DROPWIZARD AND SPRING BOOT

REVIEW AND COMPARISON

DROPWIZARD IS A JAVA FRAMEWORK FOR DEVELOPING HIGH-PERFORMANCE, RESTFUL WEB SERVICES.

Dropwizard - Highlights:

- Allows you to build one jar, that contains all needed dependencies (application has one main program which starts the jetty container)
- Simple & Lightweight
- Quick and easy to get a new http service going
- Easy Test, Deployment and Management
- Quick Project Bootstrap

DROPWIZARD CORE

Jetty – Standalone HTTP-server
Jersey – RESTful web framework
Jackson – JSON prcessing
Metrics – application metrics
Google Guava – utilities
Logback & SLF4J – logging framework
Liquibase – database migrations
JDBI, Hibernate – database access
Hibernate Validator – the reference implementation of the Java Bean Validation standard
Joda Time, Freemaker, Mustache, Jersey Client

DEPLOYMENT

1. Build executable fat JAR
2. Run database migration (if any)
3. Start application (embedded Jetty server)

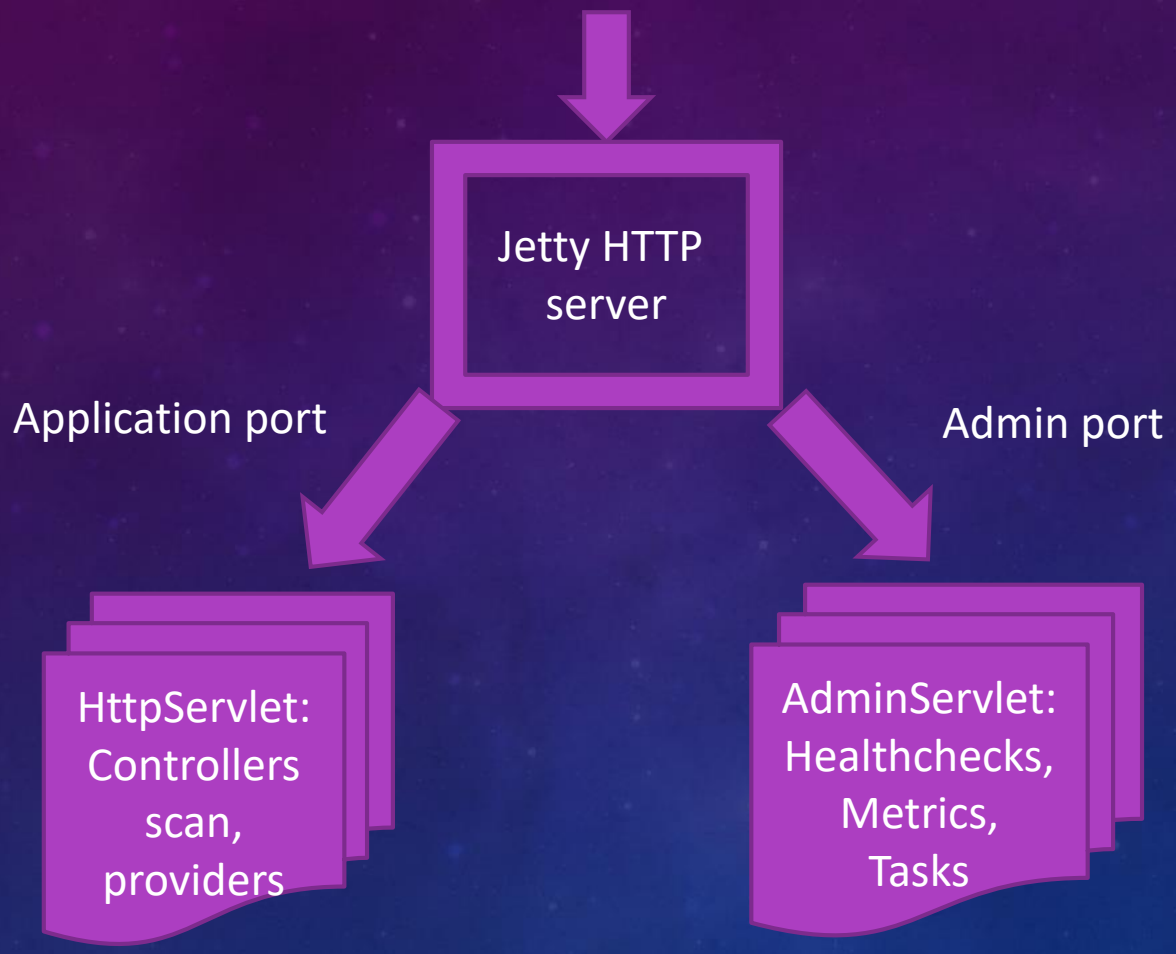
```
usage: java -jar some_service.jar [-h] [-v] {server, check, db} ...

positional arguments:
  {server, check, db}     available commands

named arguments:
  -h, --help              show this help message and exit
  -v, --version           show the application version and exit
```

BOOTSTRAP

Start up$ java –jar app.java server config.yml



PERFORMANCE

JSON serialization | Single query | Multiple queries | Fortunes | Data updates | Plaintext | Physical | Cloud

JSON serialization

Best (bar chart) | Data table | Latency | Framework overhead

Best JSON responses per second, Dell R720xd dual-Xeon E5 v2 + 10 GbE (17 tests)

Rnk	Framework	Best performance (higher is better)		Cls	Lng	Plt	FE	Aos	IA	Errors
1	gemini	914,749	100.0% (86.5%)	Ful	Jav	Svt	Res	Lin	Rea	0
2	undertow	906,076	99.1% (85.7%)	Plt	Jav	Utw	Non	Lin	Rea	0
3	netty	866,318	94.7% (81.9%)	Plt	Jav	Nty	Non	Lin	Rea	0
4	servlet	831,515	90.9% (78.6%)	Plt	Jav	Svt	Res	Lin	Rea	0
5	grizzly	731,583	80.0% (69.2%)	Mcr	Jav	Svt	Grz	Lin	Rea	0
6	wicket	344,032	37.6% (32.5%)	Ful	Jav	Svt	Res	Lin	Rea	0
7	spark	254,111	27.8% (24.0%)	Mcr	Jav	Svt	Res	Lin	Rea	0
8	jetty-servlet	245,709	26.9% (23.2%)	Plt	Jav	Jty	Non	Lin	Rea	0
9	restexpress	225,445	24.6% (21.3%)	Mcr	Jav	Nty	Non	Lin	Rea	0
10	activeweb	220,022	24.1% (20.8%)	Ful	Jav	Svt	Non	Lin	Rea	0
11	tapestry	205,351	22.4% (19.4%)	Ful	Jav	Svt	Res	Lin	Rea	0
12	dropwizard	189,934	20.8% (18.0%)	Ful	Jav	JAX	Jty	Lin	Rea	0
13	grizzly-jersey	176,523	19.3% (16.7%)	Mcr	Jav	JAX	Grz	Lin	Rea	0
14	spring	70,874	7.7% (6.7%)	Ful	Jav	tom	Non	Lin	Rea	0
15	ninja-standalone	35,124	3.8% (3.3%)	Ful	Jav	Jty	Non	Lin	Rea	0
16	play1	17,632	1.9% (1.7%)	Ful	Jav	Nty	Non	Lin	Rea	0

https://www.techempower.com/benchmarks/#section=data-r9&hw=ph&test=json&l=hra0e7

Best (bar chart) | Data table | Latency | Framework overhead

Best database-access responses per second, single query, Dell R720xd dual-Xeon E5 v2 + 10 GbE (10 tests)

Rnk	Framework	Best performance (higher is better)		Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA	Errors
1	gemini	224,112	100.0%	Ful	Jav	Svt	Res	Lin	Non	Lin	Mcr	Rea	0
2	undertow-postgresql	95,592	42.7%	Plt	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea	0
3	dropwizard	83,333	37.2%	Ful	Jav	JAX	Jty	Lin	My	Lin	Ful	Rea	0
4	tapestry	79,278	35.4%	Ful	Jav	Svt	Res	Lin	My	Lin	Ful	Rea	0
5	spark	55,938	25.0%	Mcr	Jav	Svt	Res	Lin	My	Lin	Ful	Rea	0
6	grizzly-jersey	54,150	24.2%	Mcr	Jav	JAX	Grz	Lin	My	Lin	Ful	Rea	0
7	ninja-standalone	32,768	14.6%	Ful	Jav	Jty	Non	Lin	My	Lin	Ful	Rea	0
8	spring	18,950	8.5%	Ful	Jav	tom	Non	Lin	Pg	Lin	Mcr	Rea	0
9	restexpress-mysql-ra	2,333	1.0%	Mcr	Jav	Nty	Non	Lin	My	Lin	Raw	Rea	4,901

https://www.techempower.com/benchmarks/#section=data-r9&hw=ph&test=db&l=hra0e7&f=2ups-35s-0-0-0-0-0-0

Test types | Hardware

JSON serialization | Single query | Multiple queries | Fortunes | Data updates | Plaintext | Physical | Cloud

Multiple queries

20-queries (bar) | Data table | Latency | Framework overhead

Responses per second at 20 queries per request, Dell R720xd dual-Xeon E5 v2 + 10 GbE (10 tests)

Rnk	Framework	Performance (higher is better)		Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA	Errors
1	dropwizard	11,270	100.0%	Ful	Jav	JAX	Jty	Lin	My	Lin	Ful	Rea	0
2	tapestry	10,948	97.1%	Ful	Jav	Svt	Res	Lin	My	Lin	Ful	Rea	0
3	gemini	10,675	94.7%	Ful	Jav	Svt	Res	Lin	Non	Lin	Mcr	Rea	0
4	ninja-standalone	10,412	92.4%	Ful	Jav	Jty	Non	Lin	My	Lin	Ful	Rea	0
5	undertow-postgresql	6,882	61.1%	Plt	Jav	Utw	Non	Lin	Pg	Lin	Raw	Rea	0
6	grizzly-jersey	6,333	56.2%	Mcr	Jav	JAX	Grz	Lin	My	Lin	Ful	Rea	0
7	restexpress-mysql-ra	1,551	13.8%	Mcr	Jav	Nty	Non	Lin	My	Lin	Raw	Rea	8,087
8	spring	1,416	12.6%	Ful	Jav	tom	Non	Lin	Pg	Lin	Mcr	Rea	0
9	spark	404	3.6%	Mcr	Jav	Svt	Res	Lin	My	Lin	Ful	Rea	0

https://www.techempower.com/benchmarks/#section=data-r9&hw=ph&test=query&l=hra0e7&f=2ups-35s-0-0-0-0-0-0

OTHERS

Best fortunes (db -> json -> html) responses per second, Dell R440 Xeon Gold + 10 GbE(54 tests):

1. Vertx, 336 138
2. Dropwizard 46 349
3. Spring 30 990

https://www.techempower.com/benchmarks/#section=data-r17&hw=ph&test=fortune&l=hra0e7-0&f=2usnh1-35s-0-0-0-0-0-0-0

Best plaintext responses per second, Dell R440 Xeon Gold + 10 GbE(57 tests)

1. Wizzardo-http 7 026 401
2. Dropwizard 236 960
3. Spring 158 808

https://www.techempower.com/benchmarks/#section=data-r17&hw=ph&test=plaintext&l=hra0e7-0&f=2usnh1-35s-0-0-0-0-0-0-0

Responses per second at 20 updates (update db) per request, Dell R440 Xeon Gold + 10 GbE(38 tests)

1. Vertx 18 272
2. Dropwizard 8 360
3. Spring 2 913

https://www.techempower.com/benchmarks/#section=data-r17&hw=ph&test=update&l=hra0e7-0&f=2usnh1-35s-0-0-0-0-0-0-0

HOW TO START?

PROJECT ORGANIZATION

com.example.myapplication:

- api: Representations. Request and response bodies.
- cli: Commands
- client: Client code that accesses external HTTP services.
- core: Domain implementation; where objects not used in the API such as POJOs, validations, crypto, etc, reside.
- jdbi: Database access classes
- health: Health Checks
- resources: Resources
- MyApplication: The application class
- MyApplicationConfiguration: configuration class

CONFIGURATION CLASS AND CONFIG MAPPING - THESE PARAMETERS

ARE SPECIFIED IN A YAML CONFIGURATION FILE WHICH IS DESERIALIZED TO AN INSTANCE OF YOUR APPLICATION'S CONFIGURATION CLASS AND VALIDATED.

```yaml
template: Hello, %s!
defaultName: defName

server:
  applicationConnectors:
  - type: http
    port: 9000
  adminConnectors:
  - type: http
    port: 9001

database:
  driverClass: com.mysql.jdbc.Driver
  user: root
  password: qweQWE123!@#
  url:
jdbc:mysql://localhost:3306/appcenter?allo
wMultiQueries=true
  validationQuery: "/* Health Check */
SELECT 1"
```

```java
@Data
public class AppConfig extends Configuration
{
    @NotEmpty
    private String template;

    @NotEmpty
    private String defaultName =
"AppConfigDefaultName";

    @Valid
    @NotNull
    private DataSourceFactory database = new
DataSourceFactory();
}
```

SPRING CONFIGURATION

```
server.port = 8081
email = test@hp.com
thread-pool = 10
```

```
@Data
@Validated
//@PropertySource("classpath:my.properties")
@Component
public class GlobalProperties {

    @Value("${thread-pool}")
    private int threadPool;

    @Value("${email}")
    private String email;
}
```

```
spring:
  profiles: test
name: test-YAML
environment: test
somenumber: 100
servers:
  - www.abc.test.com
  - www.xyz.test.com
```

```
@Data
@Configuration
@EnableConfigurationProperties
@ConfigurationProperties
@Validated
@PropertySource("classpath:yml.properties")
public class YAMLConfig {
    private String name;
    private String environment;
    private List<String> servers
    = new ArrayList<>();

    @Max(5)
    private Integer somenumber;
    // standard getters and setters
}
```

APPLICATION CLASS

```java
public class App extends Application<AppConfig> {
    public static void main(String[] args) throws Exception {
        new App().run(args); // path to yaml config
    }

    @Override
    public String getName() {
        return "the-app";
    }

    @Override
    public void initialize(Bootstrap<AppConfig> bootstrap)
    { // bundles
    }

    @Override
    public void run(AppConfig configuration,
                    Environment environment)
    { // set up environment
    }
}
```

APPLICATION CLASS - BUNDLES

Bundles is re-usable functionality

Examples:

- Database migrations bundle
- Hibernate bundle
- Assets bundle

INITIALIZE – ADD A MIGRATION BUNDLE

```
@Override
public void initialize(Bootstrap<AppConfig> bootstrap) {
    bootstrap.addBundle(new MigrationsBundle<AppConfig>() {
        @Override
        public DataSourceFactory getDataSourceFactory(AppConfig configuration) {
            return configuration.getDatabase(); // data source factory
        }
    });
}
```

LIQUIBASE MIGRATION

```xml
<?xml version="1.0" encoding="UTF-8"?>

<databaseChangeLog
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
         http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">

    <changeSet id="1" author="renat">
        <createTable tableName="people">
            <column name="id" type="bigint" autoIncrement="true">
                <constraints primaryKey="true" nullable="false"/>
            </column>
            <column name="fullName" type="varchar(255)">
                <constraints nullable="false"/>
            </column>
            <column name="jobTitle" type="varchar(255)"/>
        </createTable>
    </changeSet>
</databaseChangeLog>
```

src/main/resources/migratinos.xml

APPLICATION CLASS – ENVIRONMENT

```java
@Override
public void run(AppConfig configuration,
                Environment environment)
{
    // create a new DBIFactory
    final DBIFactory factory = new DBIFactory();
    final DBI jdbi = factory.build(environment, configuration.getDatabase(), "mysql"); // DataSourceFactory
    // create daos
    final UserDAO userDAO = jdbi.onDemand(UserDAO.class);

    // create controllers
    final IndexController controller = new IndexController(
            configuration.getTemplate(),
            configuration.getDefaultName(),
            configuration.getPortConfig().getServerPort());
    final UserController userController = new UserController(userDAO);

    // create health checks
    final TemplateHealthCheck healthCheck = new TemplateHealthCheck(configuration.getTemplate());

    // set up environment
    environment.healthChecks().register("template", healthCheck);
    environment.jersey().register(controller);
    environment.jersey().register(userController);
}
```

SPRING BOOT APPLICATION

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

Migration:

```
spring.liquibase.change-log=classpath:db/liquibase-changelog.xml
```

Database settings:

```
spring.jpa.hibernate.ddl-auto=none
spring.datasource.url=jdbc:mysql://localhost:3306/appcenter?useLegacyDatetimeCode=false&serverTi
mezone=UTC
spring.datasource.username=root
spring.datasource.password=qweQWE123!@#
```

HEALTH CHECKS

Runtime test of behavior, initialization happens in the run method

```
public class TemplateHealthCheck extends HealthCheck {
    private final String template;

    public TemplateHealthCheck(String template) {
        this.template = template;
    }

    @Override
    protected Result check() throws Exception {
        final String saying = String.format(template, "TEST");
        if (!saying.contains("TEST")) {
            return Result.unhealthy("template doesn't include a name");
        }
        return Result.healthy();
    }
}
```

REPRESENTATION

REPRESENTATION (POJO)

```
@Entity
@Table(name = "people")
public class PeopleTable {

    @Id
    @Column(name = "id", nullable = false)
    @NotNull
    @JsonProperty
    private Integer id;

    @Column(name = "fullName")
    @NotNull
    @JsonProperty
    private String fullName;

    @Column(name = "jobTitle")
    @NotNull
    @JsonProperty
    private String jobTitle;
```

REPRESENTATION (ADVANCED JSON)

```java
@JsonSnakeCase
public class Person {
    private final String firstName;

    @JsonCreator
    public Person(@JsonProperty String firstName) {
        this.firstName = firstName;
    }

    @JsonProperty
    public String getFirstName() {
        return firstName;
    }
}
```

```
{
"first_name":"camelCaseTo_snake_case"
}
```

DAO (HIBERNATE)

```
public class PeopleDAO extends
AbstractDAO<PeopleTable> {
    public PeopleDAO(SessionFactory factory) {
        super(factory);
    }

    public PeopleTable findById(Integer id) {
        return get(id);
    }

    public void create(PeopleTable
peopleTable) {
        persist(peopleTable);
    }

    public List<PeopleTable> findAll() {
        return list(

namedQuery("com.wiza.representation.PeopleTabl
e.findAll"));
    }
}
```

DAO (JDBI)

```
public interface UserDAO {
    @SqlQuery("select id from user_info
where user_email = :email")
    Integer
getUserIdByEmail(@Bind("email") String
email);
}
```

SPRING DATA JPA

```
public
interface PeopleRepository
extends
CrudRepository<People, Integer>
{
}
```

RESOURCE CLASSES (CONTROLLER)

Supports JAX-RS, @GET, @POST, @UnitOfWork, @Path
Dropwizard adds some additional features:
- Validation with @Valid
- Metrics supports @Timed

```
@Path("/user")
@Produces(MediaType.APPLICATION_JSON)
public class UserController {
    UserDAO userDAO;
    private final AtomicLong counter = new AtomicLong();

    public UserController(UserDAO userDAO) {
        this.userDAO = userDAO;
    }

    @GET
    @Timed
    @UnitOfWork
    public MessageDto getId(@QueryParam("email") Optional<String> email) {
        Integer userId = userDAO.getUserIdByEmail(email.orElse("renat.ashirbakiev@hp.com"));
        return new MessageDto(userId.toString(), counter.getAndIncrement());
    }
}
```

RESOURCES (CONTROLLERS) – ERROR HANDLING

If your resource class unintentionally throws an exception, Dropwizard will log that exception under the ERROR level (including stack traces) and return a terse, safe application/json 500 Internal Server Error response

ExceptionMapper allows take exceptions that your resources may throw and map them to appropriate responses.

```
public class IllegalArgumentExceptionMapper implements
ExceptionMapper<IllegalArgumentException> {
    private final Meter exceptions;
    public IllegalArgumentExceptionMapper(MetricRegistry metrics) {
        exceptions = metrics.meter(name(getClass(), "exceptions"));
    }
    @Override
    public Response toResponse(IllegalArgumentException e) {
        exceptions.mark();
        return Response.status(Response.Status.BAD_REQUEST)
                .header("some_header", "true")
                .type(MediaType.APPLICATION_JSON_TYPE)
                .entity(new ErrorMessage(Response.Status.BAD_REQUEST.getStatusCode(),
                        "You passed an illegal argument! Watch out!"))
                .build();
    }
}
```

```
environment.jersey().register(new IllegalArgumentExceptionMapper(environment.metrics()));
```

SPRING ERROR HANDLING

```
@ControllerAdvice
public class AppExceptionHandler extends ResponseEntityExceptionHandler
{

    @ExceptionHandler({IllegalArgumentException.class})
    public ResponseEntity<Object> handleIllegalArgumentException(
        Exception ex,
        WebRequest request)
    {
         return new ResponseEntity<Object>(
                  "Illegal argument" ,
                   new HttpHeaders(),
                  HttpStatus.BAD_REQUEST);
    }
}
```

RESOURCES (CONTROLLERS) – JERSEY FILTERS

There might be cases when you want to filter out requests or modify them before they reach your controllers.

```
@Provider
public class DateNotSpecifiedFilter implements ContainerRequestFilter {
    @Override
    public void filter(ContainerRequestContext requestContext) throws IOException {
        String dateHeader = requestContext.getHeaderString(HttpHeaders.DATE);
        if (dateHeader == null) {
            throw new CommonException("Date Header was not specified");
        }
    }
}
```

Another way to create filters is by creating servlet filters.

Special annotation should be used in order to filter specific endpoints.

SPRING BOOT FILTER

```
@Component
@Order(1)
public class RequestResponseLoggingFilter implements Filter {
    @Override
    public void doFilter(final ServletRequest request, final ServletResponse response, final
FilterChain chain)
            throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;
        LOG.info("Logging Request  {} : {}", req.getMethod(), req.getRequestURI());
        chain.doFilter(request, response);
        LOG.info("Logging Response :{}", res.getContentType());
    }
}
```

Filter for group of endpoints

```
@Bean
public FilterRegistrationBean<RequestResponseLoggingFilter> loggingFilter(){
    FilterRegistrationBean<RequestResponseLoggingFilter> registrationBean
      = new FilterRegistrationBean<>();
    registrationBean.setFilter(new RequestResponseLoggingFilter());
    registrationBean.addUrlPatterns("/users/*");
    return registrationBean;
}
```

VALIDATION

RESOURCES (CONTROLLERS) – VALIDATORS

```java
@GET
public String
validate(@NotEmpty
@QueryParam("v") String v) {
    return "Got: " + v;
}
```

V == null

```
422: {"errors":["query param v may not be empty"]}
```

```java
@PUT
public Person person(@NotNull @Valid
Person person) {
    return person;
}
```

Person == null

```
422: {"errors": ["The request body may not be
null"]}
```

```java
public class Person {
    @NotNull
    private String firstName;
    public Person() {
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String
firstName) {
        this.firstName = firstName;
    }
}
```

firstName == null

```
422: {"errors": ["firstName may not be null"]}
```

VALUE CONSTRAINS (VALIDATIONS)

```java
@GET
@Path("/max")
public Integer max(@QueryParam("m")
@Max(256) Integer m) {
    return m;
}
```

```java
@GET
@Path("len")
public String len(@QueryParam("l")
@Length(max = 5) String len) {
    return  len;
}
```

```java
@GET
@Path("params")
public String getBean(@Valid @BeanParam MyBeanParams
params) {
    return params.getField();
}
public static class MyBeanParams {
    @NotEmpty
    private String field;
    public String getField() {
        return field;
    }
    @QueryParam("foo")
    public void setField(String field) {
        this.field = Strings.nullToEmpty(field).trim();
    }
}
```

```java
@GET
@Path("oneof")
public String oneOf(@QueryParam("e") @OneOf(value = {"in", "out"}, ignoreCase = true,
ignoreWhitespace = true) String in) {
    return in;
}
```

ANNOTATIONS IN REPRESENTATION CLASS

```
@NotNull
private Integer id;

@NotBlank @Length(min=2, max=255)
private String firstName;

@NotBlank @Length(min=2, max=255)
private String lastName;

@Pattern(regexp=".+@.+\\.[a-z]+")
private String email;
```

LIST OF VALIDATORS

Spring:
- DecimalMax
- DecimalMin
- Digits
- Email
- Future
- FutureOrPresent
- Max
- Min
- Negative
- NegativeOrZero
- NotBlank
- NotEmpty
- NotNull
- Null
- Past
- PastOrPresent
- Pattern
- Positive
- PositiveOrZero

Dropwizard:
SizeRange
PortRange
OneOf
MixSize
MinDuration
MaxSize
MaxDuration

@NotBlank
@NotEmpty
@Range(min=,max=)
@SafeHtml
@ScriptAssert
@URL
@Mod11Check

Hibernate:
@AssertFalse
@AssertTrue
@DecimalMax(value=,inclusive=)
@DecimalMin(value=,inclusive=)
@Digits(integer=,fraction=)
@Future
@Max(value=)
@Min(value=)
@NotNull
@Null
@Past
@Pattern(regex=,flag=)
@Size(min=, max=)
@Valid
@CreditCardNumber(ignoreNonDigitCharacters=)
@EAN
@Email
@Length(min=,max=)
@LuhnCheck(startIndex=,endIndex=,checkDigitIndex=,ignoreNonDigitCharacters=)
@Mod10Check(multiplier=,weight=,startIndex=,endIndex=,checkDigitIndex=,ignoreNonDigitCharacters=)

APPLICATION CHECKING

Operational Menu

- Metrics
- Ping
- Threads
- Healthcheck
- CPU Profile
- CPU Contention

Works on admin port
/
/ping
/healthcheck
/metrics
/threads

```
Spring actuators:
http://localhost:8081/actuator
http://localhost:8081/actuator/auditevents
http://localhost:8081/actuator/beans
http://localhost:8081/actuator/caches/{cache}
http://localhost:8081/actuator/caches
http://localhost:8081/actuator/health
http://localhost:8081/actuator/health/{component}/{instance}
http://localhost:8081/actuator/health/{component}
http://localhost:8081/actuator/conditions
http://localhost:8081/actuator/configprops
http://localhost:8081/actuator/env
http://localhost:8081/actuator/env/{toMatch}
http://localhost:8081/actuator/info
http://localhost:8081/actuator/liquibase
http://localhost:8081/actuator/loggers
http://localhost:8081/actuator/loggers/{name}
http://localhost:8081/actuator/heapdump
http://localhost:8081/actuator/threaddump
http://localhost:8081/actuator/metrics/{requiredMetricName}
http://localhost:8081/actuator/metrics
http://localhost:8081/actuator/scheduledtasks
http://localhost:8081/actuator/httptrace
http://localhost:8081/actuator/mappings
```

OTHER FEATURES

1. **Environment variables** – dropwizard-configuration module provides the capabilities to substitute configuration settings with the value of environment variables
2. **SSL** support is built into Dropwizard
3. **Managed objects** –ties object's lifecycle to that of the application's HTTP server (before the server starts, the start() method is called, after the server has stopped the stop() method is called).
4. **Commands** – are basic actions which Dropwizard runs based on the arguments provided on the command line (server, db migrate)
5. **Task** – is a run-time action your application provides access to on the administrative port via HTTP.
6. **Logging** – Dropwizard can log to console, file ans syslog. DEBUG, INFO, WARN, ERROR could be written to different output. The output can be in JSON format.
7. **Testing Applications - a**ll of Dropwizard's APIs are designed with testability
8. **Dropwizard client**
9. **Proxy Authentication**
10. **Dropwizard Authentication** – provides authentication using either HTTP Basic Authentication or OAuth2 bearer tokens.
11. **Dropwizard Forms** - module provides you with a support for multi-part forms Jersey.
12. **Dropwizard validation –** NotNull, UnwrapValidatedValue, Max, DefaultValue, BeanParam, one of, ValidationMethod, Length
13. **Dropwizard Views –** dropwizard-views-mustache & dropwizard-views-freemarker modules provide you with simple, fast HTML views**.**
14. **Dropwizard & Scala**
15. **Testing Dropwizard** – provides you with some handy classes for testing your representation classes and resource classes. It also provides a JUnit rule for full-stack testing of your entire app

LOGGING

```yaml
logging:
  # The default level of all loggers. Can be OFF, ERROR, WARN, INFO, DEBUG, TRACE, or ALL.
  level: ALL
  appenders:
    - type: file # syslog, console
      currentLogFilename: ./logs/example-sql.log
      archivedLogFilenamePattern: ./logs/example-sql-%d.log.gz
      archivedFileCount: 5
```

CONSOLE LOGGING

By default, Dropwizard applications log INFO and higher to STDOUT. You can configure this by editing the logging section of your YAML configuration file:

```yaml
logging:
  appenders:
    - type: console
      threshold: WARN
      target: stderr
```

SYSLOG LOGGING

```yaml
logging:
  appenders:
    - type: syslog
      # The hostname of the syslog server to which statements will be sent.
      # N.B.: If this is the local host, the local syslog instance will need to be configured to
      # listen on an inet socket, not just a Unix socket.
      host: localhost
      # The syslog facility to which statements will be sent.
      facility: local0
```

COMBINE ANY NUMBER OF DIFFERENT APPENDERS

```yaml
logging:
# Permit DEBUG, INFO, WARN and ERROR messages to be logged by appenders.
  level: DEBUG
  appenders:
  # Log warnings and errors to stderr
    - type: console
      threshold: WARN
      target: stderr
  # Log info, warnings and errors to our apps' main log.
  # Rolled over daily and retained for 5 days.
    - type: file
      threshold: INFO
      currentLogFilename: ./logs/example.log
      archivedLogFilenamePattern: ./logs/example-%d.log.gz
      archivedFileCount: 5

  # Log debug messages, info, warnings and errors to our apps' debug log.
  # Rolled over hourly and retained for 6 hours
    - type: file
      threshold: DEBUG
      currentLogFilename: ./logs/debug.log
      archivedLogFilenamePattern: ./logs/debug-%d{yyyy-MM-dd-hh}.log.gz
      archivedFileCount: 6
```

JSON LOG FORMAT

```yaml
logging:
  appenders:
    - type: console
      layout:
        type: json
```


```
{
  "level":"INFO",
  "logger":"io.dropwizard.setup.AdminEnvironment",
  "thread":"main",
  "message":"tasks = \r\n\r\n    POST    /tasks/log-level (io.dropwizard.servlets.tasks.LogConfigurationTask)\r\n
 POST    /tasks/gc (io.dropwizard.servlets.tasks.GarbageCollectionTask)\r\n",
  "timestamp":1541062017925
}
{
  "level":"INFO",
  "logger":"org.eclipse.jetty.server.handler.ContextHandler",
  "thread":"main",
  "message":"Started i.d.j.MutableServletContextHandler@2b9ecd05{/,null,AVAILABLE}",
  "timestamp":1541062017931
}
```

ACCESS LOG

```yaml
server:
  rootPath: /api/
  applicationConnectors:
  - type: http
    port: 9000
  adminConnectors:
  - type: http
    port: 9001
  requestLog:
    appenders:
      - type: console
        layout:
          type: access-json
```


```
{
  "method":"GET",
  "userAgent":"Mozilla/5.0 (Windows NT 10.0; W
in64; x64) AppleWebKit/537.36 (KHTML, like Gec
ko) Chrome/70.0.3538.77 Safari/537.36",
  "uri":"/api/user",
  "requestTime":14,
  "protocol":"HTTP/1.1",
  "contentLength":22,
  "remoteAddress":"0:0:0:0:0:0:0:1",
  "timestamp":1541062167713,
  "status":200
}
```

LOGGING FILTERS

- Only log requests that have large bodies
- Only log requests that are slow
- Only log requests that resulted in a non-2xx status code
- Exclude requests that contain sensitive information in the URL
- Exclude healthcheck requests

```yaml
server:
  requestLog:
    appenders:
      - type: console
        filterFactories:
          - type: secret-filter-factory
```

SPRING BOOT LOGGING

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <property name="LOGS" value="./logs" />

    <appender name="Console"
              class="ch.qos.logback.core.ConsoleAppender">
        <layout class="ch.qos.logback.classic.PatternLayout">
            <Pattern>
                %black(%d{ISO8601}) %highlight(%-5level) [%blue(%t)] %yellow(%C{1.}):
%msg%n%throwable
            </Pattern>
        </layout>
    </appender>

    <appender name="RollingFile"
              class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${LOGS}/spring-boot-logger.log</file>
        <encoder
                class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <Pattern>%d %p %C{1.} [%t] %m%n</Pattern>
        </encoder>

        <rollingPolicy
                class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- rollover daily and when the file reaches 10 MegaBytes -->
            <fileNamePattern>${LOGS}/archived/spring-boot-logger-%d{yyyy-MM-dd}.%i.log
            </fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy
                    class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>10MB</maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
        </rollingPolicy>
    </appender>

    <!-- LOG everything at INFO level -->
    <root level="info">
        <appender-ref ref="RollingFile" />
        <appender-ref ref="Console" />
    </root>

    <!-- LOG "com.baeldung*" at TRACE level -->
    <logger name="com.baeldung" level="trace" additivity="false">
        <appender-ref ref="RollingFile" />
        <appender-ref ref="Console" />
    </logger>

</configuration>
```

DROPWIZARD AND SPRING VIEWS

Dropwizard supports:

- Mustache
- FreeMaker

Spring supports:

- Jsp
- FreeMaker
- Thymeleaf
- Groovy
- Jade
- JMustache

Spring Boot will provide auto-configuration for FreeMaker, Thymeleaf, Groovy

```
<dependency>
    <groupId>io.dropwizard</groupId>
    <artifactId>dropwizard-views-
freemarker</artifactId>
 <version>${dropwizard.version}</version>
</dependency>
```

```
bootstrap.addBundle(new ViewBundle<AppConfig>() {
    @Override
    public Map<String, Map<String, String>>
getViewConfiguration(AppConfig config) {
        return config.getViewRendererConfiguration();
} });
```

```
views:
  freemarker:
    strict_syntax:
true
  mustache:
    cache: false
```

```
public class PersonView extends View {
    private final Person person;

    public PersonView(Person person) {
        super("person.ftl");
        this.person = person;
    }

    public Person getPerson() {
        return person;
    }
}
```

```
<#-- @ftlvariable name=""
type="com.wiza.view.PersonView" -->
<html>
<body>
<!-- calls getPerson().getName() and
sanitizes it -->
<h1>Hello, ${person.firstName}!</h1>
</body>
</html>
```

```
@GET
public Person get() {
    Person person = new
Person();

person.setFirstName("fir
stName");
    return person;
}
```

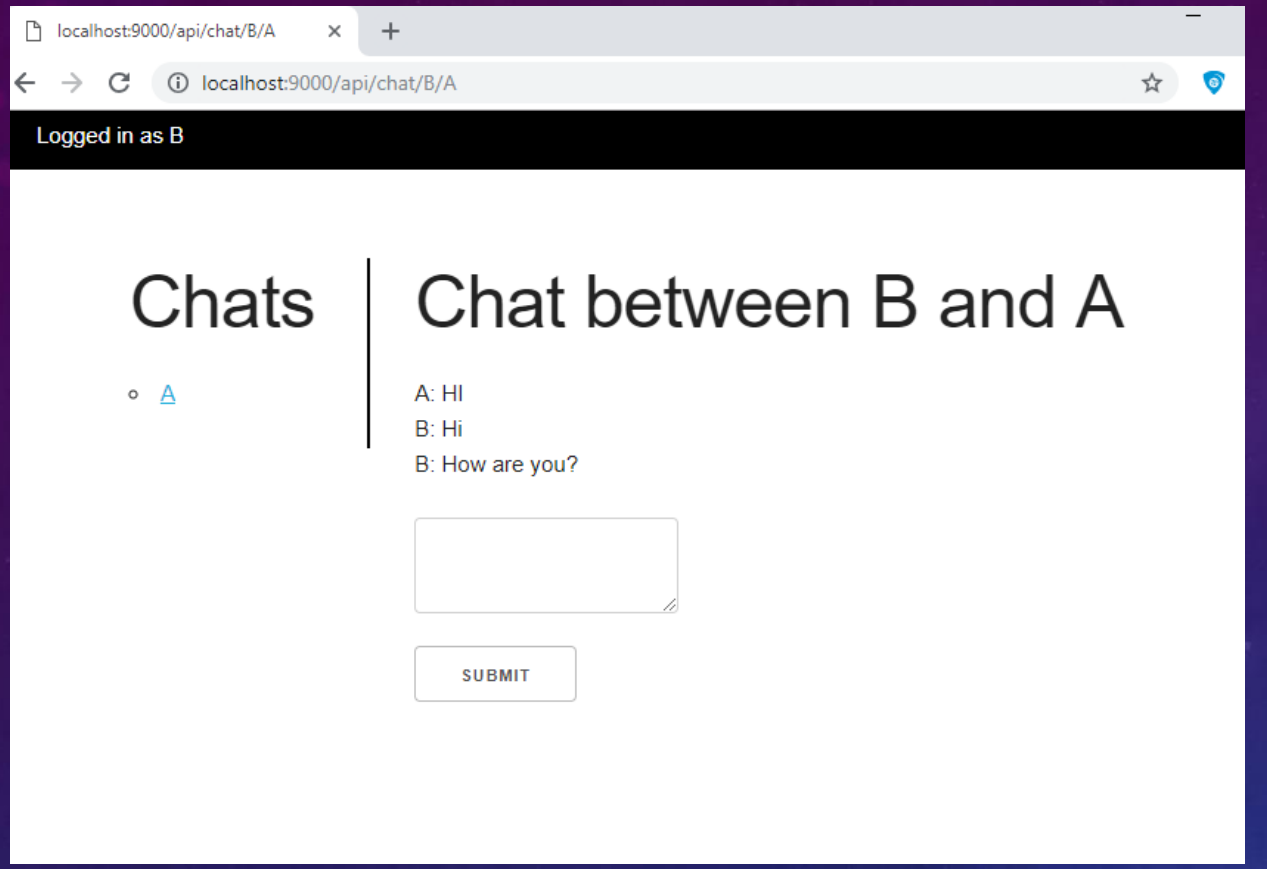
Spring, Thymeleaf:

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

```
<!DOCTYPE HTML>
<html
xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Getting Started: Serving Web
Content</title>
    <meta http-equiv="Content-Type"
content="text/html; charset=UTF-8" />
</head>
<body>
<p th:text="'Hello, ' + ${name} + '!'" />
</body>
</html>
```

```
@Controller
public class GreetingController {
    @GetMapping("/greeting")
    public String
greeting(@RequestParam(name = "name",
required = false, defaultValue =
"World") String name, Model model) {
        model.addAttribute("name",
name);
        return "greeting";
    }
}
```

MUSTACHE VIEW




```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="/assets/css/skeleton.css"/>
    <link rel="stylesheet" href="/assets/css/main.css"/>
</head>
<body>
<div class="top">
    <div class="user">Logged in as {{currentUser}}</div>
</div>
<div class="container">
<row>
    <div class = "three columns">
        <h1>Chats</h1>
        <ul>
            {{#chats}}<li><a
href="/api/chat/{{userOne}}/{{userTwo}}">{{userTwo}}</a></li>{{/cha
ts}}
        </ul>
    </div>
    <div class = "nine columns">
        {{#currentChat}}
            <h1>Chat between {{currentUser}} and {{otherUser}}</h1>
            {{#chat}}
                {{.}}<br/>
            {{/chat}}
            <br>
            <form method="post"
action="/api/chat/{{currentUser}}/{{otherUser}}">
                <textarea name="message"></textarea>
                <br/><button>Submit</button>
            </form>
        {{/currentChat}}
    </div>
</row>
</div>
</body>
</html>
```

DROPWIZARD AND SPRING TESTING

UNIT TESTS FOR SERIALIZING AND DESERIALIZING REPRESENTATION CLASSES TO AND FROM JSON

```java
public class Person {
    @NotNull
    private String firstName;
    public Person() {
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String
firstName) {
        this.firstName = firstName;
    }
}
```

CTRL + F + T

```java
public class PersonTest {

    private static final ObjectMapper MAPPER = Jackson.newObjectMapper();

    @Test
    public void serializesToJSON() throws Exception {
        final Person person = new Person();
        person.setFirstName("hp");

        final String expected = MAPPER.writeValueAsString(
                MAPPER.readValue(fixture("fixtures/person.json"),
Person.class));


assertThat(MAPPER.writeValueAsString(person)).isEqualTo(expected);
    }
}
```

```java
@Test
public void deserializesFromJSON() throws Exception {
    final Person person = new Person();
    person.setFirstName("hp");

assertThat(MAPPER.readValue(fixture("fixtures/person.json"
), Person.class))
            .isEqualTo(person);
}
```

```java
@RunWith(SpringRunner.class)
@JsonTest
public class JsonSDTest {

    private static People details = new People();
    static {
        details.setId(1);
        details.setFullName("hp");
        details.setJobTitle("hp_test");
    }

    @Autowired
    private JacksonTester<People> json;

    @Test
    public void testSerialize() throws Exception {
        Resource resource = new ClassPathResource("json/people.json");
        String json = null;
        try {
            json = StreamUtils.copyToString( resource.getInputStream(), Charset.defaultCharset());
        } catch (IOException e) {
            e.printStackTrace();
        }

        assertThat(this.json.write(details)).isEqualToJson(json);
    }

    @Test
    public void testDeserialize() throws Exception {
        Resource resource = new ClassPathResource("json/people.json");
        String json = StreamUtils.copyToString( resource.getInputStream(), Charset.defaultCharset());
        assertThat(this.json.parse(json))
                .isEqualTo(details);
    }
}
```

TESTING CONTROLLERS (RESOURCES)

```
// loads a given controller instance in an in-memory Jersey server
@ClassRule
public static final ResourceTestRule resources = ResourceTestRule.builder()
        .addResource(new UserController(dao))
        .build();
```

```
@Test
public void testGetIdMethod() {

    MessageDto expectedMessageDto = new MessageDto(String.valueOf(returnedValue), Long.valueOf(returnedValue));
    assertThat(resources.target("/user")
            .queryParam("email", emailToPath)
            .request().get(MessageDto.class))
            .isEqualTo(expectedMessageDto);

    verify(dao).getUserIdByEmail(emailToPath);
}
```

ANOTHER TEST CONTAINER

```
@ClassRule
public static final ResourceTestRule RULE = ResourceTestRule.builder()
        .setTestContainerFactory(new GrizzlyWebTestContainerFactory())
        .addResource(new ExampleController())
        .build();
```

INTEGRATION TESTING

```
public class IntegrationTest {

    @ClassRule
    public static final DropwizardAppRule<AppConfig> RULE =
            new DropwizardAppRule<AppConfig>(App.class,
ResourceHelpers.resourceFilePath("test.yml"));


    @Test
    public void firstIntegrationTest() {
        Client client = new JerseyClientBuilder(RULE.getEnvironment()).build("test client");

        String response = client.target(
                String.format("http://localhost:%d/api/validator/max?m=100",
RULE.getLocalPort()))
                .request()
                .get(String.class);

        assertThat(response).isEqualTo("100");
    }
}
```

```java
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT) // to start the server with a random port
public class ApplicationTest {

    @Autowired
    private GController controller;

    @LocalServerPort
    private int port;

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    public void contextLoads() throws Exception {
        assertThat(controller).isNotNull();
    }

    @Test
    public void greetingShouldReturnDefaultMessage() throws Exception {
        assertThat(this.restTemplate.getForObject("http://localhost:" + port + "/greeting",
                String.class)).contains("hp");
    }
}
```

TESTING CLIENT IMPLEMENTATIONS

```
public class ValidatorControllerTest {

    @ClassRule
    public static final DropwizardClientRule dropwizard = new DropwizardClientRule(new
ValidatorController());

    @Test
    public void checkMaxConstrains() throws IOException {
        final URL url = new URL(dropwizard.baseUri() + "/validator/max?m=100");
        final String response = new BufferedReader(new InputStreamReader(url.openStream())).readLine();
        assertEquals("100", response);
    }
}
```

The DropwizardClientRule takes care of:

- Creating a simple default configuration.
- Creating a simplistic application.
- Adding a dummy health check to the application to suppress the startup warning.
- Adding your JAX-RS resources (test doubles) to the Dropwizard application.
- Choosing a free random port number (important for running tests in parallel).

TESTING DATABASE INTERACTIONS

```java
public class DatabaseTest {
    @Rule
    public DAOTestRule database =
DAOTestRule.newBuilder().addEntityClass(PeopleTable.class)
.build();

    private PeopleDAO peopleDAO;
    @Before
    public void setUp() {
        peopleDAO = new
PeopleDAO(database.getSessionFactory());
    }
    @Test
    public void createsFoo() {
        Integer userId = 1123;
        PeopleTable peopleTable = new PeopleTable();
        peopleTable.setId(userId);
        peopleTable.setFullName("MrR");
        peopleTable.setJobTitle("developer");
        PeopleTable result = database.inTransaction(() ->
{
            peopleDAO.create(peopleTable);
            return peopleDAO.findById(userId);
        });
        assertEquals(result, peopleTable);
    }
}
```

```java
@RunWith(SpringRunner.class)
@DataJpaTest
public class DatabaseInteractionTest {

    @Autowired
    private TestEntityManager entityManager;

    @Autowired
    private PeopleRepository peopleRepository;

    @Test
    public void findsUserByEmail() {
        People people = new People();
        people.setId(1);
        people.setJobTitle("title");
        people.setFullName("name");
        entityManager.persist(people);
        assertEquals(people,
peopleRepository.findById(1).orElse(null));
    }
}
```

TESTING CONFIGURATIONS

```
public class YamlConfigTest {
    private final ObjectMapper objectMapper = Jackson.newObjectMapper();
    private final Validator validator = Validators.newValidator();
    private final YamlConfigurationFactory<PortConfig> factory =
            new YamlConfigurationFactory<>(PortConfig.class, validator, objectMapper, "ports");

    @Test
    public void testPortConfig() throws Exception {
        final File yml = new File(ResourceHelpers.resourceFilePath("yaml/portconfig.yml"));
        PortConfig pc = factory.build(yml);
        assertThat(pc).isInstanceOf(PortConfig.class);
        assertThat(pc.getServerPort()).isEqualTo(8000);
    }
}
```

CONCLUSION

- REST supports – frameworks are quit similar.
- DI supports – Dropwizard (DW) has only community version dropwizard-guice, but SpringBoot (SB) version works better.
- Persistence supports – DW is tightly coupled with Hibernate and JPA specification is not possible.
- Transaction supports – DW use @UnitOfWork annotation and it can be applied only to methods in REST classes.

In SB @Transactional annotation can be applied on any method or class (not only endpoints).

- Configuration management – DW has not built-in functionality to support multiple environment configs in one file.

SB has its own solution Spring Profiles.

- Metrics – SB provides only basic metrics and stats. SB suggest using DW metrics package for any advanced measurements.
- Supports – SB is sponsored by Pivotal. DW is community.