- Client connects to Sentinel server to acquire master address
- At least three Sentinel servers (in different availability zones) are recommended

redis

Redis data types



Task 4. Redis configuration

1. Run Redis 6 as Docker container
2. Run Redis CLI
3. Open new terminal and start another Redis console using *docker exec -it redis redis-cl monitor* command. It should print all the commands entered in any console.
4. Try to print all the configuration settings using *CONFIG GET ** command (or *CONFIG GET dir* for single setting).

Redis clients

- ✓ Redisson
- ✓ Jedis (lightweight implementation)
- ✓ Lettuce

Lettuce

- Netty-based
- Support sync/async/reactive approaches
- Supports Redis Standalone/Master-Slave/Cluster/Sentinel
- JDK8+
- Spring/CDI integration
- Used by Spring Data Redis

Redisson

- Java client with in-memory data grid support
- Android and JDK8+
- Replication support (AWS ElasticCache and Azure Redis Cache)
- RxJava 2/Reactive Streams API
- Asynchronous connection pool
- Used in Spring Cache/Spring Data Redis/Spring session/Hibernate Cache
- Implementation of JCache API
- Supports pipelining (command batches)
- Transactions

Redisson

- ✓ Distributed Java objects (Object, Binary stream, AtomicLong
- ✓ Distributed Java collections (Map, Multimap, Set, List, Queue, Deque)
- ✓ Distributed Java locks (Lock, ReadLock, ReadWriteLock, Semaphore)

```
<dependency>
    <groupId>org.redisson</groupId>
    <artifactId>redisson</artifactId>
    <version>3.13.0</version>
</dependency>
```

Redisson. Distributed objects


```
Map<String, String> items = new HashMap<>();
items.put("1", "DATA");
```

Local map

RMap is concurrent asynchronous map with expiration functionality

```
RedissonClient redissonClient = Redisson.create();
RMap<String, Book> map = redissonClient.getMap("books");
Book book = map.put("1", new Book());
Book currentBook = map.putIfAbsent("1",
        new Book());
```

Spring Cache. Keys


```java
public class SimpleKey implements Serializable {

    /**
     * An empty key.
     */
    public static final SimpleKey EMPTY = new SimpleKey();


    private final Object[] params;
```

Needs binary serialization

Key elements

Redisson codecs

- ✓ Jboss marshalling
- ✓ FST
- ✓ Json, Cbor, Avro, MsgPack, Smile, Amazon Ion (Jackson)
- ✓ Kryo/Kryo 5
- ✓ JDK (Java native)
- ✓ Snappy (Netty implementation)
- ✓ LZ4
- ✓ Byte array
- ✓ String
- ✓ Long

Kryo

- Fast binary object Java serialization library
- Supports deep/shallow copying/cloning
- References are not enabled by default
- Lambdas are supported since JDK 8
- Custom/nested serializers

```
<dependency>
    <groupId>com.esotericsoftware</groupId>
    <artifactId>kryo</artifactId>
    <version>4.0.2</version>
</dependency>
```

Spring Cache. Redisson configuration


```
spring.cache.type=jcache
spring.cache.jcache.provider=org.redisson.jcache.JCachingProvider
```

```
@Bean(destroyMethod = "shutdown")
public RedissonClient redisson() {
    Config config = new Config();
    config.setNettyThreads(2);
    config.setCodec(new KryoCodec());
    config.useSingleServer().setAddress("redis://127.0.0.1:6379");
    return Redisson.create(config);
}
```

32 by default

```
@Bean
public CacheManager cacheManager(RedissonClient redissonClient) {
    Map<String, CacheConfig> config = new HashMap<>();
    config.put("books", new CacheConfig(60000, 600000));
    return new RedissonSpringCacheManager(
            redissonClient, config);
}
```

Expiration time (ms)

Max idle time (ms)

Task 5. Spring Cache and Redis

1. Remove Caffeine/EHCache dependencies/configration
2. Add **Redisson** and Spring Boot Starter Cache dependency
3. Add Kryo dependency
4. Add RedissonClient/CacheManager Spring beans and JCache properties.
5. Start application and verify that caching works properly. Opens Redis CLI and check that Redis contains new cache entries

Exposing REST API

API Documentation

API Specification

API Definition

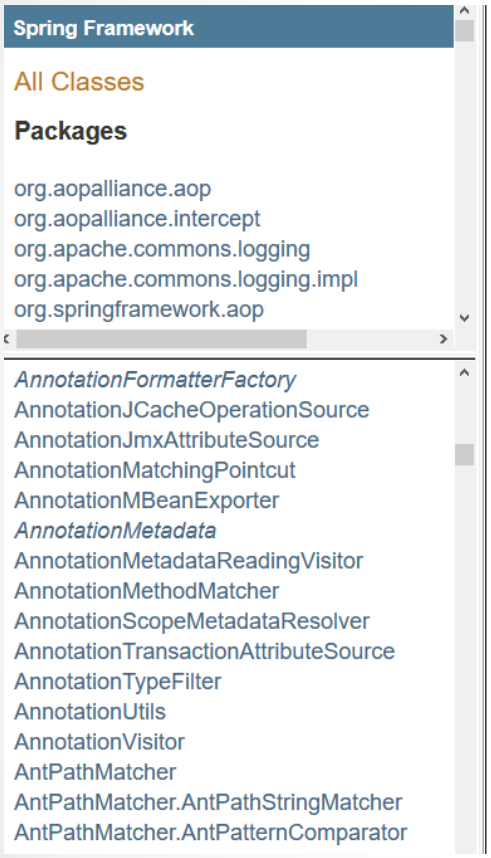
Preconditions

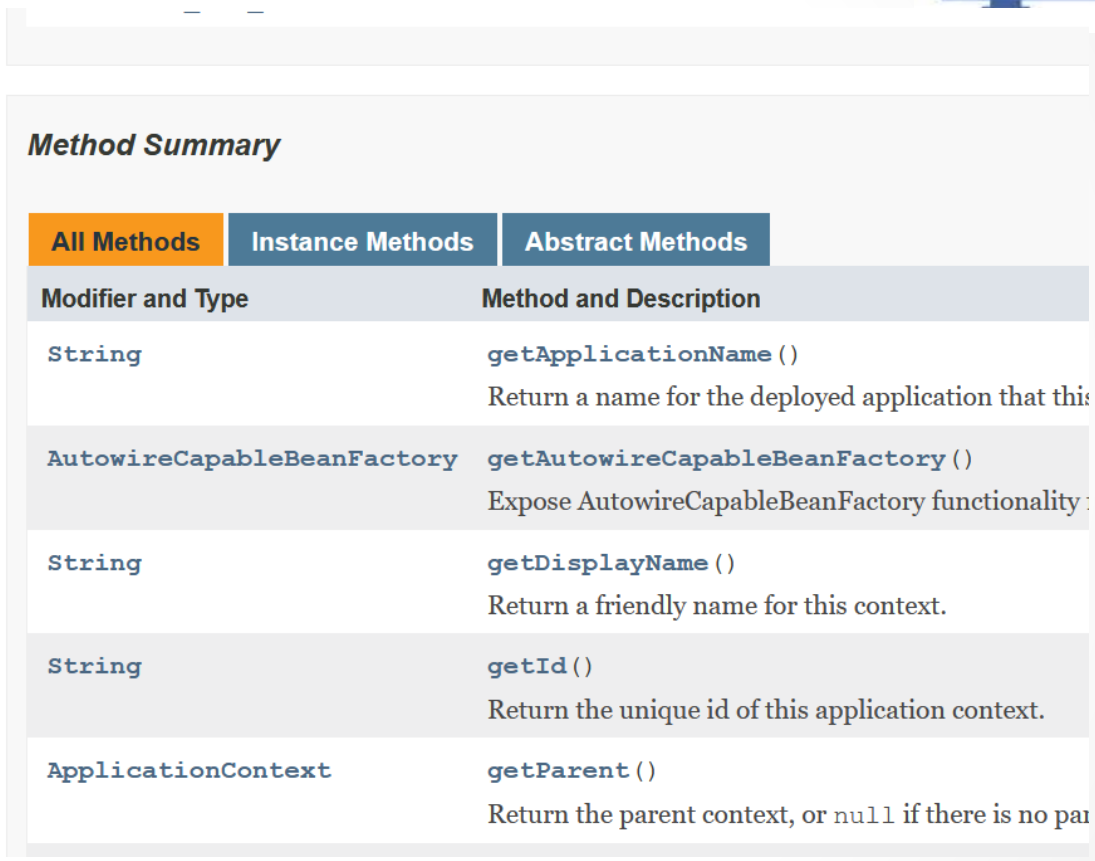

```
@Override
public void clientDepositOpen(int clientId, double sum, Currency currency,
        long accountNumberCurrent) {
    Client client = repositoryInMemory.findClientInRepository(clientId);
    long numberAcountDeposit = client.addNewAccount(TypeOfAccount.deposit,
            currency);
    Account accountDeposit = client.findAccountByNumber(numberAcountDeposit);
    Account accountCurrent = client
            .findAccountByNumber(accountNumberCurrent);
    long numberAccountCurrent = accountCurrent.getAccountNumber();
    if (accountCurrent.checkBalancyIsEnough(sum)){
    Transaction debitingTransaction = new Transaction(sum,
            "списание на пополнение депозита №" + numberAcountDeposit,
            numberAccountCurrent, currency, new Date());
    Transaction creditingTransaction = new Transaction(sum,
            "пополнение депозита", numberAcountDeposit, currency,
            new Date());
    accountCurrent.debitingAccount(debitingTransaction);
    accountDeposit.creditingAccount(creditingTransaction);
    repositoryInMemory.addAccountToRepository(accountCurrent);
    repositoryInMemory.addTransactionToRepository(debitingTransaction);
    repositoryInMemory.addTransactionToRepository(creditingTransaction);
    } else //throw Exception NoMoney;
    repositoryInMemory.addClinetToRepository(client);
```

Preconditions. JavaDocs


```
 * <p>In addition to standard {@link org.springframework.beans.factory.BeanFactory}
 * lifecycle capabilities, ApplicationContext implementations detect and invoke
 * {@link ApplicationContextAware} beans as well as {@link ResourceLoaderAware},
 * {@link ApplicationEventPublisherAware} and {@link MessageSourceAware} beans.
 *
 * @author Rod Johnson
 * @author Juergen Hoeller
 * @see ConfigurableApplicationContext
 * @see org.springframework.beans.factory.BeanFactory
 * @see org.springframework.core.io.ResourceLoader
 */
public interface ApplicationContext extends EnvironmentCapable, ListableBeanFactory,
        MessageSource, ApplicationEventPublisher, ResourcePatternResolver {

    /**
     * Return the unique id of this application context.
     * @return the unique id of the context, or {@code null} if none
     */
    @Nullable
    String getId();

    /**
     * Return a name for the deployed application that this context belongs to.
     * @return a name for the deployed application, or the empty String by default
     */
    String getApplicationName();
```

Preconditions. JavaDocs

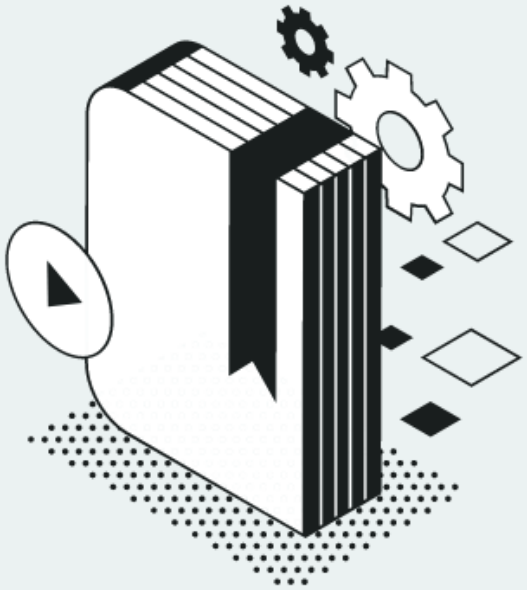




Spring Framework 5.2.6.RELEASE API

This is the public API documentation for the Spring Framework.

Preconditions. Guides & tutorials



Guides

Whatever you're building, these guides are designed to get you productive as quickly as possible – using the latest Spring project releases and techniques as recommended by the Spring team.

Getting Started Guides

15-30 minutes

Topical Guides

60 minutes or less

Tutorials

2-3 hours

Preconditions. Wiki pages

Spring Annotation Programming Model

Terminology

Meta-Annotations

A ***meta-annotation*** is an annotation that is declared on another annotation. An annotation is therefore *meta-annotated* if it is annotated with another annotation. For example, any annotation that is declared to be *documented* is meta-annotated with `@Documented` from the `java.lang.annotation` package.

Stereotype Annotations

A ***stereotype annotation*** is an annotation that is used to declare the role that a component plays within the application. For example, the `@Repository` annotation in the Spring Framework is a marker for any class that fulfills the role or *stereotype* of a repository (also known as Data Access Object or DAO).

`@Component` is a generic stereotype for any Spring-managed component. Any component annotated

Preconditions. Reference

Spring Framework Documentation

Overview	history, design philosophy, feedback, getting started.
Core	IoC Container, Events, Resources, i18n, Validation, Data Binding, Type Conversion, SpEL, AOP.
Testing	Mock Objects, TestContext Framework, Spring MVC Test, WebTestClient.
Data Access	Transactions, DAO Support, JDBC, O/R Mapping, XML Marshalling.
Web Servlet	Spring MVC, WebSocket, SockJS, STOMP Messaging.
Web Reactive	Spring WebFlux, WebClient, WebSocket.

Documentation purpose

- ✓ Use cases
- ✓ Input arguments
- ✓ Response result
- ✓ Error handling
- ✓ Consistency (atomicity)
- ✓ Thread-safe

Exposing REST API

- API documentation contains human-readable reference manual for API with detailed examples of usage (including start guide and tutorials)
- API specification contains general overview of API and relationship between API
- API definition is machine-readable representation. Can be used to generate API documentation/source code

REST API development

REST-first

- Initial discussion
- Cross-platform compatibility
- Schemas reusability

Code-first

- Auto-generation of contract
- Contract synchronization
- Consumer waits for contract

Documentation/specification

- ✓ Swagger (Open API)
- ✓ API Blueprint
- ✓ RAML
- ✓ Slate
- ✓ Spring REST Docs

SLATE

API DOCS GENERATOR

RAML

Spring REST Docs

- Document REST services by writing documentation or use auto-generated snippets provided by Spring MVC Test
- Use Asciidoctor (by default) or Markdown
- Use snippets generated by Spring MVC/WebFlux/REST Assured 3
- Use Mustache to generate snippets
- Integration tests + generated snippets + templates = HTML documentation

```
<dependency>
    <groupId>org.springframework.restdocs</groupId>
    <artifactId>spring-restdocs-mockmvc</artifactId>
    <scope>test</scope>
</dependency>
```

AsciiDoc

- Plain-text mark-up document format
- Created in 2002 by Stewart Rachham as DocBook replacement (XML-based format)
- Asciidoc was also Python-written tool for document rendering/conversion
- In 2013 AsciiDoc Python was replaced by Ruby-written AsciiDoctor
- Used for documentation, wiki-pages (Github/Gitlab), articles and blogs

Asciidoctor

- Text process that converts Asciidoc-files into HTML/DocBook/PDF/ePub
- Asciidoc is lightweight markup language (like Markdown)
- Written in Ruby and integrates with Java/Javascript
- Reads and parses plain text files (in Asciidoc format) and converts into output files

Asciidoctor conversion


```
= AsciiDoc is Writing Zen
Doc Writer <doc.writer@example.com>
:icons: font

_Zen_ in the *art* of writing `plain text` with
http://asciidoc.org[AsciiDoc].

[TIP]
Use http://asciidoctor.org[Asciidoctor] for the best AsciiDoc
experience.footnote:[Not to mention the best looking output!]
Then icon:twitter[role=aqua] about it!

== Sample Section

[square]
* item 1
* item 2

[source,ruby]
----
puts "Hello, World!"
----
```

AsciiDoc source

AsciiDoc is Writing Zen

Doc Writer – doc.writer@example.com

Zen in the **art** of writing `plain text` with AsciiDoc.

Use Asciidoctor for the best AsciiDoc experience.[1] Then about it!

Sample Section

- item 1
- item 2

```
puts "Hello, World!"
```

1. Not to mention the best looking output!

Rendered HTML

- Sergiy Morenets, 2020

Spring REST Docs. Configuration


```
@SpringJUnitWebConfig(BookApplication.class)
@AutoConfigureRestDocs
@AutoConfigureMockMvc
public class BookControllerTest {

    @Autowired
    MockMvc mockMvc;
```

Spring REST Docs. Examples


```
@Test
void findAll_RepositoryEmpty_NoBooksReturned()
        throws Exception {
    // When
    ResultActions actions = mockMvc.perform(get("/book"));
    // Then
    actions.andExpect(status().isOk()).andDo(
            document("index"));
}
```

REST API identifier
(target folder)

```
actions.andExpect(status().isOk()).andDo(
        document("{class-name}/{method-name}"));
```

target/generated-snippets ← Output directory

Spring REST Docs. Parameters


```
@Test
void findBook_validIdentifier_bookReturned()
                            throws Exception {
    // When
    ResultActions actions = mockMvc.perform(
            RestDocumentationRequestBuilders
            .get("/books/{id}", 1));
    // Then
    actions.andExpect(status().isOk()).andDo(
            document("index",
            pathParameters(parameterWithName("id")
                    .description("Unique book identifier"))));
}
```

Spring REST Docs. Input fields


```
@Test
void saveBook_validBook_bookSaved() throws Exception {
    // Given
    Book book = new Book();
    book.setTitle("Java");
    // When
    ResultActions actions = mockMvc.perform(
            RestDocumentationRequestBuilders.post("/books")
            .contentType(MediaType.APPLICATION_JSON)
            .content(jacksonTester.write(book).getJson()));
    // Then
    actions.andExpect(status().isOk())
        .andDo(document("index", requestFields(
            fieldWithPath("id").description("Book identifier"),
            fieldWithPath("title").description("Book title"))));
}
```

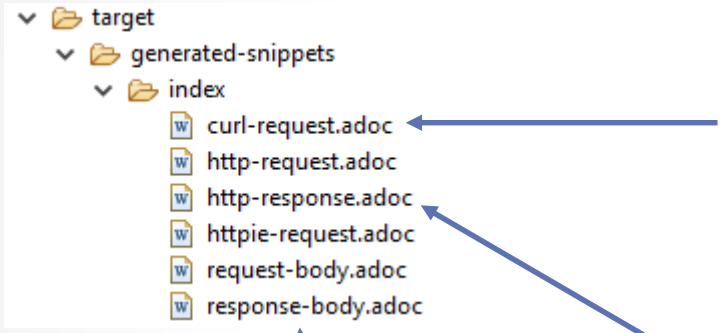
Spring REST Docs. Input fields


```
public class Book {

    private int id;

    private int year;

    private String author;

    private String title;
```

Missed in tests

```
org.springframework.restdocs.snippet.SnippetException: The following parts of
the payload were not documented:
{
  "year" : 0,
  "author" : null}
```

Generated snippets




```
[source,bash]
----
$ curl 'http://localhost:8080/book' -i -X GET
----
```

```
[source,http,options="nowrap"]
----
HTTP/1.1 200 OK
X-Total-Count: 1
Content-Type: application/json;charset=UTF-8
Content-Length: 23

[{"id":1,"title":”Java”}]
----
```

```
[source,options="nowrap"]
----
[{"id":1,"title":"Java"}]
----
```

HTML template

```
==== Links

include::{snippets}/index/http-request.adoc[]
```

src/main/asciidoc/api.html

mvn package

target/generated-docs/api.html

Links

```
GET /book HTTP/1.1
Host: localhost:8080
```

Asciidoctor plugin


```
<plugin>
    <groupId>org.asciidoctor</groupId>
    <artifactId>asciidoctor-maven-plugin</artifactId>
    <version>1.6.0</version>
    <executions>
        <execution>
            <id>generate-docs</id>
            <phase>prepare-package</phase>
            <goals>
                <goal>process-asciidoc</goal>
            </goals>
            <configuration>
                <backend>html</backend>
                <doctype>book</doctype>
            </configuration>
        </execution>
```

Task 6. Spring REST Docs

1. Add Spring REST docs dependency into your project:
2. Add Asciidocker plugin to your project
3. Create new folder **src/main/asciidoc** and create new file library-api.doc there.
4. Update your unit-tests and add code to document your REST-services/parameters/model fields
5. Run your tests and verify that new folder **target/generated-snippets** is generated.

RAML

- Introduced in 2013
- RESTful API Modeling Language
- Supports REST/SOAP specification
- Available tools are API workbench/API designer
- Uses YAML 1.2 format
- Code generators for Node.js, Java, .NET and Python

RAML. GitHub API v3


```
#%RAML 1.0
title: GitHub API
version: v3
baseUri: https://api.github.com
mediaType:  application/json
securitySchemes:
  oauth_2_0: !include securitySchemes/oauth_2_0.raml
types:
  Gist:  !include types/gist.raml
  Gists: !include types/gists.raml
resourceTypes:
  collection: !include types/collection.raml
```

API Blueprint

- ✓ Documentation-oriented web API description language
- ✓ Design-first philosophy
- ✓ Uses Markdown format

```
# Data Structures

## Blog Post (object)
+ id: 42 (number, required)
+ text: Hello World (string)
+ author (Author) - Author of the blog post.

## Author (object)
+ name: Boba Fett
+ email: fett@intergalactic.com
```

API Blueprint

- ✓ Introduced in 2013
- ✓ Focus on C++/C# implementations
- ✓ Server mocking with Apriary
- ✓ Generate client SDKs with Alpaca

```
# Blog Posts [/posts]

## Retrieve All Posts [GET]
+ Response 200 (application/json)
    + Attributes (array[Blog Post])
```

Swagger

- Swagger was created in 2011 by Tony Tam
- Automation of REST documentation and SDK generation
- Maintained by SmartBear Software company
- Includes Swagger code generator, Swagger UI and Swagger annotations
- In 2015 Swagger specification was renamed to OpenAPI Specification

Open API

- API specification language
- Provides language-agnostic interface for describing REST API services without knowing implementation details
- Allows to implement code generation, interactive documentation and test automation
- Doesn’t require changing your services
- Defines structure of OpenAPI document (definition)
- OpenAPI v3 was released in 2017
- Current version is 3.0.2

Open API. REST documentation

JSON

```
{
  "openapi" : "3.0.0",
  "servers" : [
    {
     "url" : "https://pal-test.adyen.com/pal…
    }
  ],
  "info" : {…
  },
  "x-groups" : [
    "General",
    "Modifications"
  ],
  "paths" : {
    "/adjustAuthorisation" : {
    },
    "/authorise" : {
    },
    "/authorise3d" : {
    },
    "/cancel" : {
    },
    "/cancelOrRefund" : {
    },
    "/capture" : {
    },
    "/refund" : {
    },
```

YAML

```
openapi: 3.0.0
servers:
  - url: "https://pal-test.adyen.com/pal…
info:
  version: '30'
  title: Adyen Payment Service
  description: I-
  termsOfService: "https://docs.adyen.com/le…
  contact:
x-groups:
  - General
  - Modifications
paths:
  /adjustAuthorisation:
  /authorise:
  /authorise3d:
  /cancel:
  /cancelOrRefund:
  /capture:
  /refund:
  /technicalCancel:
components:
  schemas:
    Address:
    Amount:
    BackAccount:
    BrowserInfo:
    Card:
```

Service documentation

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

Documentation/specification

- ✓ No documentation is better than wrong documentation
- ✓ Swagger doesn’t support hypermedia and links
- ✓ In Swagger annotations are put on the production code and tied with implementation details

Spring REST Docs. Open API

spring-projects / spring-restdocs

Code | Issues 44 | Pull requests 4 | Actions | Projects 0 | Wik

Build Swagger 2 / OpenAPI specs #213

Closed **kamaydeo** opened this issue on Mar 18, 2016 · 42 comments

wilkinsona commented on May 18, 2017 • edited ▾ Member

@horzsolt No updates to report at this time. The idea hasn't been rejected (hence the issue still being open) and I have given it some thought (generating a single spec from a whole suite of tests is an interesting problem) but I don't have any specific plans to tackle it at this time.

Springfox

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

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
```

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

Springfox configuration


```
@Configuration
@EnableSwagger2
public class SwaggerConfig {

}
```

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

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

JSON specification


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

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


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

Bean validation support

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

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

Override descriptors. Operation


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

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

Override descriptors. Model


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

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

Override descriptors. Response


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

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

Remove default response codes

Springfox. OpenAPI 3.0

Add support for Open API 3.0 #2022

Open **dilipkrish** opened this issue on Sep 7, 2017 · 75 comments

dilipkrish commented on Sep 7, 2017 Member

No description provided.

142 5 8 1 7 4

3.0

Past due by over 1 year **73%** complete

Java 8, Spring 5.0, Spring Boot 2.0, OAS 3.0

3.0 RELEASE #2699

Open **prafsoni** opened this issue on Sep 24, 2018 · 35 comments

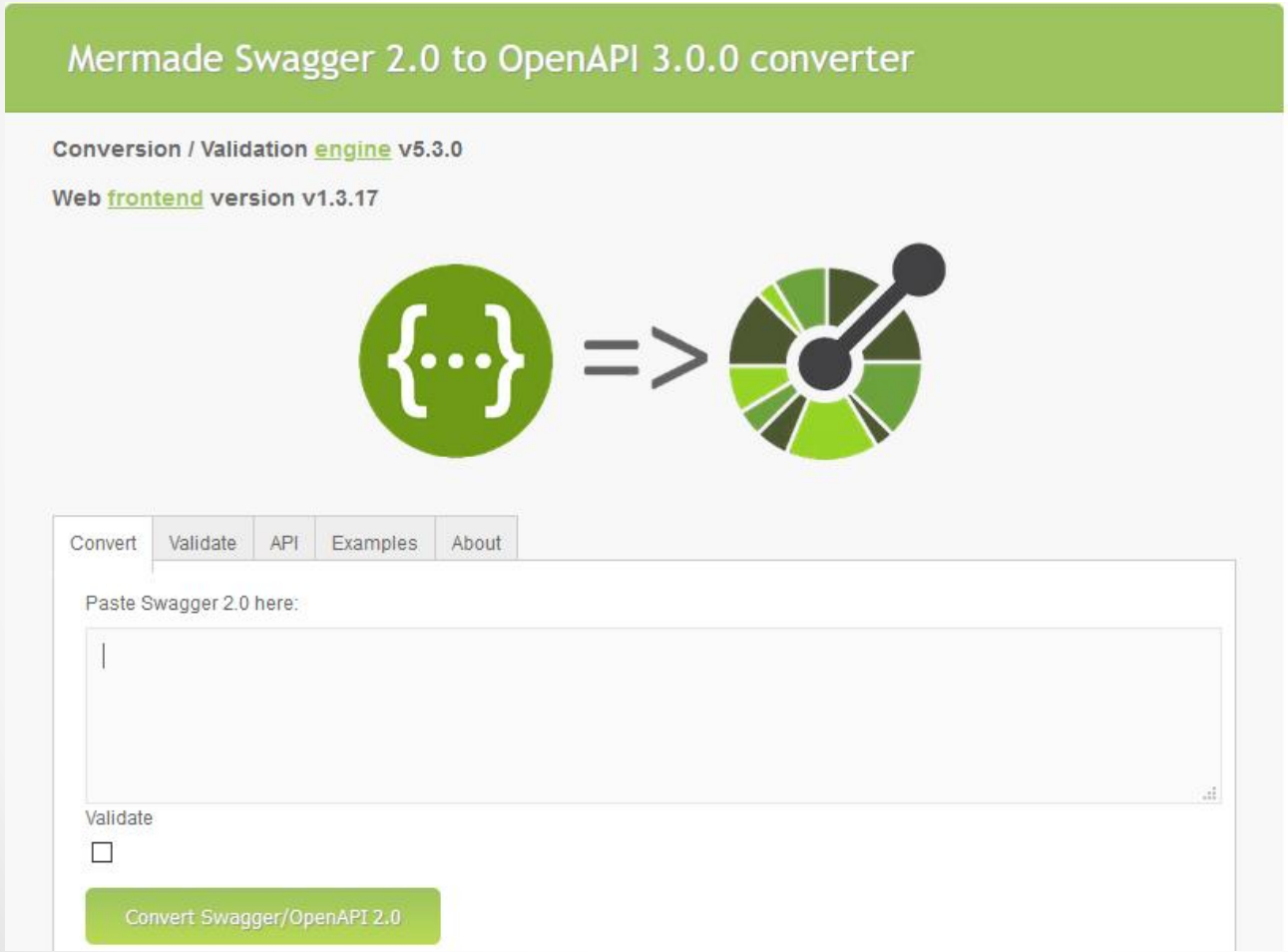
- Sergiy Morenets, 2020

Springdoc-api

- ✓ Spring Boot (1.x/2.x) integration with OpenAPI 3
- ✓ Supports JSON/YAML/HTML formats for documentation
- ✓ Swagger UI
- ✓ OAuth 2.0

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-webmvc-core</artifactId>
    <version>1.3.9</version>
</dependency>
```

Open API 2 -> 3 converter



Sergiy Morenets, 2020

https://mermade.org.uk/openapi-converter

Open API 2 -> 3 converter

- `@Api` -> `@Tag`
- `@ApiIgnore` -> `@Parameter(hidden = true)` or `@Operation(hidden = true)` or `@Hidden`
- `@ApiImplicitParam` -> `@Parameter`
- `@ApiImplicitParams` -> `@Parameters`
- `@ApiModel` -> `@Schema`
- `@ApiModelProperty(hidden = true)` -> `@Schema(accessMode = READ_ONLY)`
- `@ApiModelProperty` -> `@Schema`
- `@ApiOperation(value = "foo", notes = "bar")` -> `@Operation(summary = "foo", description = "bar")`
- `@ApiParam` -> `@Parameter`
- `@ApiResponse(code = 404, message = "foo")` -> `@ApiResponse(responseCode = "404", description = "foo")`

Springdoc-api. Configuration


```
@Configuration
public class OpenAPIConfiguration {

    @Bean
    public OpenAPI springShopOpenAPI() {
        return new OpenAPI().info(
                new Info().title("Library API")
                .description("Book Internet shop")
                .version("1.0"));
    }
}
```

Springdoc-api. Controllers


```
@GetMapping(value = "{id}")
@Operation(summary = "Returns book by identifier", responses = {
        @ApiResponse(description = "Book not found",
                     responseCode = "404"),
        @ApiResponse(description = "Book identifier not valid",
                     responseCode = "400"),
        @ApiResponse(description = "success",
                     responseCode = "default") })
public ResponseEntity<Book> findById(@Parameter(name = "id",
        example = "100", required = true, description =
        "Unique book identifier") @PathVariable int id) {
```

Springdoc-api. Model


```java
@Schema(description = "Library book element")
public class Book {

    @Schema(title = "Book unique identifier", minimum = "1",
            nullable = false, required = true)
    private int id;

    @Schema(title = "Book publishing year", minimum = "2000",
            required = true, example = "2020")
    private int year;
```

Springdoc-api. API definition


```
openapi: "3.0.1"
info:
  title: "OpenAPI definition"
  version: "v0"
servers:
  0:
    url: "http://localhost:8080"
    description: "Generated server url"
paths:
  /books/{id}:
    get:
      tags:
        0: "book-controller"
      summary: "Returns book by identifier"
      operationId: "findById"
      parameters:
        0:
          name: "id"
          in: "path"
          description: "Unique book identifier"
          required: true
```


http://localhost:8080/v3/api-docs

Task 7. SpringDoc Open API

1. Add SpringDoc dependency into your project:
2. Declare **OpenAPI** bean and specify application API version/description
3. Update Spring controllers and add @Operation/@ApiResponse/@Parameter annotations. Add @Schema annotation to DTO properties
4. Start application and verify that this URL contains OpenAPI specification

Springdoc-api. Swagger UI

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-webmvc-core</artifactId>
    <version>1.3.9</version>
</dependency>
```

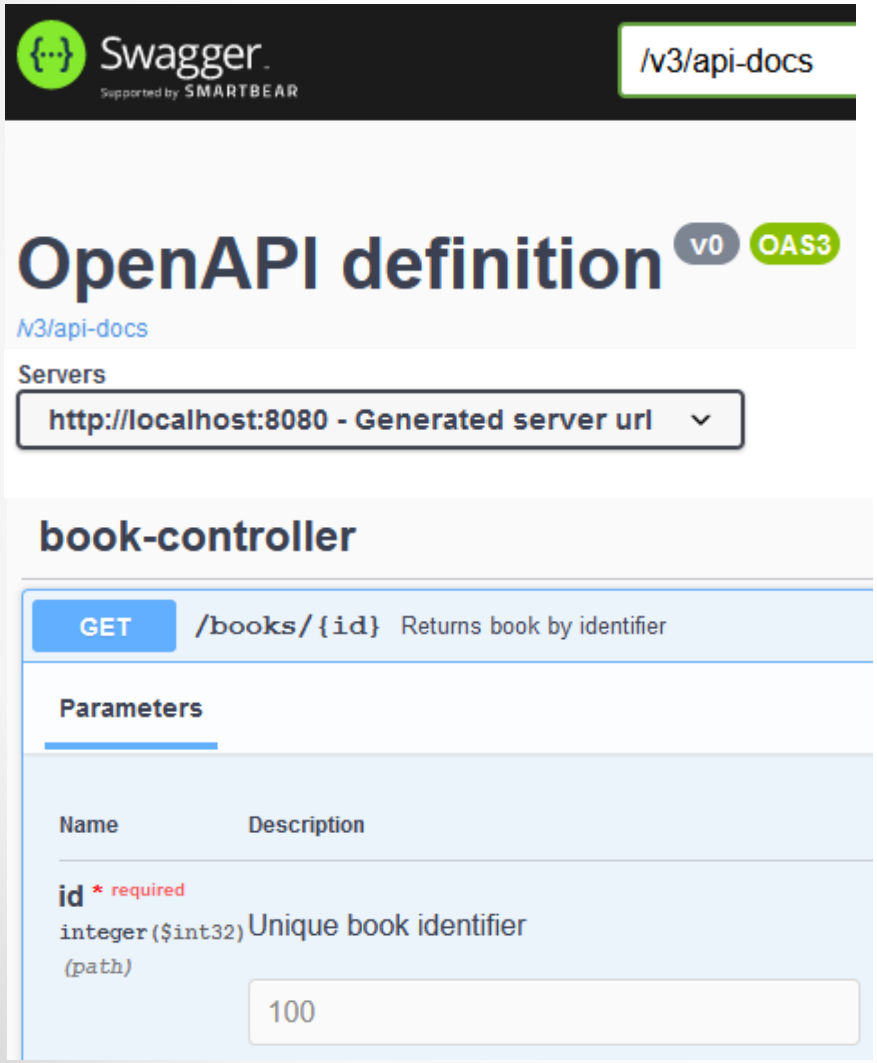
```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
    <version>1.3.9</version>
</dependency>
```

Without and with Swagger UI

```
springdoc.swagger-ui.path=/swagger.html
```

Adjust Swagger UI path

Springdoc-api. Swagger UI



http://localhost:8080/swagger.html

Springdoc-api. Spring REST Docs


```
<dependency>
    <groupId>com.epages</groupId>
    <artifactId>restdocs-api-spec-mockmvc</artifactId>
    <version>0.9.9</version>
</dependency>
```

Resides in Spring repository

```
actions.andExpect(status().isOk()).andDo(
        document("index", ResourceDocumentation
                .resource(ResourceSnippetParameters
                .builder().pathParameters(parameterWithName("id")
                .description("Unique book identifier")).build())));
```

mvn package

target/generated-snippets/index/resource.json

API definition

Sergiy Morenets, 2020

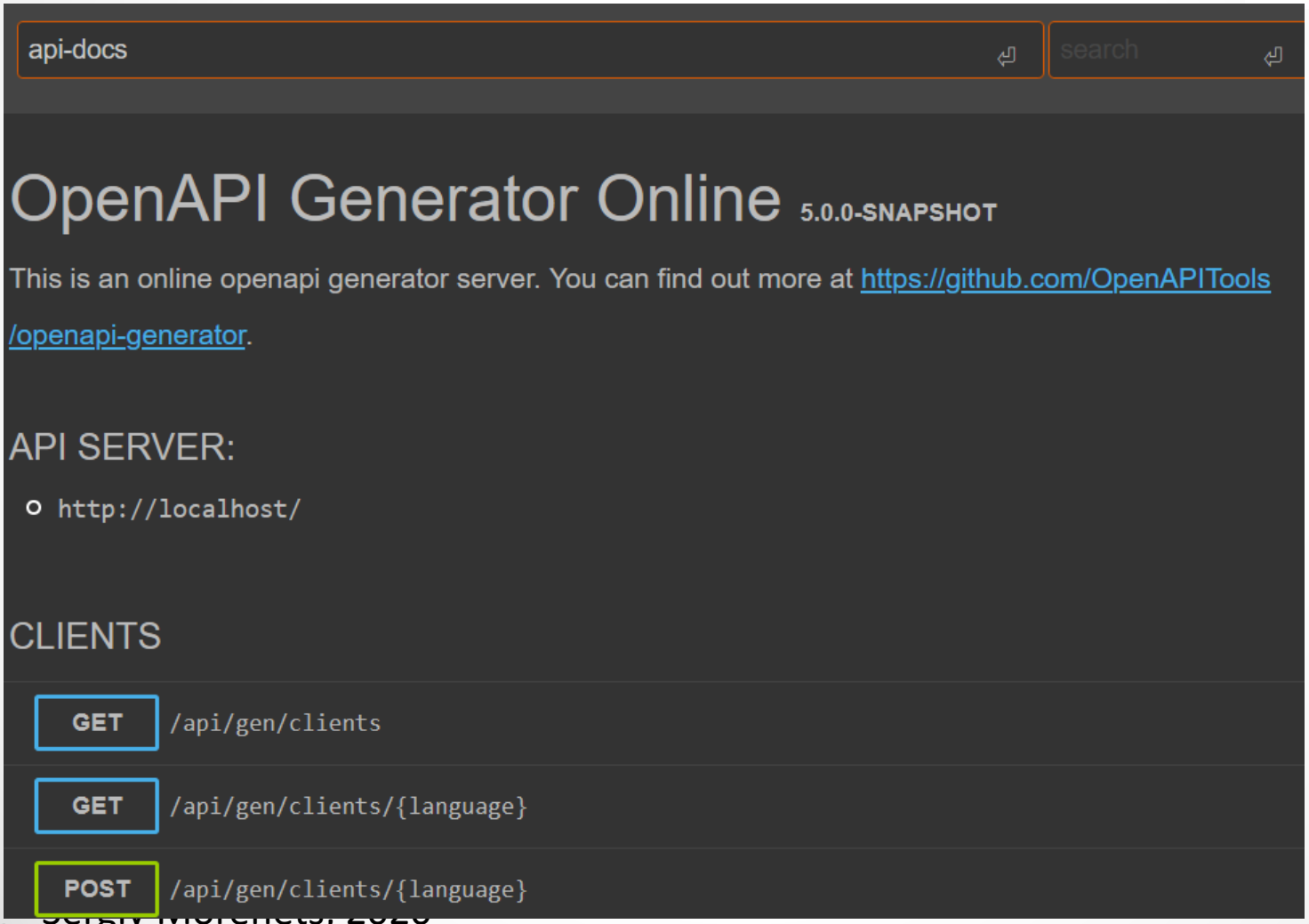
Open API generator

- ✓ Generates client/server code from OpenAPI specifications
- ✓ Supports more than 20 languages
- ✓ Based on Mustache templates
- ✓ Supports Jersey 1.x/2.x, OpenFeign 10, OkHttp 3.x, Retrofit, Google API Client and RestTemplate/WebClient as HTTP client
- ✓ Supports Gson/Jackson for serialization
- ✓ Generates integration tests

Open API generator


```
<plugin>
    <groupId>org.openapitools</groupId>
    <artifactId>openapi-generator-maven-plugin</artifactId>
    <version>4.3.1</version>
    <executions>
        <execution>
            <goals>
                <goal>generate</goal>
            </goals>
            <configuration>
                <inputSpec>${project.basedir}/src/main/resources/resource.js
                <generatorName>java</generatorName>
                <library>okhttp-gson</library>
                <generateApiTests>false</generateApiTests>
                <configOptions>
                    <sourceFolder>src/gen/java/main</sourceFolder>
                </configOptions>
            </configuration>
```

Open API generator online



Open API generator. Model


```java
@javax.annotation.Generated(value = "org.openapitools.codegen.languages.
public class Book {
  public static final String SERIALIZED_NAME_ID = "id";
  @SerializedName(SERIALIZED_NAME_ID)
  private Integer id;

  public static final String SERIALIZED_NAME_YEAR = "year";
  @SerializedName(SERIALIZED_NAME_YEAR)
  private Integer year;
```

```java
@javax.annotation.Nullable
@ApiModelProperty(value = "Publishing year")
public Integer getYear() {
  return year;
}
```

Open API generator. API client


```
public class BookControllerApi {
    private ApiClient localVarApiClient;

    public BookControllerApi() {
        this(Configuration.getDefaultApiClient());
    }
```

```
public okhttp3.Call findAllCall(final ApiCallback _callback)
    Object localVarPostBody = null;

    // create path and map variables
    String localVarPath = "/books";
```

```
public List<BookRecord> findAll() throws ApiException {
    ApiResponse<List<BookRecord>> localVarResp = findAllWithHttpI
    return localVarResp.getData();
```

Task 9. SpringDoc and REST Docs

1. Add SpringDoc Open API UI dependency
2. Start application and open Swagger UI. Try to invoke existing REST services.
3. Add SpringDoc REST Docs integration dependency
4. Update Spring REST Docs integration test to include API specification for the REST-services/DTO classes. Run mvn package task to generate API specification file

API Versioning

- Server should provide API definition and **API versioning**

- **URI** versioning (LinkedIn, Yahoo, Salesforce, Google, Dropbox, Instagram, Paypal)
- **URI parameter** versioning (Netflix, Facebook)
- **Accept header** versioning (GitHub)
- **Custom header** versioning (Microsoft Azure)

Versioning. URI and Header


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

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

Since we return the same resource(article) then URI should be the same

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

If server caches response then we can receive cached response for other version

Version could be omitted so server should choose default version

Versioning. Media type

Accept: application/json ← Use default (current) API version

Accept: application/vnd.github+json

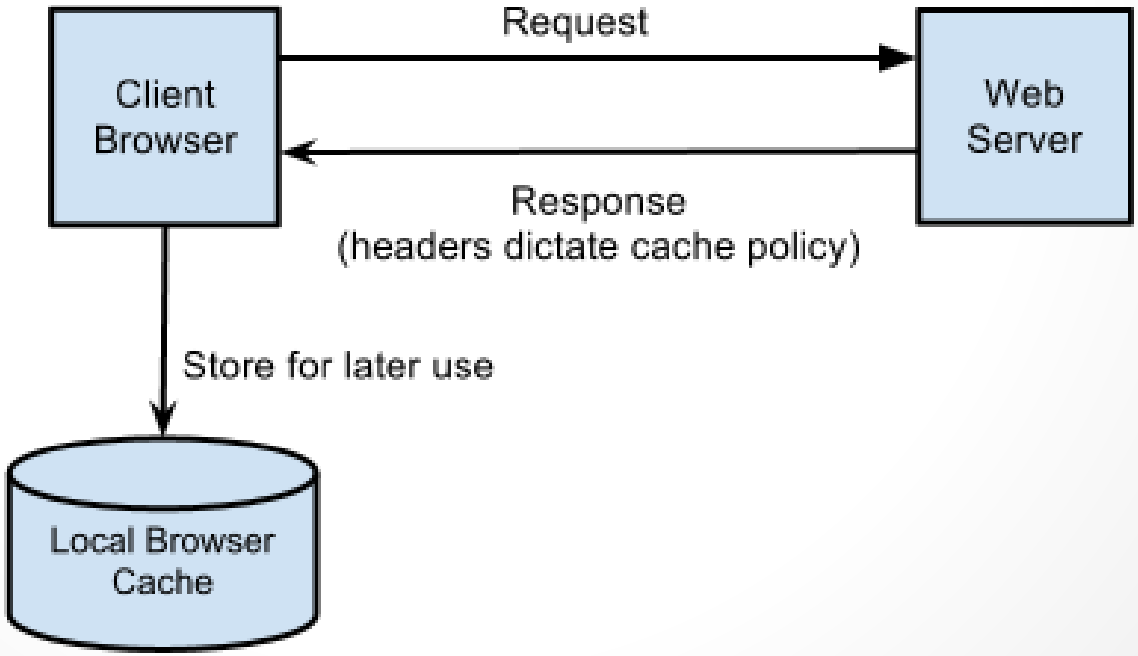
Accept: application/vnd.github.v3+json ← Use API version 3

ETag

- Part of HTTP 1.1 specification
- Enhanced replacement for Last-Modified header
- Included by server as response header
- ETag check can be weak or strong

HTTP Cache

- ✓ Managed by Expires and Cache-Control response headers
- ✓ Most CSS/JavaScript/image files are cached by the browser



ETag workflow

Client sends request

GET /URI

Server sends response

ETag header is generated

ETag: 0f15973b3f0ef34670ec2a8fa82ecfd59

Client sends request

GET /URI

Server checks header

If-None-Match: 0f15973b3f0ef34670ec2a8fa82ecfd59

Server sends reponse

304 (Not Modified)

Spring MVC and ETag

- If there is Cache-control: no-store response header then e-tag functionality is disabled
- **ShallowEtagHeaderFilter** is responsible to check If-None-Match header and generate ETag header (if it's missing in the response)
- By default ETag is weak and generated based on MD5 hash
- Strong ETag check includes request headers comparison and byte ordering
- Good candidate for entity ETag is object version

Spring MVC and ETag


```
@Configuration
public class WebMvcConfiguration implements WebMvcConfigurer {

    @Bean
    public ShallowEtagHeaderFilter etagFilter() {
        return new ShallowEtagHeaderFilter();
    }
```

```
@GetMapping(value = "{id}")
public ResponseEntity<Book> findById(@PathVariable int id) {
    Book book = bookService.findBook(id);
    return ResponseEntity.ok()
            .eTag(book.getVersion()).body(book);
```

Task 10. ETag

1. Add new field **version** to the Book class that should auto-incremented each time Book object has changed
2. Add new Spring bean ShallowEtagHeaderFilter
3. Update REST-services that returns Book objects and include ETag header
4. Run application and verify that ETag header is sent to the client

Rate limiter

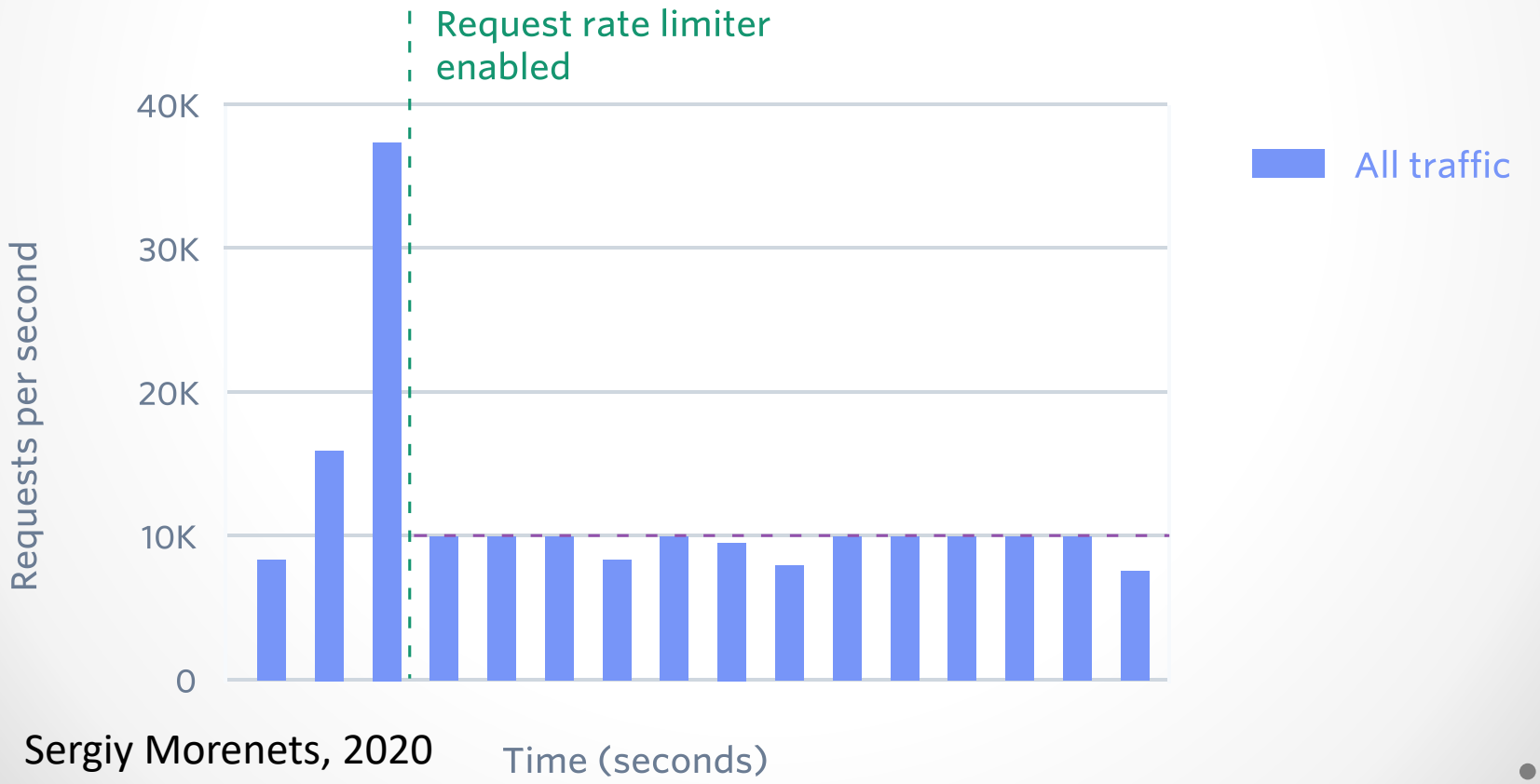
- Control rate of requests or prevents Dos attacks (or traffic spike)
- Programmatic rate limiters checks user session (or IP) in current time window (minute or hour)
- Preserves heavy-loading by automated scripts (for example, in test mode)
- Prefer high-priority requests over low-priority and optionally drops low-priority requests (load shedding)

Rate limiter types

- ✓ Request rate limiter
- ✓ Concurrent requests limiter
- ✓ Fleet usage load shedder
- ✓ Worker utilization load shedder

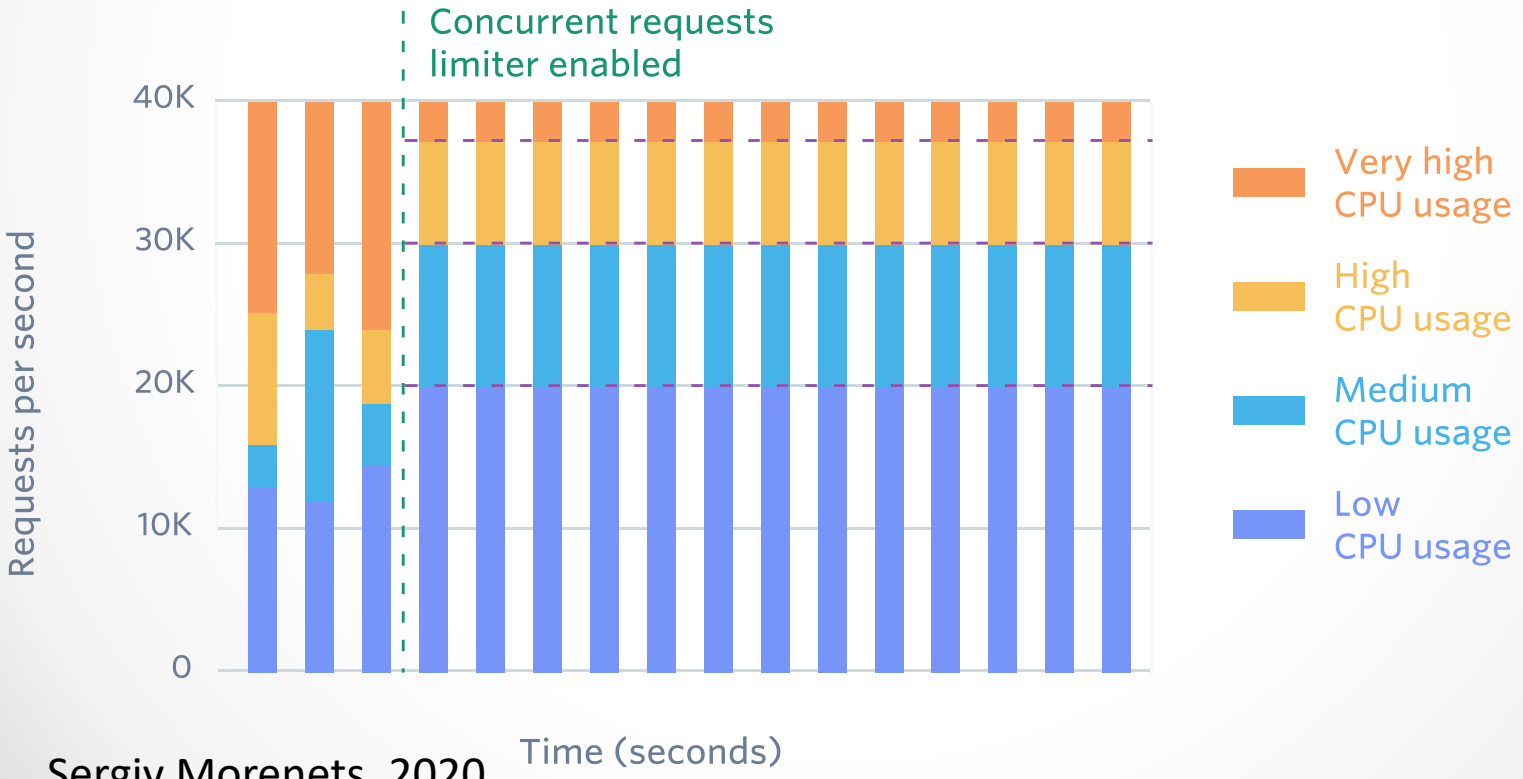
Request rate limiter

- ✔ Restricts each user to *N* requests per second (minute, hour)



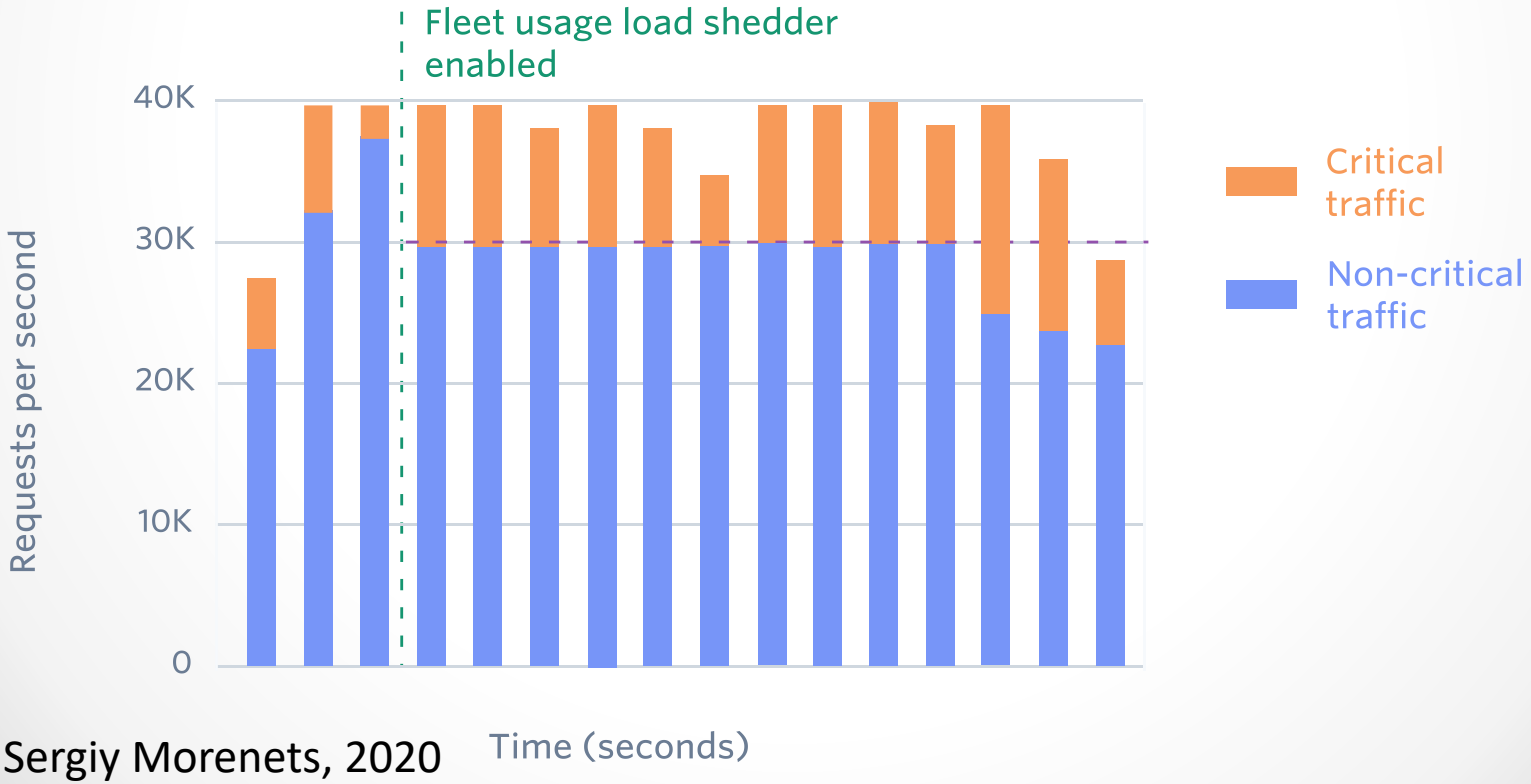
Concurrent requests limiter

- ✓ Restricts each user to *N* concurrent requests



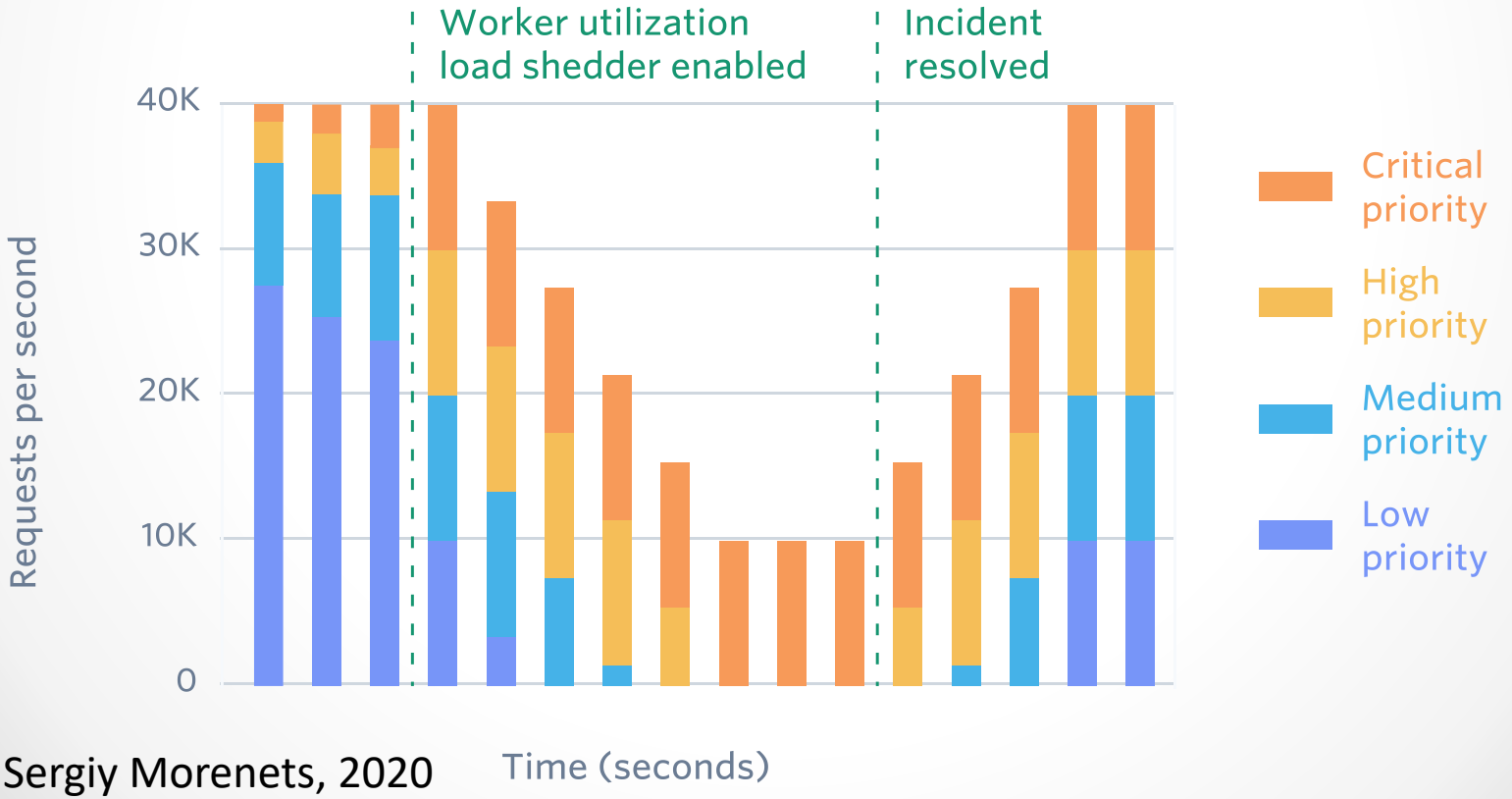
Fleet usage load shedder

- ✔ Preserves space for most critical API requests (POST, PUT)



Worker utilization load shedder

- If workers in a thread pool are over-load then drop less-critical requests



Rate limiter. Github

- GitHub allows up to 5000 requests per hour (for authenticated users) and 60 requests per (anonymous)

```
curl -i https://api.github.com/users/octocat
HTTP/1.1 200 OK
Date: Mon, 01 Jul 2013 17:27:06 GMT
Status: 200 OK
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 56
X-RateLimit-Reset: 1372700873
```

Maximum number of requests per hour

Number of remaining requests in current rate window

Time when current rate window resets

Rate limiter. Error response


```
HTTP/1.1 403 Forbidden
Date: Tue, 20 Aug 2013 14:50:41 GMT
Status: 403 Forbidden
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 0
X-RateLimit-Reset: 1377013266
{
   "message": "API rate limit exceeded for xxx.xxx.xxx.xxx. (But here's
   "documentation_url": "https://developer.github.com/v3/#rate-limiting"
}
```

```
HTTP/1.1 403 Forbidden
Content-Type: application/json; charset=utf-8
Connection: close
{
  "message": "You have triggered an abuse detection mechanism and have beer
  "documentation_url": "https://developer.github.com/v3/#abuse-rate-limits"
}
```

GitHub best practices against abuse

- ✓ Make authenticated requests
- ✓ Don’t make requests concurrently
- ✓ Provide 1 second delay for a batch of requests (automated script)
- ✓ Use Retry-after response header (number of seconds to wait until next request)
- ✓ Split low-priority (GET, test mode) and high-priority (POST) requests

Twitter. API limits

- Rate limits are divided into 15 minute intervals
- 15 requests per rate limit window
- When user exceeds rate limit it receives HTTP code 429 “Too many requests”
- If application(user) abuses rate limits it will be blacklisted
- Twitter suggest to use back-off pattern for retry attempts

https://api.twitter.com/1.1/application/rate_limit_status.json

Get application current resource limits

Twitter. API limits

GET https://api.twitter.com/1.1/application/rate_limit_status.json?resources=users

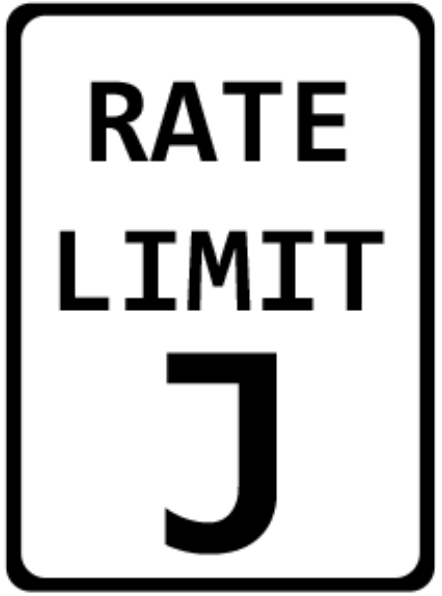

```
{
"resources": {
   "users": {
    "/users/profile_banner": {
     "limit": 180,
     "remaining": 180,
     "reset": 1403602426
    },
    "/users/suggestions/:slug/members": {
     "limit": 15,
     "remaining": 15,
     "reset": 1403602426
    },
    "/users/show/:id": {
     "limit": 180,
     "remaining": 180,
     "reset": 1403602426
    }
```

Rate limiter. Implementations

- ✓ Token bucket
- ✓ Leaky bucket
- ✓ Fixed window counters
- ✓ Sliding window log
- ✓ Sliding window counters

RateLimitJ

- ✓ Java library for rate limiting
- ✓ Pluggable architecture
- ✓ In-memory, Redis and Dropwizard rate limiter
- ✓ Support synchronous/asynchronous and reactive usage



Sergiy Morenets, 2020

Bucket4j

- Java rate limiting library based on **token bucket** algorithm
- Effective lock-free implementation
- Scalable for multi-threading usage
- Multiple bandwidths per bucket (per hour and per minute)
- Supports JCache providers (Hazelcast, Ignite, Infinispan)
- Synchronous/asynchronous API
- Monitoring and logging based on Listener API

Bucket4j

Throttle all the things!

Bucket4j. Supported back-ends

Technology	Async mode	Optimized serialization
Jcache API	No	No
Hazelcast	Yes	Yes
Apache Ignite	Yes	N/A
Infinispan	Yes	Yes
Oracle Coherence	Yes	Yes

Bucket4j. Examples


```
Bandwidth hourLimit = Bandwidth.simple(1000,
        Duration.ofHours(1));
Bandwidth minLimit = Bandwidth.simple(100,
        Duration.ofMinutes(1));

Bucket bucket = Bucket4j.builder()
        .addLimit(hourLimit)
        .addLimit(minLimit).build();

if(bucket.tryConsume(1)) {
    //success
} else {
    //limit exceeded
}
```

Bucket4j and Spring Boot

- ✔ Support Servlet API, Zuul Filter and WebFlux

```
<dependency>
    <groupId>com.giffing.bucket4j.spring.boot.starter</groupId>
    <artifactId>bucket4j-spring-boot-starter</artifactId>
    <version>0.2.0</version>
</dependency>
```

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

Bucket4j. Bandwidths


```yaml
bucket4j:
  enabled: true
  filters:
  - cache-name: buckets
    url: .*
    rate-limits:
    - bandwidths:
      - time: 1
        unit: minutes
        capacity: 1
      - time: 1
        unit: hours
        capacity: 1000
```

Cache name (optional)

URL to filter requests (optional)

20 requests per minute

1000 requests per hour

application.yml

Bucket4j. Expressions


```yaml
rate-limits:
- expression: getRemoteAddr()
  execute-condition: getRequestURI() == '/'
  bandwidths:
  - time: 1
    unit: minutes
    capacity: 20
- expression: "'default'"
  execute-condition: getRequestURI() != '/'
  bandwidths:
  - time: 1
    unit: minutes
    capacity: 5
```

Cache key is client IP

Only for root URI requests

For all IP addresses

Only for non-root URI requests

application.yml

Bucket4j. Cache configuration


```
<cache alias="buckets">
    <key-type>java.lang.String</key-type>
    <value-type>io.github.bucket4j.grid.GridBucketState</value-type>
    <heap unit="entries">10000</heap>
    <heap-store-settings>
        <max-object-size>2000</max-object-size>
    </heap-store-settings>
</cache>
```

jcache.xml

Task 11. Rate limiting

1. Add Bucket4j dependency
2. Open application.yml and put rate limiting filters for all requests.
3. Run application and try to exceed rate limits. How will application respond?
4. (Optional) Try to setup different rate limits for different REST-services

Spring Data REST

- Part of Spring Data project and based on Spring Data repositories
- Exposes domain model as HTTP resources
- Provides discoverable API with HAL support
- Supports paginated response
- Can be used with JPA, Mongo, Solr, Neo4j, Cassandra
- Current version 3.3.0

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

Spring Data REST. Repository


```
public interface BookRepository extends
                    JpaRepository<Book, Integer> {
```

```
▼ _links:
   ▼ books:
        href:        "http://localhost:8080/books{?page,size,sort}"
        templated:   true
   ▼ profile:
        href:        "http://localhost:8080/profile"
```

http://localhost:8080/

```
{
  "alps": {
    "version": "1.0",
    "descriptor": [
      {
        "id": "book-representation",
        "href": "http://localhost:8080/profile/books",
        "descriptor": [
          {
            "name": "year",
            "type": "SEMANTIC"
          },
          {
            "name": "author",
            "type": "SEMANTIC"
          },
          {
            "name": "title",
            "type": "SEMANTIC"
          },
          {
            "name": "publicationDate",
            "type": "SEMANTIC"
          }
        ]
      },
      {
        "id": "create-books",
        "name": "books",
        "type": "UNSAFE",
        "descriptor": [],
        "rt": "#book-representation"
      },
      {
        "id": "patch-book",
        "name": "book",
        "type": "UNSAFE",
        "descriptor": [],
        "rt": "#book-representation"
      },
```


http://localhost:8080/profile/books

Spring Data REST. Resources

POST /books →

```
{
  "year": 2020,
  "author": "Unknown",
  "title": "Spring Data REST",
  "publicationDate": null,
  "_links": {
    "self": {
      "href": "http://localhost:8080/books/1"
    },
    "book": {
      "href": "http://localhost:8080/books/1"
    }
  }
}
```

Spring Data REST. Resources

GET /books

```
{
  "_embedded": {
    "books": [
      {
        "year": 2020,
        "author": "Unknown",
        "title": "Spring Data REST",
        "publicationDate": null,
        "_links": {
          "self": {
            "href": "http://localhost:8080/books/1"
          },
          "book": {
            "href": "http://localhost:8080/books/1"
          }
        }
      }
    ]
  },
```

Spring Data REST. Resources


```
"_links": {
  "self": {
    "href": "http://localhost:8080/books"
  },
  "profile": {
    "href": "http://localhost:8080/profile/books"
  }
},
"page": {
  "size": 20,
  "totalElements": 2,
  "totalPages": 1,
  "number": 0
}
```

GET /books

Spring Data REST. Restrictions


```
public interface BookRepository extends
                JpaRepository<Book, Integer> {
```

Expose all DELETE, INSERT, UPDATE actions

```
public interface BookRepository extends
                JpaRepository<Book, Integer> {

    @Override
    @RestResource(exported = false)
    void deleteById(Integer id);
```

DELETE /books/1 → 405 Method not allowed

```
@RepositoryRestResource(exported = false)
public interface BookRepository extends
                JpaRepository<Book, Integer> {
```

Detection strategies

Strategy	Description
Default	Export all public repositories except marked as non-exported
All	Export all repositories
Annotation	Exports repositories annotated as @RestResource or @RepositoryRestResource with exported=true
Visibility	Exports all public repositories

Spring Data REST. Configuration


```
@RepositoryRestResource(path = "book-items",
                        itemResourceRel = "book-item")
public interface BookRepository extends
                    JpaRepository<Book, Integer> {
```

```
@Override
@RestResource(rel = "remove")
void deleteById(Integer id);
```

Spring Data REST. Configuration


```yaml
spring:
  data:
    rest:
      base-path: /api
      default-media-type:
        type: application/json
        charset: UTF-16
      detection-strategy: all
      max-page-size: 20
      return-body-on-create: false
      return-body-on-update: false
      page-param-name: page_size
```

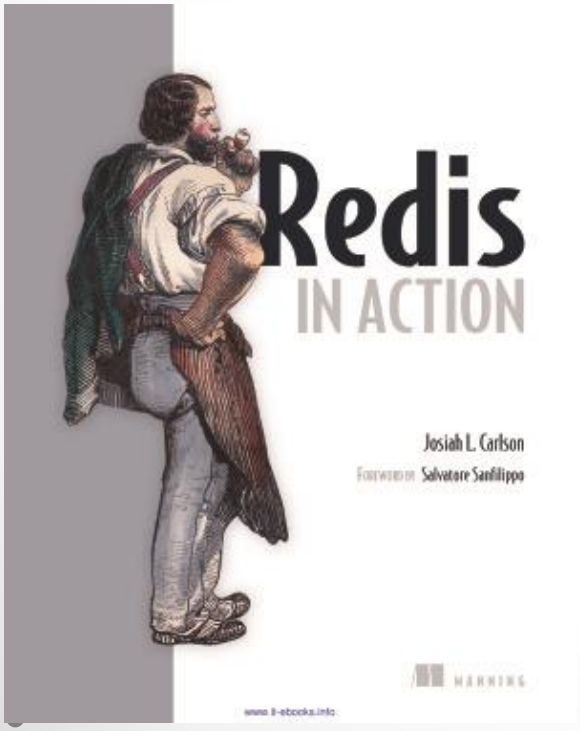
Or application/hal+json

application.yml

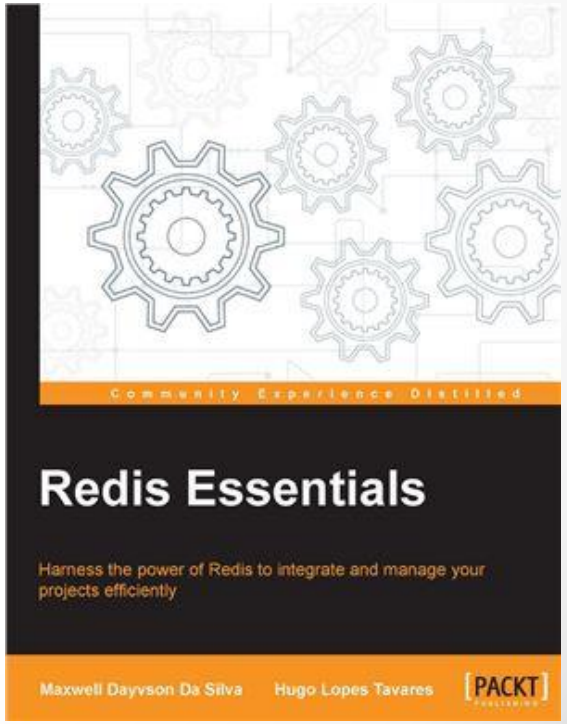
Task 12. Spring Data REST

1. Add **Spring Data REST** dependency:
2. Review existing Spring Data repositories. Which would you consider to expose using Spring Data REST?
3. Try to customize Spring Data REST behavior using **@RepositoryRestResource** and @RestResource annotation
4. Run application and verify behavior of new REST-services
5. (**Optional**) Add integration tests for new REST-services

Books


Redis
IN ACTION
Josiah L. Carlson
Foreword by Salvatore Sanfilippo
MANNING


Practical Techniques for Fast Data Manipulation
Redis
Cookbook
O'REILLY
Tiago Macedo & Fred Oliveira


Community Experience Distilled
Redis Essentials
Harness the power of Redis to integrate and manage your projects efficiently
Maxwell Dayvson Da Silva
Hugo Lopes Tavares
PACKT

✓ Sergiy Morenets, sergey.morenets@gmail.com