*INTRODUCTION*

**1. INTRODUCTION**

* 1. **Background of the Project**

In the corporate world, each organization has as much as thousands of applications; these would have been typically written in numerous languages which in turn run in diversified platforms. Despite of these complications, if we are still able to maintain these applications and provide the user, the best experience it is because of logging. The process of making an entry in the log file is called logging. These logs generated during the execution of an application, can be recorded in multiple destinations such as a local disk, console, into other sockets or files. Analysis of these log files is necessary later during debugging or enhancement of the respective application.

Logging is crucial for a developer, who has to modify the application, to add new functionality, or modify the existing one, which has been developed years ago. The main components of logging are: Logger, Formatter, and Appender. The Logger captures the information with metadata that has to logged. Formatter formats the object. Appenders publish the log information into a variety of ConsoleAppender, FileAppender, RollingFileAppender, and SocketAppender. Each logger can be associated with different levels of severity such as fatal, error, warning, info, debug, trace in the descending order. So a logger if configured to warning level would log all levels above it. Logger has overhead associated with it, so developer has to be judicious on what is being logged. Log manager creates the logger and manages corresponding configuration files.

In this application, extraction of log files and corresponding analysis is done. In the corporate world, each organization has its datacenters in different geographical locations far from each other, in order to withstand any natural calamities. Each datacenter in turn has numerous severs associated with it. Each application runs in some hosts of the server, produces log files as the application is being executed.

Our application helps the developer and the support team to derive the relevant files from loads of log files by specifying the severity level and the keywords, which can be converted into pie charts, bar graphs, and text files for further debugging, and analysis. This flexible modular application can handle complex queries by matching patterns and can be customized to organizational needs. This application can be invoked whenever necessary.

*SOFTWARE PROJECT PLAN*

**2. SOFTWARE PROJECT PLAN**

**2.1 Existing System**

Splunk is a log analyser tool. It provides facilities like indexing, searching, visualizing, analyzing and reporting of log data. It also provides a way to identify and solve the bugs. But the system is not customised for any organization so it is dearth of any application specific filters.

**2.2 Proposed System**

In the proposed system, we are going to create a customized log analyser tool which can get rid of the complexity of maintaining the application specific logs. The system will be able to filter the logs in real time. The system receives the log files, generated during the user interaction with the application. The log files are analysed and the results are filtered in accordance to the user demand.

Search is the primary way which enables user to navigate data. Search retrieves events from log file, use statistical commands to calculate metrics and generate reports, search for specific conditions within a rolling time window, identify patterns in your data, predict future trends, and so on. Searches can be saved as reports and used to power dashboard panels.

These results can be transformed into multiple formats which include excel, txt, JSON and can be sent to the support team members for further actions. The trend of the application usage can be visualized through various visual aids such as graph, pie chart, bar graph.

*SOFTWARE REQUIREMENT SPECIFICATION*

**3. SOFTWARE REQUIREMENT SPECIFICATION**

**3.1 Functional Requirements**

The application’s functional requirements include the format of the input, functions that an application can do and the format of the output.

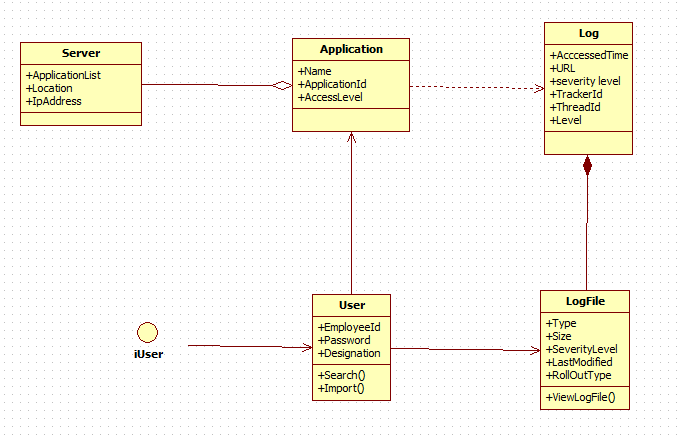
**3.1.1 Input**

The application gets the application name, for which the log file has to be searched, keyword(s) for segregation of those log files.

**3.1.1 Output**

The application upon receiving the application name, keyword to filter the log data, it displays the segregated logs and provides an option to import, as text files or csv.

**3.1.3 Class Diagram**

******

*Fig. no: 3.1.2 Class Diagram for user and system.*

**3.2 NON FUNCTIONAL REQUIREMENTS**

***3.2.1 Performance Requirements***

The key objective is to enable the user an instant access to loads of log file. It helps the developer and the support team to spot the errors and warnings if any and hence proceed with the following debugging.

***3.2.2 Interface Requirements***

The command prompt is used as an interface between the user and system where the user enters the initial key and the input file or text to be encrypted is given.

***3.2.3 Resource Requirements***

* Software Requirements:

Operating System - Windows,Linux

Front End - Web Application

Language - Java

System Type - 32, 64 bit Operating System

Memory - 8 GB RAM

Speed - 3 GHz

* Hardware Requirements:

Processor - Pentium IV or more

RAM - 512 Mb or more

Hard Disk - 50GB or Higher

Monitor - Display Panel (640 x 480)

***3.2.4 Security Requirements:***

The application needs a high level of security because it accesses critical data which includes tracker id, thread id.

***3.2.5 Quality and Reliability Requirements:***

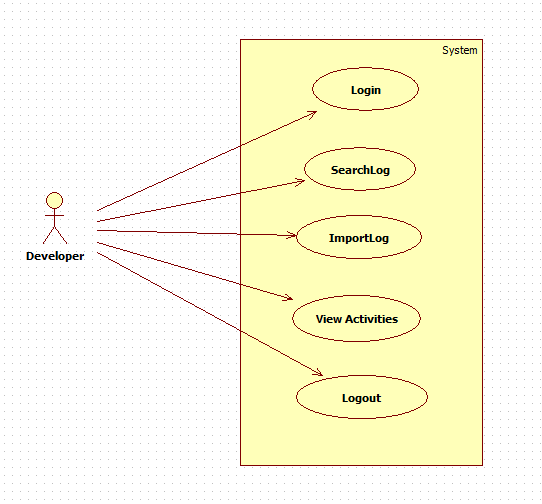
Reliability is a vital requirement of this application as it assists in the further debugging of it.

*SYSTEM ANALYSIS*

**4. SYSTEM ANALYSIS**

**4.1 Use Case Diagram**

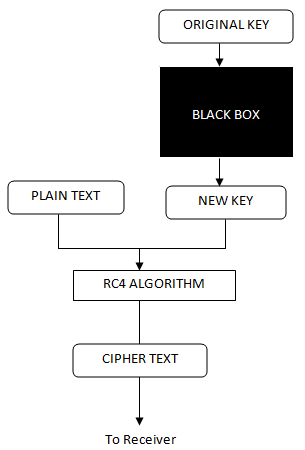
A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals, and any dependencies between those use cases.



*Fig. no: 4.1.1 Use Case diagram*

**4.2 Dataflow Diagram**

A data-flow diagram (DFD) is a graphical representation of the “flow” of data through an information system. DFDs can also be used for the visualization of data processing.



*Fig. no: 4.1.1 Dataflow diagram.*

*SYSTEM DESIGN*

**5. SYSTEM DESIGN**

**5.1 Front End Design**

*CODING*

**6. CODING**

**6.1 Sample Coding**

***POM.xml:***

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>ira.training.xeccq49</groupId>

<artifactId>SpringStudentRestWebService</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>LogAnalyser Maven Webapp</name>

<url>http://maven.apache.org</url>

<properties>

<jdk.version>1.7</jdk.version>

<spring.version>4.1.1.RELEASE</spring.version>

<jstl.version>1.2</jstl.version>

<junit.version>4.11</junit.version>

<logback.version>1.0.13</logback.version>

<jcl-over-slf4j.version>1.7.5</jcl-over-slf4j.version>

</properties>

<dependencies>

<!-- Unit Test -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

</dependency>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

<exclusions>

<exclusion>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>jcl-over-slf4j</artifactId>

<version>${jcl-over-slf4j.version}</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>${logback.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- jstl -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>${jstl.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Jackson JSON Processor -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.4.1</version>

</dependency>

</dependencies>

<build>

<finalName>LogAnalyser</finalName>

</build>

</project>