LOG LEVEL(DEBUG & INFO) EXAMPLE:

DEBUG is intended for messages that could be useful in debugging an issue.
INFO should contain messages that describe what is happening in the application

```
package com.moglix.payment.listener;

import java.util.UUID;
import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
import javax.servlet.http.HttpServletRequest;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

/**
 * @author manoranjanmahananda
 *
 */
@WebListener
public class LogRequestListener implements ServletRequestListener {
protected static final Logger LOGGER = LoggerFactory.getLogger(LogRequestListener.class);

 public void requestInitialized(ServletRequestEvent servletRequestEvent) {
   LOGGER.debug("++++++++++++ REQUEST INITIALIZED +++++++++++++++++");
   //String requestId=(String) MDC.get("RequestId");
   //LOGGER.info("Request Identifier fetched from MDC:"+requestId);
   HttpServletRequest request=(HttpServletRequest) servletRequestEvent.getServletRequest();
   String requestIdOverRest=request.getHeader("RequestId");
   LOGGER.info("Request Identifier fetched as header parameter over HTTPS rest
call:"+requestIdOverRest);
   if(requestIdOverRest == null){
         String externalRequestId=UUID.randomUUID().toString();
         LOGGER.info("Request Identifier created:"+externalRequestId);
         MDC.put("RequestId", externalRequestId);
         // while service1 hitting service2 using rest call then send requestId as request header info
   }else{
         LOGGER.info("Request Identifier set in MDC");
         MDC.put("RequestId", requestIdOverRest);
   }

         LOGGER.debug("++++++++++++ REQUEST INITIALIZED  END++++++++++++++++");
}

 public void requestDestroyed(ServletRequestEvent servletRequestEvent) {
   LOGGER.debug("------------REQUEST DESTROYED ----------");
   MDC.clear();
```

```
    LOGGER.debug("-------------REQUEST DESTROYED END------------");
  }
}
```

LOG LEVEL(WARN & ERROR) EXAMPLE:

- *ERROR* should contain technical issues that need to be resolved for proper functioning of the system
- *WARN* is best used for temporary problems or unexpected behavior that does not significantly hamper the functioning of the application (ex: failed user login)

```
@Override
	public TransactionalResponse getTransactionalId(Long userId) throws PaymentException {
		LOG.debug("userId:"+userId);
		if(userId == null){
			LOG.warn("User Id cannot be blank:"+userId);
			throw new PaymentException(PaymentError.INVALID_INPUT,"User Id cannot be
blank");
		}
		int count= payMapper.validateUser(userId);
		if (count== 0){
			LOG.warn("User Id does not exist:"+userId);
			throw new PaymentException(PaymentError.INVALID_INPUT,"User Id does not
exist");
		}
		TransactionalResponse response =null;
		try {
			TransactionRequest request= new TransactionRequest(false);
			request.setUserId(userId);
			UUID uuid = Generators.timeBasedGenerator().generate();
			request.setId(uuid.toString());
			payMapper.insertTransactionalDetails(request);
			//Long transaId = payMapper.getTransactioIdforUser(userId);
			LOG.info("TransactionId " + uuid);
			response = new TransactionalResponse();
			response.setTransactionId(uuid);
		}catch(Exception ex){
			LOG.error("Problem occured while inserting TransactionId in transaction_details
table :",ex);
			//ex.printStackTrace();
		}
		return response;
```

```
        }
```

Fatal - Any error that is forcing a shutdown of the service or application to prevent data loss (or further data loss). I reserve these only for the most heinous errors and situations where there is guaranteed to have been data corruption or loss.

UploadProductStaticDataImpl.updateProductActiveStatus{

```
…..
…..

if(failureDetails.isEmpty()){
        savedProduct.setLastUpdatedBy(fileUploadRequest.getUploadedBy());
        savedProduct.setUpdatedOn(new Date());
        logger.info("Going to add product " + savedProduct);
        iProductDataService.addProduct(savedProduct);
        logger.info("Product status updated for " + savedProduct);
        uploadedProduct++;
        if (isDefaultGroup){//update group after product update
                iProductGroup.selectNewVariant(savedProduct.getIdGroup());
        }
        //evict product from cache as it has been changed
        try {
                 logger.info("Going to evict product from cache " + savedProduct.getIdProduct());
                        cacheUtils.evictGroupedProductDetailsById(savedProduct.getIdProduct());
        } catch (Exception e) {
                e.printStackTrace();
                //logger.error("Error while evicting product " + e);
                logger.fatal("Error while evicting product :"+savedProduct.getIdProduct()+ "  from cache
and  error is:“ , e);
       }

fileResponse.adddata(new String[]
{savedProduct.getIdProduct(),savedProduct.getInternalPartNumber(),"Success"});
}

}
```

Minimum parameters to be logged

- Date and time. It doesn’t have to be UTC as long as the timezone is the same for everyone that needs to look at the logs.
- Class Name , Method Name
- Request Identifier (each request would associate with one external id- UUID)
- Arguments of method
- The service name or code, so that you can differentiate which logs are from which microservice or API.
- Stack error in case of exception or error occur.
- The IP address of the client request. This information will make it easy to spot an unhealthy server or identify a DDoS attack.
- User agent at the front end w.r.t level

External Request identifier POC

---

1. **Web.xml Changes(Add logRequestListener configuration in web.xml)**

```
<listener>
        <listener-class>com.moglix.payment.listener.LogRequestListener</listener-class>
 </listener>
```

2. **LogRequestListener Class**

```
package com.moglix.payment.listener;

import java.util.UUID;
import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
import javax.servlet.http.HttpServletRequest;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

/**
 * @author manoranjanmahananda
 *
 */
@WebListener
public class LogRequestListener implements ServletRequestListener {
protected static final Logger LOGGER = LoggerFactory.getLogger(LogRequestListener.class);

 public void requestInitialized(ServletRequestEvent servletRequestEvent) {
   LOGGER.debug("++++++++++++ REQUEST INITIALIZED +++++++++++++++++");
   //String requestId=(String) MDC.get("RequestId");
   //LOGGER.info("Request Identifier fetched from MDC:"+requestId);
   HttpServletRequest request=(HttpServletRequest) servletRequestEvent.getServletRequest();
   String requestIdOverRest=request.getHeader("RequestId");
   LOGGER.info("Request Identifier fetched as header parameter over HTTPS rest
call:"+requestIdOverRest);
   if(requestIdOverRest == null){
        String externalRequestId=UUID.randomUUID().toString();
        LOGGER.info("Request Identifier created:"+externalRequestId);
        MDC.put("RequestId", externalRequestId);
        // while service1 hitting service2 using rest call then send requestId as request header info
   }else{
        LOGGER.info("Request Identifier set in MDC");
        MDC.put("RequestId", requestIdOverRest);
   }

        LOGGER.debug("++++++++++++ REQUEST INITIALIZED  END++++++++++++++++");
```

```
 }

 public void requestDestroyed(ServletRequestEvent servletRequestEvent) {
   LOGGER.debug("-------------REQUEST DESTROYED ------------");
   MDC.clear();
   LOGGER.debug("-------------REQUEST DESTROYED END------------");
 }
}
```

3. **Changes required for logback.xml**

```
<appender name="FILE"
            class="ch.qos.logback.core.rolling.RollingFileAppender">
            <file>/var/log/moglix/online/payment/payment.log</file>
            <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>/var/log/moglix/online/payment/payment-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
                  <timeBasedFileNamingAndTriggeringPolicy
                        class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                        <maxFileSize>16MB</maxFileSize>
                  </timeBasedFileNamingAndTriggeringPolicy>
                  <maxHistory>90</maxHistory>
            </rollingPolicy>

            <append>true</append>
            <encoder>
                  <pattern> -%X{RequestId} -30(%d{dd/MM/yyyy HH:mm:ss.SSS} [%thread] ) %-5level
%logger.%M\(%line\) - %msg%n</pattern>
            </encoder>
      </appender>
```

4.**Changes required for pom.xml**

```
                  <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>servlet-api</artifactId>
                        <version>2.5</version>
                  </dependency>
```