PMR Insight Collective Knowledge (PICK)

Software Design Document

Version 2.3

3/31/2020

Document Control

Approval

The Guidance Team and the customer shall approve this document.

Document Change Control

|  |  |
| --- | --- |
| Initial Release: | 1.0 |
| Current Release: | 2.3 |
| Indicator of Last Page in Document: | $$$ |
| Date of Last Review: | 3/29/2020 |
| Date of Next Review: | 3/31/2020 |
| Target Date for Next Update: |  |

Distribution List

This following list of people shall receive a copy of this document every time a new version of this document becomes available:

Guidance Team Members: Dr. Steven Roach

Jake Lasley

Customer: Mr. Baltazar Santaella

Ms. Herandy Vasquez

Mr. Vincent Fonseca

Ms. Florencia Larsen

Mr. Eric de Nava

Software Team Members: Itzel Rivas

Eder Rodriguez

Jose Gallardo

Gilbert Alvarez

Nusrat Sarmin

Change Summary

The following table details changes made between versions of this document

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Modifier | Description |
| 1.0 | 2/27/2020 | Gilbert Alvarez | Creation of Document |
| 1.1 | 2/28/2020 | Gilbert Alvarez | Purpose and Intended Audience |
| 1.1.1 | 2/28/2020 | Itzel Rivas | Scope and helped with purpose |
| 1.2 | 2/29/2020 | Jose Gallardo | Definitions, Acronyms, Abreviation |
| 1.2.1 | 2/29/2020 | Nusrat Sarmin | Definitions |
| 1.3 | 3/4/2020 | Gilbert Alvarez | Over view and V1 of Wriffs-Brock diagram. |
| 1.3.1 | 3/4/2020 | Eder Rodriguez | Modified Wriffs-Brock diagram |
| 1.4 | 3/5/2020 | Gilbert, Eder, Itzel, Jose, Nusrat | Worked on improving CRC |
| 1.4.1 | 3/5/2020 | Gilbert Alvarez | Subsystem and comment descriptions |
| 1.5 | 3/5/2020 | Jose Gallardo | Detailed Description of components (user subsystem) |
| 1.5.1 | 3/5/20 | Eder Rodriguez | Edited System Component descriptions |
| 1.6 | 3/6/2020 | Eder Rodriguez | Described Dependencies |
| 1.7 | 3/7/2020 | Gilbert Alvarez | Modified Wriffs-Brock diagram and added description |
| 1.8 | 3/7/2020 | Itzel Rivas | 3.3 User interface Subsystem |
| 1.8.1 | 3/7/2020 | Itzel Rivas | 3.4 Ingestion/Validation/Cleansing Subsystem |
| 1.8.2 | 3/7/2020 | Nusrat Sarmin | Contract 6 |
| 1.9 | 3/8/2020 | Jose Gallardo | Contracts for Ingestion/Validation/Cleansing subsystem |
| 1.9.1 | 3/8/2020 | Gilbert Alvarez | Modify diagram again to have appropriate contract numbers, fixed contracts of classes in section 3 |
| 1.9.2 | 3/8/2020 | Gilbert Alvarez | Created Database Schema for VectorDB |
| 2.0 | 3/25/2020 | Gilbert Alvarez | Fixed Wriffs-Brock diagram |
| 2.1 | 3/26/2020 | Gilbert Alvarez | Fixed section 3.0 |
| 2.2 | 3/29/2020 | Gilbert Alverez | Continued to fix section 3.0 |
| 2.3 | 3/29/2020 | Eder Rodriguez | Database schema (crow’s foot notation) |

Table of Contents

[Document Control ii](#_Toc36395048)

[Approval ii](#_Toc36395049)

[Document Change Control ii](#_Toc36395050)

[Distribution List ii](#_Toc36395051)

[Change Summary ii](#_Toc36395052)

[1. Introduction 1](#_Toc36395053)

[1.1. Purpose and Intended Audience 1](#_Toc36395054)

[1.2. Scope of Product 1](#_Toc36395055)

[1.3. References 1](#_Toc36395056)

[1.4. Definitions, Acronyms, and Abbreviations 1](#_Toc36395057)

[1.4.1. Definitions 1](#_Toc36395058)

[1.4.2. Acronyms 2](#_Toc36395059)

[1.4.3. Abbreviations 2](#_Toc36395060)

[1.5. Overview 2](#_Toc36395061)

[2. Decomposition Description 3](#_Toc36395062)

[2.1. System Collaboration Diagram 3](#_Toc36395063)

[2.2. Subsystem and Component Descriptions 3](#_Toc36395064)

[2.2.1. Visualization Subsystem 3](#_Toc36395065)

[2.2.2. Log Entry Subsystem 4](#_Toc36395066)

[2.2.3. User Interface Subsystem 4](#_Toc36395067)

[2.2.4. Ingestion/Validation/Cleansing Subsystem 4](#_Toc36395068)

[2.2.5. Database Subsystem 4](#_Toc36395069)

[2.3. Dependencies 5](#_Toc36395070)

[3. Detailed Description of Component 6](#_Toc36395071)

[3.1. Visualization Subsystem 6](#_Toc36395072)

[3.1.1. Node 6](#_Toc36395073)

[3.1.2. Vector 6](#_Toc36395074)

[3.1.3. Connector 7](#_Toc36395075)

[3.1.4. Graph Generator 7](#_Toc36395076)

[Responsible for drawing nodes and connectors onto the UI. 7](#_Toc36395077)

[3.1.5. Graph 8](#_Toc36395078)

[3.2. Log Entry Subsystem 9](#_Toc36395079)

[3.2.1. Log Entry 9](#_Toc36395080)

[3.2.2. Log Entry Manager 10](#_Toc36395081)

[3.3. User Interaction Subsystem 10](#_Toc36395082)

[3.3.1. UI Graph and Table view 10](#_Toc36395083)

[3.3.2. UI Vector Config View 11](#_Toc36395084)

[3.3.3. UI Action Report View 11](#_Toc36395085)

[3.3.4. UI Team Configuration View 11](#_Toc36395086)

[3.3.5. UI Analysis View 12](#_Toc36395087)

[3.4. Ingestion/Validation/Cleansing Subsystem 12](#_Toc36395088)

[3.4.1. Splunk 12](#_Toc36395089)

[3.4.2. OCR (Optical Character Reader) 12](#_Toc36395090)

[3.4.3. Transcription Tool 13](#_Toc36395091)

[3.4.4. Log File 13](#_Toc36395092)

[3.4.5. Enforcement Action Report 13](#_Toc36395093)

[3.4.6. Event Configuration 14](#_Toc36395094)

[3.5. Database Subsystem 15](#_Toc36395095)

[3.5.1. Vector Database 15](#_Toc36395096)

[4. Database 17](#_Toc36395097)

[4.1. Database Schema 17](#_Toc36395098)

# Introduction

## Purpose and Intended Audience

The purpose of the Software Design Document (SDD) is to identify collaborations and subsystems within the PMR Insight Collective Knowledge (PICK) software to be able to design interactions within the system. The SDD describes what is to be built and is intended to help the development team understand what is expected to be built. Sections within the SDD include the system and subsystem collaboration graphs, component descriptions, descriptions of the classes within the components, contracts between classes, and description of database schemas. The SDD focuses on describing contracts of classes which are client server relationships that occur in order to fulfill a client’s responsibilities, and protocols of classes that state each method in a class, parameters of the methods, return value of the method, purpose of the method and pre and post conditions of a method.

## Scope of Product

This Software Design Document focuses on the core functionality of the PICK system and its main components, which is to streamline the job of an analyst to create a report from log files submitted by the defense and attacking teams. The system will utilize outside libraries and pre-existing systems, described on section 2.3, to achieve artifact ingestion, validation, cleansing and creating a visual representation of significant events for the analyst to see. Our goal is to be able to aid analysts in the creation of a report from any form of artifacts containing information of an attack response situation.

## References

[1] 08c\_SDD\_Outline.pdf

[3] 08aContracts.pdf

[4] 08b\_Protocols.pdf

[5] 09Subsystems.pdf

[6] Hasneeza. “Example of Software Design Document (SDD).” UniMAP Portal, 26 Apr. 2012, portal.unimap.edu.my/portal/page/portal30/Lecture Notes/KEJURUTERAAN\_KOMPUTER/Semester 2 Sidang Akademik 20112012/EKT420 Software Engineering/Example of Software Design Document (SDD).

## Definitions, Acronyms, and Abbreviations

### Definitions

|  |  |
| --- | --- |
| **Lead Analyst** | The analyst responsible for the oversight of work being performed by the other analysts, some other responsibilities include delegating work, and approving work |
| **Analyst** | A user that is authorized to work PICK system, they must connect to the Lead analysts' machine to gain access to the log files and databases. Some of their responsibilities include editing, creating, deleting log entries/nodes, and pushing work to the Lead analyst for approval upon synchronization. |
| **Log File** | A file that records events from a team users' machine. |
| **Log Entry** | A file that records events from a team users' machine. |
| **Log Creator** | Node attribute that states what team created the log. |
| **Node** | Visual representation of log entry on a vector graph. |
| **Node name** | Node attribute, unique name of a node. |
| **Start Date** | Start date and time of Assessment |
| **End Date** | End date and time of Assessment |
| **Timestamp** | Time the activity occurred. Recorded in Zulu time. |
| **Node ID** | Unique numerical ID created for nodes. |

### Acronyms

|  |  |
| --- | --- |
| **PICK** | PMR Insight Collective Knowledge |
| **PMR** | Prevent, Mitigate, Recover |
| **SDD** | Software Design Document |
| **UML** | Unified Modeling Language |

### Abbreviations

|  |  |
| --- | --- |
| **DB** | Database |
| **config** | Configuration |
| **e.g.** | For example |
| **i.e.** | in other words |

## Overview

The following section describes the components and the interactions of the subsystems in the PICK system using Wriffs-Brocks diagram. Wriffs-Brock diagram utilizes rounded squares to indicate classes and semicircles with numbers and arrows pointing from one class to another to indicate a contract between two classes. Subsystems on the Wriffs-Brock diagram are indicated with a dotted container surrounding the classes. This section also describes the design choices used for dividing design entities.

# Decomposition Description

This section describes the subsystems of the system and the collaboration between the components.

## System Collaboration Diagram

A close up of text on a white background

Description automatically generated

*figure 1.0*

The system can be broken down into 5 major subsystems, User interface subsystem, Ingestion/Validation/Cleansing subsystem, Visualization subsystem, Log entry subsystem, and Database subsystem.

## Subsystem and Component Descriptions

### Visualization Subsystem

**Description:** This subsystem shows the interaction of the data that was created from log files coming from the ingestion/validation/cleansing process. The visualization subsystem takes the data from the log entries and draws representations of nodes and connectors using an external library, Q Graph Viz and. From there the nodes and connectors are displayed to the user on the UI view Graph and Table. This subsystem also displays log entries to the user on a table view on the same UI view.

**Purpose:** Take in data to transform it into visuals.

**Contracts:**

* 1 (described in table 1.0)
* 2 (described in table 1.1)
* 3 (described in table 1.1)
* 4 (described in table 1.2)
* 5 (described in table 1.3)
* 6 (described in table 1.3)
* 7 (described in table 1.4)
* 8 (described in table 1.4)

### Log Entry Subsystem

**Description:** This subsystem shows the interaction between the log entries and the other components of the system. When log files are processed, they become log entries, then those log entries are associated to a vector. The relationship between a log entry and vector can be seen in the table and graph view where significant log entries are represented as nodes on a graph. The log entry subsystem is also responsible for the creation of new log entries by the user.

**Purpose:** To manage the log entries that were created as a result of ingestion/validation/cleansing and the log entries created by the user.

**Contracts:**

* 9 (described in table 2.0)
* 10 (described in table 2.1)

### User Interface Subsystem

**Description:** Then we have the User Interface subsystem that can allows the user to the interact with the system by displaying information and asking the user for input. The classes within this subsystem can display information between various views, such as the graph and table view displaying log entry and node information and the vector configuration asking the user for input to name and describe a vector.

**Purpose:** Describes the appropriate information to display to the user when they reach a certain view of the system. Asks the user for input where necessary.

**Contracts:**

* 11 (described in table 3.0)
* 12 (described in table 3.2)
* 13 (described in table 3.4)

### Ingestion/Validation/Cleansing Subsystem

**Description:** This subsystem describes the process of ingestion, validation and cleansing of log files. Classes associated with this subsystem include the Enforcement action report, Log files and Splunk. When a log file is ingested the result of the action of weather it led to an error or was successfully ingested is recorded into the Action Report.

**Purpose:** To show the interaction of log file and Splunk to create log entries that the system will use for the Graph and table view.

**Contracts:**

* 14 (described in table 4.0)
* 15 (described in table 4.1)
* 16 (described in table 4.2)
* 17 (described in table 4.4)
* 18 (described in table 4.5)

### Database Subsystem

**Description:** The database subsystem describes thedata that is stored on the database and what components push and pull data from it. The database stores information of the current event configuration, Vector configuration, and the log entries that are associated with the vectors.

**Purpose:** To Store vector and log entry data for users to push and pull to as they work on the system.

**Contracts:**

* 19 (described in table 5.0)

## Dependencies

The system runs using Python version 3 making the code portable with the ability to be able to run on the client's choice of operating system (Kali Linux). The Visual Representation component that contains classes such as Graph and node, drawing the graph utilizes the external library QGraphViz to draw nodes and edges making up the graph. The UI controller component uses an external library pyqt to draw user interface elements and create views such as Graph and table view, event configuration view, directory view, Action Report view, Log ingestion/validation/cleansing view, and Team configuration view. Ingestion of the log files in the system depend on an external system, Splunk.

# Detailed Description of Component

## Visualization Subsystem

### Node

Represents a significant log entry and can be visually seen on a graph of a specific vector. This class collaborates with the log entry class to retrieve information about the log entry (seen in signature).

*Table 1.0*

|  |  |
| --- | --- |
| **Class Name**: Node | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   1. Knows its own unique name. 2. Knows its icon type. 3. Knows its visibility. 4. Knows its unique node ID. | |
| **Contract:** 1. Get Log Entry Information | |
| **Responsibilities** | **Collaborations** |
| 1. Knows log entry | Log Entry (1) |

**Contract 1: Get Log Entry Information**

* **Method Name:** getLogEntryInfo(self);
* **Return Value Type:** LogEntry
* **Input and output Parameters:** self
* **Description of Parameters:** a reference to a node object
* **Purpose:** retrieve the start time, end time, source document, log entry description, and log entry creator

**Pre-Conditions:** Log Entries must exist and have the values start time, end time, source document, log entry description, and log entry creator.

**Post-Conditions:** returns a node object that has a sting value for the start and end time formatted by “HH:MM:SS” where the hour is on 24 hour time, a string for the path to the log entry’s source document, a string describing the log entry, and a string holding the name of the log entry creartor.

### Vector

*Table 1.1*

|  |  |
| --- | --- |
| **Class Name**: Vector | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   1. Knows its name. 2. Knows its description. | |
| **Contract:** 2. Get Vector Information | |
| **Responsibilities** | **Collaborations** |
| 1. Get vector information from user | UI Vector Configuration (1) |
| **Contract:** 3. Associate Log Entries | |
| **Responsibilities** | **Collaborations** |
| 1. Get Logs | Log Entry (1) |

**Contract 2: Get Vector Information**

* **Method Name:** getVectorInfo(string name, string description);
* **Return Value Type:** String
* **Input and output Parameters:**
* name: must be a string naming the vector
* description: must be a string describing the vector, length depends on user input
* **Description of Parameters:** the string that states the name of the vector
* **Purpose:** To get the name of the vector the user entered from the vector configuration view

**Pre-Conditions:** The user must enter a string name from the UI Vector Configuration view and create a vector.

**Post-Conditions:** Returns the vector name as a string.

**Contract 3: Associate Log Entries**

* **Method Name:** associateLogEntry(self, logEntry x);
* **Return Value Type:** self
* **Input and output Parameters:**
* self
* x: a reference to the log entry object
* **Description of Parameters:**
* self: a reference to the vector object
* x: a reference to the log entry object
* **Purpose:** To associate a log entry to a specific vector

**Pre-Conditions:** the user must enter a name and description for a vector, there must be log entries

**Post-Conditions:** Returns a vector object with the associated log entries.

### Connector

*Table 1.2*

|  |  |
| --- | --- |
| **Class Name**: Connector | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   1. Knows its Relationship ID. 2. Knows its parent node ID. 3. Knows its child node ID. 4. Knows its description. | |
| **Contract:** 4. Get Nodes Information | |
| **Responsibilities** | **Collaborations** |
| 1. Connect 2 nodes | Node (1) |

**Contract 4: Get Nodes Information**

* **Method Name:** connectNodes(self, node x, node y);
* **Return Value Type:** void
* **Input and output Parameters:**
* Self
* x: must be a node object
* y: must be a node object
* **Description of Parameters:** a connector object reference and, a reference to the first node to start the connection, a reference to the second node to connect to.

**Purpose:** to create an object to reference when using the drawer to draw a line between two nodes.

**Pre-Conditions:** At least 2 nodes must exist within a graph in a specific vector

**Post-Conditions:** Connector object has been declared

### Graph Generator

### Responsible for drawing nodes and connectors onto the UI.

*Table 1.3*

|  |  |
| --- | --- |
| **Class Name**: Graph Generator | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   1. Draws nodes. 2. Draws connectors | |
| **Contract:** 5 Draw Graph Nodes | |
| **Responsibilities** | **Collaborations** |
| 1. Get Node(s) associated to a graph for specific vector | Graph(1) |
| **Contract:** 6 Draw Graph Connectors | |
| **Responsibilities** | **Collaborations** |
| 1. Get Connector(s) associated to a graph for a specific vector | Graph(2) |

**Contract 5: Draw Graph Nodes**

* **Method Name:** drawNode(node x);
* **Return Value Type:** void
* **Input and output Parameters:**
* x: must be a node object on a graph
* **Description of Parameters:** the reference to a node on a graph

**Purpose:** to provide a service to allow nodes to be drawn

**Pre-Conditions:** there must be at least one node on a graph and the node must have been associated to a vector

**Post-Conditions:** a node has been drawn on the graph

**Contract 6: Draw Graph Connectors**

* **Method Name:** drawConnector(connector x);
* **Return Value Type:** void
* **Input and output Parameters:**
* x: must be a connector object on a graph
* **Description of Parameters:** the reference to a connector on a graph

**Purpose:** to provide a service to allow connectors to be drawn

**Pre-Conditions:** there must be at least one connector on a graph and the connector must be associated to two nodes

**Post-Conditions:** a connector has been drawn on the graph

### Graph

Holds data of nodes and their relationships.

*Table 1.4*

|  |  |
| --- | --- |
| **Class Name**: Graph | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   1. Knows what nodes that are in a graph 2. Knows the relationships of nodes in the graph | |
| **Contract:** 7. Store Node Information | |
| **Responsibilities** | **Collaborations** |
| 1. Get Nodes | Node (1) |
| **Contract:** 8. Store Connector Information | |
| **Responsibilities** | **Collaborations** |
| 1. Get Connectors | Connector(2) |

**Contract 7: Store Nodes Information**

* **Method Name:** addGraphNode(self, node x);
* **Return Value Type:** void
* **Input and output Parameters:**
* Self
* x: must be a node object
* **Description of Parameters:** a node object reference

**Purpose:** to store the references, information, of nodes that have been associated to a vector

**Pre-Conditions:** a node reference has been created from a log entry, and that node reference has been associated with a vector

**Post-Conditions:** a node reference has been added to a graph for a vector

**Contract 8: Store Connector Information**

* **Method Name:** addGraphConnector(self, connector x);
* **Return Value Type:** void
* **Input and output Parameters:**
* Self
* x: must be a connector object
* **Description of Parameters:** a connector object reference

**Purpose:** to store the references, information, of connectors that have been associated to a vector

**Pre-Conditions:** a connector reference has been created for indicating a relationship between two nodes

**Post-Conditions:** a connector reference has been added to a graph for a vector

## Log Entry Subsystem

### Log Entry

Represents an event that occurred during an assessment.

*Table 2.0*

|  |  |
| --- | --- |
| **Class Name**: Log Entry | |
| **Superclass**: None | |
| **Subclasses**: None | |
| 1. Knows start time. 2. Knows end time. 3. Knows its source document. 4. Knows a description of its activity. 5. Knows its creator.   6. Knows its log entry number. | |
| **Contract:** 9. Create Log Entry | |
| **Responsibilities** | **Collaborations** |
| 1. Create new log entry object | Log Entry Management(1) |

**Contract 9: Create Log Entry**

* **Method Name:** addLogEntry(self, s, e, source, desc, creator);
* **Return Value Type:** Log Entry object
* **Input and output Parameters:**
* Self
* s: must be a string
* e: must be a string
* source: must be a string
* desc: must be a string
* creator: must be a string
* **Description of Parameters:** a log entry object to reference, start time in “HH:MM:SS”, end time in “HH:MM:SS”, a description of the log entry can be as long as the user makes it, creator is the name of the user who made the log entry

**Purpose:** to allow the user to create new log entries that aren’t associated with a log file

**Pre-Conditions:** user must be on the UI graph and table view with a selected vector

**Post-Conditions:** returns a reference to a new log entry

### Log Entry Manager

*Table 2.1*

|  |  |
| --- | --- |
| **Class Name**: Log Entry Manager | |
| **Superclass**: None | |
| **Subclasses**: None | |
| 1. Holds all log entries. 2. Creates log entries. 3. Saves log entries into DB. 4. Filters log entries. 5. Gives ability to search Log entries. | |
| **Contract:** 10. Store Log Entries | |
| **Responsibilities** | **Collaborations** |
| 1. Save log entry to database | Vector DB Manager |

**Contract 10: Store Log Entries**

* **Method Name:** saveLogEntry(logEntry x);
* **Return Value Type:** void
* **Input and output Parameters:**
* x: must be a log entry object
* **Description of Parameters:** a reference to a log entry

**Purpose:** to allow the system to save log entries on the data base

**Pre-Conditions:** user must be on graph and table view or analysis view and click on option to create new log entry

**Post-Conditions:** a new log entry is added to the table view/analysis view

## User Interaction Subsystem

### UI Graph and Table view

*Table 3.0*

|  |  |
| --- | --- |
| **Class Name**: UI Graph and Table view | |
| **Superclass**: UI View Manager | |
| **Subclasses**: | |
| **Private Responsibilities**   1. Display information about graph. 2. Display information about table. 3. Receive user input for creating a node. 4. Receive user input for creating new log entry. | |
| **Contract:** 11. Display Graph | |
| **Responsibilities** | **Collaborations** |
| 1. Display graph information on the graph view | Graph(1) |

**Contract 11: Display Graph**

* **Method Name:** displayGraphView(graph x);
* **Return Value Type:** void
* **Input and output Parameters:**
* x: must be a graph object
* **Description of Parameters:** reference to a graph

**Purpose:** to display nodes and connectors on the graph view and log entries associated with the nodes on the table view

**Pre-Conditions:** there must be some nodes and log entries within a graph

**Post-Conditions:** Representation of a node has been drawn on the Graph/ Table view.

### UI Vector Config View

*Table 3.1*

|  |  |
| --- | --- |
| **Class Name**: UI Vector Config View | |
| **Superclass**: UI View Manager | |
| **Subclasses**: None | |
| **Private Responsibilities**   1. Ask user for Vector name input 2. Ask user for Vector description input | |
| **Contract:** | |
| **Responsibilities** | **Collaborations** |
|  |  |

### UI Action Report View

*Table 3.2*

|  |  |
| --- | --- |
| **Class Name**: UI Action Report View | |
| **Superclass**: UI View Manager | |
| **Subclasses**: None | |
| **Private Responsibilities**   1. Display error message of line from log file where ingestion failed. 2. Display progress of ingestion, validation and cleansing 3. Ask the user if they want to override error. | |
| **Contract:** 12 Display Action Report | |
| **Responsibilities** | **Collaborations** |
| 1. Display progress of ingestion | Enforcement Action Report (1) |

**Contract 12: Display Action Report**

* **Method Name:** displayActionReport();
* **Return Value Type:** void
* **Input and output Parameters:**
* **Description of Parameters:**

**Purpose:** to allow the user to see errors and progress of ingestion.

**Pre-Conditions:** log files are being ingested.

**Post-Conditions:** the user can see all the files that were processed.

### UI Team Configuration View

*Table 3.3*

|  |  |
| --- | --- |
| **Class Name**: Team Configuration View | |
| **Superclass**: UI View Manager | |
| **Subclasses**: None | |
| **Private Responsibilities**   1. Ask analyst if they are the lead. 2. Ask Lead analyst for IP address. | |
| **Contract:** | |
| **Responsibilities** | **Collaborations** |
|  |  |

### UI Analysis View

*Table 3.4*

|  |  |
| --- | --- |
| **Class Name**: UI Analysis View | |
| **Superclass**: UI View Manager | |
| **Subclasses**: None | |
| **Private Responsibilities**   1. Displays log entries to user. 2. Gives functionality to users to drag and drop entries to vectors. 3. Gives functionality to users to filter log entries. 4. Gives functionality to user to search log entries. | |
| **Contract:** 13. Display All Log Entries | |
| **Responsibilities** | **Collaborations** |
| 1. Display all logs | Log Manager(1) |

**Contract 13: Display All Log Entries**

* **Method Name:** displayAnalysisView();
* **Return Value Type:** void
* **Input and output Parameters:**
* **Description of Parameters:**

**Purpose:** to allow the user to see errors and progress of ingestion.

**Pre-Conditions:** log files are being ingested.

**Post-Conditions:** the user can see all the files that were processed.

## Ingestion/Validation/Cleansing Subsystem

### Splunk

*Table 4.0*

|  |  |
| --- | --- |
| **Class Name**: Splunk | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   1. Can Ingest Log files | |
| **Contract:** 14 Take in Log Files | |
| **Responsibilities** | **Collaborations** |
| 1. Ingest Log files | Log file(1) |

**Contract 14: Take in Log Files**

* **Method Name:** getLogfile();
* **Return Value Type:** boolean
* **Input and output Parameters:**
* **Description of Parameters:** true if the logfile was processed successfully, false if it failed.

**Purpose:** to ingest/validate/cleanse log files to turn them into log entry.

**Pre-Conditions:** Log files are stored at the directory location.

**Post-Conditions:** Log files have been ingested or resulted in an error.

### OCR (Optical Character Reader)

*Table 4.1*

|  |  |
| --- | --- |
| **Class Name**: OCR | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   1. Allows log files to be converted to text. | |
| **Contract:** 15 Take in Log File Image | |
| **Responsibilities** | **Collaborations** |
| 1. Ingest Log files | Log file(1) |

**Contract 15: Take in Log File Image**

* **Method Name:** getImage(image);
* **Return Value Type:** text file
* **Input and output Parameters:**
* **Description of Parameters:** true if the logfile was processed successfully, false if it failed.

**Purpose:** To take Log Files that are images and convert readable text on the image to a text file

**Pre-Conditions:** Log files images are stored at the directory location in a team folder.

**Post-Conditions:** returns a text file that can be ingested.

### Transcription Tool

*Table 4.2*

|  |  |
| --- | --- |
| **Class Name**: Transcription Tool | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   1. Allows log files to be converted to text. | |
| **Contract:** 16 Take in Log File Audio | |
| **Responsibilities** | **Collaborations** |
| 1. Ingest Log files | Log file(1) |

**Contract 16: Take in Log File Image**

* **Method Name:** getImage(image);
* **Return Value Type:** text file
* **Input and output Parameters:**
* **Description of Parameters:** true if the logfile was processed successfully, false if it failed.

**Purpose:** To take Log Files that are images and convert readable text on the image to a text file

**Pre-Conditions:** Log files images are stored at the directory location in a team folder.

**Post-Conditions:** returns a text file that can be ingested.

### Log File

*Table 4.3*

|  |  |
| --- | --- |
| **Class Name**: Log File | |
| **Superclass**: None | |
| **Subclasses**: | |
| **Private Responsibilities**   1. Knows its name. 2. Knows its cleansing status. 3. Knows its ingestion status. 4. Knows its validation status. | |
| **Contract:** | |
| **Responsibilities** | **Collaborations** |
|  |  |

### Enforcement Action Report

*Table 4.4*

|  |  |
| --- | --- |
| **Class Name**: Enforcement Action Report | |
| **Superclass**: | |
| **Subclasses**: | |
| **Private Responsibilities**   1. Knows the line of log file that caused and error. | |
| **Contract:** 17 Know what errors occurred while ingesting | |
| **Responsibilities** | **Collaborations** |
| 1. Know if a Log file was processed correctly | Log File(1) |

**Contract 17: Get Log File Status**

* **Method Name:** getLogfile(logfile x);
* **Return Value Type:** boolean
* **Input and output Parameters:**
* x
* **Description of Parameters:** a reference to a log file

**Purpose:** to know the ingestion, validation and cleansing status of a log file

**Pre-Conditions:** Log files are stored at the directory location

**Post-Conditions:** Log files have been ingested or resulted in an error.

### Event Configuration

*Table 4.5*

|  |  |
| --- | --- |
| **Class Name**: Event Configuration | |
| **Superclass**: | |
| **Subclasses**: | |
| **Private Responsibilities**   1. Knows the lead analysts IP Address. 2. Knows who the lead analyst is. 3. Knows the amount of connections allowed to the host. 4. Knows the location of the root directory. | |
| **Contract:** 18. Configure Assessment | |
| **Responsibilities** | **Collaborations** |
| 1. Get the name of the assessment.  2. Get the start time of the assessment.  3. Get the end time of the assessment.  4. Get the location of the root directory.  5. Get the location of the team folders.  6. Know who lead analyst is. | UI Team Configuration View(1)  UI Team Configuration View(1)  UI Team Configuration View(1)  UI Team Configuration View(1)  UI Team Configuration View(1) |

**Contract 18: Configure Assessment**

* **Method Name:** getAssesmentName();
* **Return Value Type:** string
* **Input and output Parameters:**
* **Description of Parameters:**

**Purpose:** to get the name of the current assessment from the user

**Pre-Conditions:** user is on the team configuration view

**Post-Conditions:** returns a string

* **Method Name:** getEventStart();
* **Return Value Type:** string
* **Input and output Parameters:**
* **Description of Parameters:**

**Purpose:** to get the start time of the current assessment from the user

**Pre-Conditions:** user is on the team configuration view

**Post-Conditions:** returns a string in the format “HH:MM:SS” where hours are in (0-24)

* **Method Name:** getEventEnd();
* **Return Value Type:** string
* **Input and output Parameters:**
* **Description of Parameters:**

**Purpose:** to get the end time of the current assessment from the user

**Pre-Conditions:** user is on the team configuration view

**Post-Conditions:** returns a string in the format “HH:MM:SS” where hours are in (0-24)

* **Method Name:** getRootLocation();
* **Return Value Type:** string
* **Input and output Parameters:**
* **Description of Parameters:**

**Purpose:** to get the location of the root directory that contains the folders for the log files from the different teams

**Pre-Conditions:** user is on the team configuration view, and chooses a directory from the file explorer

**Post-Conditions:** returns a string to the path of the root directory

* **Method Name:** getFolderLocation();
* **Return Value Type:** string
* **Input and output Parameters:**
* **Description of Parameters:**

**Purpose:** to get the locations of the team folders

**Pre-Conditions:** user is on the team configuration view, has chosen the root directory and has also chosen the paths to the team directories

**Post-Conditions:** returns the paths to the team folders

## Database Subsystem

### Vector Database

*Table 5.0*

|  |  |
| --- | --- |
| **Class Name**: Vector Database | |
| **Superclass**: None | |
| **Subclasses**: None | |
| **Private Responsibilities**   1. Knows the lead analysts IP Address. 2. Knows who the lead analyst is. 3. Knows the amount of connections allowed to the host. 4. Knows the location of the root directory. | |
| **Contract:** 19 Store Vector information | |
| **Responsibilities** | **Collaborations** |
| 1. Get Information about Vector | Vector(1) |

**Contract 19: Store Vector Information**

* **Method Name:** getVectorInfo(Vector obj);
* **Return Value Type:** void
* **Input and output Parameters:** vector object
* **Description of Parameters:** reference to vector object

**Purpose:** to store vector information on the vector database.

**Pre-Conditions:** at least one vector has been created by or approved by analyst.

**Post-Conditions:** the vector information has been stored in the database.

# Database

## Database Schema

