

# **Labeling Mechanism Requirement Analysis Document v1.0**

## **LEAD SOFTWARE ENGINEERS**

Abdullah CANSIZ

Ahmet AKIL

Alperen Kağan KARA

Ayşenur KARAHASAN

Efe Berke ERKESKİN

Rıdvan SAN

Zeynep ALICI

## **CUSTOMERS**

Murat Can GANİZ

Lokman ALTIN

# Content

Content	2
1. Brief Description	3
1.1 Product Perspective	
1.2 Product Functions	
1.3 User Characteristics	
1.4 General Constraints	
1.5 Assumptions and Dependencies	
2. Glossary of Terms	4
3. Functional Requirements	4
3.1 Inputs	
3.2 Processing	
3.3 Outputs	
4. Non-Functional Requirements	5
4.1 Performance and scalability	
4.2 Portability and compatibility	
4.3 Reliability, availability, maintainability	
4.4 Security	
4.5 Localization	
4.6 Usability	
Domain Model	6

## **1. Brief Description**

### **1.1 Product Perspective**

The program "Labeling Mechanism" classifies groups of instances according to predetermined labels.

### **1.2 Product Functions**

The product will simulate "Labeling Mechanism" by using the given information. The pre given input files (i.e input.json and config.json) will be parsed in order to be processed in our product. The Mechanism will label the instances that were given in the input file with one or more class labels. When the mechanism processes a task, the logging mechanism will give information about which user added which labels to the given instances and when it happened. After the execution of the labeling mechanism, all labeling that is made by the system will be written in an output file.

### **1.3 User Characteristics**

Users can attach single or multiple labels to the instances whenever he/she wants.

### **1.4 General Constraints**

Since the system is meant to be run by itself, the product can't be interrupted in any process when the user gives the command to execute it.

### **1.5 Assumptions and Dependencies**

The product can be run on any operating system since it is programmed in Java programming language.

## 2. Glossary of Terms

**assigned instance:** the instance that is labelled by the mechanism.

**assign label:** A java method that belongs to the User class that assigns some number of labels to an instance with regards to the user class type.

**config set parser:** A java deserializer instance that helps to read the config.json file

**dataset:** the object representation of the input file.

**instance:** the data that can be processed by the mechanism.

**label:** tags that will be used to stamp data.

**logger:** logs the events of the program execution

**output writer:** creates the output.json file with the dataset, users and label assignments

**parser:** reads the input file to the readable code.

**user:** the agent that labels the data.

## 3. Functional Requirements

### 3.1 Inputs

Required Json files to store predetermined labels and instances that are user provided and gets the information about users that will take place in attaching labels.

### 3.2 Processing

The data obtained from Json files will be processed via user execution command to the labeling mechanism which will execute the whole process.

### 3.3 Outputs

When a user wants to add labels to instances, we will show all the details via logging. Ultimately, when all instances are undergoing the process, the program will write the values to the output json file.

## **4. Non-Functional Requirements**

### **4.1 Performance and scalability**

The system returns the results almost immediately and our modular software architecture allows us to scale and improve the program. So it should still be functional in higher workloads.

### **4.2 Portability and compatibility**

Since the program is written in Java and Java runs on billions of devices on the planet.

### **4.3 Reliability, availability, maintainability**

As long as the configuration files are correct there is no risk of any failure.

### **4.4 Security**

Since the system is %100 offline there are no security vulnerabilities.

### **4.5 Localization**

Since the system takes the labels and instances from the input file it can be configured to any localization the user desires.

### **4.6 Usability**

Since there is no interface that the user can interact with, utilization is very simple as long as the required input files are provided correctly.

## Domain Model

