Contents

[Java Logging 4](#_Toc82017589)

[Components 4](#_Toc82017590)

[Types of Handler 4](#_Toc82017591)

[Formatters 4](#_Toc82017592)

[Logging levels 4](#_Toc82017593)

[Filter 5](#_Toc82017594)

[Logging frameworks 5](#_Toc82017595)

[AWS 6](#_Toc82017596)

[GitLab 7](#_Toc82017597)

[Veracode 7](#_Toc82017598)

[SonarQube 7](#_Toc82017599)

[API Testing 7](#_Toc82017600)

[HttpCLient 7](#_Toc82017601)

[Rest Assured 7](#_Toc82017602)

[Rest Template 8](#_Toc82017603)

[Spring Webclient 8](#_Toc82017604)

[SOLID 9](#_Toc82017605)

[Single Responsibility Principle 9](#_Toc82017606)

[Open closed Principle 9](#_Toc82017607)

[Liskov Substitution Principle 9](#_Toc82017608)

[Interface segregation Principle 9](#_Toc82017609)

[Dependency Injection 9](#_Toc82017610)

[DESIGN PATTERNS 10](#_Toc82017611)

[Creational 10](#_Toc82017612)

[Singleton 10](#_Toc82017613)

[Prototype 10](#_Toc82017614)

[Builder 10](#_Toc82017615)

[Factory 10](#_Toc82017616)

[Abstract Factory 10](#_Toc82017617)

[Object Pool Pattern 10](#_Toc82017618)

[Structural 10](#_Toc82017619)

[Adapter 10](#_Toc82017620)

[Bridge 10](#_Toc82017621)

[Composite 10](#_Toc82017622)

[Decorator 10](#_Toc82017623)

[Façade 11](#_Toc82017624)

[Behavioral 11](#_Toc82017625)

[J2EE Patterns 11](#_Toc82017626)

[Presentation Layer 11](#_Toc82017627)

[Business Layer 11](#_Toc82017628)

[Integration Layer 11](#_Toc82017629)

[REST 12](#_Toc82017630)

[Jersey 12](#_Toc82017631)

[Spring Rest 13](#_Toc82017632)

[ErrorHandling 13](#_Toc82017633)

[Validation 15](#_Toc82017634)

[Testing 15](#_Toc82017635)

[JSON 17](#_Toc82017636)

[HTTP Status codes 18](#_Toc82017637)

[Testing Types 18](#_Toc82017638)

[Smoke or Sanity Tests 18](#_Toc82017639)

[Continuos Testing 18](#_Toc82017640)

[Regression Testing 18](#_Toc82017641)

[Performance Testing 18](#_Toc82017642)

[Acceptance Testing 18](#_Toc82017643)

[Testing Levels 18](#_Toc82017644)

[Testing Frameworks 18](#_Toc82017645)

[Junit 18](#_Toc82017646)

[Spring Test 18](#_Toc82017647)

[AssertJ 19](#_Toc82017648)

[Hamcrest 19](#_Toc82017649)

[JSON Path 19](#_Toc82017650)

[Mockito 19](#_Toc82017651)

[SpringBoot 19](#_Toc82017652)

[Microservices 20](#_Toc82017653)

[Data Consistency 20](#_Toc82017654)

[Two Phase commit Pattern 20](#_Toc82017655)

[Saga Pattern 20](#_Toc82017656)

[Routing Slip Pattern 21](#_Toc82017657)

[Eventual consistency 21](#_Toc82017658)

[Microservice Elements 21](#_Toc82017659)

[Data Store 21](#_Toc82017660)

[User Interface 21](#_Toc82017661)

[Services 21](#_Toc82017662)

[Distributed Services 22](#_Toc82017663)

[Security 22](#_Toc82017664)

[Scalability and Availability 22](#_Toc82017665)

[Monitoring 23](#_Toc82017666)

[Deployment 23](#_Toc82017667)

[Security 24](#_Toc82017668)

[OpenID Connect 24](#_Toc82017669)

[OAuth 2.0 24](#_Toc82017670)

[JWT 24](#_Toc82017671)

[Keycloak 24](#_Toc82017672)

[Kong API 24](#_Toc82017673)

[MultiThreading 24](#_Toc82017674)

[Lifecycle of a Thread 24](#_Toc82017675)

[Creation of Threads 24](#_Toc82017676)

[Thread Scheduler 24](#_Toc82017677)

[Thread Pooling 25](#_Toc82017678)

# Java Logging

## Components

* Logger - object responsible for capturing log records
* Logging Handlers or Appender
* Logging Formatters or Layouts

Types of Handler – custom by extending Handler class

* StreamHandler – to output stream
* ConsoleHandler – console
* FileHandler – file in xml format
* SocketHandler – remote TCP ports
* MemoryHandler – in memory

Formatters – custome by extending Formatter

* SimpleFormatter – (console )text
* XMLFormatter – (file) xml

## Logging levels

* Nine levels , seven standard , two special. Higher values – high prioirities
* FINEST - 300
* FINER - 400
* FINE - 500
* Config - 700
* INFO – 800
* WARNING – 900
* SEVERE – 1000
* OFF – Integer.MAX\_VALUE – turns off
* ALL – Integer.MIN\_VALUE – captures everything

Logger.setLevel(Logger.INFO);

Logger.log(Level.INFO, “message”);

Logger.INFO(“message”)

## Filter

There can be an optional filter, where LogRecord is its parameter to isLoggable,

LOGGER.setFilter(new FilterExample()); - set on a class

Override public boolean isLoggable(LogRecord record) to filter messages

Record has getMessage method

## Logging frameworks

* Log4J
* Log4J2
* SLF4J – Simple Logging Façade for Java
* Logback
* tinyLog
* Preferred , Logback is default in Spring Boot
* Log4J is old ,go for Log4J2
* Logback uses SLF4j
* logback-core, slf4j-api
* logback-classic dependency for runtime
* private static final Logger logger = LoggerFactory.logger(classname)
* configuration file *logback.xml  in classpath*

# AWS

Users

Root User

IAM User

Root user creates user groups and assigns policies

Creates a user and assign user group

EC2 is created using an AMI (image) IaaS(Infrastructure as a Service)

Service roles are automatically created when you need to connect from a service to different service

Code commit – Repository

Code Build – buildspec.yml – build commands

Builds and pushes it to a bucket in S3

Code deploy – deploys the artifact from S3 bucket mentioned to the ec2 instance – deployappspec.yml

Keypair

Tags

Code pipeline – automated – cloudwatch

CloudFormation – json with all the details

# GitLab

Reposirtory as a project

Contains gitlab-ci.yml which takes care of stages and what needs to be done in each for CI/CD

# Veracode

Veracode is a cloud based solution for application security

Static analysis, compiled byte code (SAST – Satic Application security testing)

Dynamic analysis (DAST )

Intercative Analysis (IAST)

Software composition Analysis (SCA) and penetration testing

# SonarQube

Codecoverage tool to check the unit test coverage

apply the Gradle Jacoco plugin to your project and run a SonarQube scan to generate a code coverage report

# API Testing

## HttpCLient

HttpClient is a low level client for simplifying Http Communication

HTTP protocol interception

Secure HTTP connections - SSL/TLS

HTTP proxy server handling

Handles HTTP cookies

Connection pooling for different hosts, keep alive strategy,

multi-threaded request execution

## Rest Assured

simplified testing of REST based services built over HTTP

Validating REST API response using inbuilt Hemcrest Matchers

JSON & XML serialization and deserialization

Extracting JSON data using JsonPath and XML data using XmlPath

Verifying response body, cookies, headers, content-type and http status

Authentication using Basic Auth, Digest Auth, Form Authentication (CSRF support), OAuth (OAuth1 and OAuth2)

verifying multi-part form data

Request and response logging for easy troubleshooting

Session Filters

Spring Mock Mvc Module

Spring Web Test Client Module

Kotlin support

RestAssured.*baseURI* = url + "/" + tid;

StringBuilder authorization = **new** StringBuilder();

authorization.append(usr).append(":").append(pwd);

String authHeader = "Basic " + Base64.*getEncoder*().encodeToString(authorization.toString().getBytes());

RequestSpecification httpRequest = RestAssured.*given*().header("Authorization", authHeader)

.header("Content-Type", "application/json")

.header("Accept", "application/vnd.dsttechnologies.awd+xml");

Response response = httpRequest.request(Method.***GET***);

## Rest Template

RestTemplate is also a high level REST client which uses HttpClient under the hood, but it is mostly used in development rather than testing. It lacks most of the testing related features readily available in REST Assured like - Assertion capabilities - inbuilt Hemcrest matchers support, ease of use from testing perspective, out of the box support for various authentication protocols, ease of logging requests response, measuring request time, etc.

## Spring Webclient

Spring WebClient was introduced in Spring Boot 2 for reactive programming. It has following advantages over conventional RestTemplate:

It is non-blocking & reactive in nature, so you can achieve more throughput with limited threads. This should definitely reduce the hardware requirements.

It has more functional feel. WebClient has a functional, fluent API with reactive types for declarative composition.

It has all the features that RestTemplate has to offer in cloud native environment (Load balancing, token relay, oauth2 handling, etc.)

https://www.javacodemonk.com/rest-assured-vs-apache-httpclient-and-resttemplate-b50fa3b5

# SOLID

## Single Responsibility Principle

a class should only have one responsibility. Furthermore, it should only have one reason to change

Testing, Lower coupling, Organization

## Open closed Principle

Open for Extension, Closed for Modification

classes should be open for extension but closed for modification. In doing so, we stop ourselves from modifying existing code and causing potential new bugs

## Liskov Substitution Principle

if class A is a subtype of class B, we should be able to replace B with A without disrupting the behavior of our program.

## Interface segregation Principle

larger interfaces should be split into smaller ones. By doing so, we can ensure that implementing classes only need to be concerned about the methods that are of interest to them.

## Dependency Injection

The principle of dependency inversion refers to the decoupling of software modules. This way, instead of high-level modules depending on low-level modules, both will depend on abstractions.

DRY – Don’t repeat yourself

# Code Review

## SQL Injection

Use parameterized Queries – Use prepared statement

Use JPA Criteria API

User Data sanitization

* Whitelist – set of valid input checklist
* Blacklist – Not valid input checklist

## Non Functional Requirements

Scalability

Availability

Reliability.

Recoverability.

Maintainability.

Serviceability.

Capacity

# DESIGN PATTERNS

## Creational

### Singleton

### Prototype

### Builder

### Factory

### Abstract Factory

### Object Pool Pattern

## Structural

### Adapter

To connect new code to legacy code without changing the working contract of legacy code

Like a plug adapter

Can have multiple adapters

Convert interface to other interface

Client, adapter, adaptee

Ex: Arrays.asList()

### Bridge

Similar to adapter but bridge works with new code

Decouple abstraction and implementation using encapsulation, composition, inheritance

Changes in abstraction wont affect the client

Details are not available initially

Ex:JDBC drivers

### Composite

Hierarchical type pattern that deals with tree structure of information

Components represent part or whole of structure

Obj in tree structures

Ind object treated as composite

Same operations on individual and composites

Ex: component, java.awt

### Decorator

Hierarchical type pattern with functionality at each level while using composition from similar datatypes

Wrap another object and add functionality to it

Add behavior without affecting others

More than inheritance

SRP

Ex: InputStream

Inheritance based

Utilizes composition and inheritance(is-a and has-a)

### Façade

To overcome poorly designed api

Reduces dependencies on outside code

Simplify client usage

Refactoring pattern

1. Flyweight

To reduce the number of objects created

String

1. Filter
2. Proxy

### Behavioral

1. Chain of responsibility
2. Command
3. Interpreter
4. Iterator
5. Mediator
6. Momento
7. Observer
8. State
9. Strategy
10. Template
11. Visitor
12. Null Object

## J2EE Patterns

### Presentation Layer

1. Intercepting Filter
2. Front controller
3. View Helper
4. Composite View

### Business Layer

1. Business Delegate
2. Service Locator
3. Session Façade
4. Transfer Object

### Integration Layer

1. Data Access Object
2. Web Service Broker
3. MVC
4. Composite Entity

# REST

## Jersey

JAX-RS Implementation – Jersey, RestEasy

Annotations

@Path(“/hello”)

@GET

@Produces(MEDIA\_TYPE.)

@PathParam

@FormParam

@QueryParam

@HeaderParam

@CookieParam

Testing jersey

Org.glassfish.jersey.client.ClientConfig

Javax.ws.rx.client. Client, ClientBuilder, WebTarget, UriBuilder

ClientConfig config = new ClientConfig();

Client client = ClientBuilder.newClient(config)

WebTarget target = client.traget(getBaseUri);

Jersey servlet : org.glassfish.jersey.servlet.ServletContainer

1. **<init-param>**
2. **<param-name>**jersey.config.server.provider.packages**</param-name>**
3. **<param-value>**com.javatpoint.rest**</param-value>**
4. **</init-param>**

Response object building

Response.status(200)  .entity("getDate is called, year/month/day : " + date)    .build();

  ResponseBuilder response = Response.ok((Object) file);

response.header("Content-Disposition","attachment; filename=\"javatpoint\_file.txt\"");

**return** response.build();

# Spring Rest

@RestController - @Controller + @ResponseBody – no need on each method

@GetMapping

@PostMapping – accepts @RequestBody(“/pathname”)

Object create(@RequestBody Object obj)

@PutMapping

@DeleteMapping(“/pathname/{id}”)

Void delete(@PathVariable Integer id)

@RequestParam – optional parameters – required = false

## ErrorHandling

Two types of error messages

HTTP status codes

Text messages

JSON Messages

ErrorMessage classes

Response Entity – can send a happy path and exception path

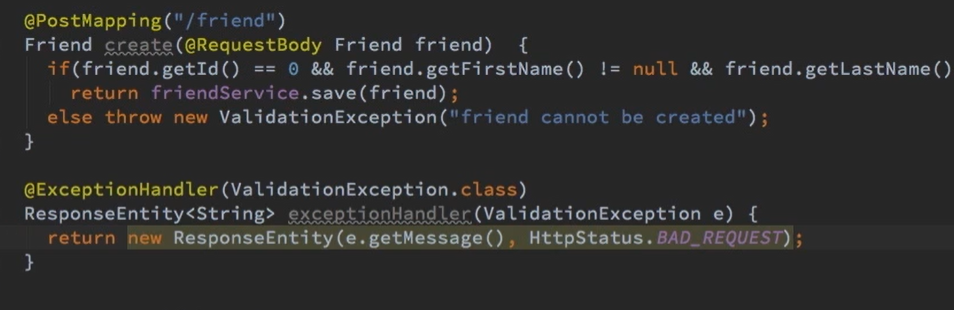
@ExceptionHandler – can handle exceptions – return error message – can use in same class in a different method

When exceptions are routed to a different class, annotate with

@ControllerAdvice – global exception handling – send error message

And has @ExceptionHandler

ResponseEntity(object, HTTPStatus)





## Validation

Validate in model,annotate constraints properties, add @valid on input ,Method ArgumentNotValidException

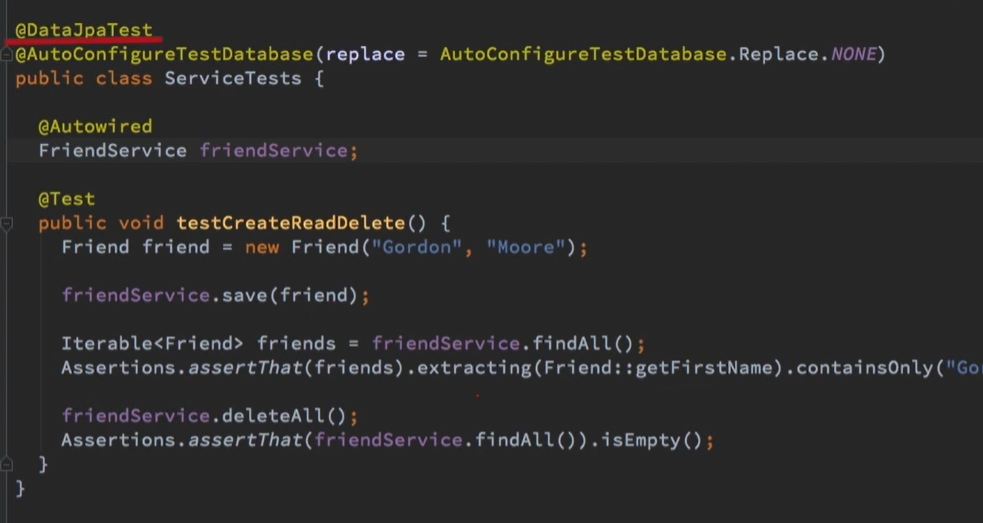
@NotNull,@AssertTreue,@Max,@Min,@size,@Digits,@Pattern,@NotEmpty, @NotBlank(Nowhitespaces),@Positive,@Negative,@Email,@Past,@Future



## Testing

Data Layer testing

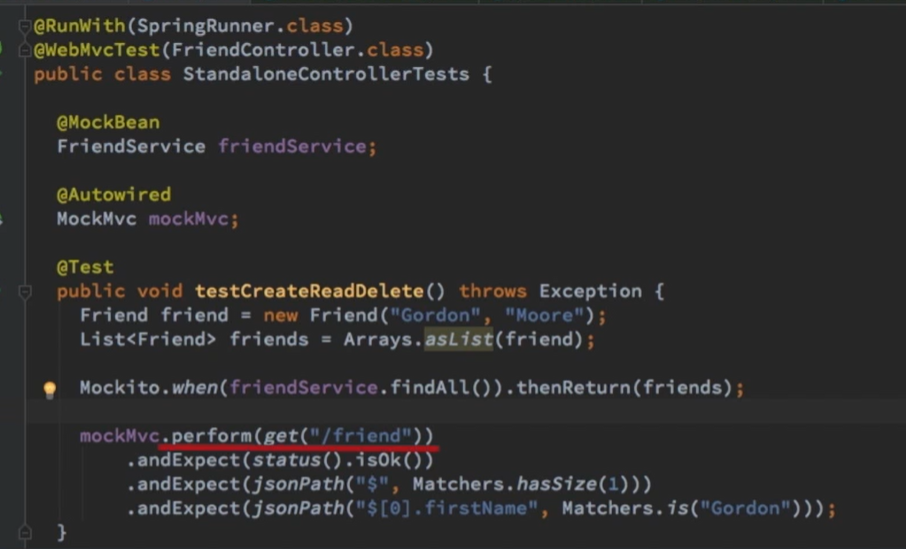
@DataJpaTest



@WebMvcTest on class

@MockBean

Autowire MockMvc



# JSON

@JsonProperty(“First”)

@JsonIgnore

@JsonIgnoreProperties

@JsonInclude(JsonInclude.Include.NON\_NULL)

@JsonManagedReference - parent child relation

@JsonBackReference – child parent relation

@Embeddable in child class and @Embedded in the composing class

For list of objects

@OneToMany(cascade = CascadeType.All)

If @ManyToOne and @OneToMany is used in same class, use @Json..reference

# HTTP Status codes

1XX – Informational

2xx – Success

3xx- Redirection

4xx- Client errors

5xx- Server errors

# Testing Types

## Smoke or Sanity Tests

## Continuos Testing

## Regression Testing

## Performance Testing

## Acceptance Testing

# Testing Levels

Units Tests

Integration Tests

System Testing

# Testing Frameworks

## Junit

## Spring Test

@RunWith(SpringRunner.class)

@SpringBootTest

@Test

## AssertJ

## Hamcrest

Matcher Library

## JSON Path

Xpath for JSOn

## Mockito

Mocking framework

# SpringBoot

@SpringBootApplication

= @EnableAutoConfiguration+ @ComponentScan + @Configuration

Scheduling

@Scheduled(cron = "${schedule.cron}")

@Scheduled(fixedRate = 1000)

@Scheduled(fixedDelay = 1000, initialDelay = 1000)

Kafka

Producer

Consumer

Broker

Cluster

Topic

Partitions

Offset

Consumer groups

# Microservices

## Data Consistency

CAP Theorem – Consistency, Availability, Partition Tolerance

Availability or consistency

ACID- Atomicity, Conistency, Isolation, Durability

## Two Phase commit Pattern

ACID is mandatory

Choosing consistency

Transaction manager manages transaction

Prepare phase

Voting phase

Not in Microservices as of reliability on transaction manager, no voting response and timeout,

Commit failure after successful vote

Pending transactions lock resources

Avoid custom impl

Has scaling out issues

Reduced throughput

Anti pattern

## Saga Pattern

Trading atomicity for availability and consistency

Chooses availability

Splits transaction into many requests called saga

Tracks each request

Centralized distributed transaction state

Using saga log

On fail we send compensate request and rollsback

Implemented by Saga execution coordinator(SEC) – state machine

## Routing Slip Pattern

Manufacturing example

Set of instructions attached to work

Determine process steps

, attach to message, message routed in sequence

No state centralization

## Eventual consistency

BASE – Basic availability, soft state, eventual consistency

## Microservice Elements

Domain driven design

### Data Store

No Distributed transaction – No two phase commit

It will have performance impact

Eventual consistency

Event sourcing – Akka, Kafka, RabbitMQ

Capture Data Change – Debezium

### User Interface

UI aggregation is necessary so user feels it consistent

Server side page composition

Client side composition

### Services

Communication using Remote procedure invocation RPC – REST, SOAP, gRPC

Messaging –

Message or Event

Broker or channel

Publish/Subscribe

Kafka, RabbitMQ

Message format – XML, YAML, JSON(Text) or Binary – gRPC

Exchanges using API’s and contracts – SOAP – WSDL, REST – Swagger, gRPC- IDL

External configuration for diff env’s

Can use DB or Archaius, Consul, Decider

### Distributed Services

**Service Registry** to discover services like a phone book

Each Services registers on start up and deregister during shutdown

Eureka, ZooKeeper, Consul

CORS comes into picture , should use HTTP headers(Access-control-Allow-Origin)

**Circuit breaker** during network issues

Invoke via proxy in order to deviate calls if needed

Hystrix, Jrugged

**Gateway**

For clients to access individual multiple services, and has unified interface

Single entry points

Handles cross cutting concerns – Authentication, Authorization

Ideal place to handle API transaltaion

Zuul, Netty, Finagle

### Security

Identity and Access Management System **IAM**

Provides single sign on

Kerberos, OpenID , Oauth 2.0, SAML - protocols

Okta, Keycloak, Shero

Identity through access token

JWT, Cookie

### Scalability and Availability

Vertical scale – add more power to existing infra – more CPU and RAm

Horizontal – more machines – service replications, clustering – client load balancing

Load balancer – Round robin, based on weight , capacity

Ribbon, Meraki – load balancing tools

### Monitoring

Needs **dashboard**

Kibana, Grafana, Splunk

**Health check – Heart bits**

**Log Aggregation**

Logstash, Splunk, PaperTrial

**Exception Tracking**

Record exceptions in centralized

**Metrics**

DropWizrd, Spring Actuator, Promoetheus

**Auditing**

**Rate Limiting**

Control API usage

Defend DoS attacks – Limit traffic from particular source

Monetize api’s

**Alerting**

**Distributed Tracing**

Trace entire request using correlation id

Dapper, Htrace, Zipkin

### Deployment

Containers

Docker, rkt

Orchestrators

Orchestrate containers – Kunernetes, Docker Swarm, Mesos

Continuous Delivery – Jenkins, Asgard, Aminator

Brown Field Microservices approach

Old monolith to micro

GreenField Microservices

New system or application

# Security

## OpenID Connect

## OAuth 2.0

## JWT

## Keycloak

## Kong API

# MultiThreading

Os contains processes

Process contains Threads

Each process can have only one thread running at a time

## Lifecycle of a Thread

New, Runnable, Running(Non existent), Non Runnable(Block), Terminated

Create a Thread Instance, Start a thread, Picked yp by thread schedular and running, (wait, sleep etc), once run method is executed

## Creation of Threads

Thread can be created in two types

Extend Thread class – M extends Thread have one run method , in main method M a = new M(), a.start()

Implement Runnable Interface

M class implements Runnable, have one run method

M a = new M()

Thread b = new Thread(a), b.start()

## Thread Scheduler

Part of JVM that decides which thread should run

Preemptive scheduling – higher priority task runs until it eneters wait

Time slicing Scheduling – predefined slice of time and gets back to ready, then next thread starts based on prio

## Thread Pooling

If we create many threads , OS may run out of system level threads

Or each thread gets very less time for execution

Thread Pool Pattern helps to save resources in multithreaded app

We write concurrent code in form of parallel tasks and submit execution it to the instance of thread pool

Instance controls the threads to execute these tasks

**Task Submitters** – create and submit tasks

**Executor service** – contains Task Queue and Thread pool, returns a future object for fine tuning the result

Can take a runnable or callable object as input thread for execution

**Executors**, when the fine tuning of result is not required

# Kafka