# Data Corruption Recovery

Project Documentation Submitted

To the Faculty of School of

Computing and Information Technologies

Of

Asia Pacific College

In Partial Fulfillment of the Requirements for the subject

Applied Projects 2 or Software Development

By

Alberca, Reginald John Steven R.  
Miculob, Kent Michael P.

Laureta, Maria Letty M.

Table of Contents

[Data Corruption Recovery 1](#_Toc459605601)

[Abstract 3](#_Toc459605602)

[List of Figures, List of Tables, List of Notations 4](#_Toc459605603)

[Event Table 4](#_Toc459605604)

[Activity Diagram 5](#_Toc459605605)

[Class Diagram 10](#_Toc459605606)

[Communication Diagram 11](#_Toc459605607)

[Component Diagram 11](#_Toc459605608)

[Composite Diagram 12](#_Toc459605609)

[Context Diagram 13](#_Toc459605610)

[Data Dictionary 13](#_Toc459605611)

[Data Flow Diagram 14](#_Toc459605612)

[Deployment Diagram 15](#_Toc459605613)

[Entity-Relationship Diagram 16](#_Toc459605614)

[Interaction Diagram 16](#_Toc459605615)

[Object Diagram 16](#_Toc459605616)

[Package Diagram 17](#_Toc459605617)

[Sequence Diagram 18](#_Toc459605618)

[State Diagram 18](#_Toc459605619)

[Timing Diagram 21](#_Toc459605620)

[Use Case 21](#_Toc459605621)

[I. Introduction 24](#_Toc459605622)

[1.1 Project Context 27](#_Toc459605623)

[1.2 Purpose and Description 27](#_Toc459605624)

[1.3 Objectives 27](#_Toc459605625)

[1.4 Scope and Limitations 27](#_Toc459605626)

[II. Related Literature 28](#_Toc459605627)

[III. Technical Background 31](#_Toc459605628)

[IV. Design and Methodology 31](#_Toc459605629)

[4.1 Include discussions on conceptual design/system architecture/block diagrams and algorithms 31](#_Toc459605630)

[V. Results and Discussion 31](#_Toc459605631)

[VI. Conclusions and Recommendations 32](#_Toc459605632)

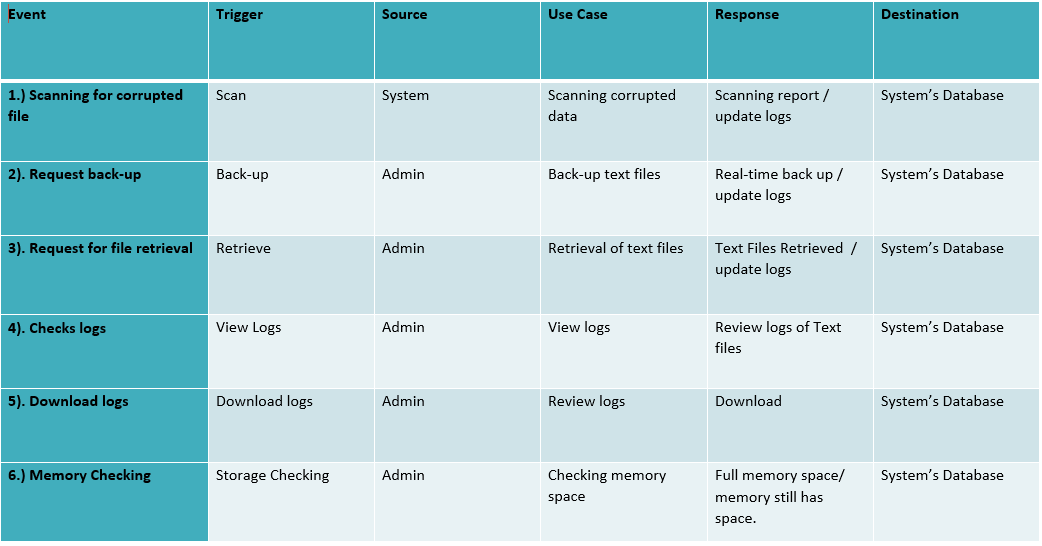
[VII. Appendices 32](#_Toc459605633)

# Abstract

This project aims to create a forensic tool capable of preventing data corruption. The existence of data corruption began since the beginning of the automated technologies, And in the world manipulated by virtual information it is necessary for us to secure the information that we have. Information stored virtually are so fragile that in fact a single error or point of failure could cause your information to be corrupted. Data corruptions are something you’ll definitely hated if you’re one of those people who doesn’t do backup or does backup occasionally. The efforts that you’ve inserted and the time allotted for the file or document would gone to waste, and how devastating could the event if it we’re to happen to you? The researches of this paper would want to create a solution on this. And they figure out that the data corruption itself are inevitable since there are external factors that aren’t preventable. As the researchers continue the study, they’ve realized that data corruption prevention aren’t just limited on how could you avoid the event of corruption, but you could minimize or neglect the impact by considering the fact that it’s unavoidable and be always prepared of it.

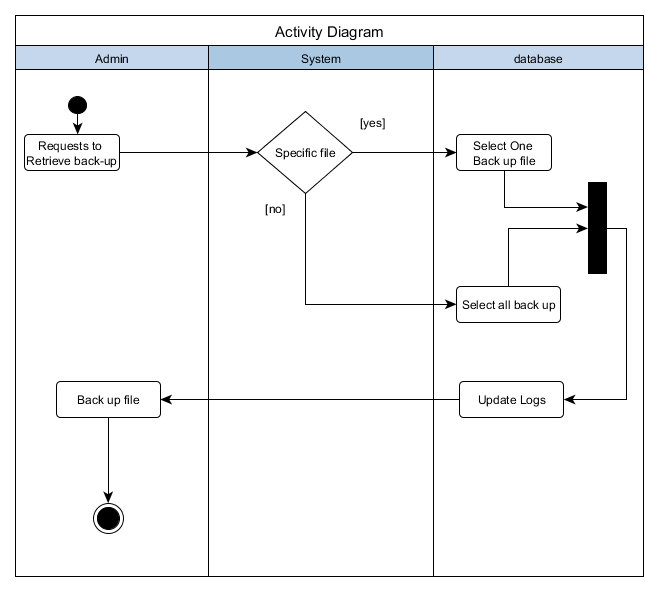
# List of Figures, List of Tables, List of Notations

### Event Table

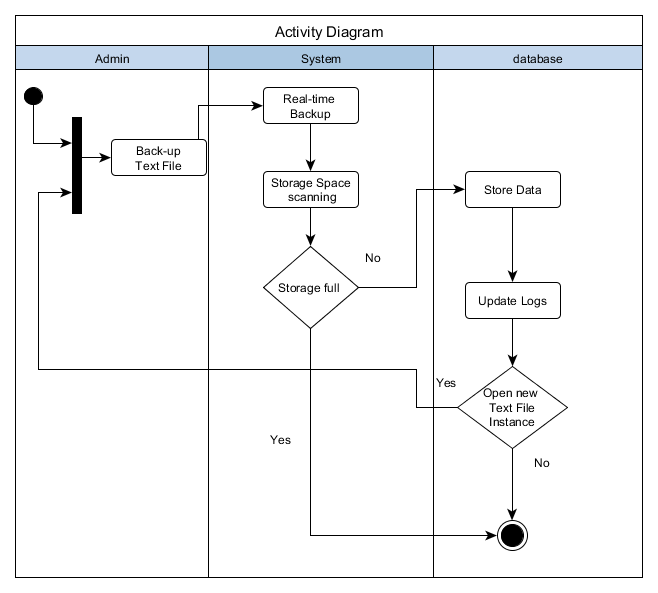


### Activity Diagram

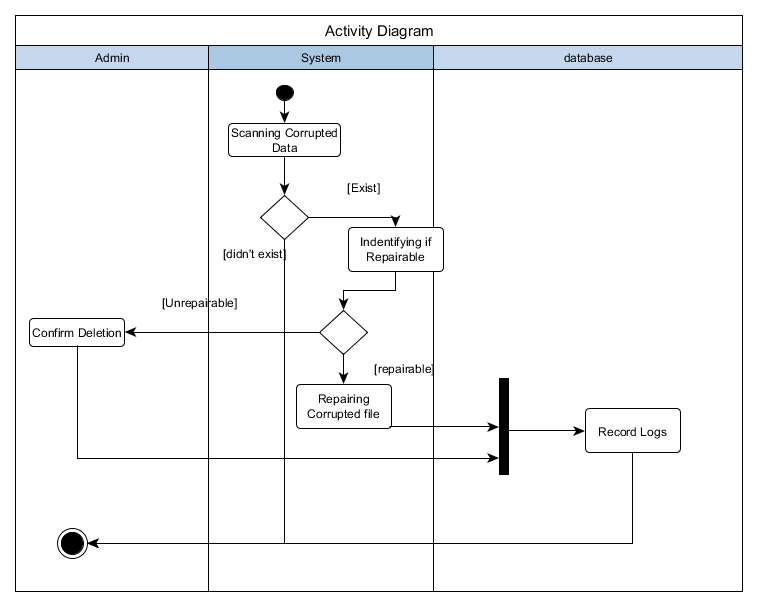
Back-up



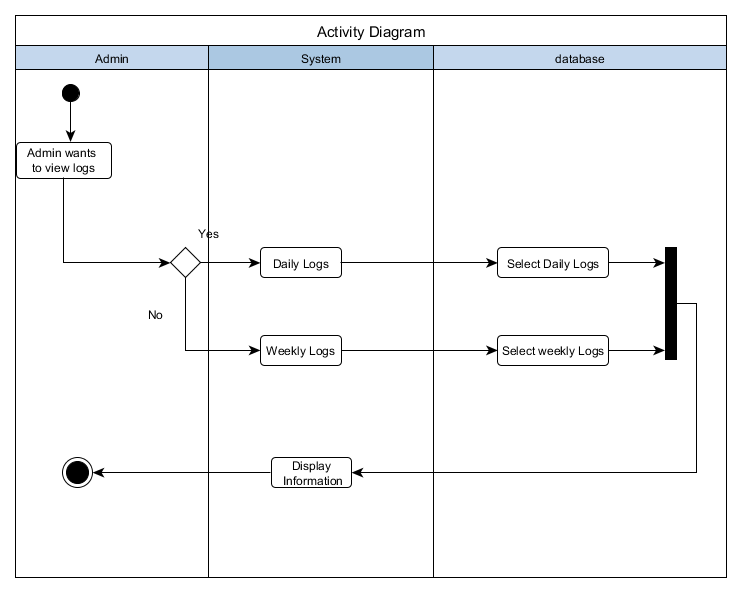
Memory Check and Scanning



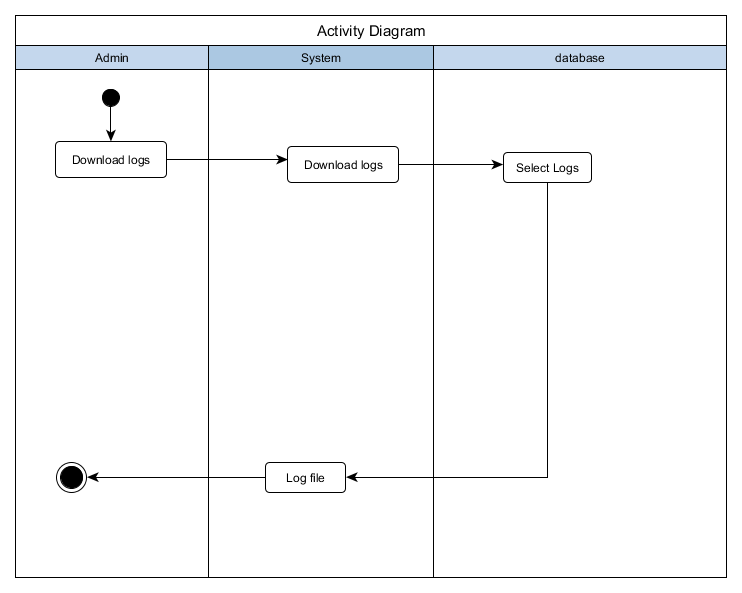
Scanning



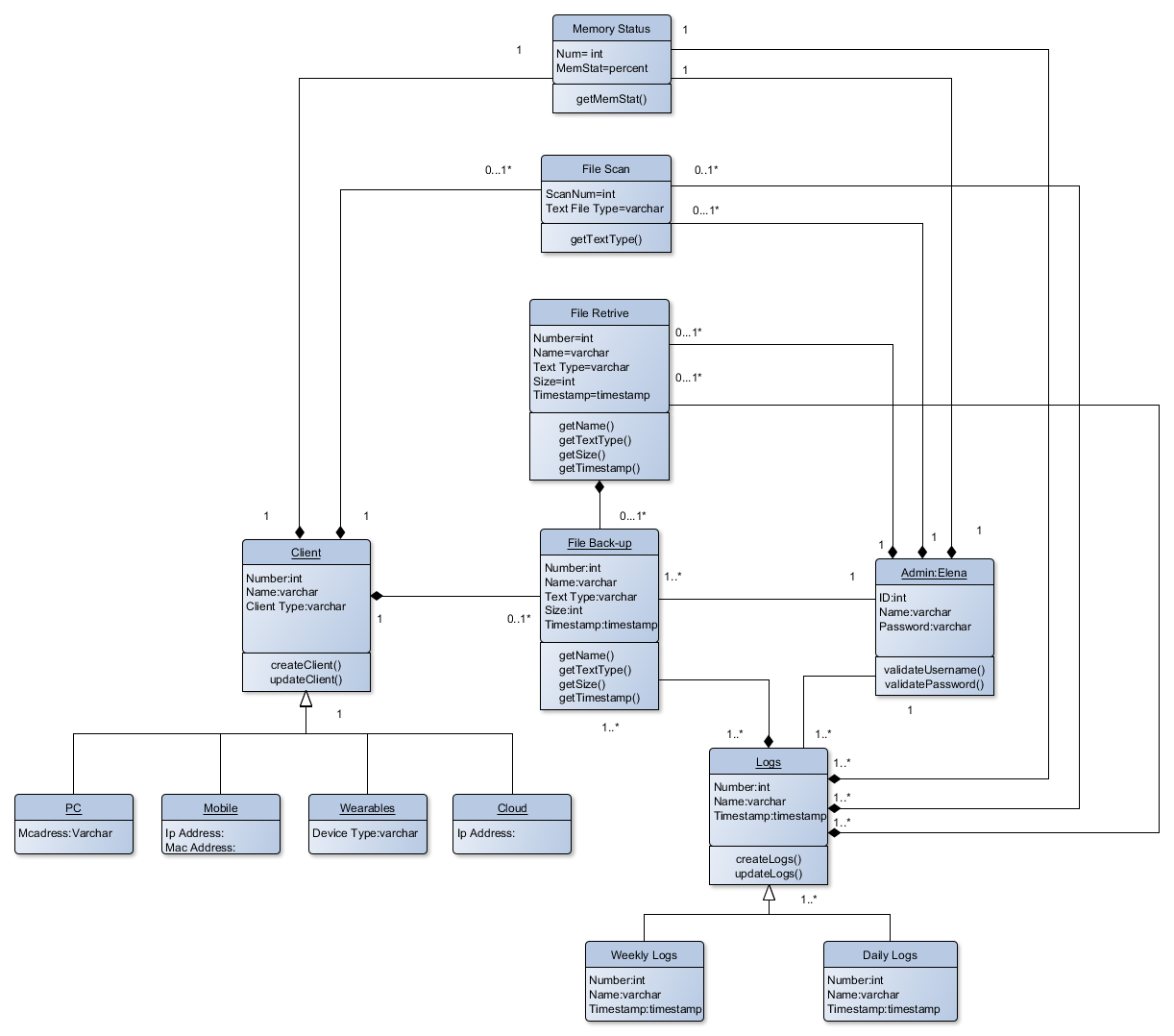
View Logs



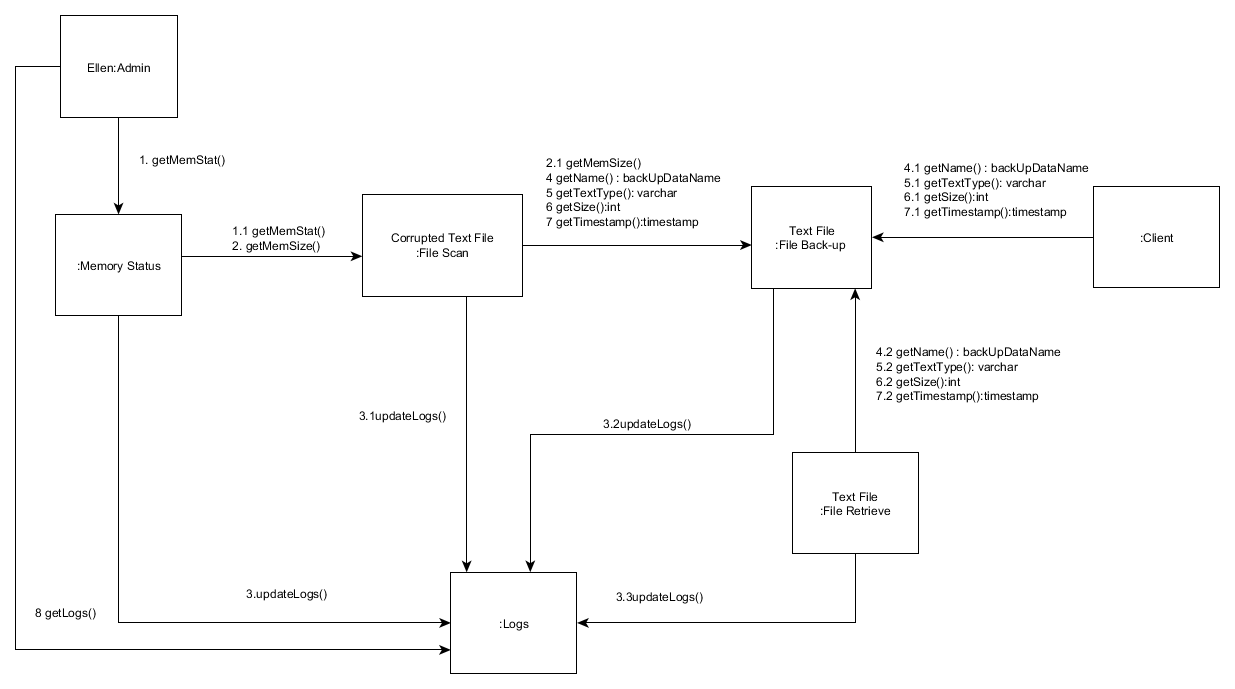
Download Logs



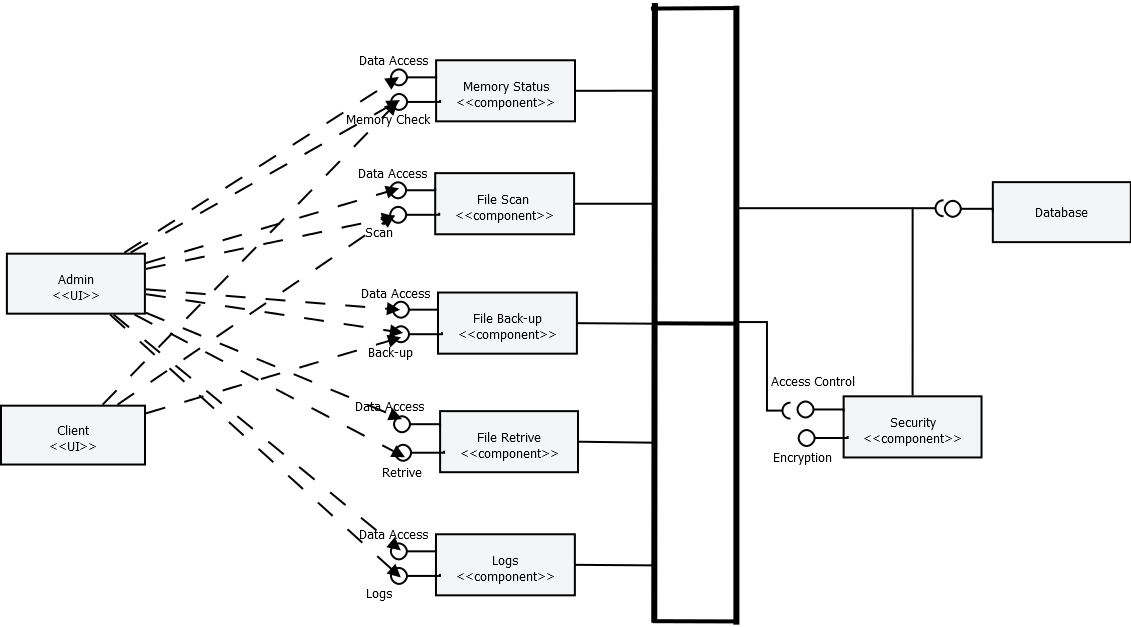
### Class Diagram



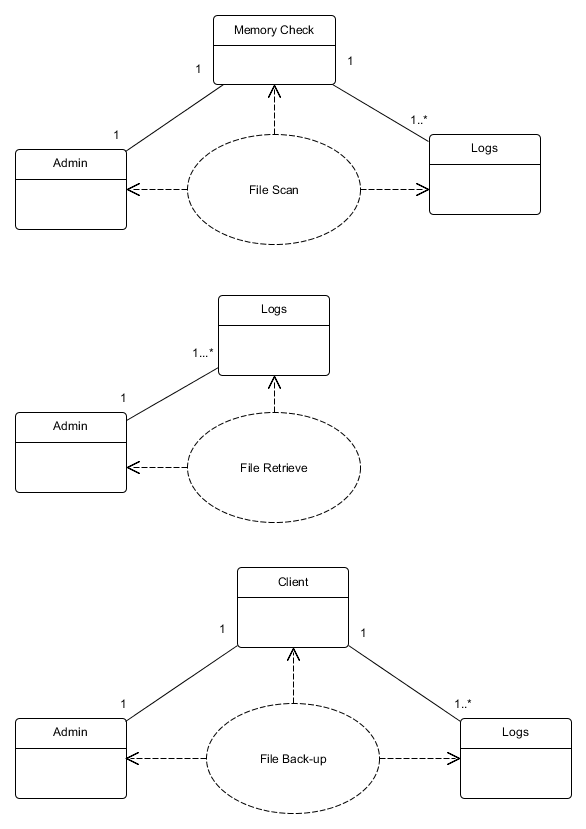
### Communication Diagram



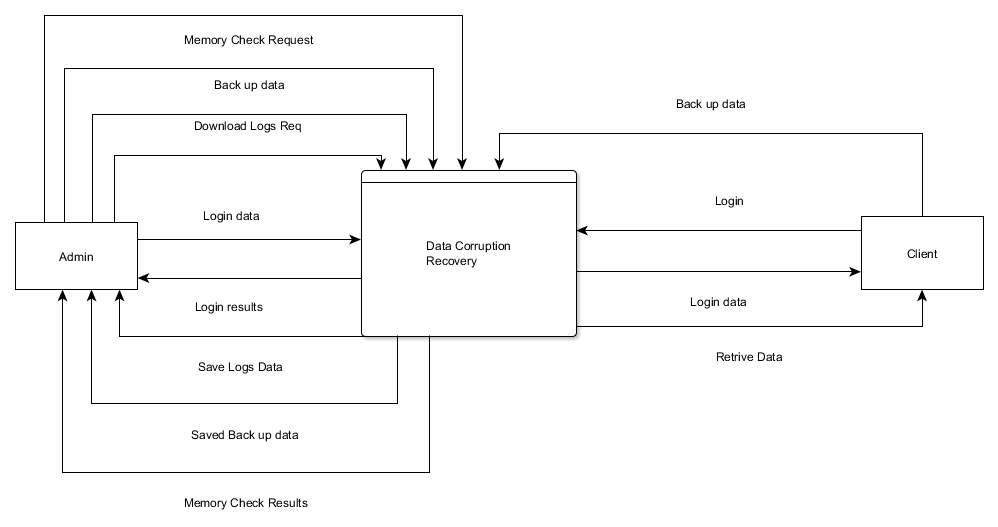
### Component Diagram



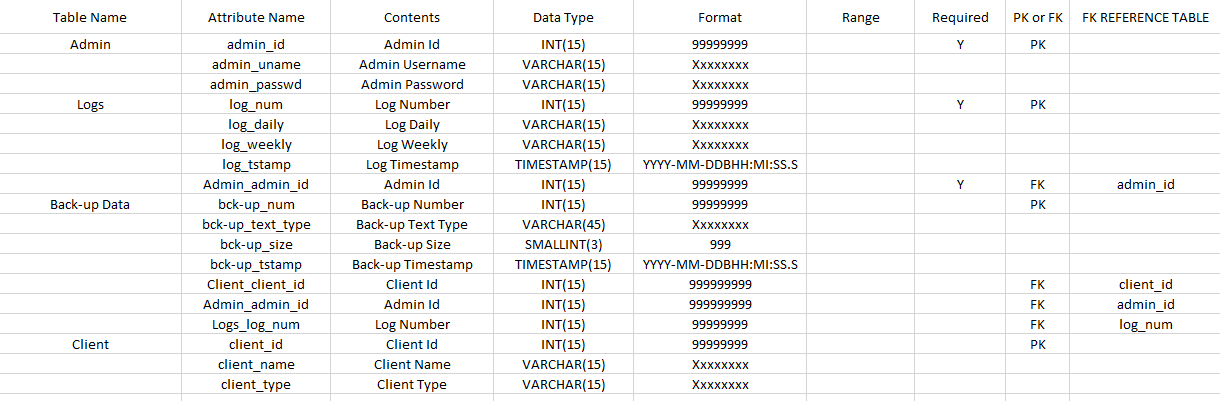
### Composite Diagram



### Context Diagram



### Data Dictionary



### Data Flow Diagram

DFD Fragment

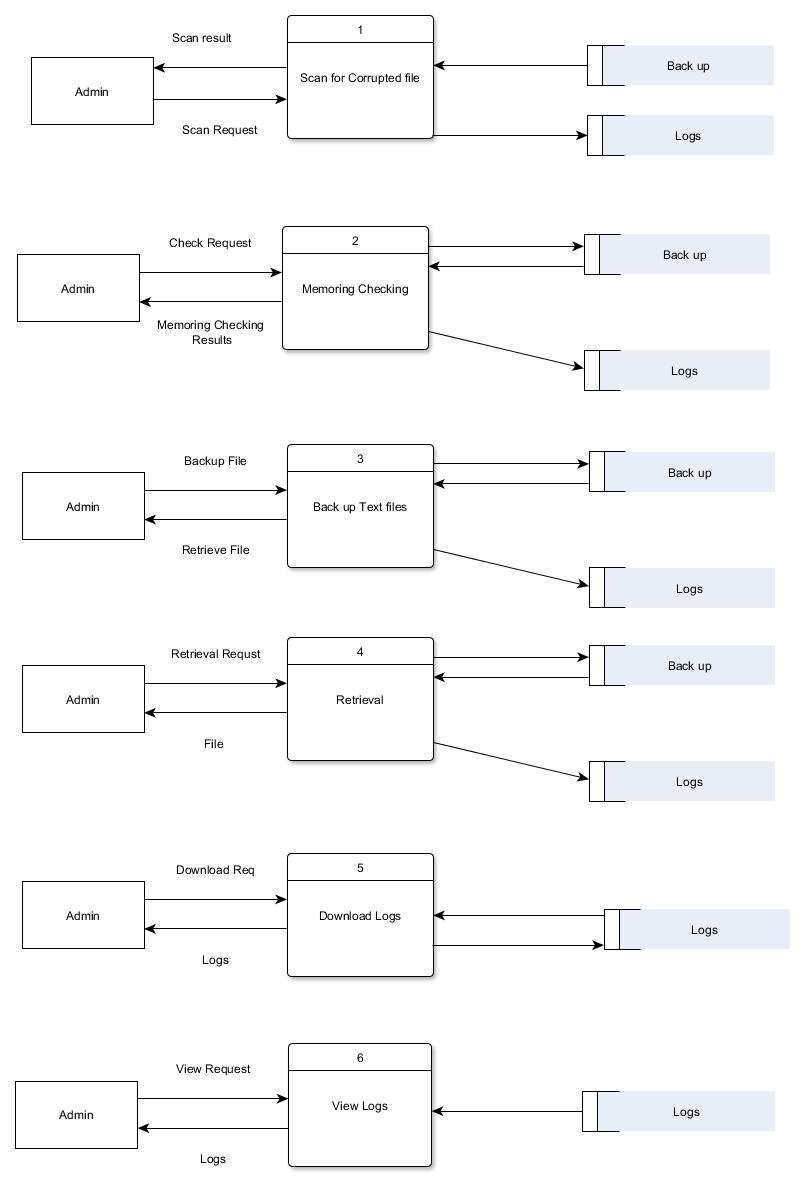
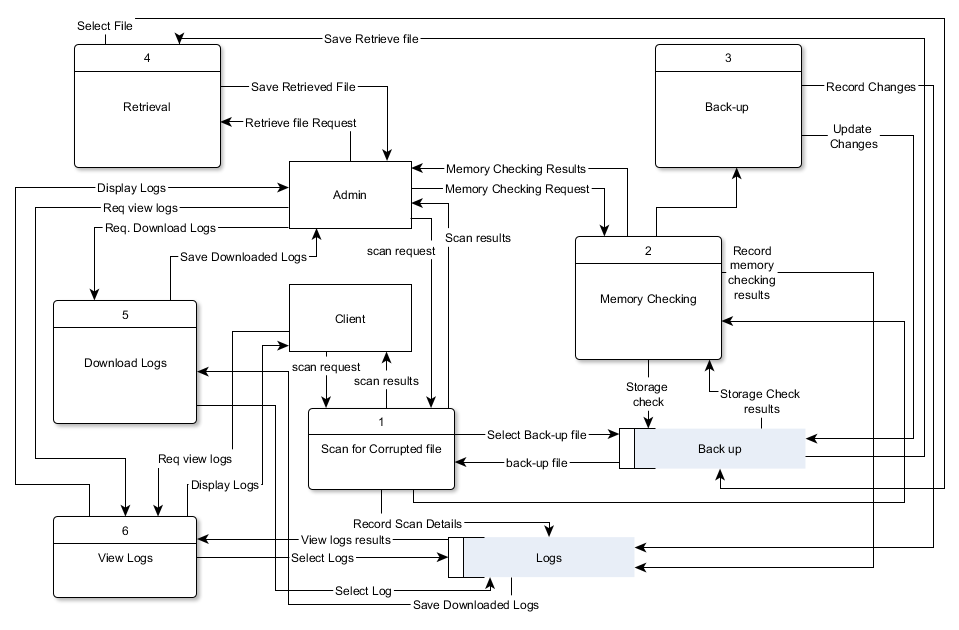
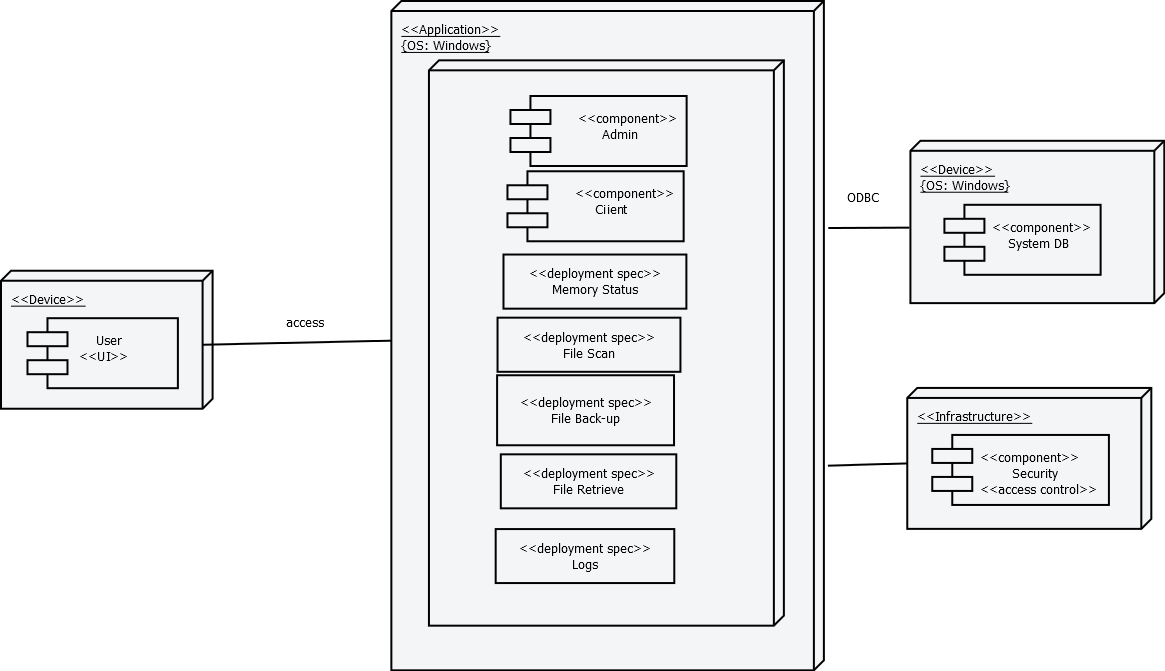


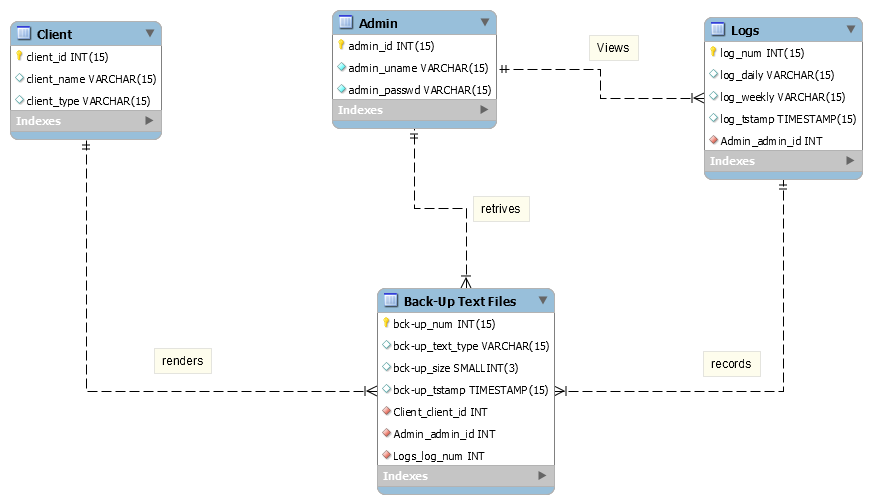
Diagram 0



### Deployment Diagram

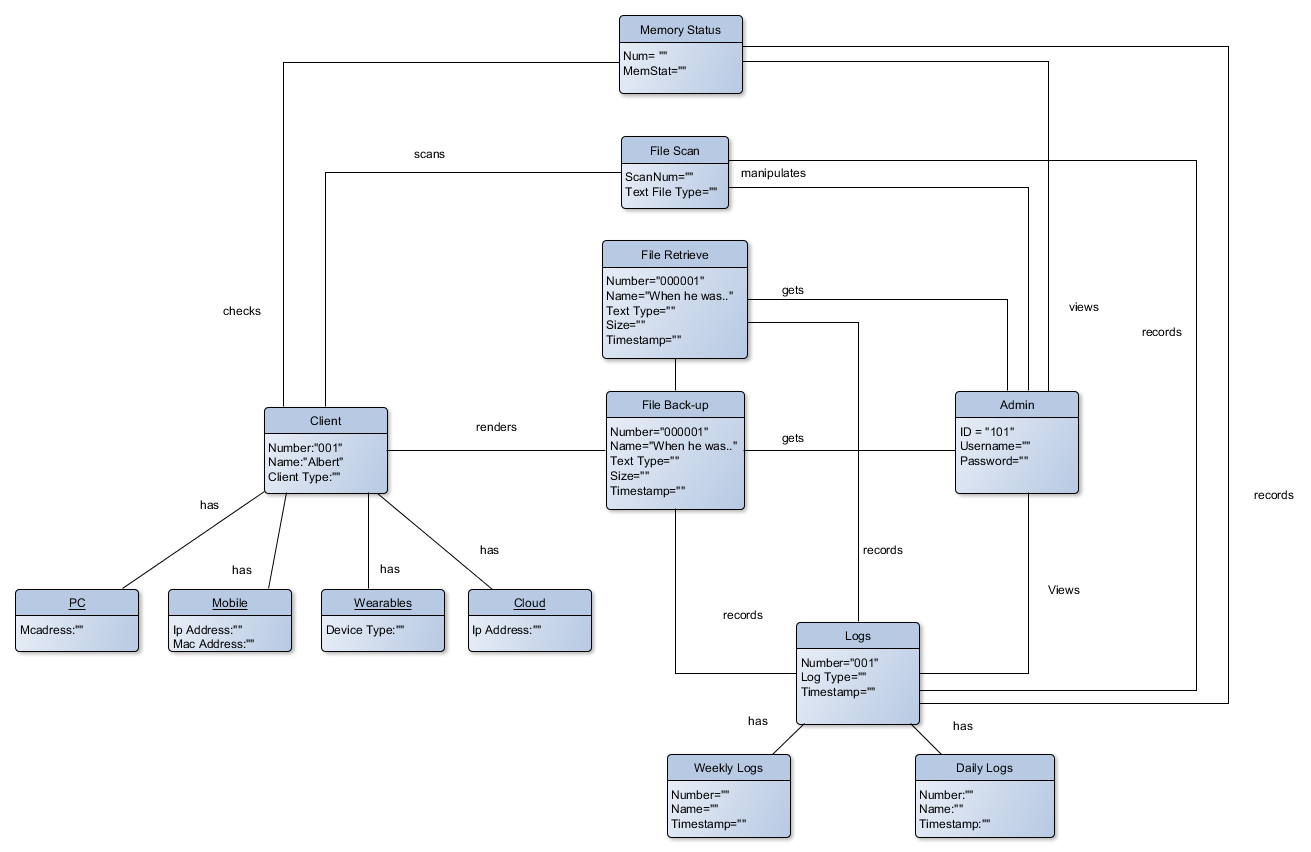


### Entity-Relationship Diagram

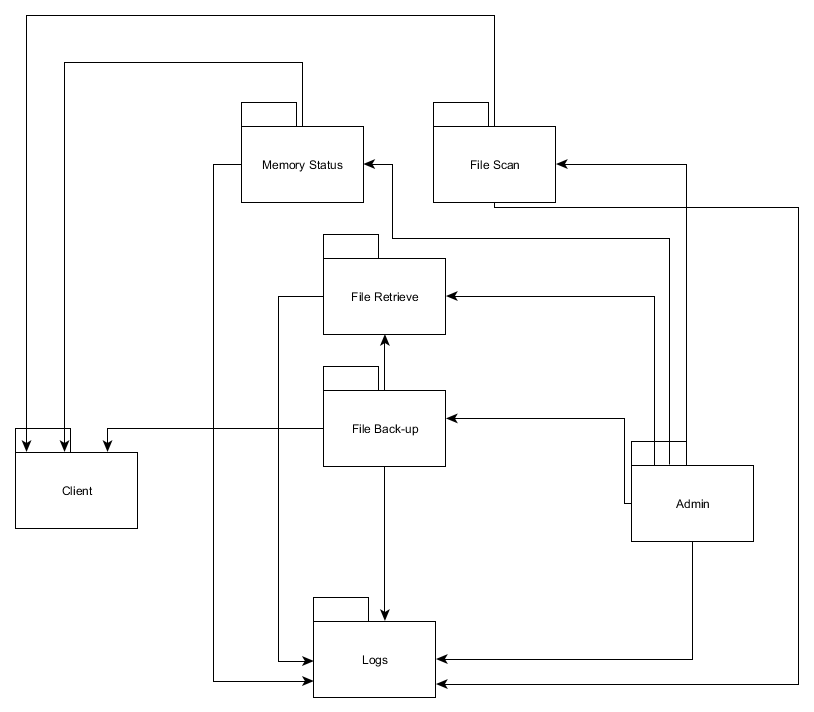


### Interaction Diagram

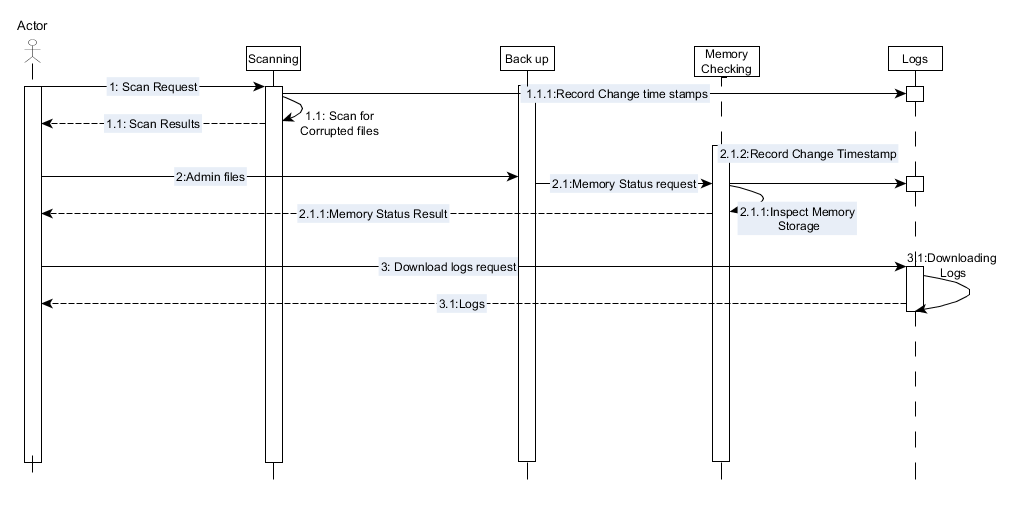
### Object Diagram



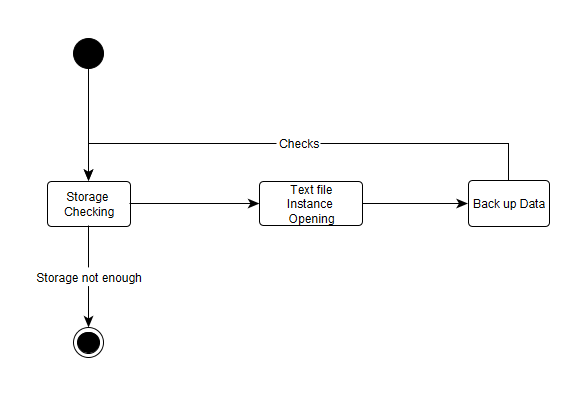
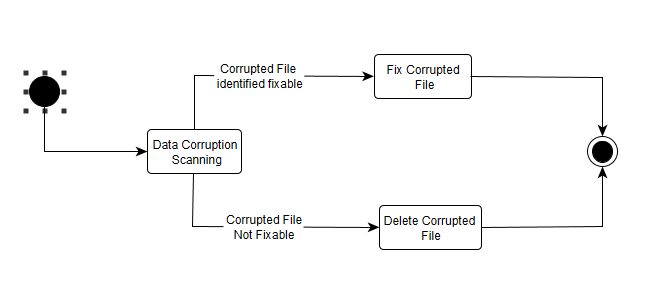
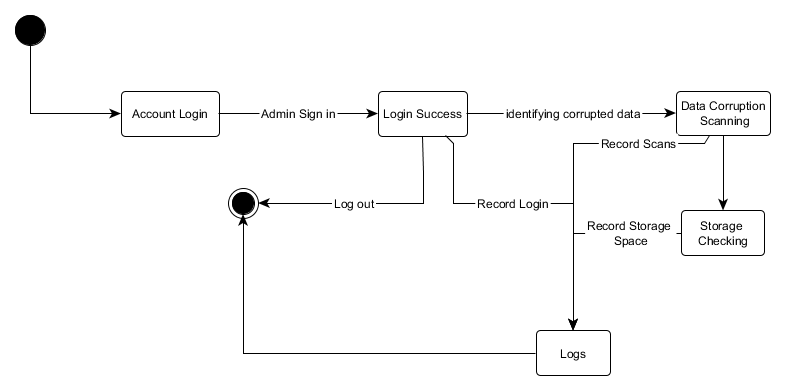
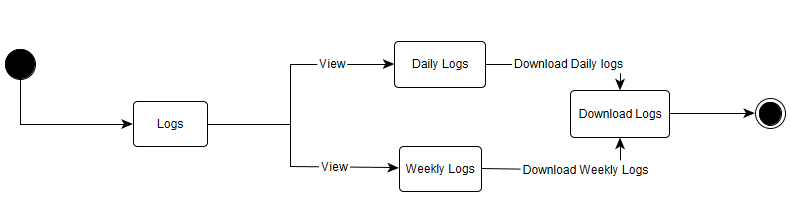
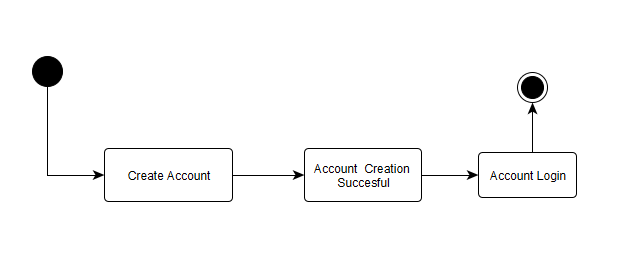
### Package Diagram



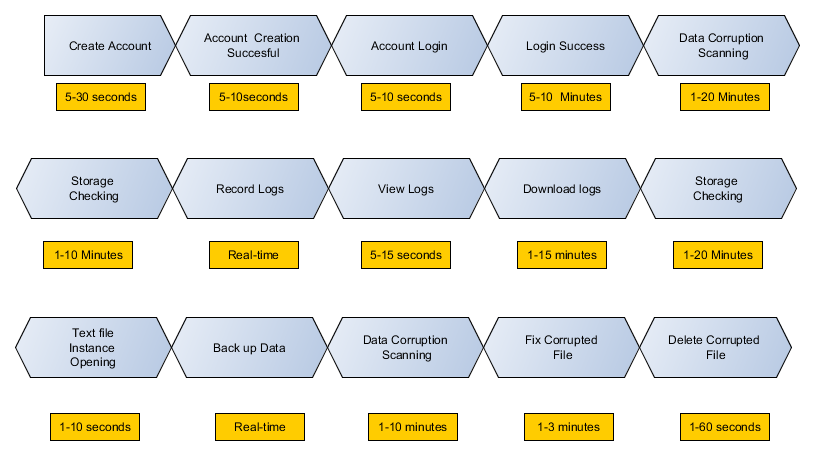
### Sequence Diagram



### State Diagram

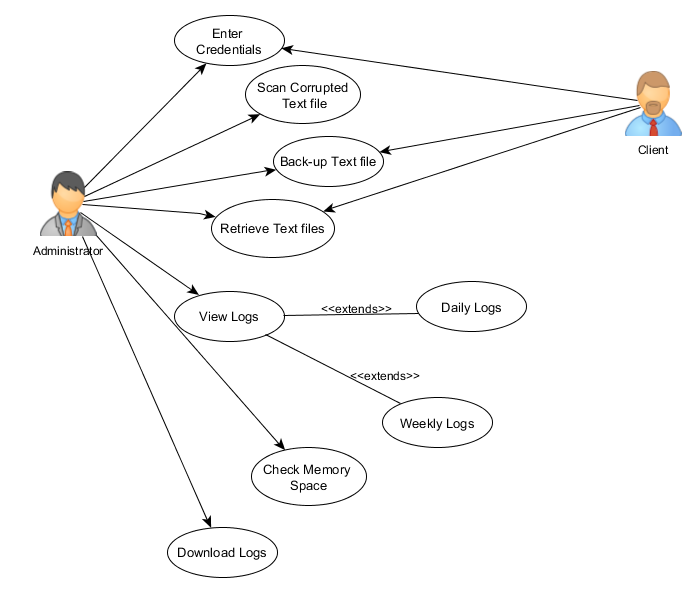


### Timing Diagram



### Use Case

Use Case Diagram



Use case full description

|  |  |  |
| --- | --- | --- |
| Number | PICDS-UC01 | |
| Use Case Name | User Authentication | |
| Scenario | User wants to Log in to the system | |
| Triggering Event | Log In | |
| Brief Description | When the admin access his files authentication is required. | |
| Actor(s) | Administrator | |
| Related Use Cases | ---- | |
| Stakeholders | Administrator, Client | |
| Precondition | User must plugin the Flash drive to the PC | |
| Post Condition | Administrator must able to access the system | |
| Basic Flow: | Actor’s Action | System’s Response |
|  | 1: Opens the application | 1.1: System displays the login page of the application |
| 2: User types its username/password |  |
| 3: User clicks login button | 3.1: System checks the database if the username and password is correct |
|  | 3.2: username and password matched System will message (SM01) |
|  | 3.3: System appears the main page of the application |
|  | 3.4: System scans for corrupted text file |
|  |  |  |
| Alternative Flow: |  | 3.2: If username and password of Administrator didn’t match |
|  |  | 3.3: System displays message (SM02) |

|  |  |  |
| --- | --- | --- |
| Number | PICDS-UC02 | |
| Use Case Name | User back-ups text files | |
| Scenario | User request for back-up | |
| Triggering Event | Back-up Text files | |
| Brief Description | As a prevention mechanism on data corruption, an automatic real-time backup is running on background. | |
| Actor(s): | Administrator | |
| Related Use Cases | User Authentication | |
| Stakeholders | Administrator, Client | |
| Precondition | User opens an text application | |
| Post condition | User was able to back-up the text file automatically while working on text application | |
| Basic Flow: | Actor Action | System Response |
|  | Step 1: User Plug’s in the Flash Drive | 1.1: System checks if the flash drive is less than 80% of the storage |
|  | 1.2: System scans for corrupted text file |
|  | 1.3: System will identify If the corrupted data can be fixed or not. |
|  | 1.4 Fix corrupted data |
| 2: User opens text file application | 2.1: System will start executing automatic real-time back up |
|  | 2.2: System will calculate if the storage can accumulate the back-up file. |
|  | 2.3: System backed-up text file |
|  |  |
|  |  |  |
| Alternative Flow: |  | 1.3: System will identify If the corrupted data can be fixed or not. |
|  | 2. User Agrees to delete corrupted data | * 1. Delete Corrupted data. |

# 

|  |  |  |
| --- | --- | --- |
| Number | PICDS-UC03 | |
| Use Case Name | User views log | |
| Scenario | Admin views logs | |
| Triggering Event | View Logs | |
| Description | The changes made in the Flash drive are recorded in the logs. | |
| Actor(s) | Administrator | |
| Related Use Case | Use Authentication | |
| Stakeholders | Administrator | |
| Precondition | User is logged in as Admin | |
| Post Condition | User is able to view the in/out of text file | |
| Basic Flow: | Actor Action | System Response |
|  | 1: User view logs. | 1.1: System displays “Daily logs” and “Weekly logs” |
| 2. User select “Daily logs” | 2.1: System displays “Daily logs” page |
| 3. User select “Weekly logs” | 3.1: System displays “Weekly logs” page |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Number | PICDS-UC04 | |
| Use Case Name | User retrieve text file | |
| Scenario | User wants to get the back-up text file | |
| Triggering Event | Retrieve Text File | |
| Brief Description | Admin retrieves files that we’re backed up. | |
| Actor(s) | Administrator | |
| Related Use Case | User Authentication, User Back-Up | |
| Stakeholders | Administrator, Client | |
| Precondition | User has a back-up text files | |
| Post condition | Admin was able retrieve the back-up files | |
| Basic Flow: | Actor Action | System Response |
|  | 1. Admin will Select & Copy the file to retrieve |  |
| 1. Admin will Paste the retrieved file. |  |

|  |  |  |
| --- | --- | --- |
| Number | PICDS-UC05 | |
| Use Case Name | Scan Corrupted Text file | |
| Scenario | User wants to know if there’s any corrupted text file | |
| Triggering Event | Scan | |
| Brief Description | System scans for the corrupted text file | |
| Actor(s) | Administrator | |
| Related Use Case | User Authentication | |
| Stakeholders | Administrator, Client | |
| Precondition | Admin is logged in | |
| Post condition | Admin was able to log out the system | |
| Basic Flow: | Actor Action | System Response |
|  | 1. Log In’s the application |  |
|  | 1. Automatically scans for the corrupted text file |

|  |  |  |
| --- | --- | --- |
| Number | PICDS-UC06 | |
| Use Case Name | Download Log | |
| Scenario | User wants to review logs | |
| Triggering Event | Logs | |
| Brief Description | To review the systems activity | |
| Actor(s) | Administrator | |
| Related Use Case | User Authentication, View logs | |
| Stakeholders | Administrator, Client | |
| Precondition | User must have logged in as Administrator | |
| Post condition | Admin was able to download the logs | |
| Basic Flow: | Actor Action | System Response |
|  | 1. Admin clicks “View Logs” | 1.1 Displays the View Logs page |
| 1. Admin clicks “Download Logs” | * 1. System download logs |

|  |  |  |
| --- | --- | --- |
| Number | PICDS-UC07 | |
| Use Case Name | Memory Check | |
| Scenario | User wants to know the memory space of the flash drive | |
| Triggering Event | Memory Checking | |
| Brief Description | Able to identify the available memory space of the flash drive | |
| Actor(s) | Administrator, client | |
| Related Use Case | User Authentication | |
| Stakeholders | Administrator, Client | |
| Precondition | User must have logged in | |
| Post condition | User was able to know the memory space of the flash drive | |
| Basic Flow: | Actor Action | System Response |
|  | 1. Enter credentials | 1.1 Automatically Scans for the corrupted text file |
|  | 1. Scanned Complete |
|  |  | 1. Checks the memory space |

|  |  |  |
| --- | --- | --- |
| Number | PICDS-UC08 | |
| Use Case Name | Log out | |
| Scenario | User wants to log out from the system | |
| Triggering Event | Log out | |
| Brief Description | Log out from the system | |
| Actor(s) | Administrator | |
| Related Use Case | User Authentication | |
| Stakeholders | Administrator, Client | |
| Precondition | Admin is logged in | |
| Post condition | Admin was able to log out the system | |
| Basic Flow: | Actor Action | System Response |
|  | 1. Clicks the Logout button |  |
|  | 1. Destroys the session and logs out from the system 2. Displays the Log In page |

|  |  |
| --- | --- |
| System Message | |
| SM01 | You logged in successfully! |
| SM02 | Username or password is incorrect! |

# Introduction

## Project Context

Data corruption has always been a problem in computing. With the imminent implementation of internet of things, solutions on data corruption must be address to avoid future complications regarding data loss.

## Purpose and Description

Technology advances rapidly, this also means that security issues such as data corruption are evolving as well and we need to address this issue since huge consequences are at risk. We may not be able to prevent the event of corruption but we might able to minimize or neglect the impact by being prepared for it.

## Objectives

General Objectives:

* + - * This Project aims to create a forensics tool capable of saving critical information from data loss.

Specific Objectives:

* + - * To be able to provide a real-time back up.
      * To be able to create a flexible forensic tool in terms of mobility.
      * To be able to produce a seamless data corruption prevention tool.

## Scope and Limitations

This Project aims to help students and/or office-workers to secure their information or data on availability issues, whom are using windows based OS- 7 and higher. The project primarily focuses on data corruption and prevention. It also assumes that the interaction are just between the players of the system such as the admin, system, and user interactions and no other irrelevant factors are included. The prevention of data corruption are only focus on the text documents, and it is beyond the systems capability if the flash drive itself became corrupted since it should be the users responsibility to protect the hardware itself.

The flash drive that will be used should have a NTFS file structure with a memory space not below 4gig. The optimal performance of the application could be achieve if the memory space of the flash drive is less than 90%.

# Related Literature

**Causes of Data Corruption**

The common causes of corrupted data is power outages or other related to power failure, Inappropriate shutdown caused by the user, hardware problems or failures such as hard drive failure, bad sectors, and etc. another reason is software failures caused by bad programming, particularly if it results in either hard restarts or data that is saved incorrectly. Any of these causes can result in a corrupted hard drive directory. A corrupted hard drive directory can cause files to apparently "go missing" and lead to further data loss or corruption. (Wikipedia : Data Corruption, n.d.), (Smith, 2014)

**Data Corruption**

Some file systems, such as Btrfs, HAMMER, ReFS, and ZFS, use internal data and metadata check summing to detect silent data corruption. In addition, if a corruption is detected and the file system uses integrated RAID mechanisms that provide data redundancy, such file systems can also reconstruct corrupted data in a transparent way. This approach allows improved data integrity protection covering the entire data paths, which is usually known as end-to-end data protection, compared with other data integrity approaches that do not span different layers in the storage stack and allow data corruption to occur while the data passes boundaries between the different layers. (Wikipedia : Data Corruption, n.d.)

**Checksum**

A checksum is a count of the number of bits in a transmission unit that is included with the unit so that the receiver can check to see whether the same number of bits arrived. If the counts match, it's assumed that the complete transmission was received. Both TCP and UDP communication layers provide a checksum count and verification as one of their services. (Search Security, n.d.)

**File System**

A file system is a part of the operating system that determines how files are named, stored, and organized on a volume. A file system manages files and folders, and the information needed to locate and access these items by local and remote users. Microsoft Windows Server 2003 supports both the FAT and NTFS file systems. NTFS allows you to gain the maximum benefits for the needs of today’s enterprise business environments from Windows Server 2003, such as increased security, more robust and reliable performance, as well as a design for greater storage growth, features not found in FAT. (How to Repair Corrupted Windows System Files with the SFC and DISM Commands, n.d.)

**RAID Technology**

With RAID technology, data can be mirrored on one or more disks in the same array, so that if one disk fails, the data is preserved. RAID also offers the option of reading or writing to more than one disk at the same time in order to improve performance. In this arrangement, sequential data is broken into segments which are sent to the various disks in the array, speeding up throughput. A typical RAID array uses multiple disks that appear to be a single device so it can provide more storage capacity than a single disk. (Beal)

**Assembly Language**

In implementing the corrupted data retrieval we can use the Assembly language. It is a low leveled programming language for a computer, or other programmable device specific to a particular computer architecture meaning you can communicate through the device one on one. Each assembly language is specific to a particular computer architecture, in contrast to higher level of programming which are generally portable across multiple architectures, but require interpreting or compiling. Assembly language may also be called symbolic machine code. Assembly language also use mnemonic to show a list of each low leveled machine instruction or operation. Many operation takes one or more operands to be able to complete instruction. Assembly language is converted into executable machine code by a utility program referred to as an assembler like NASM, MASM, etc. (Assembly Language, n.d.) , (Assembly Programming Tutorial, n.d.)

**An Analysis of Data Corruption in the Storage Stack**

An important threat to reliable storage of data is silent data corruption. In order to develop suitable protection mechanisms against data corruption, it is essential to understand its characteristics. In this paper, we present the ﬁrst large-scale study of data corruption. We analyze corruption instances recorded in production storage systems containing a total of 1.53 million disk drives, over a period of 41 months. We study three classes of corruption: checksum mismatches, identity discrepancies, and parity inconsistencies. We focus on checksum mismatches since they occur the most.

We found more than 400,000 instances of checksum mismatches over the 41-month period. We ﬁnd many interesting trends among these instances including: (i) near line disks (and their adapters) develop checksum mismatches an order of magnitude more often than enterprise class disk drives, (ii) checksum mismatches within the same disk are not independent events and they show high spatial and temporal locality, and (iii) checksum mismatches across different disks in the same storage system are not independent. We use our observations to derive lessons for corruption-proof system design. (Bairavasundaram, Arpaci-Dusseau, Arpaci-Dusseau, Goodson, & Schroeder, 2008)

**Method and apparatus for recovering data from damaged or corrupted file storage media**

An automated method and apparatus for identifying and copying lost files from a mass data storage device of a computer when file system information (as opposed to the actual data files) stored on the mass data storage device has been corrupted or destroyed. The mass data storage device is scanned on a sector-by-sector basis in order to attempt to identify sectors containing file system data structures and file attributes. Identification is made using data signature and/or pattern matching filters. The location of, and any valid information found in, such sectors is used then to derive information useful in locating files to be copied to another storage device. For example, in a FAT, NTFS or other cluster-oriented file system, if information on the number of sectors per cluster (SPB) is not available from a boot directory, it and a cluster base (the starting sector of cluster 0) are calculated using the physical location of the beginning sectors of the directories or folders. When a starting cluster is known from a directory entry, but not additional file allocation information, a cluster chain may be reconstructed utilizing one or more of several disclosed methods.

**Systems and methods for automated maintenance and repair of database and file systems**

The present invention relates generally to database and file system management and, more particularly, to automatic database and file system maintenance and repair to ensure data reliability. Various aspects of the present invention relate to responding and correcting data corruptions at a data page level for all data page types, as well as to recovery (including rebuild or restore operations) for various scenarios including, without limitation, index page corruptions (clustered and non-clustered), data page corruptions, and page corruptions in the log file.

**System and method for detecting correcting and discarding corrupted data packets in a cable data delivery system**

A method and apparatus for detecting errors and improving quality in real-time data transmissions is provided. In one embodiment, the packet header checksum field is turned off to allow uninterrupted transmission of data packet payloads. A checksum added to each independent data segment in the datagram payload permits each data packet to be examined separately, resulting in improved transmission quality.

**VALIDATING POPULATION VIABILITY ANALYSIS FOR CORRUPTED DATA SETS**

Diffusion approximation (DA) methods provide a powerful tool for population viability analysis (PVA) using simple time series of population counts. These methods have a strong theoretical foundation based on stochastic age-structured models, but their application to data with high sampling error or age-structure cycles has been problematic. Recently, a new method was developed for estimating DA parameters from highly corrupted time series. We conducted an extensive cross-validation of this new method using 189 long-term time series of salmon counts with very high sampling error and non-stable age-structure fluctuations. Parameters were estimated from one segment of a time series, and a subsequent segment was used to evaluate the predictions regarding the risk of crossing population thresholds. We also tested the theoretical distributions of the estimated parameters. The distribution of parameter estimates is an essential aspect of a PVA because it allows one to calculate confidence levels for risk metrics. This study is the first data-based cross-validation of these theoretical distributions. Our cross-validation analyses found that, when parameterization methods designed for corrupted data sets are used, DA predictions are very robust even for problematic data. Estimates of the probability of crossing population thresholds were unbiased, and the estimated parameters closely followed the expected theoretical distributions.

# Technical Background

There are existing back up applications right now such as DirSyncPro. That application required you to create a job where the user needs to create a jobset containing the source and destination paths or directories, it also allows you to filter the information being backup. You can also choose what type of synchronization you want.

The difference between our application and the DirSync Pro is that, the back-up files aren’t focus on the existing once. We focus our back-up on the text files that are currently creating or writing by the user, hence creating a backup that could incrementally mirror the files in real-time, the advantage of this back is that, your files are safe if an event causing corruption happen during the creation period of the file. The second advantage of this application is that it backups your file seamlessly allowing you to back up each and every file you create. Since this application monitors backup real-time.

Most of the backup applications right now often backup huge amount of information or specifying a complete file, which results to a slower file transfer or mirroring. But since the application-backup’s real-time it only transfers small amount of information making it convenient.

# Appendices