**Master java**

Chapter 1: Java Basics

<https://www.w3schools.com/java/java_ref_keywords.asp>

https://www.w3resource.com/java-exercises/#google\_vignette

**Environment**

1. Java
2. Maven
3. Git
4. IntelliJ
5. **OOP**

* **Inheritance**

Class can be extended and the subclass can use variable/methods from super class by specifying “super”

Use case: Animal class as a super class that has one common method (sleep method) for every animal. Pig, Cow, Dog class that have only specific methods like sound. They extend Animal class to have access to sleep method.

* **Polymophism**

Happens when there are 2 or more classes related through inheritance.

Use case: In the above use case, Pig, Cow and Dog class are release because they extend the same Animal class.

* **Encapsulation**

This makes sure that sensitive data is hidden from the user by declaring variable as private and writes its setter and getter methods

* **Abstraction**

This is a process of hiding certain details and showing only essential information. You can achieve abstraction by Abstract classes and interface.

|  |  |
| --- | --- |
| Abstract | Interface |
| Can have both abstract (without implementation) and non-abstract (with implementation) methods | Interface does not have implementation |
| Subclass of abstract class can write implementation of the abstract method in the super class by Overriding the method. | A different class must implement the interface and write implementations of the methods in interface by Overriding them. |
| You can access the methods/ variable of the abstract class through its subclass. | You can access through the interface |

1. **Design Patterns/Princiles**

Solid is an acronym for the following five principle

* Single Responsibility

A class or method should have one responsibility. Instead of having one big method with multiple functionalities, we can create multiple small methods that have only one responsibility.

* Open-Closed Principle

The Open-Closed Principle (OCP) states that classes should be open for extension but closed for modification. “Open to extension” means that you should design your classes so that new functionality can be added as new requirements are generated. “Closed for modification” means that once you have developed a class you should never modify it, except to correct bugs.

* Liskov Substitution

Mandates that objects of a superclass can be replaceable with objects of its subclasses without altering the correctness of the program.

* Interface Segregation

No code should be forced to depend on methods it does not use. ISP splits interfaces that are very large into smaller and more specific ones so that clients will only have to know about the methods that are of interest to them.

* Dependency Inversion

High-level modules (Controllers), which provide complex logic, should be easily reusable and unaffected by changes in

low-level modules(Service), which provide utility features.

There should be no dependency between the business logic java classes and the client classes (controllers). These two modules should not be

tightly coupled to each other. Making changes on one of the modules should not force changes on the other module.

We can achieve this by having an interface between the two modules and create objects on the client using dependency enjection.

This way, the control on how dependencies are acquired by client classes no longer resides in these classes. It resides in the

underlying injectors / DI framework(s) (Spring framework) instead.

1. **Constructors**

Constructor in java is used **to create the instance of the class** and can also be used to set variables while creating the object. Constructors are almost similar to methods except for two things - its name is the same as the class name and it has no return type. Sometimes constructors are also referred to as special methods to initialize an object.

1. **Override vs Overload**

|  |  |
| --- | --- |
| Override | Overload |
| Overriding occurs when the method signature is the same in the superclass and the child class. | Overloading occurs when two or more methods in the same class have the same name but different parameters. |
|  | It is used **so that the programmer does not have to remember various function names**. |
|  |  |
|  |  |

1. Collections

Collections are implementation of Data Structures.

|  |  |  |  |
| --- | --- | --- | --- |
| Arrays | Lists | Maps | Sets |
| Fixed size | Not Fixed in size | A value is assigned to a key. |  |
| Specify size or initialize on creation | Implementation: ArrayLists, Stacks  Vector | Keys can not be duplicated and can assign one value in a key. |  |
|  |  | Get value using the key |  |
|  |  |  |  |
|  |  |  |  |

1. **Treads**

Threads allows a program to operate more efficiently by doing multiple things at the same time. Threads can be used to perform complicated tasks in the background without interrupting the main program.

There are two ways to create a thread.

* Extending the Tread class and overriding the run class.
* Implement the Runnable interface and override the run method

**Use case:** One of the most common scenarios where using multiple threads significantly improves an application’s performance is a client-server application. A single-threaded application means only one client can connect to the server at a time, but a multi-threaded server means multiple clients can connect to the server at the same time.

This means the next client doesn’t have to wait until your application finish processing the request of the previous client.

By the way, Threading is not free, it comes with its own challenges. You can only maximize the throughput of your application up to a certain extent, once the numbering of the thread increases up to a certain threshold, they will start competing for CPU, and context switching will occur.

Context switching means one thread that is using CPU is suspended and the CPU is allocated to another thread for execution. When this happens, the thread generally loses all of its cached data. If that thread resumes on another core then it has to build its cache from the start.

Threading also introduces a special set of a problem known as multi-threading issue e.g. deadlock, livelock, memory inconsistency error, race conditions, and starvation.

**Callenge**: Because threads run at the same time as other parts of the program, there is no way to know in which order the code will run. When the threads and main program are reading and writing the same variables, the values are unpredictable. The problems that result from this are called concurrency problems.

**Solution**:  If attributes need to be shared, one possible solution is to use the isAlive() method of the thread to check whether the thread has finished running before using any attributes that the thread can change.

1. **Difference between JVM, JRE, JDK**
2. **JDK** (Java Development Kit) is a Kit that provides the environment to **develop and execute (run)** the Java program. JDK is a kit(or package) that includes two things
3. Development Tools(to provide an environment to develop your java programs)
4. JRE (to execute your java program).
5. **JRE** (Java Runtime Environment) is an installation package that provides an environment to **only run (not develop)** the java program (or application)onto your machine. JRE is only used by those who only want to run Java programs that are end-users of your system.
6. [**JVM** (**Java Virtual Machine)**](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line, hence it is also known as an [**interpreter**](https://www.geeksforgeeks.org/compiler-vs-interpreter-2/)**.**
7. **Garbage collection**

Garbage Collection tracks each and every object available in the JVM heap space, and removes the unused ones.

Basically, GC works in two simple steps, known as Mark and Sweep:

* **Mark –**this is where the garbage collector identifies which pieces of memory are in use and which aren’t.
* **Sweep –**this step removes objects identified during the “mark” phase.

**Advantages:**

* No manual memory allocation/deallocation handling because unused memory space is automatically handled by GC
* No overhead of handling [Dangling Pointer](https://en.wikipedia.org/wiki/Dangling_pointer)
* Automatic [Memory Leak](https://en.wikipedia.org/wiki/Memory_leak) management (GC on its own can’t guarantee the full proof solution to memory leaking; however, it takes care of a good portion of it)

**Disadvantages:**

* Since JVM has to keep track of object reference creation/deletion, this activity requires more CPU power than the original application. It may affect the performance of requests which require large memory.
* Programmers have no control over the scheduling of CPU time dedicated to freeing objects that are no longer needed.
* Using some GC implementations might result in the application stopping unpredictably.
* Automatized memory management won’t be as efficient as the proper manual memory allocation/deallocation.

1. **Data Structures**

They define how data is organized, stored, retrieved, and manipulated within a program.



1. **Arrays**

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

**Array** is a linear data structure that stores a collection of elements of the **same**data type and of same memory size. Each element has a unique **index**number.

Arrays are Static data structure (has a fixed memory size).

* Runtimes and memory

Elements are allocated **contiguous memory**, allowing for **constant-time access**.

* Advantage Disadvantage

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Less memory | Arrays fixed size |
| Faster access of elements ( 0(1) time complexity |  |

* Use case
  + - * 1. Keep data that is likely not change or increase as arrays have a fixed size. Also data that will need less manipulation/ operations as arrays have lesser operations compared bvto arraylist and hash maps.

1. **ArrayList**

The ArrayList class is a resizable [array](https://www.w3schools.com/java/java_arrays.asp), which can be found in the java.util package.

The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an ArrayList whenever you want.

* Runtimes and memory

Whenever an instance of ArrayList in Java is created then by default the capacity of Arraylist is 10. Since ArrayList is a growable array, it automatically resizes itself whenever a number of elements in ArrayList grow beyond a threshold. However, ensureCapacity() method of java. util.

Avoiding Waste. We should note that ArrayList is a good solution for a flexible-sized container of objects that is to support random access. It consumes slightly more memory than an array but provides a richer set of operations.

Array cloning is much faster than ArrayList because array creation is a simpler operation that involves allocating a contiguous block of memory. In contrast, ArrayList creation involves additional overhead, such as initializing internal data structures and dynamically resizing the list as elements are added.

* Advantage Disadvantage

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Size can be changed | Use more memory |
|  | slower access of elements ( 0(n) time complexity |

* Use case

1. Keep Employee names. This data can be increase or decrease.
2. **Linked list**

* Runtimes and memory
* Use case

1. **Hash table**

A Hash Table is a data structure designed to be fast to work with.

The reason Hash Tables are sometimes preferred instead of arrays or linked lists is because searching for, adding, and deleting data can be done really quickly, even for large amounts of data.

In a [Linked List](https://www.w3schools.com/dsa/dsa_theory_linkedlists.php), finding a person "Bob" takes time because we would have to go from one node to the next, checking each node, until the node with "Bob" is found.

And finding "Bob" in an [Array](https://www.w3schools.com/dsa/dsa_data_arrays.php) could be fast if we knew the index, but when we only know the name "Bob", we need to compare each element (like with Linked Lists), and that takes time.

With a Hash Table however, finding "Bob" is done really fast because there is a way to go directly to where "Bob" is stored, using something called a hash function.

The most important reason why Hash Tables are great for these things is that Hash Tables are very fast compared Arrays and Linked Lists, especially for large sets. Arrays and Linked Lists have time complexity O(n)O(n) for search and delete, while Hash Tables have just O(1)O(1) on average! Read more about time complexity [here](https://www.w3schools.com/dsa/dsa_timecomplexity_theory.php).

**Collision:**

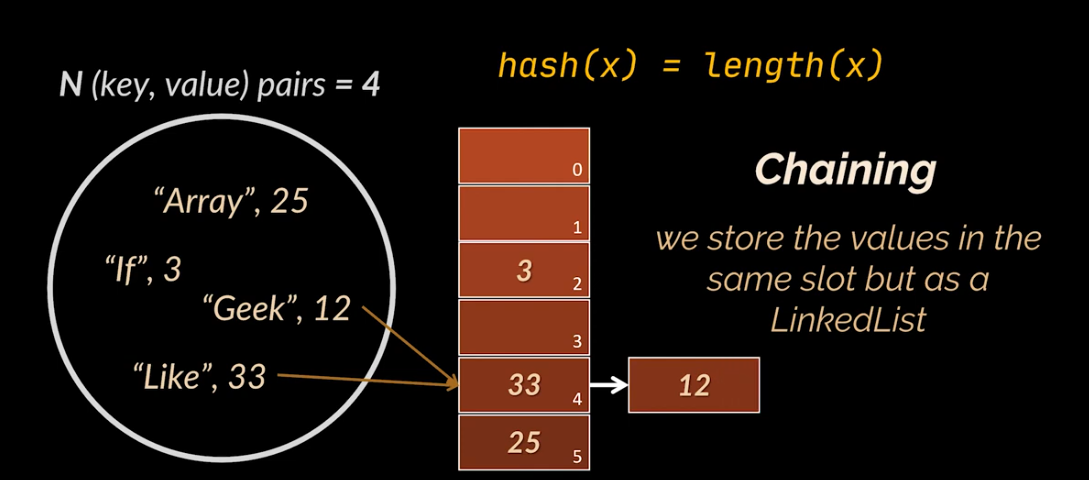
Happens when a new key is given an index that is already occupied by another key. Indexes are calculated by a hash function to determine the position of the new key and value that needs to be added. This is called collision when the index given is already occupied.

**Resolution:**

* Chaining

The new key is given space in the same slot which results in more the one key in the same bucket. This creates a linked list inside the bucket.

Disadvantage of this approach is when you search the hash map, on the bucket where there is multiple keys, it will go through each and every key to search is it exists, just like arraylist. Searching will take longer.



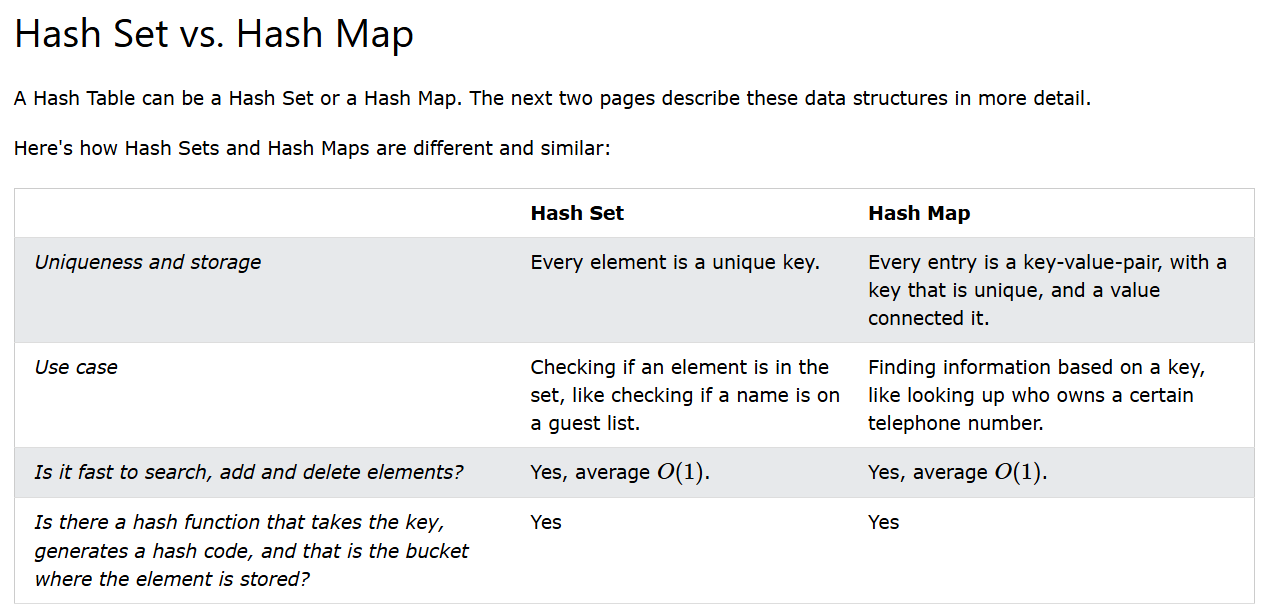
* Open-addressing

Searches another empty or available slot/bucket in the array and place the new key

Linear Probing – searching for free slots in a linear manner

Quadratic Probing

Double Probing



* Runtimes and memory

One might ask, why not simply use a List and get rid of the keys all together? Especially since HashMap consumes more memory for saving keys and its entries are not ordered. The answer lies in the performance benefits for searching elements.

HashMap is very efficient at checking if a key exists or retrieving a value based on a key. Those operations take O(1) on average.

Adding and removing elements from a HashMap based on a key takes O(1) constant-time. Checking for an element without knowing the key takes linear time O(n), as it’s necessary to loop over all the elements.

* Advantage Disadvantage

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Size can be changed | Use more memory (keys and entries) |
| Faster access of elements. 0(1) time complexity. (Don’t have to loop through all elements, can just use containKey or containValue which goes straight to the value). |  |

* Use case
* Checking if something is in a collection (like finding a book ins a library).
* Storing unique items and quickly finding them (like storing phone numbers).
* Connecting values to keys (like linking names to phone numbers).

1. **Queue**

It stores and processes the data in FIFO(First In First Out) order. It is an ordered list of objects limited to inserting elements at the end of the list and deleting elements from the start of the list.

No Null Elements: Most implementations like Priority Queue do not allow null elements.

* Runtimes and memory
* Use case

1. Commonly used for Task scheduling, Message passing, and Buffer management in applications.
2. **Stack**

The class is based on the basic principle of **LIFO**(last-in-first-out). In addition to the basic push and pop operations, the class provides three more functions of empty, search, and peek.

* The Stack class extends Vector and provides additional functionality specifically for stack operations, such as **push, pop, peek, empty,**and **search**.
* Runtimes and memory
* Use case

 A deck of cards, piles of books, piles of money, and many more. This example allows you to perform operations from one end only

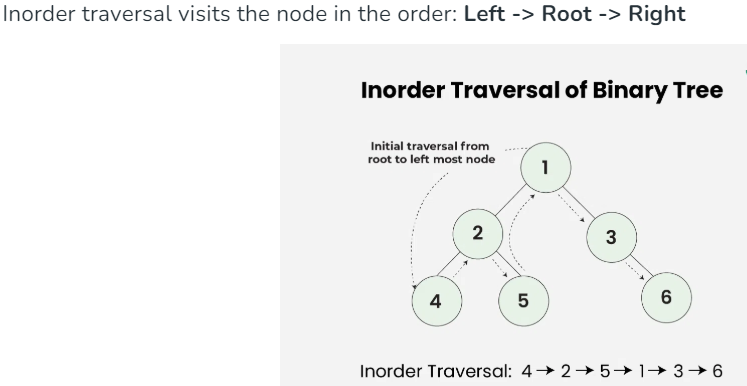
1. **Algorithms**

An algorithm is a procedure used for solving a problem or performing a computation. Algorithms act as an exact list of instructions that conduct specified actions step by step in either hardware- or software-based routines.

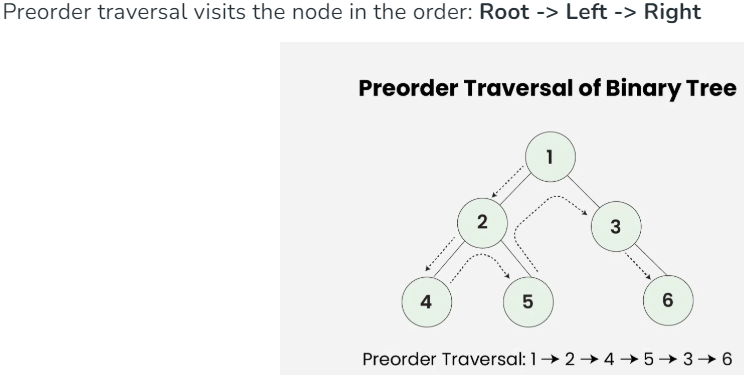
* Traversals

**Tree Traversal**refers to the process of visiting or accessing each node of the tree exactly once in a certain order. Tree traversal algorithms help us to visit and process all the nodes of the tree. Since tree is not a linear data structure, there are multiple nodes which we can visit after visiting a certain node. There are multiple tree traversal techniques which decide the order in which the nodes of the tree are to be visited.

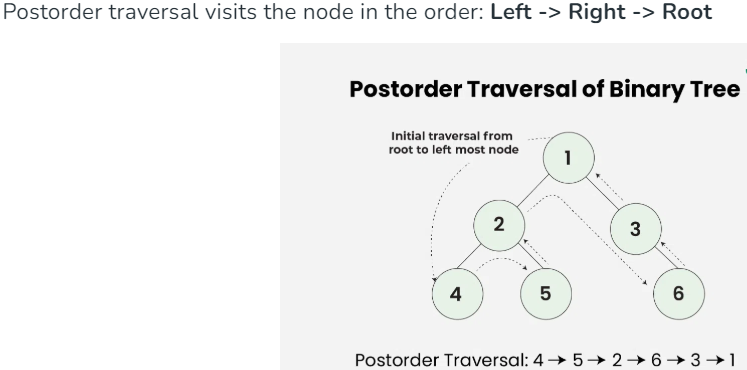
* Inorder Traversal



* Preorder Traversal



* Postorder Traversal

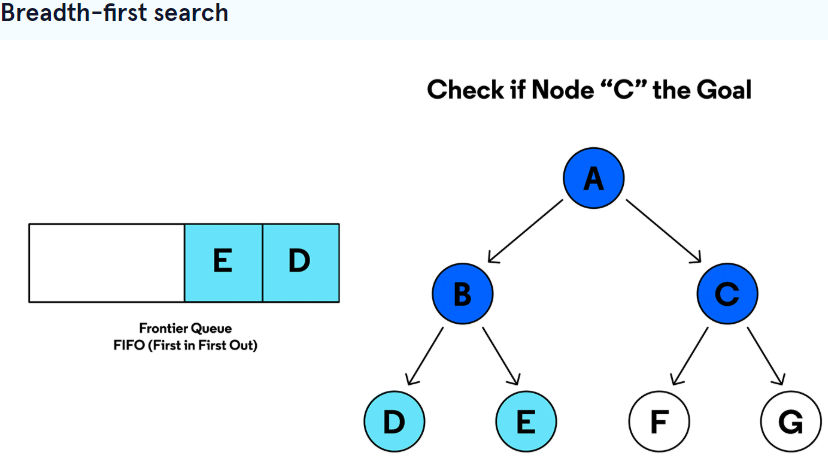


* Dived and conquer

A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

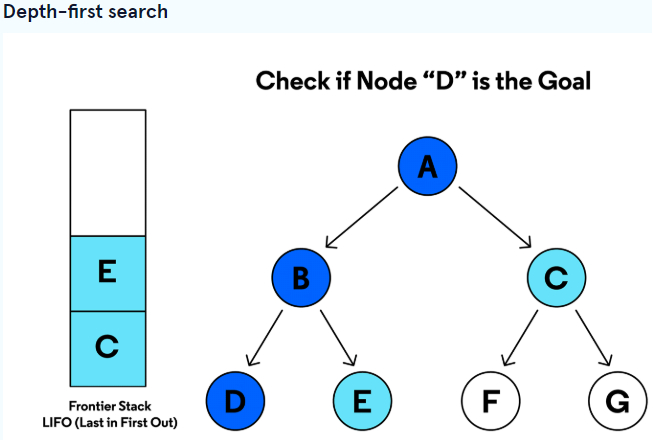
* Breadth first search

A breadth-first search is when you inspect every node on a level starting at the top of the tree and then move to the next level.



* Add A to the queue
* Check if A is the goal searching for.
* Add C and B to the queue
* Check if B is the goal searching for.
* Add E and D to the queue
* Check if C is the goal searching for.
* Depth first search

 A depth-first search is where you search deep into a branch and don’t move to the next one until you’ve reached the end.



* Add A to the stack
* Check if A is the goal searching for.
* Add B and D to the stack
* Check if B is the goal searching for.
* Add E and C to the stack
* Check if D is the goal searching for.

Chapter 2: Service

**Soap**

Soap (Simple Object Access Protocol) service

**SOAP** is a protocol which was designed before REST to ensure that programs built on different platforms and programming languages could exchange data in an easy manner. SOAP stands for Simple Object Access Protocol.

1. Advantages and Disadvantages

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| Platform independent  -HTTP – Transport independent  -XML – Data Independent (Java sends xml data to .Net app and the .Net app can convert data to .Net objects) | Performance impact due to serialization and deserialization (convert xml to objects and objects to xml) |
|  |  |
|  |  |
|  |  |
|  |  |

1. When to use soap web service?
   1. When formal contract is required before client app development. WSDL
   2. When we have a lot of non-functional requirements like security and transaction management. Can Rely on the web service standards that come with the soap web service and we can use these right out of the box. Web Service engines like apache cfx provide implementation for these standards and developers can use these standards with minimal configuration.
   3. When a Reliable Asynchronous Processing is required. Can use the Web Service Messaging standard
2. WS Standards

WS standards helps SOAP Web Service by providing non-functional requirements in a standard fashion. This helps developer to just use these Standards with minimal configuration.

Reason why these WS Standards are important.

For a payment application for example, when it processes a payment, the Bank Payment service using WS Security is expecting to receive the payments details in the standard format and in the standard place i.e. Soap Header or Soap Body. If there are no standard, every bank can have deferent formats and will need to keep making changes on your application.

WS Standard:

* 1. WS –Security
     1. Authentication
        1. User Name Token Profile – for username and password
        2. X 508 Certificate – For authentication
        3. SAML – Allows for single sign on in web applications
     2. Confidentiality
        1. Encryption and Decryption
     3. Integrity – make sure data received is the data sent by other app
        1. XML Signature – signature is sent with the message
  2. MTOM

For exchanging files.

* 1. Addressing

For Asynchronous call-backs

* 1. WS-Policy-Assertion

Assert and mandate certain rules to consume our web service.

* 1. WS-Secure Conversation

Improves performance while encrypting and decrypting by negotiating a key in the beginning.

* 1. WS-Security Policy

Assert WS-Security requirements.

1. Soap WS Designs
   1. Top Down / WSDL First / Contract First
      1. Create the WSDL file
      2. Generate the java stubs using tools like wsdl2java (Add CFX plugin in pom file)
      3. Create the endpoint
      4. Create the config class

Advantage:

* + Faster Integration – consumers don’t have to wait for the service to be completed.
  + Better Interoperability – Consumers can quickly is all required data structure and fields.
  + Contract with the consumer signed off – Easy to read for business and sign off.
  1. Bottom Up / Code First
     1. Write java code and annotate
     2. Generate the WSDL from the code using java2wsdl (provided by java CFX)

Advantage:

* + - Legacy applications to be providers or consumers

1. WSDL

Web Service Description Language, is an XML based definition language. It's used for describing the functionality of a SOAP based web service.

* Type - used to define the data types that will be used in the web service operations.
* Message - defines the data elements that are being exchanged between the web service provider and the service user.
* Port Type - describes a set of operations that can be performed by the web service
* Binding - specifies how the abstract operations defined in portType are mapped to a concrete protocol for communication. It defines details such as the message format and the protocol (e.g., SOAP).
* Port - specifies the network address where the web service can be accessed.

1. JAXB

Converts java object to XML Schema and from XML Schema to java objects.

JAXB is similar to Hibernate, Hibernate converts java objects to sql scripts.

1. Apache CFX

Open source framework for developing web services.

* JAX-WS and JAX-RS

Implements both of these

* SOAP Engine

Serialization and Deserialization and dispatches the incoming requests to the correct endpoint.

* WS Standards

Implements all the WS Standards

* Tools

Provide tools like java2wsdl and wsdl2java. Add CFX dependency and plugin in pom file

* Spring Configuration

Easy to configure CFX since its using sping.

1. Challenges of SOAP service

* WSDL file: REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. Making changes to the WSDL file will cause clients to make changes.
* Size of SOAP message: The large size of soap messages transferred from client to server can cause performance problems.

**Rest**

Rest (Representational State Transfer)

**REST** was designed specifically for working with components such as media components, files, or even objects on a particular hardware device. Any web service that is defined on the principles of REST can be called a RestFul web service. A Restful service would use the normal HTTP verbs of GET, POST, PUT and DELETE for working with the required components. REST stands for Representational State Transfer.

1. Advantages

* CRUD methods
* Easy to access. Use similar url and specify the CRUD method
* Multiple transfer data types (JSON, XML, CSV)
* Stateless. The rest server does not need to maintain any state.
* Scalability. Can use load balancer to distribute work load.

1. Steps to create a Rest service

Example, Hospital management application

* Identify the resource – Patient, Prescription, Medicine
* Create the URl’s to access the resources
* Assign the HTTP meths – POST, PUT, GET, DELETE
* Choose the data type – Create requests and response

1. Challenges of REST service

* Lack of security: Rest does not impose any sort of security like SOAP.
* Lack of state: Most web applications require a stateful mechanism. For example, if you had a purchasing site which had the mechanism of having a shopping cart, it is required to know the number of items in the shopping cart before the actual purchase is made. Unfortunately, the burden of maintaining this state lies with the client, which just makes the client application heavier and difficult to maintain.

**Soap vs Rest**

|  |  |
| --- | --- |
| Soap | Rest |
| SOAP can only work with XML format. As seen from SOAP messages, all data passed is in XML format. | REST permits different data format such as Plain text, HTML, XML, JSON, etc. But the most preferred format for transferring data is JSON. |
| Performance: slower than Rest  SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the amount of data transfer using SOAP is generally a lot. | Performance: Quick  REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. |
| API class cannot be cached | API calls can be ceched |
| Stateless by default but its possible to make a SOAP API stateful | stateless |
| WS-security with SSL support. Its also has a built in asset compliance. | Supports HTTPS and SSL |
| Transfer protocols: HTTP, SMTP, UDP and others. | Transfer protocol: HTTP |
|  |  |

**Recommendations to use SOAP and REST?**

Soap is recommended for high security applications like payment/financial applications and other application with sensitive data.

Rest is recommended to be used is public API’s for web services and applications that does not contain sensitive data.

**Query parameter vs path parameters vs request parameters**

**Styles of writing rest invokers in spring**

RestTemplate and FeignClient and WebClient

<https://medium.com/thefreshwrites/resttemplate-feignclient-webclient-with-spring-boot-f34979040b9e>

<https://www.baeldung.com/spring-boot-feignclient-vs-webclient>

**Stateless vs State full**

|  |  |
| --- | --- |
| State full | Stateless |
| The server maintains info about previous requests. | Server does not maintain info about previous requests. Requests are in depended of each other. |
| Caches user info on the server | Use A DB or Caching to maintain user state. |
| If server breaks, user will be routed to a different server by the load balancer and the user’s state will be gone with the broken server. User will have to redo the process. | There is one common place that maintains the user state DB or Caching (quicker and improved performance). This way user data maintenance is not tightly coupled with a single server. |

**Asynchronous vs synchronous**

|  |  |
| --- | --- |
| synchronous | Asynchronous |
| Synchronous operations are performed one at a time. One task finishes, the next step begins. | Asynchronous operations can happen at the same time — you can move to the next step while another step finishes. |
| Asynchronous example  AsyncRestTemplate restTemplate = new AsyncRestTemplate();  or  client.sendAsync(request, BodyHandlers.ofString()) | In a microservices architecture, async APIs allow microservices to communicate efficiently without blocking each other. This is crucial for building scalable and responsive systems. |
|  | Use Case: When the next operation doesn’t depend on the previous operation or rest api request. |

1. Swagger documentation
2. Exceptions
3. Logging
4. Auditing

Chapter 3: Testing

**Testing**

1. Integration testing – Testing controller and api endpoints
2. Unit testing – testing business logic methods
3. Repository testing – Testing JPA Repository methods

Chapter 4: Framework

**Java Frameworks**

1. Spring
2. Play
3. Quarks

**Spring framework**

1. Why we need spring framework?

With spring, we get **Dependency Injection** (identify the beans, identify their dependencies and wire them together). With this, we don’t have to focus on creating objects and managing their dependencies. Spring has a number of modules/projects like Spring Data JPA, Spring boot, Spring MVC, spring starter test (Mockito, Junit).

1. Types of dependency injections

* **Constructor-based**: Dependencies are set by creating a constructor on your bean and setting the variable bean in the constructor. You can have “@Autowire” in you bean variable declaration or on the constructor.
* **Setter-based**: Dependencies are set on a setter method on your bean class. You can have “@Autowire” in you bean variable declaration or on the setter method.
* **Field**: No setter or constructor. Dependencies are injected using reflection.

1. Recommended type of dependency injection.

Constructor-based because all dependencies are automatically set when an object is created.

1. How spring works

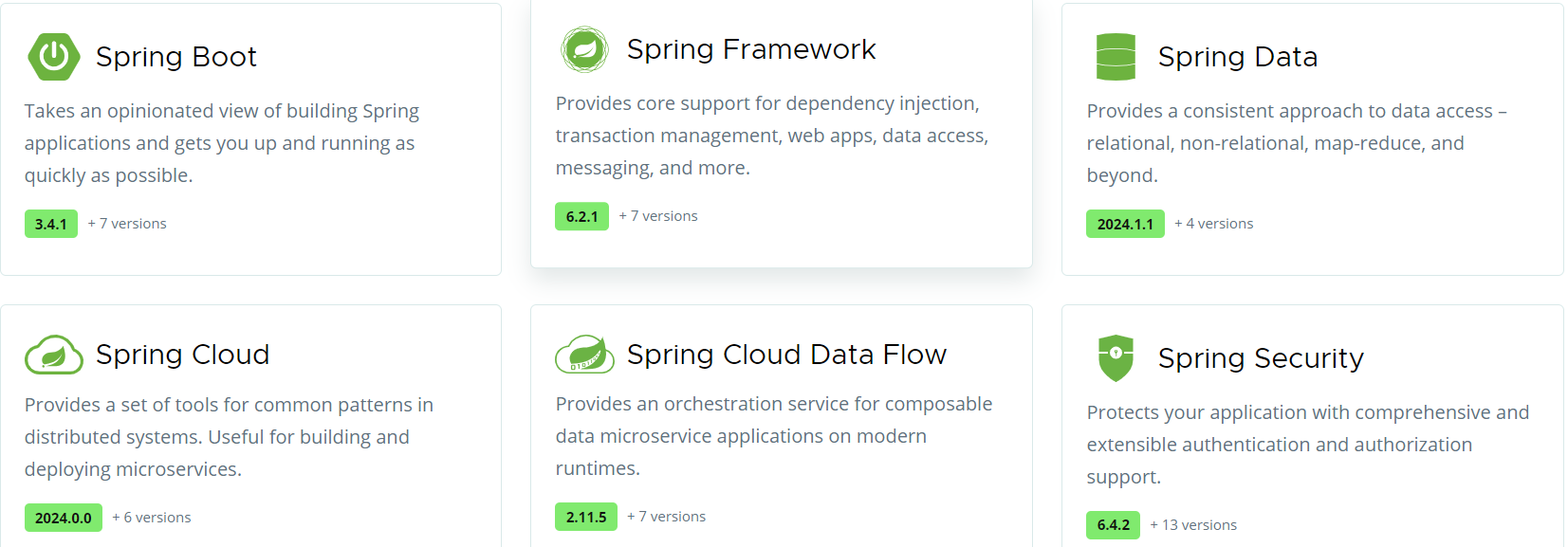
* **@Component** to allow spring to create a bean of the class
* Component scan will scan the package to look for all classes with annotation @Component.
* **Application context** creates the beans.
* **@Autowire** – Identifying the spring dependencies and wiring them.

1. Spring jars

We use maven to download spring jars to support our spring implementation.

**Spring Projects**

1. Some Spring projects



1. Spring MVC

A Spring MVC is a Java framework which is used to build web applications and REST API. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

1. Spring boot

* World before spring boot.

Setting up an project was not easy and took time.

* **Dependency Management**: Need add, manage frameworks/Dependencies and their versions in the pom file., e.g to build REST API (dependencies: Spring framework, Spring MVC, JSON binding framework). Unit tests (Spring test, Mockito, Junit)
* **Web.xml**: Configure dispatcher sevlet for spring MVC
* **Spring Configuration**: Define component scan, view resolver(jsp path)
* **None-functional requirements**: configure Logging, Error handling, Monitoring
* Spring boot helps to build **Production-Ready** apps **Quickly**.
* Build **Quickly**
* **Spring initializer**

You can create a spring project by specifying the name, build tool and dependencies in the spring initializer website.

* **Spring boot starter project**

Spring boot starter projects are dependencies that contain a list of pre-defined dependencies inside them for a specific feature.

**REST API and Web Applications** – Spring Boot Starter Web (spring-webmvc, spring-web, spring-boot-starter-tomcat, spring-boot-starter-json)

**Unit Test** – Spring Boot Starter Test (Spring test, Mockito, Junit)

**Database connection** – Spring Boot Data JPA

**Security** – Spring Boot Starter Security

* **Spring boot Auto Configuration**

By adding starter projects in the pom file, maven will download the dependencies for that starter project and place in the class path. Spring boot auto configures based on the frameworks that exist in the class path.

Auto Configuration decide bases on:

* Which frameworks are in the class path
* What is the existing configuration

Example: Spring Boot Starter Web will configure:

* Dispatcher servlet (DispatcherServletAutoConfiguration)
* Tomcat
* Default error page ( when you hit incorrect url endpoint)
* Bean to and from JSON (JacksonHttpMessageConvertersConfiguration)

All logic for spring boot auto configuration sits in a jar called SpringBootAutoConfiguration.jar

* **Spring boot DevTool**

Spring Boot devtools increate developer productivity. You don’t have to manually restart the services after writing a new code change. After the change, spring boot will auto restart the application.

* Build **Production-Ready**
* Logging

You can specify the logging level you want on your application property file. Level log:

Trace, debug, info, warning, error, off

* Different Configuration for different Environment (Profiles, ConfigurationProperties)

Provides easy configuration of profiles for different environments. Create separate application.properties for DEV, QA and PROD with their own configurations ( eg different databases, log level). You can set the application property file you want to active on your main property file.

* Monitoring (Spring Boot Actuator)

Spring boot starter actuator provides list of endpoints to monitor your application:

* Beans – Complete list of Spring beans in the app
* Health – Application health information
* Metrics – Application metrics
* Mappings – Details around Request mappings.

By default when you launching the actuator url, it gives the health url. For more features, you need to enable them in application.properties. For all endpoints:

Management.endpoints.web.exposure.include=\*;

1. Spring security

To authentication your api

Add spring security dependency on your pom file, it generates a password for you to use so that you can access the rest api methods.

You can override the generated password and useraname by specifying spring security username and password on application.property file.

At this point the application provides form based authentication

**Basic authentication**

Implement Basic authentication because "form based authentication" belongs to web services.

done in "SecurityConfig" class and "configure" method

At this point we still use the unersame and password specified in the application.properties with basic authentication

Now we want to have two users, "User" and "Admi".

User must not be able to create,update and delete Posts, they must only get the posts.

**In Memory authentication**

For this we can start by creating a In memory authentication where we specify the username, password and role for the two user("USRE,ADMIN).

So the application first checks/authenticates the username and password, if they are incorrect, it throws an unauthorised exception.

If its ussername and password are correction, it then checks the role.

If the role is USER then you can only get posts. theis is specified in the post controller

**Database authentication**

Remember its in a good idea to use an In memory authentication, rather use database authentication.

For database authentication, create USER and ROLE enteties/tables

Since relationship between ROLE and USER is many to many, have a third joint table.

Create USER and ROLE Repository and specify the needed methods.

create your interface and Impl class with the method of retrieving users with their roles from the DB using the methods in the Repository.

Call the method in your Interface into ""SecurityConfig" " to accomodate the database authentication, and you can remove the IN MEMORY AUTHENTICATION.

**JWT(JASON WEB TOKEN)**

Client uses their credentials(username/password) to login. If login is successful, they are given a token that expires after a certain time.

They use the token to make queries to the api unlike Basic Auth that uses the credentials every time they make a query which is not good for security.

Advantages of JWT over Basic authentication:

1 JWTs are generally specific to one service. Passwords (sadly) get reused all over.

2 JWTs are time-limited, and expire. If you look at logs for user credentials, you will find a token which is useless because it expired.

3 JWTs can be issued by one service and verified by another.can login with a defferent application(eg facebook, google) and your credentials can never been seen by the application you are using

4 JWTs can be both secure and fast to verify.

5 Relatedly, JWTs can be verified without a database lookup at all.

Passwords, including ones sent in basic auth, can't. You can cache them, but that takes server-side resources and breaks down when you have a cluster of servers behind a load balancer.

Thus, JWTs are much better for scaling to large numbers of users, where DB hits are relatively expensive and need to be minimized.

6 Logging out a user who uses a JWT is simple. Rely on token expiring. with basic auth, user has to logout when they want.

**Implement JWT**

1 add JWT Dependency

2 create jwtAuthenticationEntryPoint - to handle the exception that occured due to unautherised access from the client

3 add JWT properties is application.properties file

4 create JwtTokenProvider - create methods to generate JWT tokens

5 create JwtAuthenticationFilter - garantees a single execution per request dispatch

6 create JwtAuthResponse DTO

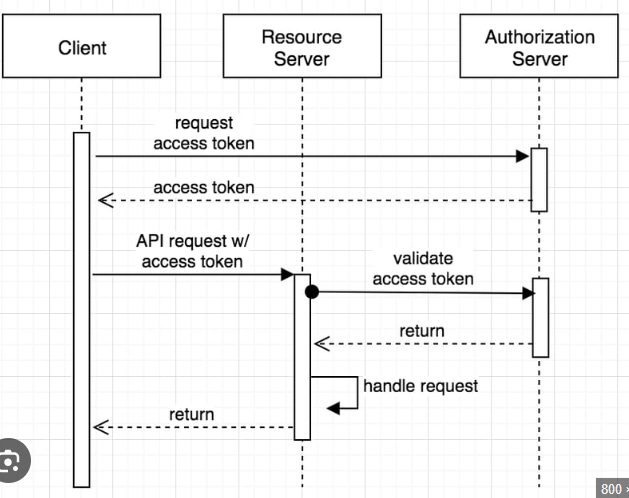
7 Configure JWT in Spring Security Configuration -

In the configuation, we specify the authenticationEntryPoint, specify the sessionCreatePolicy.stateless and added the JwtAuthenticationFilter

8 Change login API to return token to client

* **Aouth2**

In a Spring Boot application, OAuth2 can be integrated to handle both authentication and authorization. The framework relies on Spring Security, which provides robust mechanisms for securing applications.



Client application

* **Client Registration:** You need to register the application with the OAuth2 provider (e.g., Google, GitHub) to obtain a client ID and client secret. These credentials are used to identify the application to the authorization server.
* **OAuth2 Client Setup in Spring Boot:** In Spring Boot, configure the OAuth2 client settings in the application.properties or application.yml file. This includes specifying the client ID, client secret, authorization server URLs, and scopes of access. The Spring Security OAuth2 client automatically handles the redirection to the authorization server and manages the exchange of authorization codes for access tokens.

Authentication server

* <https://wkrzywiec.medium.com/create-and-configure-keycloak-oauth-2-0-authorization-server-f75e2f6f6046>

Resource server

The server hosting the protected resources. It validates the access token and grants access to the resources if authentication is successful.

Steps to implement

* Add this property to interact with authentication server: spring.security.oauth2.resourceserver.jwt.issuer-uri: ${JWT\_ISSUER\_URI}
* Add Spring Security dependencies.
* Write Configurations to validate tokens, roles.

Chapter 5: Database

**Databases**

1. **What are databases and why we need them?**

Used to store information

1. **SQL vs NoSQL**

|  |  |  |
| --- | --- | --- |
| **Category** | **SQL** | **NoSQL** |
| **Structure** | Relational (Tables that can have relationships) | Implementation dependent (documents, Graphs, tables) |
| **Storage** | Concentrated (One node contain the entire copy of the data. It is not partitioned) | Hashing input. Data is partitioned into multiple nodes. |
| **Scale** | Vertical (increase RAM, CPU in your server) | Horizontal (add more servers to handle traffic) |
| **Access** | SQL (Structured query language), JPQL, Direct connection | Rest API  Simple queries |
| **Advantage** | Access data (flexible queries) | 1. Horizontal scaling.  2.Schemaless (items in the DB don’t have to have the same structure)  3.High performance (using key to search instead of going through all rows) |
| **Disadvantage** | 1.Vertical scaling  2.Schema base (All data have to be in a table structure) | 1.Access data (flexible queries) |

1. **When to use SQL and NoSQL**

|  |  |
| --- | --- |
| **SQL** | **NoSQL** |
| When your access patterns are not defined | When your access pattern is defined |
| When you want to perform flexible queries | When your primary key is known |
| When you want to perform relational queries | When your data model fits (graph) |
| When you want to enforce field constraints | When you need high performance and low latency |
| When you want to use well documented access language (SQL) |  |

1. **SQL and NoSQL database engine examples**

|  |  |
| --- | --- |
| **SQL** | **NoSQL** |
| MySQL | mangoDB |
| SQLServer | DynamoDB |
| PostgreSQL | Elasticsearch |
| Amazon Aurora | Amazon ElastiCeche |

1. **NoSQL DB Types**

* key/value
* graph
* column
* document

1. **JPA and Hibernate**

|  |  |
| --- | --- |
| **JPA** | **Hibernate** |
| JPA is described in **javax.persistence** package. | Hibernate is described in **org.hibernate** package. |
| It describes the handling of relational data in Java applications. | Hibernate is an Object-Relational Mapping (ORM) tool that is used to save the Java objects in the relational database system. |
| It is not an implementation. It is only a Java specification. | Hibernate is an implementation of JPA. |
| It is a standard API that permits to perform database operations. | It is used in mapping Java data types with SQL data types and database tables. |
| As an object-oriented query language, it uses **Java Persistence Query Language (JPQL)** to execute database operations. | As an object-oriented query language, it uses **Hibernate Query Language (HQL)** to execute database operations. |
| To interconnect with the entity manager factory for the persistence unit,ituses **EntityManagerFactory**  interface. Thus, it gives an entity manager. | To create Session instances, it uses **SessionFactory** interface. |
| To make, read, and remove actions for instances of mapped entity classes, it uses **EntityManager** interface.  This interface interconnect**s** with the persistence condition. | To make, read, and remove actions for instances of mapped entity classes, it uses **Session**interface. It acts as a runtime interface between a Java application and Hibernate. |

1. **Entity lifecycle**

* Transient/New State

When an entity object is initially created, its state is new. In this state the object is not yet associated with an Entity Manager and has no representation in the database.

* Managed State

This is when the entity is persisted to the database via an Entity Manager persist() method which must be invoked within an active transaction.

* Detached

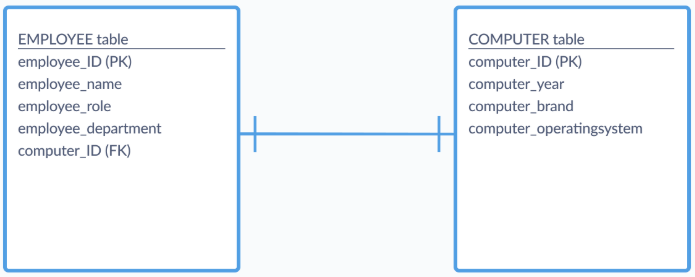
This is when the entity is disconnected from the entity manager

* Removed

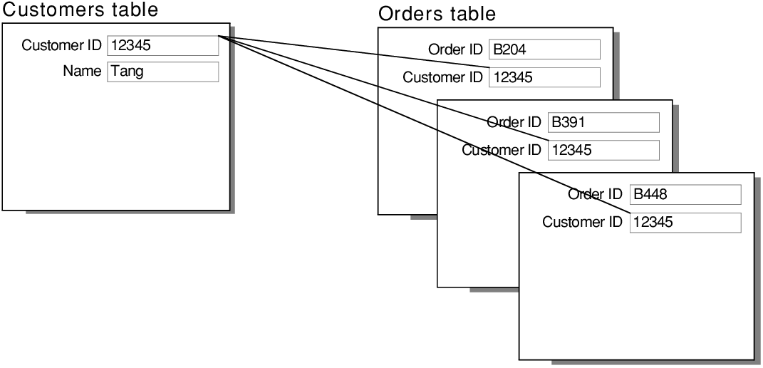
Entity can be marked as removed by using the entityManager remove() method within an active transaction. The entity object changes from managed state to remove state and physically deleted from the database during commit.

1. **Relationships**

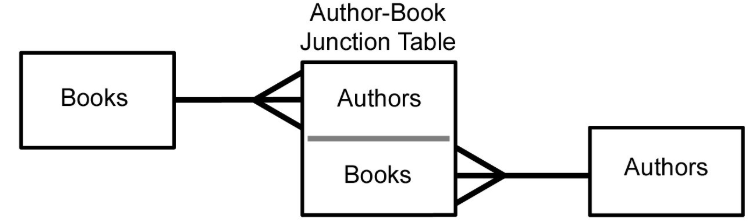
* One to One



* One to Many



* Many to Many



1. **Database Constraints**

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

* [NOT NULL](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [UNIQUE](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [FOREIGN KEY](https://www.w3schools.com/sql/sql_foreignkey.asp) - Prevents actions that would destroy links between tables
* [CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that the values in a column satisfies a specific condition
* [DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified
* [CREATE INDEX](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

1. **Database design and normalization**

Database normalization is the process of organizing the attributes of the database to reduce or eliminate data redundancy (having the same data but at different places).Data redundancy unnecessarily increases the size of the database as the same data is repeated in many places. Inconsistency problems also arise during insert, delete, and update operations.

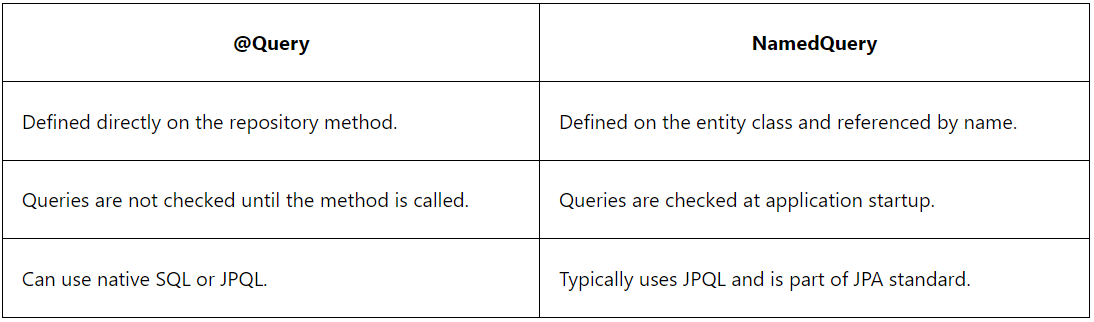
1. **Native Queries**

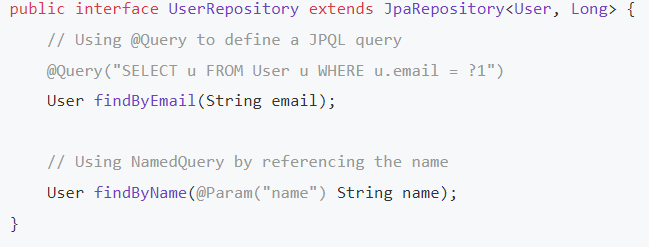
A native query is a SQL statement that is specific to a particular database like MySQL. It varies a little from JPQL (Java Persistence Query Language) which is used by Spring Data JPA by default. Below is an example of a native SQL query SELECT \* FROM Student ORDER BY age.

While JPQL can help simplify queries, it may not always fully utilize the power of SQL. Using native queries can offer more flexibility in terms of performance optimization and handling complex queries, as they can leverage the maximum benefits of DB configurations such as table indexes, etc.

1. **JPA (Name query vs @Query)**

In Java Persistence API (JPA), *@Query* and *NamedQuery* are both ways to execute database operations using JPQL (Java Persistence Query Language). *@Query* is an annotation used on repository methods to specify a JPQL or native SQL query. *NamedQuery*, on the other hand, is defined statically in the entity class and referenced by a unique name.





* When to use them?
  + Use *@Query* when you need flexibility and dynamic queries that might change at runtime, or you are heavily using Spring Data features.
  + Use *NamedQuery* when you have a static query that won't change and you want to ensure the query is valid at application startup.

1. **Stored procedure**

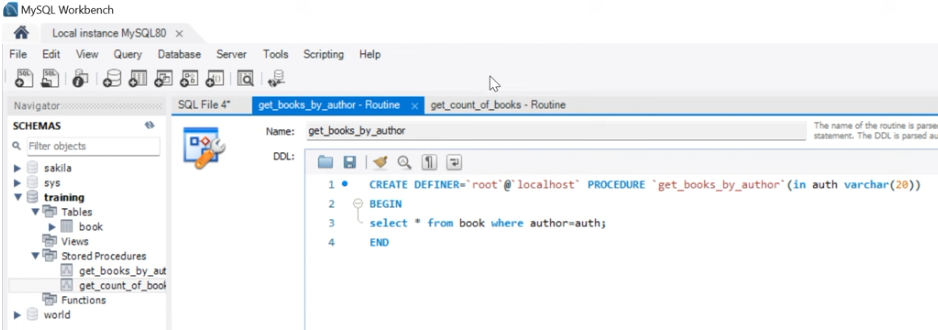
A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

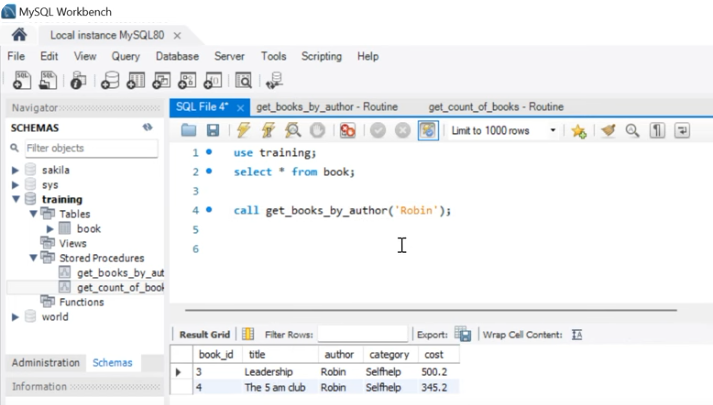
So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

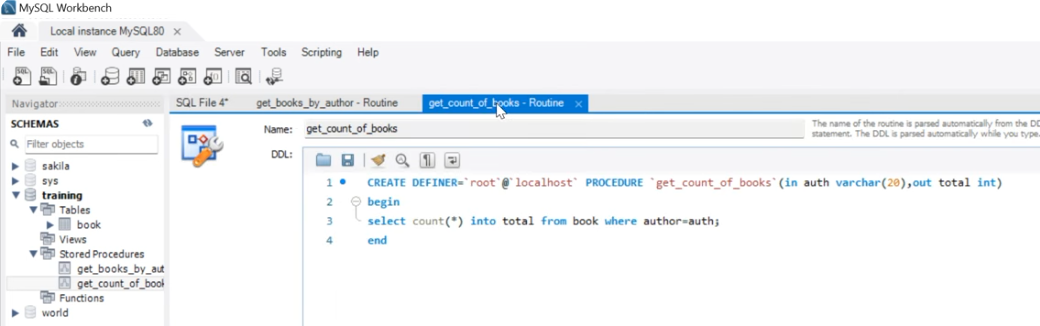
* Create and Query on MySQL

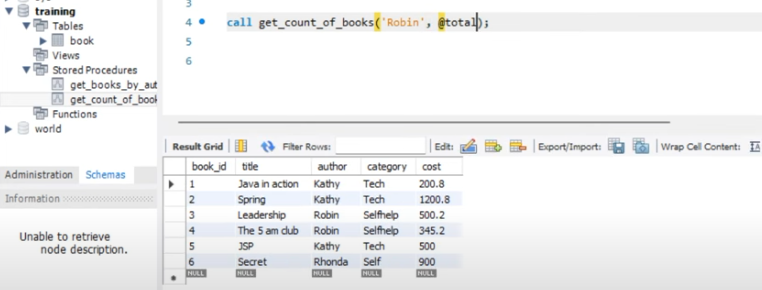
Stored procedure 1

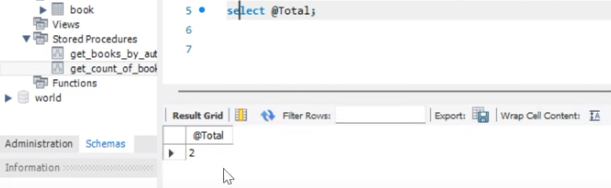




Stored procedure 2

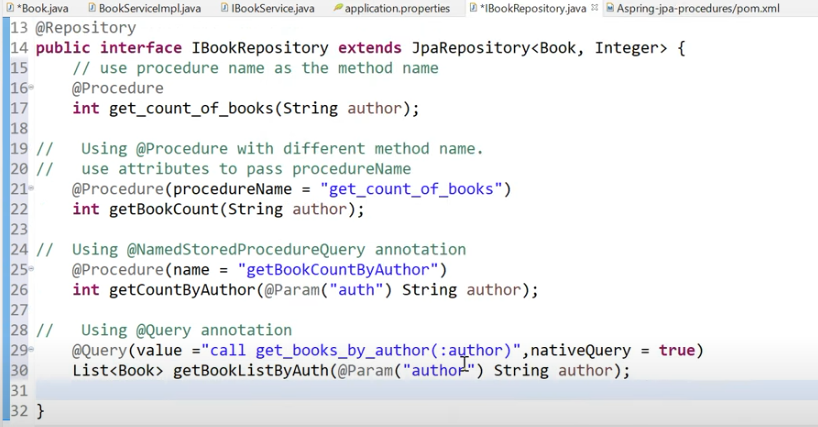




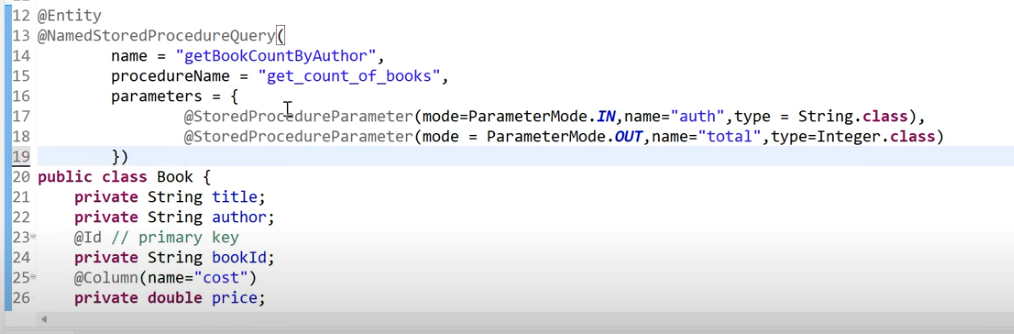


* Query using java

Different methods of calling a stored procedure in spring boot application

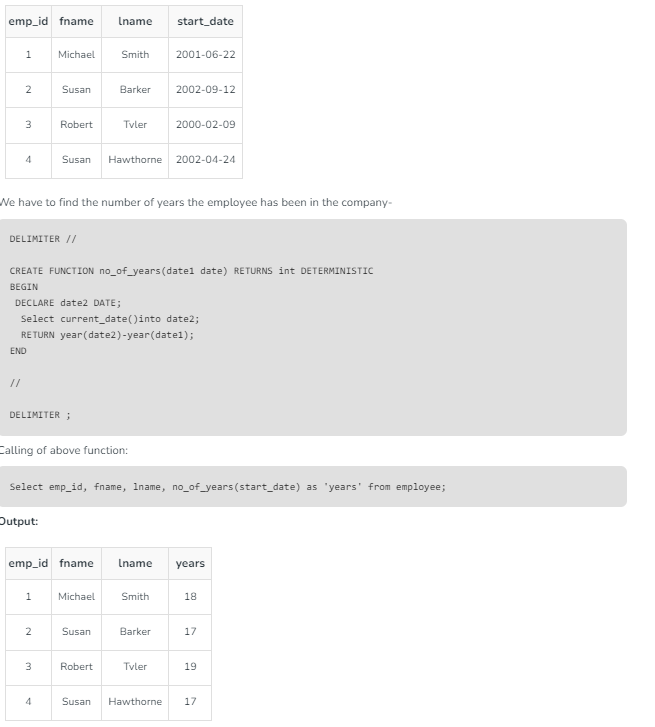


This is needed for the @NamedStoreProcedureQuery



1. **Stored function vs procedure**

Functions cannot change anything and must have at least one parameter. Also, it must return a result. Stored procedures take no parameters, can modify database objects, and need not return results. Stored procedures join SQL queries into transactions and communicate with the outside world.

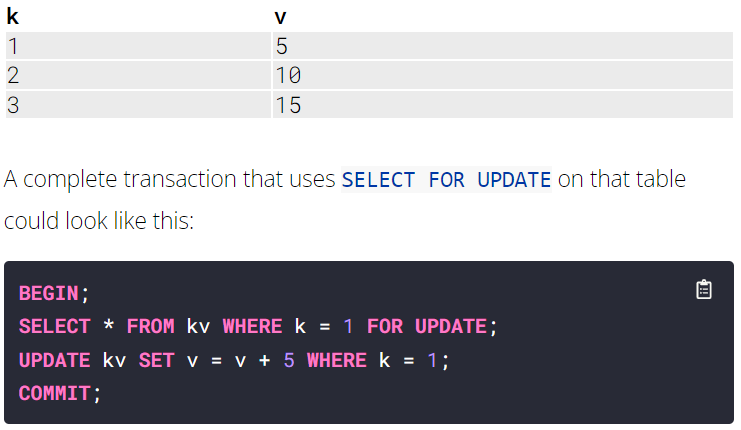


1. **Select for update**

Locking a specific row just before you update it to avoid other people/ scripts updating the same row at the same time which will cause incorrect data and confusion.

In general, SELECT FOR UPDATE is useful for any transactional workload where multiple transactions might attempt to access the same rows at the same time.

Example:



1. **Joins**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

* (INNER) JOIN: Returns records that have matching values in both tables
* LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
* RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
* FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

1. **Transactions**

A SQL transaction is a grouping of one or more SQL statements that interact with a database. A transaction in its entirety can commit to a database as a single logical unit or rollback (become undone) as a single logical unit. In SQL, transactions are essential for maintaining database integrity.

1. **Indexes**

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

A table or view can contain the following types of indexes:

* Clustered
  + Clustered indexes sort and store the data rows in the table or view based on their key values. These are the columns included in the index definition. There can be only one clustered index per table, because the data rows themselves can be stored in only one order.
  + The only time the data rows in a table are stored in sorted order is when the table contains a clustered index. When a table has a clustered index, the table is called a clustered table. If a table has no clustered index, its data rows are stored in an unordered structure called a heap.
* Nonclustered
  + Nonclustered indexes have a structure separate from the data rows. A nonclustered index contains the nonclustered index key values and each key value entry has a pointer to the data row that contains the key value.

1. **Connection pool**

Connecting to a backend service is an expensive operation, as it consists of the following steps:

* Open a connection to the database using the database driver.
* Open a TCP socket for CRUD operations
* Perform CRUD operations over the socket.
* Close the connection.
* Close the socket.

In a production environment where we expect thousands of concurrent open and close connections from clients, doing the above steps for every single connection can cause the database to perform poorly.

We can resolve this problem by pooling connections from clients. Instead of creating a new connection with every request, connection poolers reuse some existing connections. Thus there is no need to perform multiple expensive full database trips by opening and closing connections to backend service. It prevents the overhead of creating a new connection to the database every time there is a request for a database connection with the same properties (i.e name, database, protocol version).

Implementing connection pool



1. **Caching**

Caching is a system design concept that involves storing frequently accessed data in a location that is easily and quickly accessible. The purpose of caching is to improve the performance and efficiency of a system by reducing the amount of time it takes to access frequently accessed data.

* Why you cannot store all the data in cache?
* Hardware of the cache which is much more expensive than a normal database.
* Also, the search time will increase if you store tons of data in your cache.
* Cache is typically a volatile storage, meaning data is lost if the system crashes or restarts. For critical and long-term data, storing it only in cache would risk data loss.
* So in short a cache needs to have the most relevant information according to the request which is going to come in the future.
* How Does Cache Work?
* Toreduce the number of calls to the database, we can use cache and the tweets can be provided much faster.
* In a typical web application, we can add an application server cache, and anin-memory store like [Redis](https://www.geeksforgeeks.org/introduction-to-redis-server/" \t "_blank) alongside our application server.
* When the first time a request is made a call will have to be made to the database to process the query. This is known as a cache miss.
* Before giving back the result to the user, the result will be saved in the cache.
* When the second time a user makes the same request, the application will check your cache first to see if the result for that request is cached or not.
* If it is then the result will be returned from the in-memory store. This is known as a **cache hit.**
* The response time for the second time request will be a lot less than the first time.

* distributed caching

A distributed cache is a cache shared by multiple app servers, typically maintained as an external service to the app servers that access it. A distributed cache can improve the performance and scalability of an ASP.NET Core app, especially when the app is hosted by a cloud service or a server farm.

1. **How to increase DB performance?**

Connection pooling

Indexing

Caching

1. **DB dead locking**

In a database, a deadlock is a situation in which two or more transactions are waiting for one another to give up locks.

For example, Transaction A might hold a lock on some rows in the Accounts table and needs to update some rows in the Orders table to finish. Transaction B holds locks on those very rows in the Orders table but needs to update the rows in the Accounts table held by Transaction A. Transaction A cannot complete its transaction because of the lock on Orders. Transaction B cannot complete its transaction because of the lock on Accounts. All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions.

* Deadlock avoidance

The deadlock avoidance method is suitable for smaller databases whereas the deadlock prevention method is suitable for larger databases.

One method of avoiding deadlock is using application-consistent logic. In the above-given example, Transactions that access Accounts and Orders should always access the tables in the same order.

* Deadlock prevention

For a large database, the deadlock prevention method is suitable. A deadlock can be prevented if the resources are allocated in such a way that a deadlock never occurs. The DBMS analyses the operations whether they can create a deadlock situation or not, If they do, that transaction is never allowed to be executed.

1. **Multiple data sources**

Chapter 6: Micro-service

**Microservices**

1. **Different architectures**

* Monolithic

Monolithic software architecture is a traditional model for designing software programs that uses a single codebase to perform multiple functions. The word "monolith" is often used to describe something large and glacial, which is similar to the complexity of a monolithic architecture.

* SOA

Service-oriented architecture (SOA) is a method of software development that uses software components called services to create business applications. Each service provides a business capability, and services can also communicate with each other across platforms and languages.

* Microservice

With a microservices architecture, an application is built as independent components that run each application process as a service. These services communicate via a well-defined interface using lightweight APIs. Services are built for business capabilities and each service performs a single function.

* Map reduce

MapReduce is a programming model and software framework for processing large amounts of data in parallel on computer clusters

1. **Why microservice architecture is the best for big projects**

* Accelerate scalability

DevOps teams seamlessly introduce new components without causing any downtime, thanks to the independent operation of each service within the microservices architecture. They can choose each service's best language or technology without compatibility concerns.

Deploying services across multiple servers can mitigate the performance impact of individual components and help companies avoid vendor lock-in.

* Improve fault isolation

Microservices architecture is compartmentalized — if one service encounters a fault or failure, it doesn’t propagate across the entire system.

* Enhance team productivity

Microservices architecture allows small, focused teams to concentrate on a particular service’s development, deployment, and maintenance without being burdened by the complexities of the entire system.

Microservices architecture fosters a sense of ownership and expertise within teams, enabling specialized team members to make informed decisions, iterate quickly, and maintain a high quality of service within their domain.

* Quicker deployment time

In monolithic architectures, changing necessitates redeploying the entire application. Microservices architecture enables faster releases because each service evolves and deploys independently, reducing the risk and time associated with coordinating changes across an entire application.

Decoupling services in this manner enhances agility. You can swiftly roll out updates or fixes with minimal disruption to the overall system.

* Increase Cost-efficiency

Microservices architecture optimizes resource allocation and maintenance because teams work on small, well-defined services. Efforts are localized to specific services, reducing overall development and system maintenance costs. Teams focus on specific functionality, ensuring resources are used efficiently without redundancy or excess capacity.

1. **Challenges**

* Sizing

Solution: You can choose per feature. E.g Cards, accounts, loans

* Deployment

How do we deploy all the 100s of microservices?

Solution: use docker compose to run all your images/services with only one command (docker-compose up).

If services were stopped, you and start all of them at the same time. docker start fee e // You can run one command to deploy all container and specify.

* Portability

How do we move the 100s of services between environments

Solution: The same docker image you can use in different environments. You don’t need to install the project dependencies, you just download the docker image, install docker and run/convert the image to a docker container.

* Scalability

How do we scale our applications based on requirements

Solution: docker run -p 8081:8080 lwazi/accounts //can create another instance of the same service and provide difference port number. With virtual machine, you need to create a virtual machine, set up the environment and run another instance. You will have to buy all the deferent virtual machines but with docker you will use/run the the different container instances

* Performance

Caching responses between services.

Create multiple instances of the services so load balancer can distribute the requests.

* A client need to remember all host address's for all microservices.

Use API Gateway to accept requests from client and rout the requests to the appropriate microservice

* How is the load being distributed to the service instances?

Use API Gateway for load balancing

* All mricoservices need security. Its good practise to have a centralised security for all microservices.

Use API Gateway for centralised security

* All microservices have their own configurations. Its best practise to have a central project for configuration

Use Spring cloud to create a central place for configuration.

* Microservices communicate with each other and if M2 is down, M1 will throw an exception because M1 did not get the response from M2.

M1 will respond to API Getway with an error and API Getway responds to the client with an error. This is a waste of resources.

M1 must not call M2 if M2 is down.

Solve by implementing Circut Breaker pattern

* Services and instances of services may be down and we need a way to know which ones are up and running

Use Service Registry and Service Discovery to check which services are up and running

* M1 can call M2. We need loggings of the call hierarchy

Use Distributed tracing to get all the call hierarchy

* Improve performance on micro-service architecture.
  + **Caching**: Store frequently accessed data in a temporary cache to speed up data retrieval. This can reduce the need for time-consuming operations like database queries.
  + **Asynchronous requests**: Use asynchronous communication, which uses intermediaries like message queues.
  + **Load balancing**: Use load balancing to ensure high availability.
  + **Monitoring and optimization**: Monitor and optimize resource usage.
  + **Best coding practices**: Apply best coding practices.
  + **Testing and benchmarking**: Test and benchmark your services.
  + **Updates and upgrades**: Update and upgrade your services.
  + **Redundancy and replication**: Use redundancy and replication to ensure high availability.
  + **Auto-scaling**: Use auto-scaling to ensure high availability.
  + **Circuit breakers**: Use circuit breakers to ensure high availability.
  + **Health checks and self-healing**: Use health checks and self-healing to ensure high availability.
  + **Geographic distribution**: Use geographic distribution and multi-region deployment to ensure high availability.
  + **Data replication and backup**: Use data replication and backup to ensure high availability.
  + **Failover mechanisms**: Use failover mechanisms to ensure high availability.

1. **Spring Cloud**

Spring cloud provide implementation for features that can be used in microservice architecture. Features like Spring Cloud Gateway, Spring Cloud Config. We don’t need to implement these features manually when creating microservices because its already implemented by spring cloud.

* **Some of the Spring Cloud models/features**
* Spring Cloud Gateway - API Gateway
* Spring Cloud Eureka Netflix - Service Registry
* Spring Cloud Config - Centralized Configuration
* Spring Cloud Bus - refresh all the services and its instances to get the updated configurations
* Spring Cloud Sleuth - for Distributed tracing
* Spring Cloud Circut Breaker – Call retries and stop calling if not response
* Spring Cloud OpenFeig - to make rest calls to other microservices
* **API Gateway (Spring Cloud Getway)**
* Responsibilities
  + Route requests from clients to service (Clients dont have to rememeber port address for all services and instances. Clients dont need to know all details of the micro services)
  + Load balancing
  + Central Security (all services need security and it is tedious to have security for each service. Client will authenticate with the API Gateway only and not for each and every service).
* How API Gateway works:
  + Client will send a request to API Gateway.
  + API Gateway will discover the correct IP address and port using Service Registry to communicate and route the request to the correct micro-service instance.
* **Service Registry (Spring Cloud Eureka Netflix)**
* Responsibilities
  + It keeps track of all running instances of a service
  + It allows services to discover each other (Specify only the service name you are requesting to (no need for a url) and eureka server will forward the request to the correct instance)
  + It provides load balancing (will direct queries to only up and running instances)
* **Centralized Configuration (Spring Cloud Config Server)**
* Problem 1: When updaing configuration file of a service, we need to restart the service and its instances

Solution 1: No need to restart services with centralized configuration. Call the refresh endpoint to get the updated configurations

* Problem 2: Each service has its own configuration file. When updating we need to update all configuration file in all services.

Solution 2: Make changes in one project (git repo)

* **Spring Cloud Bus**
* Problem: You need to refresh all the services and its instances to get the updated configurations (localhost:8080/actuator/refresh)

Solution: Spring Cloud Bus module can be used to link multiple applications with a message brocker (i.e RabbitMQ) and we can broadcast the configuration changes by hiting /busrefresh and rabbitmq will update all services without restarting them. http://localhost:8080/actuator/busrefresh

* **Distributed Tracing (Spring Cloud Sleuth and Zipkin)**
* Problem: requests can be passed between micro services. Response can be slow or services can be down.

Solution: Use Distributed Tracing to trace request from start to finish. Also used to check how much time the service take to process and respond.

* **Spring Cloud Circut Breaker**

Circuit breaker using Resilience4J implementation

* Problem: m1 calls m2 calls m3 calls m4. If m4 is down, this will cause failure in the call from m3 to m4. This will then affect all the other micro-services because of dependency

Solution:

1. fallback - m3 will respond to m2 with a fallback response and m2 will respond to m1 with the fallback response

2. circuit breaker - this wont allow m3 to hit/call m4 continuously when m4 is down.

3. retry - when m4 is down, we can implement a number of retries in m3 incase m4 responds.

4. rate limit - This will limit the number of calls from m3 to m4

* How circuit breaker works?

Example: circuit breaker is implemented in m3. m3 makes calls to m4.

maintains 3 states (closed, open, half open):

Closed - both m3 and m4 are up and running. Circuit breaker will allow m3 to make calls to m4.

We maintain a threshold of 50% for example. If m3 makes 5 calls to m4 and only 2 passed and 3 failed, the circuit breaker will move to open state.

Open - We can set 5 seconds to be in a open state. Automatically after 5s, it will move to Half open state.

Everytime there is a failed results, circuit breaker moved to a open state and then after 5s, it automatically moved to half open state.

Half Open - In this state, m3 is given a chance to make the calls again to m4. We can set the number of calls try to 3.

If we get 2 successful calls (PASSED), it will move to closed state.

If we get 2 unsuccessful calls (FAILED), it will move to Open state.

1. **What is docker image**

Docker image is package which has all the dependencies that your application need (java, spring boot libraries, business logic). With one image you can create multiple containers (instances) with different port numbers. These containers run in their own isolated environment and they don’t need operating systems unlike virtual machines.

1. What is a container

Is a environment that allows us to build and run software packages. These packages include the code and all dependencies to run the code. We call these package container images.

1. What is Docker

Docker is a tool that allows applications to be packaged with all dependencies and ran wherever wanted

1. Create docker image

* Option1 (Docker file)
* Create a docker file inside your project folders
* docker images //check if you don’t have images
* docker build . -t lwazi/accounts //create image
* Option2 (Build-packs)
* What are build-packs

Buildpacks are provided by spring-boot and the purpose of build-packs is that you don’t have to write a docker file definition. Build-packs detects all the dependencies you have in your application, business logic and base on that, they will create a docker image without you creating a docker file.

* Sas
* as

1. Create docker container from docker image

Docker run -p 8080:8080 lwazi/accounts

* Docker ps //check all containers that are running
* Docker run -p 8081:8080 lwazi/accounts //can create another instance of the same service and provide different port number

1. Docker

* **docker images** //check if you don’t have images
* **docker build . -t lwazi/accounts** //create image
* **Docker ps** //check all containers that are running
* **Docker ps -a** //returns all containers present in the system/server
* **Docker run -p 8090:8080 lwazi/accounts** //creates docker container from the docker images
* **Docker run -p 8081:8080 lwazi/accounts**  //can create another instance of the same service/container and provide different port number.
* **Docker run -d -p 8082:8080 lwazi/accounts**  //run container without see/returning logs.
* **Docker logs fc** //checks the current logs of the specified container. “fc” is the container id.
* **Docker logs -f fc** //returns latest and new container logs.”-f” means follow.
* **docker stop fc** //gradually (takes some time) stops the running the container. It will shutdown the container first then stops the container. Check the logs.
* **docker kill fc** //immediately stops/kill the running container. Does not wait to shutdown the container.
* **docker start fee e** //deploy both containers at the same time
* **docker container pause fc** // pause the container so that it does not take any requests.
* **Docker container unpause fc** //un-pause the container
* **Docker container inspect fc** //returns container details
* **Docker stats** // returns the stats of all running containers e.g CPU used.
* **Docker rm fc** // removes/deletes the container
* **Docker push docker.io/lwazi/accounts** // pushes image to remote repo so that someone else can download and run it.
* **Docker compose**

With docker compose you can create a yml file with all the services/images defined in it. You can run a single command to execute to yml file which will run all specified images.

* docker-compse up //this will read the file and execute and run all specified images in the yml file.
* **Cd**

JMS (Java Message Serve)

JMS is a java messaging oriented middleware API for sending messages between applications.

JMS Components

* JMS Provider – Is the JMS middleware API
* JMS clients – Java applications that produces or receive messages
* JMS Producers – application that creates the and send the message
* JMS Consumer - application that received the message
* JMS Application – System composed of many JMS clients and one JMS provider.

Messaging Models

|  |  |
| --- | --- |
| PTP (Point to Point) | Publish-Subscribe |
| Each message has only one consumer | Each message can have multiple consumers. |
| Messages are first sent to the Queue | Messages are first published to the Topic |
| A sender and a receiver of the message have no timing dependencies. The receiver can fetch the message whether or not it was running when the client sent the message | Publishers and Subscribers have a timing dependency. |
| The receiver acknowledges the successful processing of a message | Does not provide acknoledgement |
|  |  |

Chapter 6: Cloud Computing

Cloud Computing

This is when you get computing services, DB storage, software applications and other IT resources from the cloud. So you don’t have to have your own hardware servers in your building.

Benefits

* No hardware maintenance and security needed
* Pay as you go
* Auto scaling – right size
* Access the resources instantly

Public cloud providers

* AWS
* Google cloud
* Microsoft azure

Kubernetes

Event driven architecture

NoSQL