**How do you implement logger in the project?**

logging.config=classpath:logback-expresconnect-shipping.xml

customize exception in your project?

* Generic compile time exception which will be thrown from the source code in case of compile time exception
* CustomExceptionHandler **extends** ResponseEntityExceptionHandler

get error message.

String errorDescription = this.messageSource.getMessage(String.valueOf(errorcode), null, locale);

**Approach 1. Within your Controller method.**

@RequestMapping("/car/{id}")

public ResponseEntity<?> getCar(@PathVariable String id) {

Car car = carService.getCar(id);

if (car == null) {

ErrorResponse errorResponse = new ErrorResponse();

errorResponse.setMessage("Record not found");

return new ResponseEntity<>(errorResponse, HttpStatus.NOT\_FOUND)

}

return new ResponseEntity<>(car, HttpStatus.OK);

}

**Approach 2: Handle exceptions globally.**

Step 1: Create NotFound exception class and extend to RunTime Exception.

public class NoRecordFoundException extends RuntimeException {

public NoRecordFoundException() {

super();

}

}

@RestControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(NoRecordFoundException.class)

@ResponseStatus(HttpStatus.NOT\_FOUND)

@ResponseBody

public ErrorResponse handleNoRecordFoundException(NoRecordFoundException ex)

{

ErrorResponse errorResponse = new ErrorResponse();

errorResponse.setMessage("No Record Found");

return errorResponse;

}

Step 3: throw Not found exception from your controller or service:

@RequestMapping("/car/{id}")

public ResponseEntity<?> getCar(@PathVariable String id) {

Car car = carService.getCar(id);

if (car == null) {

throw new NoRecordFoundException();

}

return new ResponseEntity<>(car, HttpStatus.OK);

}

Response code?

INTERNAL\_SERVER\_ERROR(500, "Internal Server Error"),

SERVICE\_UNAVAILABLE(503, "Service Unavailable"),

INTERNAL\_SERVER\_ERROR(500, "Internal Server Error"),

BAD\_GATEWAY(502, "Bad Gateway")

Java coding Questions:

<https://examhelperrr.blogspot.com/2021/10/wipro.html>

**What is** [**Autowired**](https://www.java67.com/2019/04/top-10-spring-mvc-and-rest-annotations-examples-java.html)**?**

[Autowired](https://www.java67.com/2019/04/top-10-spring-mvc-and-rest-annotations-examples-java.html)provides us Object initialization. Autowired used to find the object which is already created in spring container.

**@****Transactional(readonly=true)?**

* Spring will set the JDBC transaction into a read-only mode,
* @Transactional annotation is used when you want the certain method/class(=all methods inside) to be executed in a transaction.
* Let's assume user A wants to transfer 100$ to user B. What happens is:

1. We decrease A's account by 100$
2. We add 100$ to B's account

Let's assume the exception is thrown after succeeding 1) and before executing 2). Now we would have some kind of inconsistency because A lost 100$ while B got nothing. Transactions means all or nothing. If there is an exception thrown somewhere in the method, changes are not persisted in the database. Something called rollback happens.

If you don't specify @Transactional, each DB call will be in a different transaction.

**What are @Transactional Propagation Levels used for?**

When looking at the Spring source code, you’ll find a variety of propagation levels or modes that you can plug into the @Transactional method.

@Transactional(propagation = Propagation.REQUIRED)

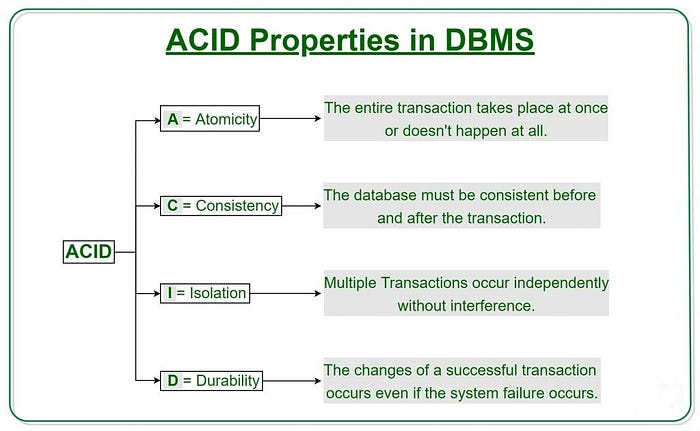
*// or*

@Transactional(propagation = Propagation.REQUIRES\_NEW)

*// etc*

The full list:

* REQUIRED
* SUPPORTS
* MANDATORY
* REQUIRES\_NEW
* NOT\_SUPPORTED
* NEVER
* NESTED



**More Idea :**

<https://www.marcobehler.com/guides/spring-transaction-management-transactional-in-depth>

**Caching**?

is a mechanism to enhance the performance of a system. It is a temporary memory that lies between the application and the persistent database.

**Submit soap Request?**

Need to know host url and wsdl file.

JPA?

JPA over JDBC is that, in JPA, data is represented by objects and classes while in JDBC data is represented by tables and records.

**Immutable classes?**

are those classes, whose [object](http://javarevisited.blogspot.com/2012/12/what-is-object-in-java-or-oops-example.html) can not be modified once created, it means any modification on an immutable object will result in another immutable object.

public final class ImmutableReminder{

    private final Date remindingDate;

    public ImmutableReminder (Date remindingDate) {

        if(remindingDate.getTime() < System.currentTimeMillis()){

            throw new IllegalArgumentException("Can not set reminder” +

                        “ for past time: " + remindingDate);

        }

        this.remindingDate = new Date(remindingDate.getTime());

    }

public Date getRemindingDate() {

        return (Date) remindingDate.clone();

    }

}

<https://medium.com/@cs.vivekgupta/everything-about-immutable-classes-in-java-9f5fe8e6ca54>

class final Student {   
 private final int id;  
 private final String name;  
 private final int deptNo;  
 private Student(int id, String name, int deptNo) {  
 this.id = id;  
 this.name = name;  
 this.deptNo = deptNo;  
 }  
 public int getId() {  
 return id;  
 }  
 public String getName() {  
 return name;  
 }  
 public int getDeptNo() {  
 return deptNo;  
 }  
 public static Student with(int id, String name, int deptNo) {  
 return new Student(id, name, deptNo);  
 }  
}

public Student(int id, String name, int deptNo, List<String> courses) {  
 this.id = id;  
 this.name = name;  
 this.deptNo = deptNo;  
 this.courses = new ArrayList<>(courses); // DO THIS.  
}

Comparable vs comparator:

A comparable object can compare itself with another object. The class itself must implements the **java.lang.Comparable** interface to compare its instances.

**java.util.Comparator** interfaces to sort array/list of custom classes.

package com.journaldev.sort;

import java.util.Comparator;

public class Employee implements Comparable<Employee> {

private int id;

private String name;

private int age;

private long salary;

public int getId() {

return id;

}

public String getName() {

return name;

}

public int getAge() {

return age;

}

public long getSalary() {

return salary;

}

public Employee(int id, String name, int age, int salary) {

this.id = id;

this.name = name;

this.age = age;

this.salary = salary;

}

@Override

public int compareTo(Employee emp) {

//let's sort the employee based on an id in ascending order

//returns a negative integer, zero, or a positive integer as this employee id

//is less than, equal to, or greater than the specified object.

return (this.id - emp.id);

}

@Override

//this is required to print the user-friendly information about the Employee

public String toString() {

return "[id=" + this.id + ", name=" + this.name + ", age=" + this.age + ", salary=" +

this.salary + "]";

}

/\*\*

\* Comparator to sort employees list or array in order of Salary

\*/

public static Comparator<Employee> SalaryComparator = new Comparator<Employee>() {

@Override

public int compare(Employee e1, Employee e2) {

return (int) (e1.getSalary() - e2.getSalary());

}

};

/\*\*

\* Comparator to sort employees list or array in order of Age

\*/

public static Comparator<Employee> AgeComparator = new Comparator<Employee>() {

@Override

public int compare(Employee e1, Employee e2) {

return e1.getAge() - e2.getAge();

}

};

/\*\*

\* Comparator to sort employees list or array in order of Name

\*/

public static Comparator<Employee> NameComparator = new Comparator<Employee>() {

@Override

public int compare(Employee e1, Employee e2) {

return e1.getName().compareTo(e2.getName());

}

};

}

Here is the separate class implementation of Comparator interface that will compare two Employees object first on their id and if they are same then on the name.

package com.journaldev.sort;

import java.util.Comparator;

public class EmployeeComparatorByIdAndName implements Comparator<Employee> {

@Override

public int compare(Employee o1, Employee o2) {

int flag = o1.getId() - o2.getId();

if(flag==0) flag = o1.getName().compareTo(o2.getName());

return flag;

}

}

**Sort a List With Chain of Comparators**

In the last example, we want to have starred movie at the top and then sort by rating.

1

List<Movie> movies = Arrays.asList(

2

new Movie("Lord of the rings", 8.8, true),

3

new Movie("Back to the future", 8.5, false),

4

new Movie("Carlito's way", 7.9, true),

5

new Movie("Pulp fiction", 8.9, false));

6

​

7

movies.sort(Comparator.comparing(Movie::getStarred)

8

.reversed()

9

.thenComparing(Comparator.comparing(Movie::getRating)

10

.reversed())

11

);

12

​

13

movies.forEach(System.out::println);

**Exception**:

<https://www.mygreatlearning.com/blog/exception-handling-in-java/>

class Main {

public static void main (String[] args) {

try{

System.out.println(4/0);

try{

int[] a={1,2,3};

System.out.println(a[3]);

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println("Out of bounds");

}

}

catch(ArithmeticException e)

{

System.out.println("ArithmeticException : divide by 0");

}

}

}

**Output:**

ArithmeticException: Divide by 0

**Note – If we put code of outer try before inner try, then if an exception occurred, it will ignore the entire inner try and move directly to its catch block.**

**How to debug stream? Peek method.**

List<String> result = Stream.of("EURO/INR", "USD/AUD", "USD/GBP", "USD/EURO") .filter(e -> e.length() > 7) .peek(e -> System.out.println("Filtered value: " + e)) .map(String::toLowerCase) .peek(e -> System.out.println("Mapped value: " + e)) .collect(Collectors.toList());  
  
Read more: <https://www.java67.com/2016/09/java-8-streampeek-example.html#ixzz7OBSDFKzy>

**If I** **remove @springbootApplication, what will happen?**

**Why do I need to use spring cloud config just for accessing properties? it creates performance issue? Is there any way to handle it?**

**How to get specified column only from one entity?**

**Example: employee has address, I need only empid,emp name,city(address entity). How do I get it? @Notnull on the field.**

# [**Difference between FetchType LAZY and EAGER in Java Persistence API?**](https://stackoverflow.com/questions/2990799/difference-between-fetchtype-lazy-and-eager-in-java-persistence-api)

**How to convert the below json object into java pojo. Steps?**

{

"name":"John",

"age":30,

"cars":[ "Ford", "BMW", "Fiat" ]

}

How to convert this below mutable class into immutable class?

**publicfinalclass**User {

**privatefinal**String username;

**privatefinal**String password;

**private**String firstname;

**private**String lastname;

**private**String email;

private List<String> addressList;

**public**User(String username, String password) {

**this**.username = username;

**this**.password = password;

    }

**public**String getUsername() {

**return**username;

    }

**public**String getPassword() {

**return**password;

    }

**public**String getFirstname() {

**return**firstname;

    }

**publicvoid**setFirstname(String firstname) {

**this**.firstname = firstname;

    }

**public**String getLastname() {

**return**lastname;

    }

**publicvoid**setLastname(String lastname) {

**this**.lastname = lastname;

    }

**public**String getEmail() {

**return**email;

    }

**publicvoid**setEmail(String email) {

**this**.email = email;

    }

**public**String getAddressList () {

**return**addressList;

    }

**publicvoid**setEmail(List<String>addressList) {

**this**.addressList = addressList;

    }

}

**@PropertySources (For Multiple Property Locations)**

@Configuration

**@PropertySources({**

**@PropertySource("classpath:/com/dev/javatechonline/app1.properties"),**

**@PropertySource("classpath:/com/dev/javatechonline/app2.properties")**

**})**

public class MyClass { }

### @Value for default Value

Suppose we have not defined a property in the properties file. In that case we can provide a default value for that property. Here is the example:

**@Value("${emp.department:Admin}")**

private String empDepartment;

## @SpringBootApplication (@Configuration + @ComponentScan + @EnableAutoConfiguration)

### Use of ‘exclude’ in @EnableAutoConfiguration

@Configuration

**@EnableAutoConfiguration(exclude={WebSocketMessagingAutoConfiguration.class})**

public class MyWebSocketApplication {

public static void main(String[] args) {

...

}

}

## @ConfigurationProperties

**dev.name=Development Application**

**dev.port=8090**

**dev.dburl=mongodb://mongodb.example.com:27017/**

**dev.dbname=employeeDB**

**dev.dbuser=admin**

**dev.dbpassword=admin**

Now, create a bean class with getter and setter methods and annotate it with @ConfigurationProperties.

**@ConfigurationProperties(prefix="dev")**

public class MyDevAppProperties {

private String name;

private int port;

private String dburl;

private String dbname;

private String dbuser;

private String dbpassword;

//getter and setter methods

}

Here, Spring will automatically bind any property defined in our property file that has the prefix ‘*dev’*and the same name as one of the fields in the MyDevAppProperties class.

Next, register the *@ConfigurationProperties* bean in your *@Configuration* class using the *@EnableConfigurationProperties* annotation.

@Configuration

**@EnableConfigurationProperties(MyDevAppProperties.class)**

public class MySpringBootDevApp { }

How to restrict access

**The below to refer spring boot:**

<https://www.javainuse.com/spring/SpringBootInterviewQuestions>

Similarly, why we should not make entity class final?

The subclass overrides all the methods of the parent, and when any of the methods are accessed, the proxy loads the real object from the DB and calls the actual method on that object. If **we make entity class** as **final** then that **entity class** can´t be subclassed because **final classes** can´t be inherited.

Similarly, can we extend entity class? **Entity** Inheritance. **Entities** support **class** inheritance, polymorphic associations, and polymorphic queries. **Entity classes can extend** non-**entity classes**, and non-**entity classes can extend entity classes**. **Entity classes can** be both abstract and concrete.

Also to know, what impact does making an entity bean final in hibernate?

Yes, a **Hibernate Entity** class **can** be declared **final**, however it is not a good practice. **Hibernate** uses the proxy pattern for performance improvement during lazy association, by **making an entity final**, **Hibernate** will no longer be able to use a proxy as **Java** doesn't allow the **final** class to be extended.

#### Loubna Simonetti

**Professional**

## [What will happen if we don't have no args constructor in Entity Bean?](https://everythingwhat.com/open-detail/393968A4)

**what will happen if we don't have no**-**args constructor in entity bean**? newInstance() is used for creating the instance of **Entity beans** which requires **no**-**args constructor**. So **if we** won't **have no**-**args constructor in entity beans**, hibernate **will** fail **to** instantiate it and **we will get** HibernateException.

#### Isacia Sarrionandia

**Professional**

## [What is hibernate proxy and how it helps in lazy loading?](https://everythingwhat.com/open-detail/393968A5)

A **proxy** is a subclass implemented at runtime. **Hibernate** creates a **proxy** (a subclass of the class being fetched) instead of querying the database directly, and this **proxy** will **load** the "real" object from the database whenever one of its methods is called.

#### Laira Obaldia

**Professional**

## [How many layers are there in hibernate architecture?](https://everythingwhat.com/open-detail/393968A6)

four layers

#### Sheron Rezac

**Explainer**

## [What are different types of bean injections?](https://everythingwhat.com/open-detail/393968A7)

**It exits in two major types :**

* Setter Injection.
* Constructor Injection.

#### Upa Pedersen

**Explainer**

## [What is JPA specification?](https://everythingwhat.com/open-detail/393968A8)

The Java Persistence API (**JPA**) is a Java **specification** for accessing, persisting, and managing data between Java objects / classes and a relational database. **JPA** was defined as part of the EJB 3.0 **specification** as a replacement for the EJB 2 CMP Entity Beans **specification**. **JPA** also requires a database to persist to.

#### Barb Engelschall

**Explainer**

## [What is abstract entity?](https://everythingwhat.com/open-detail/393968A9)

Noun. 1. **abstract entity** - a general concept formed by extracting common features from specific examples. **abstraction**. **entity** - that which is perceived or known or inferred to have its own distinct existence (living or nonliving)

#### Tempie Tevs

**Pundit**

## [What is MappedSuperclass?](https://everythingwhat.com/open-detail/393968A10)

@Target(value=TYPE) @Retention(value=RUNTIME) public @interface **MappedSuperclass**. Designates a class whose mapping information is applied to the entities that inherit from it. A **mapped superclass** has no separate table defined for it.

#### Antima Vassaio

**Pundit**

## [What is a discriminator column?](https://everythingwhat.com/open-detail/393968A11)

**Discriminator**. The **discriminator column** is always in the table of the base entity. It holds a different value for records of each class, allowing the JPA runtime to determine what class of object each row represents. The **DiscriminatorColumn** annotation represents a **discriminator column**.

Difference between Put and post method?

Invoke multiple db from a microservices?

<https://www.kindsonthegenius.com/microservices/multiple-database-configuration-for-microservice-in-spring-boot/> ?

# [**How can I create a memory leak in Java?**](https://stackoverflow.com/questions/6470651/how-can-i-create-a-memory-leak-in-java)

**Wipro Questions:**

Table

Description automatically generated with medium confidence

Explain about @configuarion, @autowire, @qualifier annotations in spring internal working of Rest?

What is Static initializer block?

* It will be run only one time before the constructor or main class.

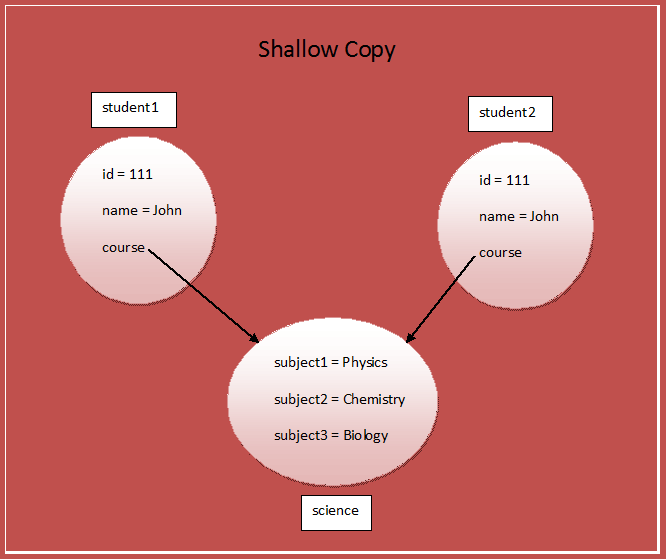
Where the static blocks are stored?

* Metaspace

Text

Description automatically generated

<https://javaconceptoftheday.com/difference-between-shallow-copy-vs-deep-copy-in-java/>



**Describe A Problem You Faced And How You Deal With It?**

We migrated mainframe application into MS java application.

1. Understand requirement and Design doc
2. Converted into java app from mainframe app
3. Design Architecture
4. Understand MS pattern

**How Concurrent HashMap Works** ?

Diagram

Description automatically generated

Text

Description automatically generated

- Rehashing

- collision

- load factor

**how hashmap works internally in java?**

**<https://codingnconcepts.com/top-spring-boot-interview-questions/>**

**<https://github.com/learning-zone/spring-interview-questions>**

Graphical user interface

Description automatically generated with medium confidence

A picture containing graphical user interface

Description automatically generated

Text

Description automatically generated

# **Circuit Breaker Pattern - Fault Tolerant Microservices**

Diagram

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

CQRS Pattern:

Publishing event by using event handler while doing the below

CREATE,DELETE, UPDATE

While event querying we do normal query to db.

Drawbacks : data inconsistency

**eureka.client.register-with-eureka is mandatory in eureka server?**

Default value of property ‘eureka.client.register-with-eureka’ is true. Please note that this property is mandatory to include in Eureka Server in order to make its value as false.

#### **eureka.client.fetch-registry=false**

This property indicates that Eureka Server is supported to fetch instance details of microservice to make intra-communication between microservices happen. If one microservice wants to communicate with another microservice by using Eureka then inside microservice we should add this property (eureka.client.fetch-registry) and set it to true.

**Important Spring boot starters?**

<!-- dependency to support web and restful applications using spring mvc -->

        <**dependency**>

            <**groupId**>org.springframework.boot</**groupId**>

            <**artifactId**>spring-boot-starter-web</**artifactId**>

        </**dependency**>

        <!-- dependency to support eureka server -->

        <**dependency**>

            <**groupId**>org.springframework.cloud</**groupId**>

            <**artifactId**>spring-cloud-starter-eureka-server</**artifactId**>

            <**version**>1.4.6.RELEASE</**version**>

        </**dependency**>

        <!-- dependency to support feign client -->

        <**dependency**>

            <**groupId**>org.springframework.cloud</**groupId**>

            <**artifactId**>spring-cloud-starter-feign</**artifactId**>

            <**version**>1.4.6.RELEASE</**version**>

        </**dependency**>

        <!-- dependency to support ribbon -->

        <**dependency**>

            <**groupId**>org.springframework.cloud</**groupId**>

            <**artifactId**>spring-cloud-starter-ribbon</**artifactId**>

            <**version**>1.4.6.RELEASE</**version**>

        </**dependency**>

        <!-- dependency to support hystrix -->

        <**dependency**>

            <**groupId**>org.springframework.cloud</**groupId**>

            <**artifactId**>spring-cloud-starter-hystrix</**artifactId**>

            <**version**>1.4.6.RELEASE</**version**>

        </**dependency**>

## 1. Difference between PUT and POST

| **PUT** | **POST** |
| --- | --- |
| RFC-2616 clearly mention that PUT method requests for the attached entity (in the request body) to be stored into the server which hosts the supplied [Request-URI](https://restfulapi.net/resource-naming/).  If the Request-URI refers to an already existing resource – an update operation will happen, otherwise create operation should happen if Request-URI is a valid resource URI (assuming the client is allowed to determine resource identifier).  PUT /questions/{question-id} | The POST method is used to request that the origin server accept the entity attached in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.  It essentially means that POST request-URI should be of a collection URI.  POST /questions |
| PUT method is [**idempotent**](https://restfulapi.net/idempotent-rest-apis/). So if we retry a request multiple times, that should be equivalent to a single request invocation. | POST is **NOT idempotent**. So if we retry the request N times, we will end up having N resources with N different URIs created on the server. |
| Use PUT when we want to **modify a singular resource** that is already a part of resources collection.  PUT replaces the resource in its entirety. Use PATCH if request updates part of the resource. | Use POST when you want to **add a child resource under resources collection**. |
| Though PUT is idempotent, we **should not cache** **its response**. | Responses to this method are **not**[**cacheable**](https://restfulapi.net/caching/), unless the response includes appropriate Cache-Control or Expires header fields.  However, the 303 (See Other) response can be used to direct the user agent to retrieve a cacheable resource. |
|  |  |
| Generally, in practice, use PUT for **UPDATE** operations. | Always use POST for **CREATE** operations. |

**The word SOLID acronym for:**

* Single Responsibility Principle (SRP) employe { not dept}
* Open-Closed Principle (OCP) shape example
* Liskov Substitution Principle (LSP) bird,sparrow
* Interface Segregation Principle (ISP).Account->saving ac and Current Ac
* Dependency Inversion Principle (DIP)- car { engine; public car(Engine e) {}} car class depend on engine. If engine will be changed deisol and petrol engine. Car should be updated.

Saga pattern:

Choreography - event based (async)

Orchorestor -central service – coordinates with other service.

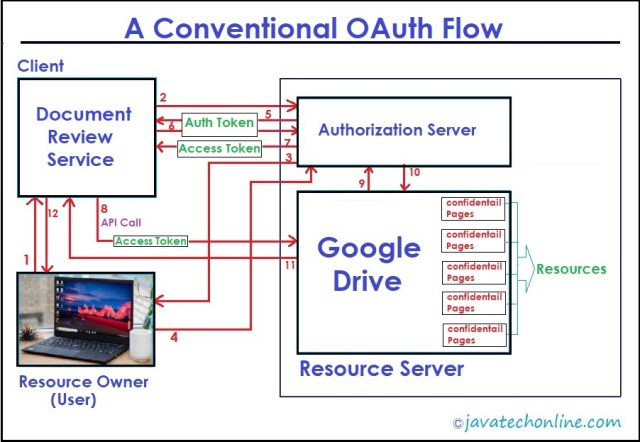
Diagram, Teams

Description automatically generated

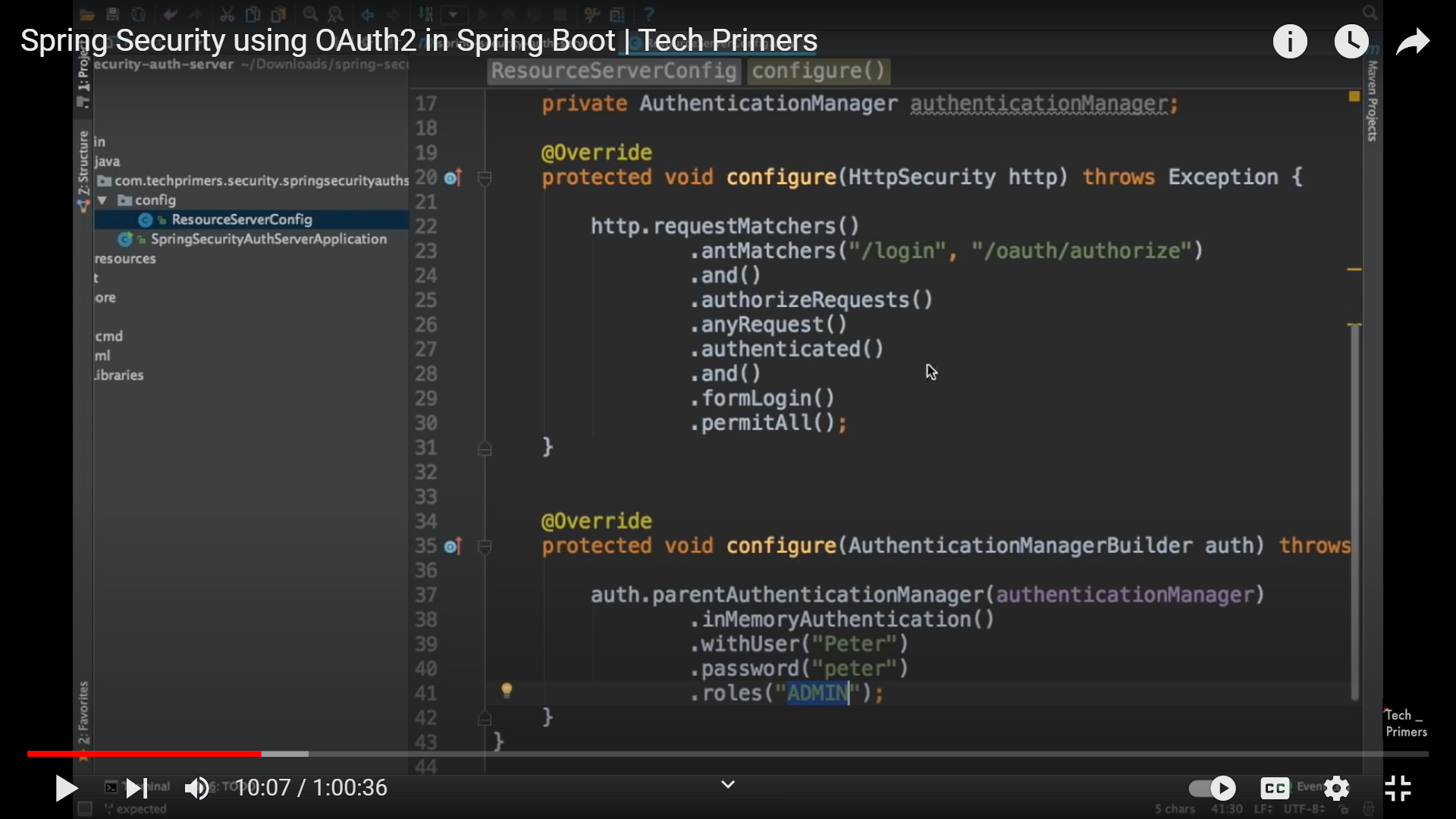
## What is OAuth ?

OAuth is a technique to authorize web applications, servers, devices, APIs etc. via access tokens rather than credentials.

If you want to provide access of an API to 3rd party clients/applications, you should implement OAuth also.



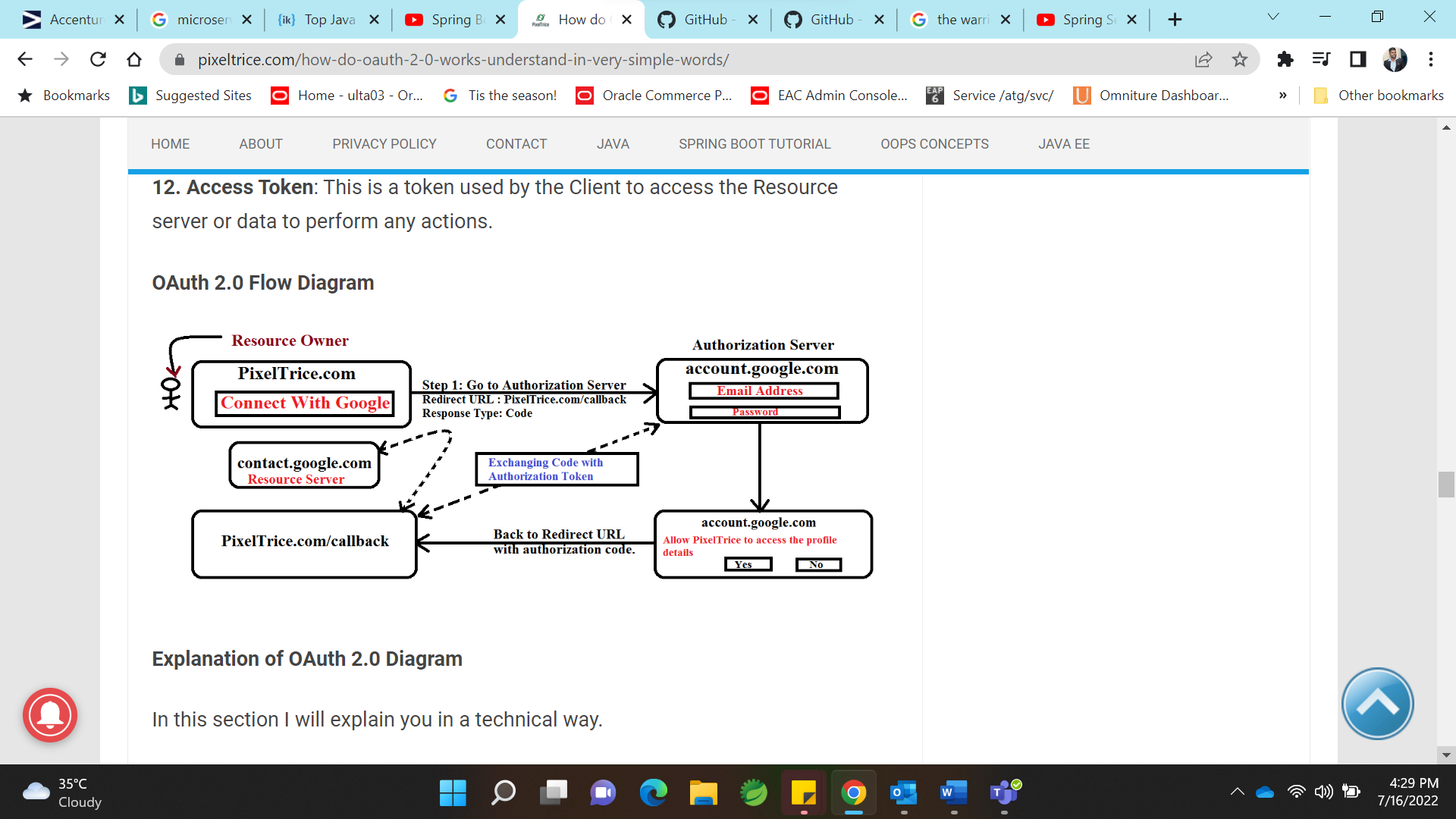
Authenticate -> providing credentials and get roles.



Best oath implementation videos :

<https://www.youtube.com/watch?v=af_2f1rrZdw&t=498s>

<https://www.pixeltrice.com/>



Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

<https://howtodoinjava.com/java/cloning/a-guide-to-object-cloning-in-java/>