**Contents**

1. MySQL
2. Java Collection Framework
3. Java Multithreading
4. Java Spring
5. Spring Boot & Microservices
6. React.js

Pre-requisites

* OOPS concepts
  + Inheritance
  + Encapsulation
  + Polymorphism – Overriding
  + Abstraction – abstract class & interface
* Exception Handling
  + try, catch, finally, throw & throws
* Object – root class
* String, LocalDate, LocalDateTime, LocalTime

Exceptions – Types

1. Checked Exceptions – Need to handle it at the compilation time
2. Unchecked Exceptions – These are ignored by the compiler

ArithmeticException, NullPointerException, ArrayIndexOutpBoundsException – Unchecked Exceptions

SQLException, IOException, Exception – Checked Exceptions

Naming conventions:

Keywords: lowercase, example: private, public, int, long, void, double, …

Classnames & Constructors: Starts with uppercase & follows the camel case

Method names & Variable names: Starts with lowercase & follows the camel case

Final variables: Every letter will be in uppercase & uses \_ for multiple words

Example: HELLO\_WORLD

Predefined-classes

1. LocalDate: It is to represent the date like doj, dob, current date
2. LocalDateTime: It is to represent the date & time both like login times
3. LocalTime: It is to represent the time

LocalDate follows ISO standard format to represent the date, i.e, yyyy-MM-dd

ISO format is followed by all the applications & devices

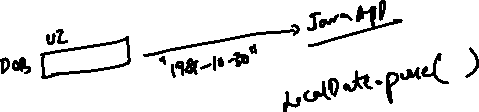
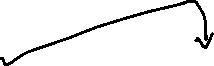
How to create LocalDate

LocalDate dob = LocalDate.of(1980, 11, 25); // 1980-11-25

LocalDate dob2 = LocalDate.parse(“1981-10-30”); // 1981-10-30

LocalDate dob3 = LocalDate.parse(“20-11-1989”, DateTimeFormatter.ofPattern(“dd-MM-yyyy”))

// dob3 = 1989-11-20



LocalDate is present in java.time package which you must import.

Note: Object, String, System and etc are part of java.lang which you don’t have to import

toString(): It is called when you print the object, by default it returns memory address & it is present in Object class, but you can override to return the object property so that when you print object it can print the object property by calling toString().

Signature of toString(): public String toString()

Creating multiple layers to maintain the user

Since we don’t have the database we are maintaining the user data in an array.

class UserService {   
 User[] users = new User[5];   
 store(User user) { // store user in the array }  
 findAll() { // returns the array }   
}

How to handle exception

You need to handle exceptions using try catch, however the code can propagate a checked exception to the caller so that caller will be notified at the compilation time to handle the exception

Scenario 1:   
User findByName(String name) {   
 try {   
 if(..) { throw check-exception }  
 } catch (…) { }  
}

Scenario 2:

User findByName(String name) throws checked-exception {   
 if(…) { throw checked-exception }  
}

You can’t throw predefined exceptions because they don’t look valid for all the business requirements, hence you must create your own checked exceptions by extending Exception class

class UserNotFoundException extends Exception {   
 UserNotFoundException(String message) { … }   
}

We can use this constructor with String to pass error messages so that caller can get those error messages.

If(…) {   
 throw new UserNotFoundException(“User with “+name+” is not found”);  
}

Note: You must create all the exception classes related to your requirement in a separate package like com.npci.exceptions

Database

* Types of SQL
* Joins
* Views
* Stored Procedures
* Indexes

SQL: Structured Query Language which is used to communicate the RDBMS

SQL types

1. DDL (Data Definition Language): CREATE, ALTER, DROP, TRUNCATE
2. DML (Data Manipulation Language): INSERT, DELETE, UPDATE
3. DQL/DRL (Data Query/Retrieval Language): SELECT
4. TCL (Transaction Control Language): COMMIT, ROLLBACK, SAVEPOINT
5. DCL (Data Control Language): GRANT, REVOKE

Constraints: These are rules applied on the table & their data

1. PRIMARY KEY: No null values & must be unique
2. UNIQUE: Supports null values
3. NOT NULL: Needs value mandatorily
4. CHECK: Conditions on the value like gender must be male or female, age must be between 18 to 60
5. FOREIGN KEY: To reference the primary key of a table

SQL Queries

mysql> create database npci\_db;

Query OK, 1 row affected (0.00 sec)

mysql> use npci\_db;

Database changed

mysql> create table employees(employee\_id int primary key, employee\_name varchar(15), phone\_number int unique);

Query OK, 0 rows affected (0.04 sec)

mysql> insert into employees values(500, 'Alex', 99008800);

Query OK, 1 row affected (0.01 sec)

mysql> insert into employees values(501, 'Bruce', null);

Query OK, 1 row affected (0.00 sec)

mysql> insert into employees values(502, 'Charles', null);

Query OK, 1 row affected (0.00 sec)

mysql> select \* from employees;

+-------------+---------------+--------------+

| employee\_id | employee\_name | phone\_number |

+-------------+---------------+--------------+

| 500 | Alex | 99008800 |

| 501 | Bruce | NULL |

| 502 | Charles | NULL |

+-------------+---------------+--------------+

3 rows in set (0.00 sec)

mysql> update employees set phone\_number=8899888 where employee\_id=501;

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> update employees set phone\_number=8899888 where employee\_id=502;

ERROR 1062 (23000): Duplicate entry '8899888' for key 'employees.phone\_number'

mysql> update employees set phone\_number=8899898 where employee\_id=502;

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> alter table employees add column salary double;

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> select \* from employees;

+-------------+---------------+--------------+--------+

| employee\_id | employee\_name | phone\_number | salary |

+-------------+---------------+--------------+--------+

| 500 | Alex | 99008800 | NULL |

| 501 | Bruce | 8899888 | NULL |

| 502 | Charles | 8899898 | NULL |

+-------------+---------------+--------------+--------+

3 rows in set (0.00 sec)

mysql> alter table employees add column email\_id varchar(20), add column pan varchar(10);

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> desc employees;

+---------------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------------+-------------+------+-----+---------+-------+

| employee\_id | int | NO | PRI | NULL | |

| employee\_name | varchar(15) | YES | | NULL | |

| phone\_number | int | YES | UNI | NULL | |

| salary | double | YES | | NULL | |

| email\_id | varchar(20) | YES | | NULL | |

| pan | varchar(10) | YES | | NULL | |

+---------------+-------------+------+-----+---------+-------+

6 rows in set (0.01 sec)

mysql> alter table employees drop column pan, drop column email\_id, drop column salary;

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> desc employees;

+---------------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------------+-------------+------+-----+---------+-------+

| employee\_id | int | NO | PRI | NULL | |

| employee\_name | varchar(15) | YES | | NULL | |

| phone\_number | int | YES | UNI | NULL | |

+---------------+-------------+------+-----+---------+-------+

3 rows in set (0.00 sec)

mysql> select \* from employees;

+-------------+---------------+--------------+

| employee\_id | employee\_name | phone\_number |

+-------------+---------------+--------------+

| 500 | Alex | 99008800 |

| 501 | Bruce | 8899888 |

| 502 | Charles | 8899898 |

+-------------+---------------+--------------+

3 rows in set (0.00 sec)

mysql> select databases();

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'databases()' at line 1

mysql> select database();

+------------+

| database() |

+------------+

| npci\_db |

+------------+

1 row in set (0.00 sec)

mysql> select PI();

+----------+

| PI() |

+----------+

| 3.141593 |

+----------+

1 row in set (0.00 sec)

mysql> select now();

+---------------------+

| now() |

+---------------------+

| 2023-10-13 17:44:02 |

+---------------------+

1 row in set (0.00 sec)

mysql> select current\_date;

+--------------+

| current\_date |

+--------------+

| 2023-10-13 |

+--------------+

1 row in set (0.00 sec)

Day wise agenda

Day1: MySQL

Day2: MySQL & Collection Framework

Day3: Threads, Filehandling, JDBC

Day4: ORM & Spring

Day5: Spring Boot

Day6 & Day 7: Microservices

Day8 & Day9: React.js

Day 2 Agenda

Joins

Views

Procedures

Index

Joins:

It is used to join multiple tables based on some conditions

Ex: Student & Marks table

Ex: User & Roles table

Ex: Employee & Department table

Syntax:

Select column\_names from table1 join table2 on condition

[or]

Select table\_name.column\_names from table1 join table2 on condition

[or]

Select alias.column\_names from table1 alias join table2 alias on condition

Views: It is a virtual table created from the select statements

Syntax: create view view\_name as (query)

Select \* from view\_name runs the query provided while creating view

Index: it helps you to search the data quickly, because it uses binary search

Syntax:

Create table students(column, … index index\_name(column));

Alter table students add index index\_name(column));

Ex: alter table students add index student\_name(name));

Select \* from students where name = “Yuvraj”;

The above query uses index to search as name is configured with an index

Stored Procedures

These are reusable programs that you can store & call whenever you need.

Syntax:

Create procedure procedure\_name(arguments, arguments)

Statements;  
Statements;

Note: Since ; is terminal statement & its used in the procedures for terminating multiple statements we must use a different delimiter to terminate the procedure

delimiter $ - now $ is the delimiter to end

Loosely coupled applications with design patterns

Whenever the client code depends on another code it has to create an object, but if the new implementation of another code should be used, then client code must also be changed, hence client should always use interfaces at their side & abstract the object of its implementations through some design patterns like factory, singleton

Note: Create all the design pattern classes in a separate package like com.npci.utility

Drawbacks of an array

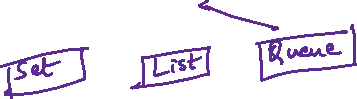
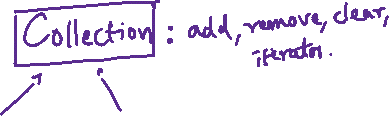
1. Size is fixed
2. It can only store similar data-types
3. No inbuilt algorithms to maintain data like maintaining unique elements, maintaining elements in sorted order, maintaining elements in FIFO, maintaining elements in random order and so on.

Collection Framework

These provide set of API’s to maintain the data in various way

* Maintaining duplicate elements, unique elements, in sorted order, random order
* You get inbuilt methods to add, remove & iterate
* Size grows and shrinks at run time as and when you add or remove elements

Collection Framework hierarchy



Set: It maintains only unique elements

List & Queue: They can maintain duplicates, List is index based, Queue maintains elements that needs to be removed either in FIFO or Sorted order

Set implementation

HashSet: It maintains the elements in random order, its retrieval is faster compare to other implementations

LinkedHashSet: it maintains the elements in insertion order

TreeSet: It maintains the elements in sorted order

Set API internal implementation

Set.add(obj) >> obj.hashCode() == if object collides >> obj.equals(obj2) adds it if equals is false else ignores it

Set.add(obj) >> obj.hashCode() == if doesn’t collide >> adds it

Test cases: These helps you to determine whether your methods are working as per your requirement.

Note: Currently we are testing save, findAll, findUserByName using main method, but we can write test cases for each methods.

Test Cases will give Green report if test passes else gives Red report when the test fails

How to test the code

Suppose we have a Calculator class

class Calculator {   
 public int add(int x, int y) { return x + y; }   
}

From main: Calculator c = new Calculator(); c.add(20, 30);

From test case Calculator c = new Calculator();

int result = c.add(20, 30);

assertEquals(50, result); // Green report or Red report

Activity 1

Create a Profile that will have the following properties: profileId, name, password, dob, phone.

Use the layered architecture and in the service layer you must be able to perform following operations

1. save
2. delete

Maintain the Profile in a Collection framework in the service layer.

Write the test cases for save, delete and also for not saving the duplicate profile.

Design the classes & interfaces accordingly, write the code such that it must be loosely coupled apply appropriate design patterns so that the client must able to get the multiple implementation of the service layer without changing the code

Activity 2

To the existing Profile activity provide a login method in the service layer which must accept profileId & password on a valid profileId and password it must return the Profile else throw a checked exception

Ensure you have a test case to pass profileId and password that expects Profile on those inputs or assert the error messages you are expecting when profileId or password is wrong.

Day Agenda

Comparable & Comparator

Spring Framework

TreeSet: It is used to store elements in sorted order

Note: Complex objects can’t be sorted unless it implements Comparable & Comparator

Note: Simple types can be sorted automatically

Hello, Test, Demo, Apple, Grapes – String

15, 7, 3, 8, 9, 12 – Integer

String, Integer and other classes implements Comparable.

Implementing Comparator

Comparator<String> comparator = new Comparator<String>() {   
 public int compare(String s1, Strings s2) { return s1.compareTo(s2)}  
};

Implementing with Lambda expression

Comparator<String> comparator2 = (a, b) -> b.compareTo(a) // return b.compareTo(a) // -ve goes towards right side

Pass the comparator to the TreeSet

Set<String> string1 = new TreeSet() // uses Comparable of String  
Set<String> string2 = new TreeSet(comparator2) // uses Comparator

Note: You can directly pass the lambda expression as below

Set<String> string3 = new TreeSet( (a, b) -> b.compareTo(a) );

Comparable vs Comparator

|  |  |
| --- | --- |
| Comparable | Comparator |
| It is implemented in the same class that needs to be sorted  Ex: class Employee implements Comparable<Employee> Ex: class String implements Comparable<String> | It is implemented in a separate class |
| It has a method compareTo(T t) | It has a method compare(T t1, T t2) |
| It is for natural ordering as it is part of the class that needs to be sorted | It is to customize the natural ordering into various types of sorting  Ex: sort based in ascending / descending & sort based on other properties like price, ratings |

Spring Framework

Framework: It is a semi-implemented application with lot of common features which every application needs, you don’t have to write codes for those common features like-

1. Design patterns
2. Handling all the predefined checked exceptions
3. Creating database connections & closing their resources
4. Type conversions

Spring Framework: It is an application framework used to develop various types of applications for different platforms like web, mobile, cloud, desktop and so on

The main thing in spring framework is Spring Container

Spring Container: It is an environment where spring creates & maintains all the reusable objects and supplies those objects to another objects if required (Supplying Dao to the Service)

Without spring how this Dependency Injection works

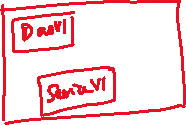
// Dao layer code  
class EmployeeDaoV1 implements EmployeeDao { }   
  
//Service layer code

class EmployeeServiceV1 implements EmployeeService {   
 EmployeeDao dao = factory.getInstance();  
}



With spring how this dependency injection works

// Dao layer code  
class EmployeeDaoV1 implements EmployeeDao { }   
//Service layer code



class EmployeeServiceV1 implements EmployeeService {   
 EmployeeDao dao;  
 setDao(EmployeeDao dao) { this.dao = dao; } // spring supplies the DAO object  
}



Two important things in spring framework

1. Spring Container or IoC (Inversion of Control): It takes care of maintaining the beans(objects)
2. Dependency Injection: process of supplying an object to another object

Ex: Dao is supplied to Service, then service is supplied to controller

Spring framework modules: These are to add features to your application

1. Spring Context: It takes care of all the design patterns & DI
2. Spring Web: It is to create REST & Web based applications
3. Spring Data JPA: It is to interact with the database using ORM framework
4. Spring Boot: It is to automate the spring features in your application
5. Spring Cloud: It is to create microservices
6. Spring Security: It is to apply Authentication & Authorization features in the application

Maven project

Dependencies: spring-context, junit

How to get the object from the spring container

1. Using ApplicationContext context = new ClassPathXmlApplicationContext(“beans.xml”);

context.getBean(“b1”);

1. Using Dependency Injection

EmployeeDao: save, findById, findAll, remove methods

EmployeeDaoListImpl: Use List<Employee> to maintain employee

TestCases: To test all the 4 methods

Employee: id, name, salary with constructors, setters & getters

Day 4 Agenda

ORM Framework

Spring Boot & Annotation configurations

ORM (Object Relational Mapping):

It is a Java framework for the database which takes care of

* converting Java objects to SQL objects & vice versa
* establishing connections to the database based on the datasource information’s & closing them when not in use
* separates the application logic from the database configuration logic(url, username, password, driver-class)
* ORM users HQL/JPQL(database independent) instead of SQL queries (which are database dependent)
* ORM takes care of generating the SQL queries based on the database its connected to
* HQL/JPQL: Hibernate Query / Java Persistence Query language: They are the queries written for the java objects which are mapped to the table

How to create an object that maps to the table?

Entity class must be created which is a Java bean with setters, getters, constructors but maps to the database table

Table employee – id(int - primarykey), name(varchar), birthday(date), salary(double)

@Entity // convert java to sql & sql to java  
@Table(name = “employee”) // optional if table & class names are same  
class Employee {   
 @Column(name = “id”) // optional if column name is same as variable name  
 @Id // to determine primary key variable   
 int employeeId;  
 @Column(name = “name”) // optional because variable & column names are same.  
 String name;  
 @Column(name = “birthday”)  
 LocalDate dob; // maps to birthday column  
 double salary; // maps to salary column  
}

ORM – crud operations without using any queries

Spring with ORM framework

1. Spring-orm library – classes & interfaces to perform CRUD operations
2. Hibernate-core library - @Entity, @Table, @Id and so on
3. Spring-context library
4. Mysql-connector-j library – provides jdbc driver to interact with mysql database
5. XML file with – datasource information – username, password, url, driver-class name
6. HibernateTemplate – it is an object that provides save, delete, find methods, it uses the database connection created in the spring container

Note: HibernateTemplate object is created in spring-container, we must create <bean> for it

Steps to interact with the database

1. Create employee table
2. Add necessary libraries in the pom
3. Configure XML with DriverManagerDataSource, LocalSessionFactoryBean, HibernateTemplate, EmployeeDaoORMImpl
4. Create EmployeeDaoORMImpl that performs the CRUD operations via HibernateTemplate
5. Entity class - @Entity, @Table, @Id, @Column
6. Main method that acts like a Service layer and needs an object of the DAO

Hibernate Template has methods to perform CRUD operations

1. save(T t): saves the entity
2. get(ID, T): returns the entity based on the primary

Business layer / Service layer

It will have business logics which are run before or after accessing the DAO

1. adding one or more entities
2. transactions
3. calculating some results like applying tax on the total price
4. throwing the exceptions on some conditions

Service layer must have an interface that is used by the controller

1. Create a checked exception EmployeeNotFoundException
2. Create an interface EmployeeService that will have methods
   1. void createEmployees(Employee employee1, Employee… employee);
   2. Employee createEmployee(Employee employee);
   3. Employee findEmployee(int id) throws EmployeeNotFoundException
   4. void deleteEmployee(int id) throws EmployeeNotFoundException
   5. List<Employee> findEmployeesByName(String name);
   6. List<Employee> findEmployees();
3. Implement EmployeeService that depends on EmployeeDao

DDD: Domain Driven Design helps you to analyze the requirements and create models, tables, interfaces as per the business needs.

Note: You will not be creating the DAO layer implementation however you will create DAO layer interface in the Spring Boot because Spring Boot provides an auto-implementation for the DAO layer

As per the DDD we need to create following resources

1. Table & their relationship between other tables
2. Models – Java Beans / Entity classes
3. DAO interface – this will have CRUD methods
4. Service interface – this will have all the operations client wants to do

Table creation

1. Profile / User – profileId, name, emailId, password, dob, phone, wallet\_id\_ref, account\_ref
2. Wallet – wallet\_id, account\_number, wallet\_amount
3. Account – account\_number, amount
4. Contact – contact\_id, profile\_id\_ref , wallet\_id\_ref

Note: When Profile is registered, Account & Wallet entries should be created

Account table – account\_number(primary key), amount

Wallet table – wallet\_id(primary key), account\_number(foreign key), wallet\_amount

Profile table – profile\_id(primary key), name, email\_id, password, dob, phone, wallet\_id\_ref(foreign key), account\_ref(foreign key)

Contact – contact\_id(primary key), profile\_id\_ref(foreign key), wallet\_id\_ref(foreign key)

Spring annotation configurations/ Java configurations

It simplifies the bean configurations with the help of annotations instead of declaring the beans in the XML file

XML based configuration

<bean id = “b1” class = “com.EmployeeDaoImpl”>  
 <property name = “template” ref = “hibernateTemplate” /> <!—setTemplate(hibernateTemplate) -->  
</bean>

<bean id = “b2” class = “com.EmployeeServiceImpl”>  
 <property name = “employeeDao” ref = “b1” /> <!—setEmployeeDao(b1) -->  
</bean>

Annotation based configuration

Spring provides annotations to mark the class which needs to be maintained by spring container i.e., spring container creates object of such classes

@Repository – write this on the dao classes

@Service – write this on the service classes

@RestController – write this on the controller classes

@Component – write this on the class which may not belong to any layer, but spring wants to maintain its instance

@Repository(name = “xyz”) // spring container creates the object

// bean id = xyz or employeeOrmImpl if name is not provided  
public class EmployeeOrmImpl implements EmployeeDao {   
 @Autowired // injects the object matching to the type, you don’t need setter method  
 HibernateTemplate template;   
}   
  
@Service // bean id = employeeServiceImpl  
public class EmployeeServiceImpl implements EmployeeService {   
 @Autowired // injects the object matching to the type  
 private EmployeeDao dao;  
}

XML >> DriverManagerDataSource >> HibernateTemplate

Note: These annotations work only if you have written   
 <context:component-scan base-package=”com”/>

Spring Boot

It provides an environment such that you can quickly create product-grade spring applications

Spring Boot takes care of automatically configuring your applications based on the library you add in your class path, you don’t need xml file instead you will use property files

Things spring boot can automatically configure

1. Component Scan starts automatically – so that any class having component type annotations like @Repository, @Service, @RestController, @Component will be instantiated in the spring-container
2. Provides web server for web application internally which is called as embedded server(Tomcat by default), you can change your embedded servers to JBoss(Undertow), Eclipse(Jetty)
3. Supplies the dependencies as per your library – Suppose you want to interact with the Database, then spring boot takes care of configuring DriverManagerDataSource, LocalSessionFactoryBean, HibernateTemplate, TransactionManager everything automatically
4. Spring Boot also takes care of implementing the DAO layer for you, so that you can only create Service & Controller
5. Spring Boot can download the right spring library version based on the spring-boot-parent project version

List of libraries provided in spring-boot to auto-configure the application

1. Spring Boot Starter Web: Component Scan, Web application
2. Spring Data JPA: Configuration for the database

Spring Boot uses application.properties: It is a property file to configure your application like datasource properties, server properties, cloud properties

Note: Spring.io(Spring Initializr) website provides you an empty spring boot project template, from which you can create spring-boot project for your requirement

Spring Data JPA

It provides auto-implementation for DAO layer, you need to extend the interface it provides

1. CrudRepository<T, ID>
2. JpaRepository<T, ID> extends CrudRepository<T, ID>

These interfaces has crud methods like save(T t), deleteById(ID id), findById(ID id), findAll()  
T: entity class  
ID: primary key class

You must only extend any one of these interfaces

public interface EmployeeDao extends JpaRepository<Employee, Integer> { }

Now spring data jpa implements save(Employee t), deleteById(Integer id), findById(Integer id), findAll() methods automatically so that when you call them they perform CRUD operations on Employee entity

@Autowired  
private EmployeeDao employeeDao; // in the service layer

employeeDao.save(emp); // emp will be mapped to employee table

List<Employee> list = employeeDao.findAll();  
Optional<Employee> op = employeeDao.findById(200);

op.get(); // returns employee or null  
op.isPresent(); // returns true if data is present

Things to create

1. Employee entity
2. EmployeeDao extends JpaRepository<Employee, Integer>
3. EmployeeService & EmployeeServiceImpl (already present in the previous project)
4. EmployeeController who can handle requests from any type of clients and return response to them

Webservices:

These are online services which helps in sharing the data between heterogenous applications

Ex: Java to C# to C++ to Python to Javascript and so on

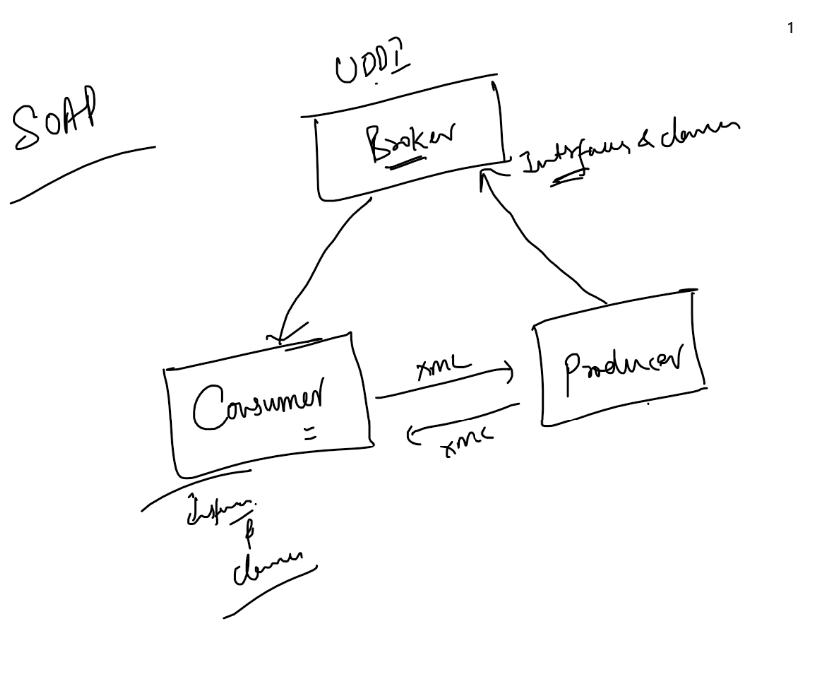
There two types of Webservices

1. SOAP webservice (Simple Object Access Protocol)
2. ReST based webservice (Representational State Transfer)

SOAP webservices

It shares the data in XML format i.e., Java will be converted to XML and XML will be converted C#

* You can integrate only enterprise applications in SOAP

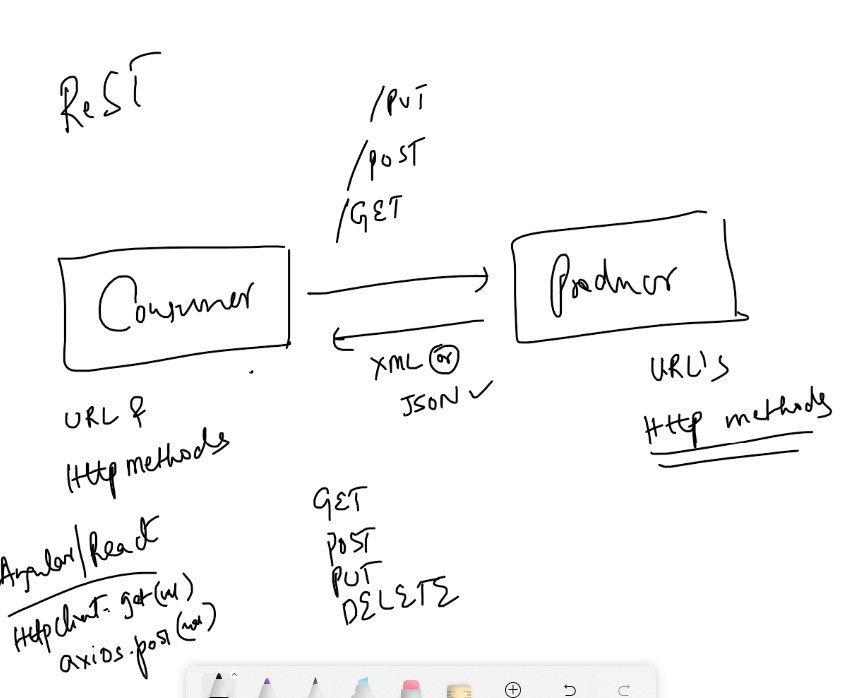


UDDI: Uniform Description Discovery Integration, which is used to register the interfaces & classes so that Consumer can use them to send/receive the data to/from the Producer , the data will be XML format

ReST webservices

It shares the data in various formats like JSON, XML, TEXT, HTML, CSV, but more preferred one is JSON

* You can integrate any types of applications like sharing data from mobiles, web and any other devices



ReST webservice uses HTTP methods to specify the type of operation

i.e., CRUD operations

1. GET : Retrieval

2. POST : Create / Store new resource

3. PUT : Update the resource

4. DELETE : Delete the resource

Every webservice must have URL & HTTP methods which consumers will use to access

Webservice:   
@GetMapping(“/find”)  
public Employee findEmployee() {   
 return …;  
}

Consumer:

http.get(“/find”): sends HTTP GET request to the /find service

In Spring how to create a webservice

@RestController  
@RequestMapping(“/api”)  
public class EmployeeController {   
   
 @GetMapping(“/employees”)  
 public ResponseEntity<Object> getEmployees() { service.findAll(); }  
   
 @PostMapping(“/employees”)  
 public ResponseEntity<Object> storeEmployee(Employee employee) { service.createEmployee(employee); }  
}

<http://ip:port/api/employees> GET >> getEmployees

<http://ip:port/api/employees> POST >> storeEmployee

Association Mappings

One to One Mapping : @OneToOne: Employee & EmployeeID

One to Many Mapping: @OneToMany: State & City

Many to One Mapping: @ManyToOne: City & State

Many to Many Mapping: @ManyToMany: Student & Course

public class Employee {   
 id, name, salary  
 @OneToOne  
 private Address address;   
 @OneToMany // looks for the table having multiple entries of contact for the same employee  
 @JoinTable(name = “contact”,….)  
 private List<Contact> contactsList;  
}

When you store employee, then address table also will be updated

employeeDao.save(employee);

Activity 3:

Implement contact transaction application backend with all the necessary DAO, Service & Controller

1. Create tables
2. Create entity classes & map the relationships like One to One & One to Many
3. Create Service interfaces that will have all the operations you want to do
4. Create an implementation for all the service interfaces you create
5. Create webservices with appropriate URL’s & HTTP methods that will perform all the operations through Postman

Writing our own queries in DAO which returns the result based on some condition

You need to write JPQL which is a query for the entity not for the table.

Examples

Select e from Employee e: It selects all the entities of Employee

Select e from Employee e where e.name = ?1: This takes a value to the ?1 and applies a where condition on the name property of Employee entity

Select e from Employee e where e.email = ?1 and e.password = ?2: This takes 2 values for ?1 & ?2 to get an entity that matches to email & password properties

Microservices:

These are small independent services from other services which are loosely coupled

Advantages

1. You can create the services that are independent from other services
2. If any service go down it doesn’t affect the other services
3. You can scale a particular service you want
4. You can create the services from any technology/language you want
5. Microservices are webservices hence there wouldn’t be any problem in exchanging the data between other services

Design patterns in Microservice

1. Service Discovery: It is a program that registers all the microservices with their instance-ids & physical address
2. Client Side Load Balancer: It is a program that resolves the physical address of the remote service using their instance-id by querying the service discovery
3. Circuit Breaker: It is a program that breaks the request flow whenever is required
4. Distributed Configuration
5. Distributed Tracing
6. Security

Note: All the above design patters are implemented by spring framework using a library called spring-cloud

Service Discovery: It is a program to register all the microservices

Spring Cloud gives you a program which is called as Eureka Server that acts like Service Discovery

Libraries required

1. Eureka Server
2. Dev tools

Microservice: It is a webservice that will register to the service discovery with an instance-id

Libraries required

1. Eureka Client (but this is part of spring-cloud, hence you need to add spring cloud version in the pom.xml)

application.properties

spring.application.name = instance-id-name

Steps to develop service discovery & microservice

Service Discovery

1. Eureka Server – need to add this library
2. @EnableEurekaServer – need to be added on the main class
3. Disable client features – fetch-registry & register with eureka must be made false

Microservice

1. Eureka Client – this automatically registers & fetches registry
2. application.properties – instance-id

Second Microservice: This can also be registered in the service discovery

Libraries –

1. Eureka Client
2. Web
3. Dev tools

Intercommunication between the microservices

When a client microservice service wants to access the remote microservice it must use remote microservice instance-id

In Spring Boot you will use RestTemplate object to access the remote service

RestTemplate: It is the class that helps to make HTTP calls it provides various methods for HTTP methods   
  
@Bean  
@LoadBalanced  
public RestTemplate createTemplate(RestTemplateBuilder builder) {   
 return builder.build();  
}  
  
@Autowired // spring supplies RestTemplate  
RestTemplate rest;  
  
//http GET request

rest.getForObject(“http://EMPLOYEE-APP/api/findAll “) >> LoadBalanced queries the Service Discovery & determines the physical address of EMPLOYEE-APP >> and prepares the request as   
http://ip:port/api/findAll

4 tables

Account: account\_number(primary key), balance

create table account(account\_number int primary key auto\_increment, balance double)auto\_increment=5000;

Wallet: wallet\_id(primary key), amount, account\_number\_ref(foreign key)

create table wallet(wallet\_id int primary key auto\_increment, amount double, account\_number\_ref int, foreign key(account\_number\_ref) references account(account\_number))auto\_increment=1000;

Profile: profile\_id(primary key), name, email, dob, phone, wallet\_id\_ref(foreign key)

create table profile(profile\_id int primary key auto\_increment, name varchar(15), emailId varchar(20), password varchar(20), dob date, phone bigint, wallet\_id\_ref int, foreign key(wallet\_id\_ref) references wallet(wallet\_id));

Contact: contact\_id(primary key), profile\_id\_ref(foreign key), wallet\_id\_ref(foreign key)

create table contact(contact\_id int primary key auto\_increment, profile\_id\_ref int, wallet\_id\_ref int, foreign key(profile\_id\_ref) references profile(profile\_id), foreign key(wallet\_id\_ref) references wallet(wallet\_id))auto\_increment=500;

@Entity  
public class Account {   
 int accountNumber; // @Id, @GeneratedValue,..  
 double balance;  
}  
@Entity   
public class Wallet {   
 int walletId; // @Id, @GeneratedValue  
 double amount;  
 @OneToOne  
 @JoinColumn(“accNum”)  
 private Account account;  
   
}  
@Entity  
public class Profile {   
 int id; // @Id,…  
 @OneToOne  
 @JoinColumn(“walletRef”)  
 private Wallet wallet;  
}  
// while storing profile, we need to have wallet & account object to update those tables

Wallet wallet = new Wallet(100);  
 Account account = new Account(3000);  
 wallet.setAccount(account);  
 profile.setWalelt(wallet);  
save(profile); // profile.wallet & profile.wallet.account

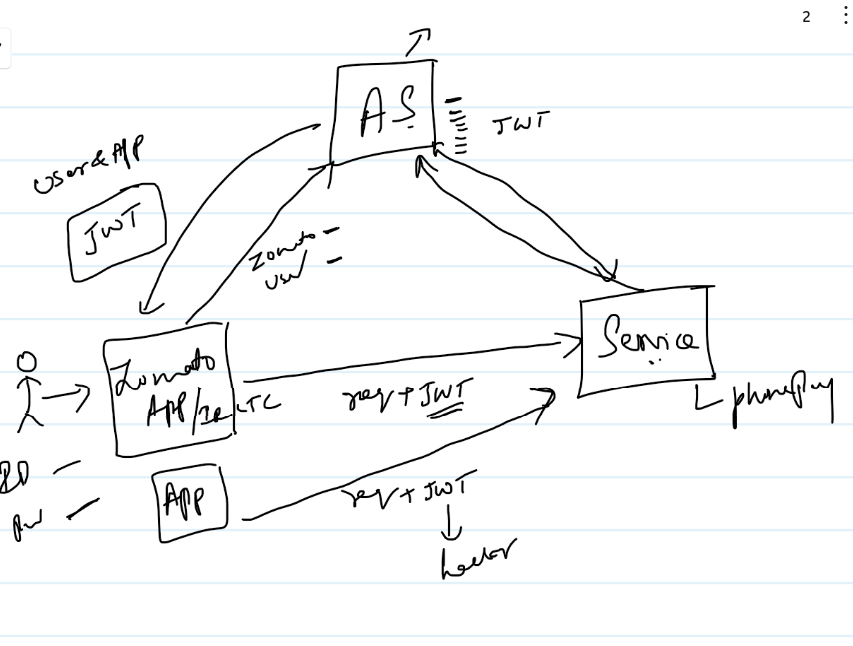
class Contact {   
 int contactId// @id  
 int profileId // maps to profile ref  
 int walletId // maps to wallet ref  
}  
@Entity  
class Profile {   
 …  
@OneToMany(…)  
 @JoinTable(name = “contact”, joinColumns = @JoinColumn(name = “profile\_ref”),   
 inverseJoinColumns = @JoinColumn(name = “wallet\_ref”))  
 List<Contact> contacts;  
}

Securing Microservices

Microservices uses token based authentication & authorization to access the microservice

Authentication Service: Identifying the right user/application

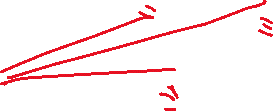
Authorization service: Identifying whether the user has right permission to access the service



Any application who wants to access a microservice must send the JWT to the service, the service will send it to the auth-server to validate, if the auth-server was the one who generated token then it specifies the token is valid so that service can allow the applications to access it else it says unauthorized

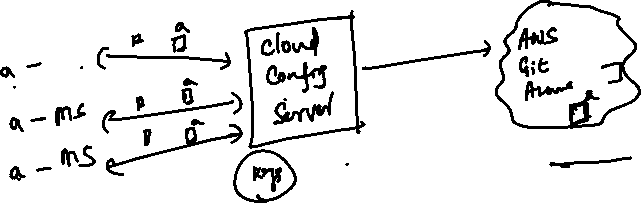
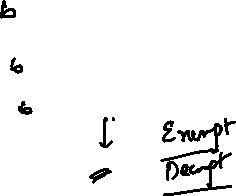
Distributed Cloud Configuration

It is used when multiple microservices wants to share common configurations



Encryption: Sensitive informations must be stored in encrypted format, so that no body should able to decrypt it except the program that is encrypted

Spring uses Cloud Configuration Server to perform encryption & decryption



Things to create

1. Cloud Config Server
2. Microservice that use Cloud Config Server to fetch the configuration file

Creating Config Server

1. Config Server library
2. @EnableConfigServer should be used
3. application.properties : use the GIT URL that will have the configuration file

Creating microservice

1. Spring Cloud Starter Config, Eureka Client, Devtools, Web
2. application.properties:
   1. Use the url of the config server
   2. Use the name of the configuration file the config server wants to fetch

React.js

It is a Javascript library to create UI components to develop single page applications at the front-end

Components: These are reusable UI’s which you can independently create & add any where you want

Technologies React.js use

1. HTML
2. JSX: Javascript XML : It is an extended form of Javascript which helps writing HTML in Javascript in an easier way

Note: Browser don’t understand JSX, hence React.js uses a transpiler called “Babel” which converts JSX to Javascript which is understood by browser.

Pre-requisites

1. HTML
2. New features of Javascript (ES6 features)

Software’s requirement

1. Node.js
2. VS Code

Creating react project

npx create-react-app project-name

Note: Don’t use underscore for project name

npm start: It is used to launch the react application, you will get a default welcome page that you can change as per your requirement.

Features you get from the react project created using create-react-app

1. Embedded light server: runs in 3000 port
2. Babel library to convert JSX to Javascript
3. Auto-compilation feature when you save the changes in your application
4. Live re-load of your output
5. React & ReactDOM library which helps to create UI’s in React.

Sharing data from one component to another component

In React.js we use props to share the data from one component to another component

<Hello name = “Raj” age = “30”/>

Here Hello component will receive the props : {name : “Raj”, age : “30” }

function Hello(props) {   
 // props.name  
 // props.age  
}

Dynamically passing the data

let userName = dynamicValue;  
let userAge = dynamicValue;

<Hello name = {userName} age = {userAge} />

Hello(props) {   
 props.name  
 props.age  
}

Component will have 2 types of data

1. props: read-only
2. state: read-write both

useState(..): It is an inbuilt React function called React hook which helps you to create a state & a function to update the state.

let [id, setId] = useState(“”)

Here id, setId are the two properties of the state, where id is a variable to store the data, setId is a method to modify the variable.

setId(200); then id value will be 200

let [users, setUsers] = useState([ ]);

Initial value of users is an empty array

Event handling: These are the callback functions that needs to be called when you want to update the state

React Router:

It helps you to navigate from one component to another component without reloading the entire page

<BrowserRouter>

<App />

</BrowserRouter>

Link: It helps you to create a clickable link which sends the path to the router

<Link to = “/login”>Login</Link>  
<Link to = “/register”>Register</Link>

Routes & Route  
<Routes>  
 <Route path = “/login” element = { <LoginComponent /> } />  
 <Route path = “/register” element = { <RegisterComponent /> } />  
 <Route path = “/success” element = { <SuccessComponent /> } />  
</Routes>

List of components provided react router

1. BrowserRouter
2. Link
3. Routes
4. Route

Note: react router is a library which we need to install using npm i react-router-dom

Dynamic path parameters

<Route path = “/success/:id” element = { <SuccessComponent /> } />

It can accept any urls that start with success/100, success/200, success/300

<Route path = “/success/:id/\*” element = {<SuccessComponent /> } />  
It can accept any URLs that start with success/100/wallet, success/200/wallet, success/200/addContact

Some of the use-full bootstrap classes

.container-fluid  
.container  
.btn btn-primary  
.btn btn-danger  
.btn btn-secondary  
.row  
.col  
.text-center  
.text-danger  
.text-success  
.form-control

List of components

1. ProfileRegistration : form
2. ProfileLogin: form
3. ProfileSuccess: child components like Dashboard, AddContact, ViewContacts, Wallet

useParams: It is a function which helps you to read the path parameters

/success/:emailId matches to below

[/success/abc@gmail.com](mailto:/success/abc@gmail.com)

[/success/xyz@gmail.com](mailto:/success/xyz@gmail.com)

let { emailId } = useParams();

/success/:emailId/:profileId matches to below

[/success/abc@gmail.com/123](mailto:/success/abc@gmail.com/123)

[/success/xyz@gmail.com/456](mailto:/success/xyz@gmail.com/456)

let { emailId, profileId } = useParams();

axios: It is used to access backend webservices, it sends HTTP requests with different HTTP methods

axios.get(url).then( callback ).catch( callback )

axios.post(url, body).then( callback ).catch( callback )

axios.put(url, body).then( callback ).catch( callback )

axios.delete(url, body).then( callback ).catch( callback )

callback is an arrow function that is executed based on the response status code, if 2xx then() is executed, if 4xx catch() is executed

CORS (Cross Origin Resource Sharing): This error you will get when cross domain are sharing the data, you need to enable the cors at the backend so that CORS is allowed

@CrossOrigin( origins = { “ip”, “ip”, … }) // only set of ip’s can share the set

Topics

1. Exception Handling & Collection Framework
2. MySQL
3. Spring Core – ApplicationContext, ClassPathXmlApplicationContext, layered architecture, scope, annotations
4. Spring Boot – Webservices – REST & SOAP
5. Microservices – Service Discovery, Load Balancer, Circuit Breaker, Distributed Configuration & Encryption
6. React.js – Components, props, state, routing, axios, useState, useParams, useEffect