# Exception Handling in Java

**package** net.javaguides.springboot.exception;

@ResponseStatus(value = HttpStatus.***NOT\_FOUND***)

**public** **class** ResourceNotFoundException **extends** RuntimeException{

**private** String resourceName;

**private** String fieldName;

**private** Long fieldValue;

**public** ResourceNotFoundException(String resourceName,String fieldName,Long fieldValue)

// public ResourceNotFoundException(String resourceName, String fieldName, Long fieldValue){

// super(String.format("%s not found with %s : '%s'", resourceName, fieldName, fieldValue));

// this.resourceName = resourceName;

// this.fieldName = fieldName;

// this.fieldValue = fieldValue;

// }

{

**super**(String.*format*("%s not found with %s : '%s'", resourceName, fieldName, fieldValue));

**this**.resourceName = resourceName;

**this**.fieldName = fieldName;

**this**.fieldValue = fieldValue;

}

}

## Notes:-

1)Here we will get internal server error generally and we want to handle that manually,so from first line we can see @ResponseStatus(value = HttpStatus.***NOT\_FOUND***) used to generate not found Exception.

2) and it will generate respective string.

**super**(String.*format*("%s not found with %s : '%s'", resourceName, fieldName, fieldValue));

3)In Repository the data type must be Optional<user> to use the function.

For example-public Optional<Consumer> findItByEmail(String email);

We can call this function while updating the user where we need to first search the user before updating.

User existingUser=userRepository.findById(user.getId()).orElseThrow(()-> new ResourceNotFoundException(“User”,”id”,userId));

@Override

**public** UserDto updateUser(UserDto user) {

User existingUser = userRepository.findById(user.getId()).orElseThrow(

() -> **new** ResourceNotFoundException("User", "id", user.getId())

);

existingUser.setFirstName(user.getFirstName());

existingUser.setLastName(user.getLastName());

existingUser.setEmail(user.getEmail());

User updatedUser = userRepository.save(existingUser);

//return UserMapper.mapToUserDto(updatedUser);

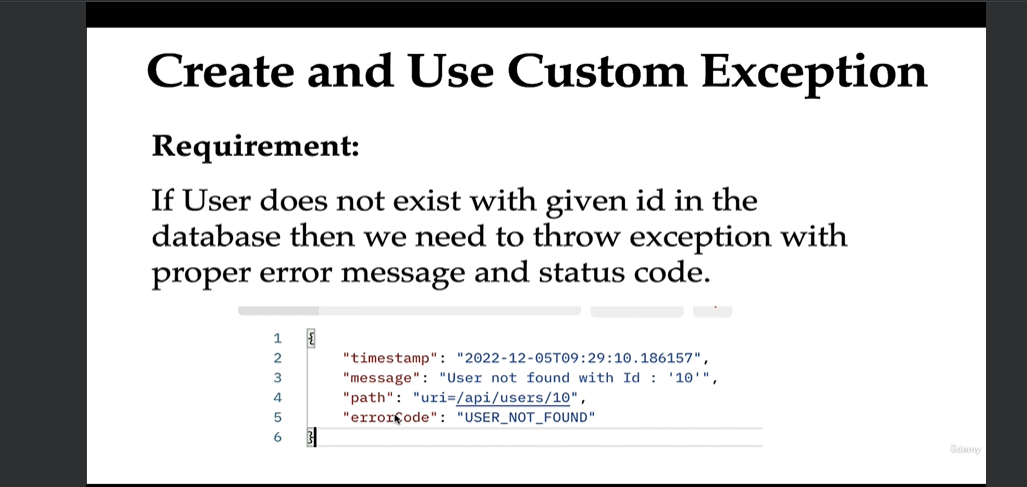
//return modelMapper.map(updatedUser, UserDto.class);

**return** AutoUserMapper.***MAPPER***.mapToUserDto(updatedUser);

}

# Development Steps

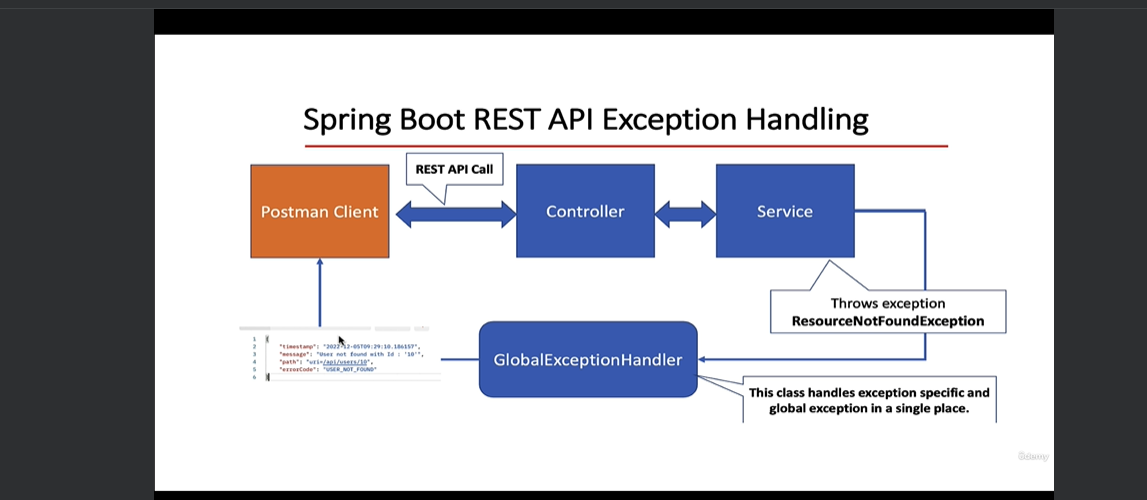
Q)we want Resource not found Exception in an specific format like-



1)Create and use ResourceNotFoundException Custom Exception.

2)Create Error Details class to hold of custom error response.

3)Create GlobalExceptionHandler class to handle specific and global exceptions



If we want to handle exceptions specific to controller than we can write the code to handle exception within controller only.

1)Create error Details class with the parameters mentioned as **private** LocalDateTime timestamp;**private** String message;**private** String path;**private** String errorCode;

With constructors and getters and setters

2)go to controller and specific code to handle specific exceptions

@ExceptionHandler(ResourceNotFoundException2.**class**)

**public** ResponseEntity<ErrorDetails> handleResourceNotFoundException(ResourceNotFoundException2 exception,

WebRequest webRequest){

ErrorDetails errorDetails=**new** ErrorDetails(

LocalDateTime.*now*(),

exception.getMessage(),

webRequest.getDescription(**false**),

"USER\_NOT\_FOUND"

);

**return** **new** ResponseEntity<>(errorDetails, HttpStatus.***NOT\_FOUND***);

}

3)We use @ExceptionHandler annotation to handle an specific exception with in a controller and we use @ControllerAdvice annotation to handle all the exceptions globally.

# EmailAlreadyExistsException:-

Create an exception with in a class.

@ResponseStatus(value = HttpStatus.***BAD\_REQUEST***)**public** **class** EmailAlreadyExistsException **extends** RuntimeException {

**private** String message;

**public** EmailAlreadyExistsException(String message){

**super**(message);}

We can create exceptions globally using GlobalExceptionHandler.class

@ControllerAdvice

**public** **class** GlobalExceptionHandler {

@ExceptionHandler(EmailAlreadyExistsException.**class**)

**public** ResponseEntity<ErrorDetails> handleEmailAlreadyExistsException(EmailAlreadyExistsException exception,

WebRequest webRequest){

ErrorDetails errorDetails=**new** ErrorDetails(

LocalDateTime.*now*(),

exception.getMessage(),

webRequest.getDescription(**false**),

"EMAIL\_ALREADY\_EXISTS "

);

**return** **new** ResponseEntity<>(errorDetails, HttpStatus.***BAD\_REQUEST***);

}

}

We need to use it in service to check if email exists .

**public** **void** saveConsumer(Consumer consumer) {

Optional<Consumer> optionalUser =cRepo.findItByEmail(consumer.getEmail());

**if**(optionalUser.isPresent()) {

**throw** **new** EmailAlreadyExistsException("Email Already Exists for a user");

}

cRepo.save(consumer);

}

We need to take care of Internal Server Error:-

@ExceptionHandler(Exception.**class**)

**public** ResponseEntity<ErrorDetails> handleGlobalException(Exception exception,

WebRequest webRequest){

ErrorDetails errorDetails=**new** ErrorDetails(

LocalDateTime.*now*(),

exception.getMessage(),

webRequest.getDescription(**false**),

"INTERNAL\_SERVER\_ERROR"

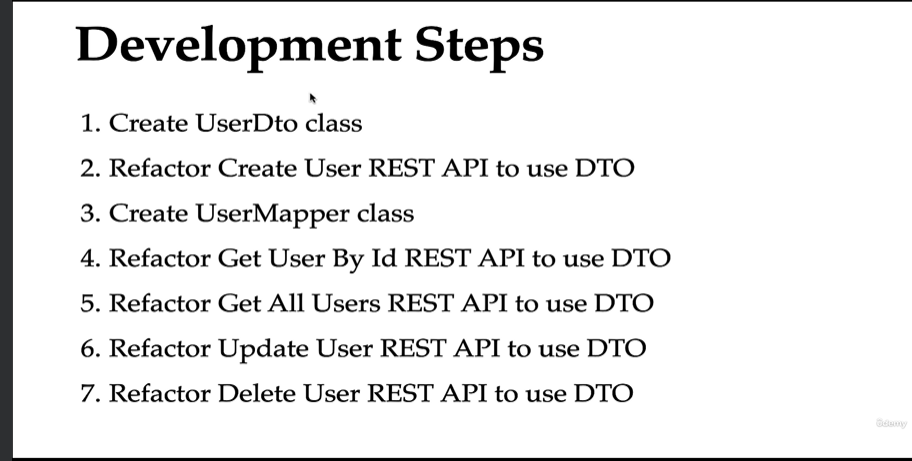
);

**return** **new** ResponseEntity<>(errorDetails, HttpStatus.***INTERNAL\_SERVER\_ERROR***);

}

DTO🡪(Design Transfer Object)

We must not use Jpa Objects directly to pass onto the client. we must use self designed objects for security purposes.



In this we create our personal objects to return to results.