Exception Handling

Prepared by

Divyashree.M.D

Associate Software Developer

Exception Handling for API's

Exceptions are the errors that happen at runtime. Exception handling is the technique to handle this runtime error in our application code. If any error is thrown in web API that is caught, it is translated into an HTTP response with status code.

Some of the common exceptions are:

- **IllegalArgumentException**: This exception is thrown when an input argument is invalid, such as a null or empty value.
  HTTP Status Code: 400 Bad Request.
- **NotFoundException**: This exception is thrown when a requested resource is not found, such as a non-existent endpoint or a missing record in a database.
  HTTP Status Code: 404 Not Found.
- **UnauthorizedException:** This exception is thrown when the user is not authorized to perform a certain action, such as accessing a protected endpoint or performing an operation without sufficient permissions.
  HTTP Status Code: 401 Unauthorized.
- **InternalServerErrorException:** This exception is thrown when an unexpected error occurs on the server, such as a database connection failure or a system-level error.
  HTTP Status Code: 500 Internal Server Error.
- **TimeoutException:** This exception is thrown when a request takes too long to complete, such as when a connection times out or when a service is unavailable.
  HTTP Status Code: 503 Service Unavailable.
- **NoSuchElementException:** This exception is thrown when an attempt is made to access an element that does not exist in a collection.
- **ArrayIndexOutOfBoundsException:** This exception is thrown when an attempt is made to access an array index that is out of bounds.
- **ClassCastException:** This exception is thrown when an object is cast to an incompatible class.
- **NumberFormatException:** This exception is thrown when a string that is not a valid number is parsed.
- **SQLException:** This exception is thrown when an error occurs while accessing a database.

- **DataAccessException:** This exception is thrown when there is an error accessing data from a database or other data source.
- **RuntimeException:** This is a general exception that is thrown when an unexpected error occurs. It is the superclass of all unchecked exceptions.
- **MethodArgumentNotValidException**: This exception is thrown when a method argument fails validation.
- **MissingServletRequestParameterException:** This exception is thrown when a required request parameter is missing.
- **TypeMismatchException:** This exception is thrown when a method argument or property is of the wrong type.
- **HttpMediaTypeNotSupportedException:** This exception is thrown when a request is made with an unsupported media type.
- **ConversionFailedException:** This exception is thrown when a type conversion fails.
- **SecurityException:** This exception is thrown when a security violation occurs.
- **RemoteAccessException:** This exception is thrown when a remote access error occurs.
- **AccessDeniedException:** This exception is thrown when an authenticated user tries to access a resource or perform an action that they are not authorized to access. This can happen when a user tries to access a resource that they don't have the required permissions for.
- **MissingAuthenticationTokenException:** This exception is thrown when an authentication token is missing or not found. This can happen when a user tries to access a protected resource without providing authentication credentials.
- **InvalidAuthenticationException:** This exception is thrown when authentication fails due to invalid credentials. This can happen when a user provides incorrect credentials during authentication.
- **InvalidGrantException:** This exception is thrown when an OAuth2 grant is invalid. This can happen when a user provides an invalid grant during the OAuth2 authentication process.
- **InvalidTokenException:** This exception is thrown when an authentication token is invalid. This can happen when an authentication token has been tampered with or has expired.
- **TokenExpiredException:** This exception is thrown when an authentication token has expired. This can happen when an authentication token has been issued with an expiration time and that time has passed.

- **BadCredentialsException:** This exception is thrown when authentication fails due to invalid credentials. This can happen when a user provides incorrect credentials during authentication.

Steps to handle exceptions:

- Create an exception class: Define a custom exception class that extends the base Exception class or any of its subtypes such as RuntimeException, IOException, etc.
- Use the @ControllerAdvice annotation: Annotate a class with the @ControllerAdvice annotation to mark it as a global exception handler for the application.
- Define exception handler methods: In the class annotated with @ControllerAdvice, define one or more methods that handle specific types of exceptions. Use the @ExceptionHandler annotation to mark these methods as exception handlers.
- Customize the response: In the exception handler methods, customize the response returned to the client. Set the response status code, add error messages, and return a custom response object.
- Use the appropriate HTTP status code: In the exception handler methods, use the appropriate HTTP status code to indicate the nature of the exception. For example, use 404 Not Found for resource not found exceptions, 401 Unauthorized for authentication exceptions, and 500 Internal Server Error for server-side exceptions.

IllegalArgumentException:

IllegalArgumentException is a runtime exception that is thrown to indicate that a method has been passed an illegal or inappropriate argument. It can be thrown for a variety of reasons, such as passing an invalid argument to a method, or providing an incorrect configuration value. To handle IllegalArgumentException in a Spring application, use a try-catch block to catch the exception and handle it appropriately, such as returning an error response.

```
    @ResponseStatus(HttpStatus.BAD_REQUEST)

    @ResponseBody

public ErrorResponse handleBadRequestException(BadRequestException ex) {

   return new ErrorResponse("BadRequest", ex.getMessage(), "ERR_001");

  }
```

NotFoundException:

NotFoundException is a type of exception that is commonly used in web applications to indicate that a requested resource could not be found. This exception is often used to handle 404 errors in web applications, where a user requests a resource that does not exist, such as a web page or REST endpoint.

```
@ResponseStatus(HttpStatus.NOT_FOUND)
  @ResponseBody
public ErrorResponse handleNotFoundException(NotFoundException ex) {
   return new ErrorResponse("NotFound", ex.getMessage(), "ERR_002");
 }
```

UnauthorizedException:-

UnauthorizedException is an exception that is commonly used in web applications to indicate that a user is not authorized to perform a particular action or access a particular resource. It can be thrown for a variety of reasons, such as when a user attempts to access a protected resource without providing valid authentication credentials, or when a user provides invalid or expired authentication credentials.

```
   @ResponseStatus(HttpStatus.UNAUTHORIZED)
   @ResponseBody
public ErrorResponse handleUnauthorizedException(UnauthorizedException ex) {
   return new ErrorResponse("Unauthorized", ex.getMessage(), "ERR_003");
 }
```

InternalServerErrorException:-

InternalServerErrorException can be thrown for a variety of reasons, such as when an unexpected exception occurs during request processing, or when a third-party service or dependency fails to respond or behaves unexpectedly.

```
@ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
  @ResponseBody
  public ErrorResponse handleInternalServerErrorException(InternalServerErrorException ex)
{
   return new ErrorResponse("InternalServerError", ex.getMessage(), "ERR_005");
 }
```