**Images POC**

This is a mini project that retrieves images from a third party web service, processes them based on end user's input and sends them back to the user as a zipped folder which can be downloaded on the user's system

**Technologies:**

Core Java (Muthithreading using ExecutorService, Collections framework, Exception Handling, etc.)

Advanced Java (Servlets, Web services)

Java 8 concepts (Streams ,Limit , forEach ,Lambda Expression ,Functional Interface)

Restful web services (Spring Rest)

Swagger (API Documentation)

**Tools:**

Git repository: https://github.com/mounicakv/ImagesPOC

IDE: Spring Tool Suite

Client: Postman

Server: Apache Tomcat (version 8.5)

Build tool: Maven

Repository: gitbash for pushing in the code changes

Spring version: 4.3.5

Java version: JDK 1.8

**List of Services Exposed**:

* Get All Images From Internet

[/images/retrieveImagesFromInternet](http://localhost:8080/ImagesPOC/swagger-ui.html#/operations/image-controller/getAllImagesFromInternetUsingGET)

Purpose: To retrieve images using RequestParam

* Get Required Images based on keyword

[/images/retrieveCountableImagesFromInternet/count/{countID}](http://localhost:8080/ImagesPOC/swagger-ui.html#/operations/image-controller/getRequiredImagesFromInternetUsingGET)

Purpose: To retrieve images using path variable and request parameter

* Get All Images From Internet Optimized

[/images/retrieveImagesfromInternetOptimized](http://localhost:8080/ImagesPOC/swagger-ui.html#/operations/image-controller/getAllImagesFromInternetOptimizedUsingGET)

Purpose: To retrieve images in parallel using multithreading to achieve faster retrieval of images

* Get All Images From Resource

[/images/retrieveImagesFromResource](http://localhost:8080/ImagesPOC/swagger-ui.html#/operations/image-controller/getAllImagesUsingGET)

Purpose: To retrieve stored images from resources folder in project directory

**Source code details:**

**Package: com.capgemini.imagespoc.controller**

**ImageController Class:**

This class is annotated with "@RestController" to let the spring container know that this is the Controller class and to eliminate the need to annotate every request handling method of the controller class with the @ResponseBody annotation

**Handler methods:**

**getAllImages (HttpServletResponse response):**

This method handles below incoming uri request from client

@RequestMapping (value = "/images/retrieveImagesFromResource", method = RequestMethod.GET, produces="application/zip")

This method calls the method "getImagesFromResource ()" in service class to fetch the stored images in resources folder of project directory and returns this as a byte [] of images

Response is sent back to client using HttpServletResponse object by setting the content type as a zipped folder

**getAllImagesFromInternet (@RequestParam String keyword, HttpServletResponse response)**

This method handles below uri request from client

@RequestMapping (value = "/images/retrieveImagesFromInternet", method = RequestMethod.GET, produces="application/zip")

This method takes Request parameter(String) which is the search keyword and calls the method "getImagesFromInternet(keyword)" in service class by passing the parameter given in the client to obtain all images from third party service for the given input string

This method takes each image from the third party service sequentially and calculates the total time taken to retrieve all images and process them

Response(Image files) is sent back to client using HttpServletResponse object by setting the content type as a zipped folder

**getAllImagesFromInternetOptimized (@RequestParam String keyword, HttpServletResponse response)**

This method handles below uri request from client

@RequestMapping (value = "/images/retrieveImagesfromInternetOptimized", method = RequestMethod.GET, produces="application/zip")

This method takes Request parameter(String) which is the search keyword and calls the method "getImagesFromInternetOptimized" in service class by passing the request parameter given in the client to obtain all images from third party service by using multithreading so that the images are retrieved parallel using threads

The effective time taken for this processing is calculated and the performance is compared against the time taken for images retrieval without using threads

Response (Image files) is sent back to client using HttpServletResponse object by setting the content type as a zipped folder

**getRequiredImagesFromInternet (@RequestParam String keyword, @PathVariable int countID, HttpServletResponse response)**

This method handles below uri request from client

@RequestMapping(value = "/images/retrieveCountableImagesFromInternet/count/{countID}", method = RequestMethod.GET, produces="application/zip")

In this method, a path variable is passed along with request parameter(String)

The path variable is an integer that the user specifies in order to retrieve only that many number of images from the third party service

This method calls the service class method "getRequiredImagesFromInternet (keyword ,countID)", passing the search keyword and count of images to be retrieved and gets the image in byte[]

Response (Image files) is sent back to client using HttpServletResponse object by setting the content type as a zipped folder

**Package: com.capgemini.imagespoc.service**

**ImageServiceImpl Class:**

This class is annotated with "@Service" to let the Spring Container know that this is the service class used

Below are the methods present in this class

**getImagesFromResource ():**

Return type: byte []

Parameter: nil

This method takes the "jpeg" type image files stored in the resources folder and stores them in String [] named files

Calls the method "zipFiles(files)” in the imageConverter helper class by passing the String [] files which in turn returns byte [] format of the passed jpeg image files

This byte [] output files is returned to the calling handler method “getAllImages ()” in controller class

**getImagesFromInternet (String keyword):**

Return type: byte []

Parameter: String

This method calls the retrieveImageFromSplash(keyword) method in same class which takes the passed keyword and returns the images list by accessing the third party webservice "http://www.splashbase.co/api/v1/images/search?query={queryparam}"

Images that are returned by the third party service are stored in ImageList object (DTO class) that holds the received images data in a list

If the obtained images list is not empty then, the images list is taken as a Stream and the images url present for each image is taken and passed to method "invokeImage(img.getUrl (), img.getId())" present in the same class which reads each image based on url and writes the image into Image IO stream and returns 1 or 0 depending on the success or failure of writing images to ImageIO

This method then calls the method "zipFiles ()", in the imageConverter helper class which in turn returns byte [] of images

This byte [] output files is returned to the calling handler method in controller class

**getImagesFromInternetOptimized (String keyword):**

Return type: byte []

Parameter: String

This method also calls the retrieveImageFromSplash(keyword) method in same class which takes the passed keyword and returns the images list by accessing the third party webservice "http://www.splashbase.co/api/v1/images/search?query={queryparam}"

Images that are returned by the third party service is stored in ImageList object (DTO class) that holds the received images data in a list

If the obtained images list is not empty then,

Size of the images received from third service is stored in count variable

Number of threads to be created are taken based on the core processors available in system. This is taken from Runtime class's method "Runtime.getRuntime().availableProcessors ()"

ExecutorService is created using the Executors newFixedThreadPool () factory method. This creates a thread pool with the thread count configured as above

Threads are created using ExecutorService interface as below

ExecutorService es = Executors.newFixedThreadPool (threadCount)

Minimum images to be stored in a thread is calculated by dividing the count of images to the thread count created

Maximum images to be stored in a thread is obtained by adding 1 to the minimum number of images

Thread with maximum images stored is obtained by “count - (threadCount \* minItemsPerThread)”

Thus we have threads created by configuring the images to be held by each thread

For every thread created, a sublist of images is created from Images list and assigned to each thread

The submit(Callable) method is invoked and the Callable's result is stored in a Map of Future objects

Each future object is then obtained using get ()

This method then calls the method "zipFiles ()", in the imageConverter helper class which in turn returns byte[] of images

This byte [] output files is returned to the calling handler method in controller class

**getRequiredImagesFromInternet (String keyword, int count)**

Return type: byte []

Parameters: String,Integer

This method also calls the retrieveImageFromSplash(keyword) method in same class which takes the passed keyword and returns the images list by accessing the third party webservice "http://www.splashbase.co/api/v1/images/search?query={queryparam}"

Images that are returned by the third party service is stored in ImageList object (DTO class) that holds the received images data in a list

If the obtained images list is not empty then, the following checks are performed:

If the returned image list size is less than the required number of images specified by user, then the images list is taken as a Stream and the images url present for each image is taken and passed to method "invokeImage (img.getUrl (), img.getId ())" present in the same class which reads each image based on url and writes the image into Image IO stream and returns 1 or 0 depending on the success or failure of writing images to ImageIO

If the returned image list size is greater than count then, limit(count) is invoked to the stream output of images list and then each image is taken and passed to method "invokeImage (img.getUrl(), img.getId())" present in the same class which reads each image based on url and writes the image into Image IO stream and returns 1 or 0 depending on the success or failure of writing images to ImageIO

This method then calls the method "zipFiles ()", in the imageConverter helper class which in turn returns byte [] of images

This byte [] output files is returned to the calling handler method in controller class

**retrieveImageFromSplash(String keyword):**

Return type: ImageList

Parameter: String

RestTemplate object is created

The URL of third party service is assigned to a String variable

Hashmap is created with key and value pairs as String in order to hold queryparam (exposed by third party api) as key and the keyword(value) passed by user

Using the object created for RestTemplate, getForObject (resourceURL, ImageList.class, urlVariables) method is called by passing the third party url string, hashmap of url variables and the response stored in "ImageList" class which holds each image object returned from the third party api

**invokeImage(String imageUrl, int id)**

Return type: Integer

Parameter: String, Integer

This method creates a url object with the url string passed as input and reads the image from url using ImageIO utility's read method

The read image is stored as a BufferedImage object

A new file object is created by specifying the temporary directory path, file name and extension in which each image file will be written as read

The buffered image object is then written to the directory using ImageIO utility's write method

Returns 1 on successful read and write

Returns 0 in case of any exception while reading or writing

**Package: com.capgemini.imagespoc.helper**

**Image Converter class**

This class is annotated with "@Component" to auto detect for dependency injection

Below are the methods in this class:

**zipFiles (String [] files):**

Return type: byte []

Parameter: String []

This method zips the files which are available as part of source files and returns the images as byte []

**zipFiles ():**

Return type: byte []

This method creates a new File object with the path where each image was written when invokeImage (String imageUrl, int id) was called

and saves all the image files created with extension as "jpg". It also creates a string list of file names.

It then calls the method "zipFiles (File directory, List<String> files)" to zip the files and return the images as byte []

This method also removes all the files from the created temporary directory after the byte [] is obtained using method "FileUtils.cleanDirectory (directory)"

**zipFiles (File directory, List<String> files):**

Return type: byte []

This method zips all the files present in particular directory and return images as byte []

**DTO classes:** com.capgemini.imagespoc.dto

**Image.java:**

The class is annotated with @JsonIgnoreProperties(ignoreUnknown=true) so as to take only the required json properties (as data members) from the images returned by third party service

Below are the data members for image object and their corresponding getters and setters

id: int

url: String

large\_url: String

source\_id: String

copyright: String

site: String

**ImageList.java:**

This class is also annotated with @JsonIgnoreProperties (ignoreUnknown=true)

This class contains images list as it's only data member and the corresponding getter and setter method

images: List<Image>

**ImageProcessingTask.java:**

This is the thread's worker class which implements callable interface

Data members:

imgService: ImageServiceImpl

imagesBucket: List<Image>

**Parameterized Constructor:** ImageProcessingTask (ImageServiceImpl imgService, List<Image> imagesBucket)

When the object for this class is created at the time of thread creation, it takes the sublist of images passed from calling method and assigns it to data member "imagesBucket" and assigns the object of calling class to "imgService" data member of this class

**Call () method:**

Return type: Map<Integer,Integer>

This method iterates over each image present in the sublist of imagesList and stores the image id and the success/failure(1/0) status in a hashmap object

This method will run simultaneously for each created thread

Additionally, Swagger classes are configured within the package “com.capgemini.imagespoc.helper” in order to generate the API documentation and for execution of the project by the end user. URL for the same is <http://localhost:8080/ImagesPOC/swagger-ui.html>