<Database Management System>

<Nikita Voronin>

|  |  |
| --- | --- |
| **Modification date** | **Modification description** |
| 02/15/2019 | Document Created |
| 03/02/2019 | Introduction and Overall Description are updated. |
| 03/04/2019 | Use – Cases added |
| 03/07/2019 | Class UML and description added |
|  |  |
|  |  |
|  |  |

Table of Contents

1. Introduction 3

1.1 Purpose 3

1.2 Scope 3

1.3 Definitions, Acronyms and Abbreviations 3

1.4 References 4

1.5 Technologies to be used 4

1.6 Overview 4

2. Overall Description 4

2.1 Use-Case Model Survey 4

2.2 Architecture diagram 4

2.3 Assumptions and Dependencies 5

3. Specific Requirements 6

3.1 Use-Case Reports 7

3.2 Class UML and Description 8

3.2 Supplementary Requirements 9

1. Introduction

## Purpose

To build a generic database management application that allows user to save and manipulate his data using a console or graphic user interface.

## Scope

* Create/Delete custom databases
* Create/Delete custom tables with user specified columns
* Add/Delete records to the user defined tables
* Edit records
* Observe database contents
* Save all data in text and binary files
* Load all data from files
* Utilize OpenGL or another graphics library
* Registration and Log In systems
* Password encryption

The scope of this project is creating an efficient multi-user database management system, in which the user can perform basic operations stated above. Application must be user friendly even for people with no to low programming experience.

## Definitions, Acronyms and Abbreviation

**User** - Someone who interacts with the GUI or Command Line

**OOP** - Object Oriented Programming

**C++17** – Programming Language used in this project

**DBMS** - Database Management System

**GUI** - Graphical User Interface

**Storage** – “Container” of all the databases

## References

## IEEE SRS Format.

## Technologies to be used

# Programming Language - C++17

Language Libraries - Standard Library

# Development tool - Visual Studio 2017 IDE

# Graphics Framework - OpenGL

## Overview

This project is an application to create and manage databases. This application consists of many classes to separate functionality according to the OOP paradigm.

# Overall Description

## Use-Case Model Survey

// To Be Added

## Integrated Application Environment

This DBMS will be built using C++, an OOP language, which will allow this application to run on devices running operative systems such as Windows, Mac OS, or Linux.

## Architecture Diagram

// To Be Added

## Assumptions and Dependencies

* This project will not have an ability to query the database. Instead, user will use the console or graphic interface to choose which database, table, or record to add, delete, or in case of the record, edit.
* Application may turn out to be not usable as a library since all the data manipulations are done via the user interface. However, if needed, application’s architecture allows such modifications.
* Application may be modified to be multi-user. Each user will have his own storage. If this concept will be implemented, each user will have to log in the DBMS system using his personal login and password.

# Specific Requirements

## Use-Case Reports

**Use-Case #1: Register and log in into the DBMS account.**

**Pre-Condition** – Having the DBMS application up and running

**Description –** User is responsible for creating new account or logging into the existing account whenever he starts the application.

**Scope:**

* **Register** – if the User wants to register the new account, he will be prompted to enter new username and new password. The password will be encrypted using the sha-256 algorithm. Both username and the encrypted password will be added to the file called “user.txt” which has credentials of all users in a following format: “<username:password>”.
* **Log In** – if the User wants to log in his existing account, he will be prompted to enter his username and password. Entered password will be encrypted and compared to the saved encrypted password. This way we avoid saving unencrypted password anywhere in the application for the security reasons. If both the username and password are correct, the main menu will open. Else, user will be prompted to enter his credentials once more.

**USE CASE NORMAL EVENT FLOW DIAGRAM**

DBMS application starts.

User is prompted to choose if he wants to register or log in.

Register.

Input new username and password.

Log in.

Input existing username and password.

Save new credentials to the file user.txt

Compare entered data with data saved in user.txt

Open main menu.

**Use-Case #2: Create and manage User’s storage.**

**Description –** The empty storage is created for every new registered user. User can customize it by adding custom databases and tables. User can manipulate existing data.

**Pre-Condition** – User must be logged in to the DBMS.

**Scope:**

* + **Creating databases** – the User will be able to create new database by choosing a corresponding option from the menu and entering new database’s name.
  + **Creating tables**- the User will be able to create new tables once by choosing a corresponding option from the menu, entering the name of the database in which table will be added, entering the name of the new table, and specifying the columns of the new table.
  + **Adding records to tables -** the User will be able to add records to any of his tables by choosing a corresponding option from the menu, entering the name of the database in which the desired table is, entering the name of the table in which record will be added, and finally specifying values for each column of the table. Each record will be assigned a new record id, which is unique for every record in the table.
  + **Deleting databases -** the User will be able to delete any of his databases by choosing a corresponding option from the menu and entering the name of the database that is to be deleted.
  + **Deleting tables** – the User will be able to delete any of his tables by choosing a corresponding option from the menu, entering the name of the database in which the desired table is, and finally, entering the name of the table that is to be deleted.
  + **Adding and Deleting records** – the User will be able to add and delete any of the records from the specified table and specified database.

**Normal Flow of Events –**

* User chooses an option from the main menu:

1. Show storage contents.
2. Add database to the storage.
3. Delete database from the storage.
4. Add table to the database.
5. Delete table from the database.
6. Add data into table.
7. Delete data from table.
8. Edit data.
9. Exit.

* User changes the storage contents by performing a chosen operation.
* Current storage content printed to the screen.
* User chooses another operation until he decides to exit.
* Storage is saved to the text and binary file.

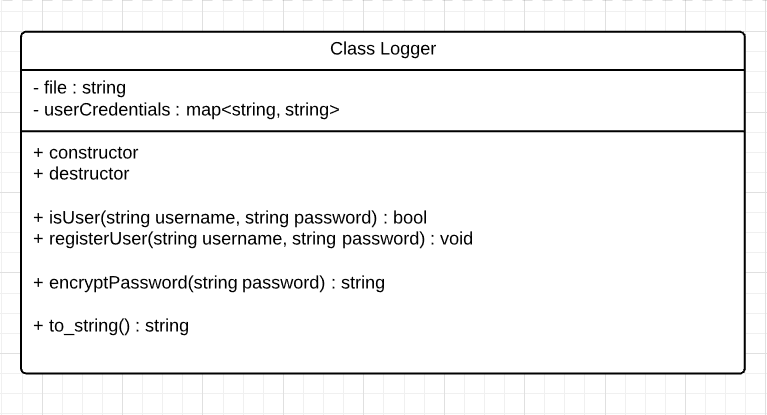
## Class UML and Description

# 3.21 Register and log in into the DBMS account.

**Logger**

Class Logger contains all the methods necessary to register a new User, check credentials of the existing user, and log the existing user in. When registering a new User, Logger uses a picosha class which is copyrighted under MIT license. All the code for picosha can be found here:

[**https://github.com/okdshin/PicoSHA2/blob/master/picosha2.h**](https://github.com/okdshin/PicoSHA2/blob/master/picosha2.h)

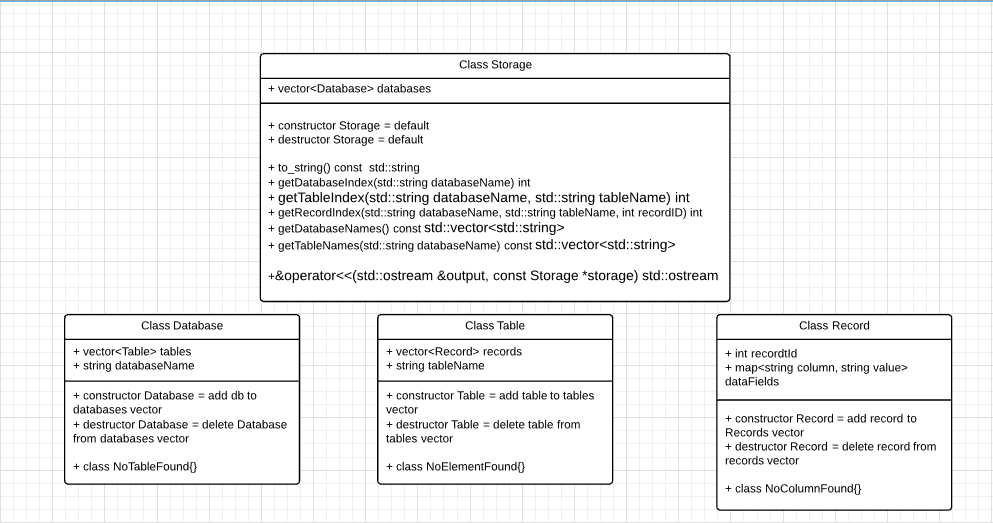
****

# 3.22 Create and manage User’s storage.

**Storage – database file system.**

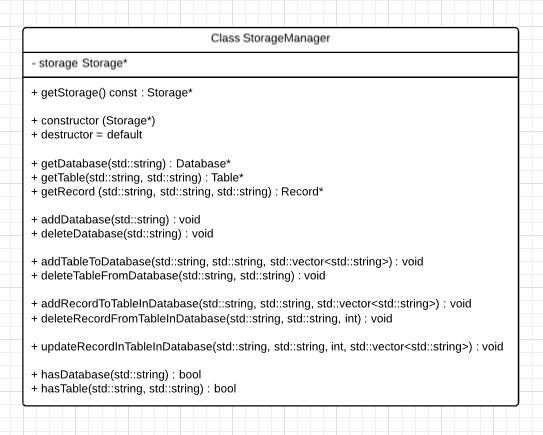
As I have already stated above, all the user data is represented as a single storage. Every user has a storage file “username.txt”. Storage has a list of databases and some methods for outputting the storage and getting some contents from it.

The storage, database, table, and record class are not directly related, meaning that none of them is a subclass or a parent class of another one. However, each Storage has list of databases, each database has a list of its tables, each table has a list of its records, and each record has a list of its values.



**Storage Manager – controller for the data in the Storage**

The Storage Manager is the class which contains the Storage of the User and has all the necessary methods for manipulating the databases, tables, records, and their contents. Storage Manager does not read Storage from file, nor save Storage to File.



## Supplementary Requirements