Account Manager

Technical Specification

Damian Zylski and Barry Hoinacki JR

2019

Table of Contents

[Revision History 2](#_Toc7455811)

[Introduction 3](#_Toc7455812)

[Project Scope 3](#_Toc7455813)

[Glossary 3](#_Toc7455814)

[Requirements Description 4](#_Toc7455815)

[Summary 4](#_Toc7455816)

[Assumptions and Dependencies 4](#_Toc7455817)

[Use Cases 4](#_Toc7455818)

[Actors 5](#_Toc7455819)

[Use Case Details 5](#_Toc7455820)

[Use Case 1 5](#_Toc7455821)

[Use Case 2 6](#_Toc7455822)

[Use Case 3 8](#_Toc7455823)

[Use Case 9 9](#_Toc7455824)

[Use Case 10 10](#_Toc7455825)

[External Interface Requirements 11](#_Toc7455826)

[Technical Solutions Description 11](#_Toc7455827)

[Interfaces 11](#_Toc7455828)

[Architecture 11](#_Toc7455829)

[User 12](#_Toc7455830)

[Client 12](#_Toc7455831)

[Account Manager 12](#_Toc7455832)

[Database File 12](#_Toc7455833)

[Data Model 14](#_Toc7455834)

[Appendix A: Issues List 14](#_Toc7455835)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Date | Summary |
| 1.0 | Damian Zylski, Barry Hoinacki Jr | 4/23/2019 | Initial document creation, Introduction. |
| 1.1 | Damian Zylski | 4/23/2019 | Typed up the project scope and out of scope fields. |
| 1.2 | Barry Hoinacki Jr | 4/28/2019 | Started to update glossary, typed up summary, started on assumptions and dependencies. |
| 1.3 | Barry Hoinacki Jr | 4/29/2019 | Updated glossary more, External interfaces updated, Interfaces updated, and did Technical solution description. |
| 1.4 | Damian Zylski | 4/29/2019 | Updated architecture diagram and use case documents |
| 1.5 | Barry Hoinacki Jr | 4/29/18 | Updated actors in use case documents and updated the various components in the Architecture section. |
| 1.6 | Damian Zylski | 4/29/2019 | Updated use case diagram, corrected typos in use cases |
| 1.7 | Damian Zylski | 4/29/2019 | Added use cases to document. Updated summary, technical solutions, architecture, glossary, dependencies and assumptions, and data model |
| 1.71 | Barry Hoinacki Jr | 4/29/2019 | Changed wording in glossary a bit and updated table of contents. |
| 1.8 | Barry Hoinacki Jr | 4/30/2019 | Updated database file in architecture and included a JSON file example. |
| 1.9 | Damian Zylski | 4/30/2019 | Updated glossary, and Database File section in Architecture section, issues list, and data model section |

# Introduction

An account manager is a software application that stores a list of accounts, such as a gmail account, and their respective details, such as passwords, to a database file. This document describes requirements and technical documentation for a new account manager.

## Project Scope

The software considers the following to be in scope

* The ability to use the account manager anywhere (portable)
* The ability create a database and add an account to it
* The ability to load an existing account
* The ability to edit and delete accounts in a database
* The ability to store account information securely using Blowfish encryption

The software considers the following to be out of scope

* The ability to use autofill on a particular website or login screen
* The ability to export database file in different file formats
* The ability to set database policies and usage rules
* File recovery in the case the file is accidently edited or deleted
* Client distribution and installation.

# Glossary

For this document, the terms below are defined as shown.

|  |  |
| --- | --- |
| Term | Definition |
| Encryption | Process of taking words and other such data and masking it (using random characters) by using a specific method (in this program Blowfish is used) so whatever was in the file would look incomprehensible so the sensitive data in the file cannot be viewed. |
| Decryption | Process of taking information that was encrypted and trying to reverse the effects of masking it to make it readable/usable, usually by using the reverse of the same method used to encrypt it. |
| Blowfish | A method of encryption/decryption |
| Account | A collection of information that is needed to either label or be used to login like a username and password tied to a specific user in a service such as an email or social media website or a login screen like for logging into Windows. |
| Database | A storage of various information. In this case a JSON file was used in storing the master password along with various accounts and their information. The JSON file is always encrypted |
| Master Password | The password used to be able to load a database so you can add, edit, delete, or see the accounts in the database. |
| Scrum | An incremental software development process with defined work periods after which a workable piece of software is delivered to the user. |
| JSON | A specific file type that is used to sort out information in a specific way. It simplifies reading and writing and is always encrypted. |
| Account Manager | The software that is used by the user to store, display, and edit the database of user accounts. |
| System | Another name for the account manager. |
| Client | The machine running the Account manager software. |

# Requirements Description

This section describes the business requirements.

## Summary

An account manager is used to store accounts for various things such as email or social media sites so that you will be able to, using encryption, securely keep all the login information for all the accounts the user adds. This way the user would not need to constantly try different passwords/username combinations to log into an account. As such the requirements center around how the account manager will securely create databases of accounts.

## Assumptions and Dependencies

The project assumes the following

* The user will be able to remember where the database file is located and the master password for that database file. The software cannot connect remotely to a database file
* The user already has accounts that they want to store into the account manager.
* The target operating system is Windows 7 or newer. There is no mobile version of this program

## Use Cases

This documentation focuses on the key use cases the software will meet. While other use cases are present, and documented in the scrum tool, the following have been identified as either being key functionalities or the most difficult to implement. For details on the other use cases see the project on [Taiga](https://tree.taiga.io/project/bart-c137-s2019-cmpsc390-team3/timeline).

|  |  |
| --- | --- |
| Use Case ID | Use Case Summary |
| 1 | Create Database |
| 2 | Add Account |
| 3 | Edit Account |
| 9 | Encrypt |
| 10 | Decrypt |

The interactions between the use cases are shown in the diagram below.

C:\Users\Zylski\Desktop\Use case.tif

### Actors

All the use cases identified in section Use Cases reference one of the following actors.

|  |  |
| --- | --- |
| Actor | Description |
| User | The user who is using the software. |
| System | The software the user is using. This is also the software described in this document. |

# Use Case Details

## Use Case 1

|  |  |
| --- | --- |
| **Use Case** | Create Database |
| **Primary Actor** | Database user |
| **Goal in Context** | To allow the user to create a new database to store passwords whether it’s the first database or a completely new database |
| **Preconditions** | 1. User has portable application on client 2. User has program launched |
| **Trigger** | User wants to create a database to use the account manager for storing passwords. |
| **Scenario** |  |
| 1. User initiates “Create database option” 2. User is prompted for a database name, master password, and location for the new database. 3. User inputs name and database location 4. User inputs a master password and then inputs it again to verify it. 5. Database is created | |
| **Exceptions** |  |
| 1. User decides to cancel database creation    1. User selects cancel option    2. User is taken back to the main menu 2. User leaves master password field blank    1. Password field is left blank and user selects submit option.    2. User is prompted with error to input a password 3. User does not input name for the database.    1. User leaves name field for the database blank.    2. User is prompted of error to reinput name    3. Database name ultimately becomes the filename. 4. File name for database already exists.    1. User types in name for a database.    2. User is prompted that the name already exists.    3. User is prompted to overwrite, rename, or choose a new file location 5. When user input master password and then inputs it again to verify it, then the passwords do not match    1. User is notified of error that passwords do not match    2. User re-inputs both password fields | |
| **Priority** | Essential |
| **When Available** | First increment |
| **Frequency of Use** | Many times a day |
| **Channel to Actor** | Via program interface |
| **Secondary Actors** | n/a |
| **Channels to Secondary Actors** | n/a |
|  | |
| **Open Issues** | Should the program allow the user not to have a master password for a database?  What kind of encryption to use?  Should a password generator be used? |

## Use Case 2

|  |  |
| --- | --- |
| **Use Case** | Add account to existing database |
| **Primary Actor** | Database user |
| **Goal in Context** | To add an account and login info for that account to an existing database |
| **Preconditions** | 1. Portable application exists on client 2. User has program launched 3. User’s database already exists 4. User knows the master password |
| **Trigger** | User wants to add an account to the database to store the login info for an account |
| **Scenario** |  |
| 1. User has already created or loaded the database they want to access 2. User has already logged into the database using the master password 3. User initiates “add account option” 4. User is prompted to enter an account label, username for the account, password, confirm password, and URL, a description, and category for account to log into. 5. User inputs required account label, username, password, confirm password, and category for the account they want to add. Description and URL fields are optional. 6. Password field and confirm password field match 7. New entry added with account information that user inputted. | |
| **Exceptions** |  |
| 1. User decides to cancel adding a new entry.    1. User selects cancel option    2. User is taken back to the main menu 2. User doesn’t input any information for 1 or more of the required fields.    1. User does not enter in any information for one or all fields.    2. User is prompted an error to input required information. 3. Verify password field does not match password field.    1. User input passwords into the password field and verify password field.    2. User is notified that they do no match and has to input both passwords correctly. 4. User has no account entries in the database    1. User selects edit account with no existing account in database    2. User is prompted with error message. | |
| **Priority** | Essential |
| **When Available** | First Increment |
| **Frequency of Use** | Many times a day |
| **Channel to Actor** | Via program interface |
| **Secondary Actors** | n/a |
| **Channels to Secondary Actors** | n/a |
|  | |
| **Open Issues** | Should there be an option for generating a random password?  Should password be masked as it is typed in?  Should grouping system be implemented for added entries? Such as a general group, banking group, etc.  Should URL and description field be required or optional? |

## Use Case 3

|  |  |
| --- | --- |
| **Use Case** | Edit an account in an existing database |
| **Primary Actor** | Database user |
| **Goal in Context** | To edit an account entry in an existing database |
| **Preconditions** | 1. Portable program exists on client 2. User has program launched 3. User’s database already exists 4. User knows the master password 5. Account entry already exists |
| **Trigger** | User wants to edit an already existing account entry in the database |
| **Scenario** |  |
| 1. User has selected the database they want to access 2. User has logged into the database using the master password 3. User selects account entry and initiates “edit account“ option. 4. User is prompted with a screen of preexisting information of the account to edit, which is the label, username, password, verify password, URL, description, category. 5. User inputs new information or leaves old information as is 6. Password field and confirm password field match 7. User then submits the information and the account entry is successfully edited. | |
| **Exceptions** |  |
| 1. User decides to cancel the editing of an account and the information the user edited is not implemented in the account entry.    1. User wants to cancel editing the account    2. User is taken back to the main menu 2. Verify password field does not match password field.    1. User inputs passwords into the password field and verify password field.    2. User is notified that they do no match and has to reinput both passwords correctly. 3. User doesn’t input any information for 1 or more of the required fields.    1. User does not enter in any information for one or all fields.    2. User is prompted with error message to input required information 4. User has no account entries in the database    1. User selects edit account with no existing account in database    2. User is prompted with error message. | |
| **Priority** | Essential |
| **When Available** | Second Increment |
| **Frequency of Use** | Sometimes in a day |
| **Channel to Actor** | Via program interface |
| **Secondary Actors** | n/a |
| **Channels to Secondary Actors** | n/a |
|  | |
| **Open Issues** | Should there be an option for generating a random password?  Should password be masked as it is typed in?  Should this be multi platform or just windows? |

## Use Case 9

|  |  |
| --- | --- |
| **Use Case** | Encrypt |
| **Primary Actor** | Database user |
| **Goal in Context** | To encrypt the content that is being written to the database file, whether via adding an account, creating, editing, or deleting. |
| **Preconditions** | 1. Portable program exists on client 2. User has program launched 3. User has initiated an option that uses Encrypt use case |
| **Trigger** | User has created, edited, or deleted an account or database and needs the contents to be encrypted as it is saved |
| **Scenario** |  |
| 1. User initiates add account, create database, edit account, delete account or save account or adds or edits security questions or properties 2. User successfully carries out adding, creating, editing, or a deleting operation. 3. Account/database details are ready to be written to database file 4. Details are encrypted by the System before being written to the file 5. Details are encrypted using the Blowfish algorithm, and using the master password as the key 6. Encrypted contents are successfully written to the file by the system | |
| **Exceptions** |  |
| 1. User decides to cancel database creation, or account adding, editing, deleting    1. User selects cancel option    2. User is taken back to the main menu    3. Contents are not encrypted | |
| **Priority** | Essential |
| **When Available** | Second increment |
| **Frequency of Use** | Many times a day |
| **Channel to Actor** | Via program interface |
| **Secondary Actors** | System |
| **Channels to Secondary Actors** | Via encryption method |
|  | |
| **Open Issues** | Is handling of the master password within the Account Manager completely secure?  Do all the prompts successfully prevent a blank or non-existent password from being used. |

## Use Case 10

|  |  |
| --- | --- |
| **Use Case** | Decrypt |
| **Primary Actor** | Database user |
| **Goal in Context** | To decrypt the content that is being read from the database file, which is done while loading a database file |
| **Preconditions** | 1. Portable application exists on client 2. User has program launched 3. User has initiated the load account option that uses the Decrypt use case |
| **Trigger** | User wants to load a database |
| **Scenario** |  |
| 1. User initiates the load database option 2. User chooses the database file they want to load 3. User successfully inputs the correct master password 4. Contents are decrypted by the system using the Blowfish algorithm where the master password acts as the key 5. Contents are read from the file and loaded into the account manager by the system | |
| **Exceptions** |  |
| 1. User decides to cancel loading a database.    1. User selects cancel option    2. User is taken back to the main menu    3. Contents are not decrypted 2. User input the wrong master password    1. User is given an error message and taken back to the main menu.    2. Contents are not decrypted | |
| **Priority** | Essential |
| **When Available** | Second increment |
| **Frequency of Use** | Many times a day |
| **Channel to Actor** | Via program interface |
| **Secondary Actors** | System |
| **Channels to Secondary Actors** | Via decryption method |
|  | |
| **Open Issues** | Are exceptions for decryption properly thrown?  Does the error message always display correctly? |

# External Interface Requirements

There are no external interface requirements

# Technical Solutions Description

The Account Manager is developed as a [JavaFX](https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm) application designed to run on an individual user’s machine. The Account Manager makes use of the [GSON](https://github.com/google/gson) library to make formatting, reading, and writing files the database easier to simplify and shorten the development cycle. The Account Manager automatically saves any edits to the database that the user is working on. The Account manager also allows to user to save the file anywhere and remembers the last working directory to make loading future accounts easier.

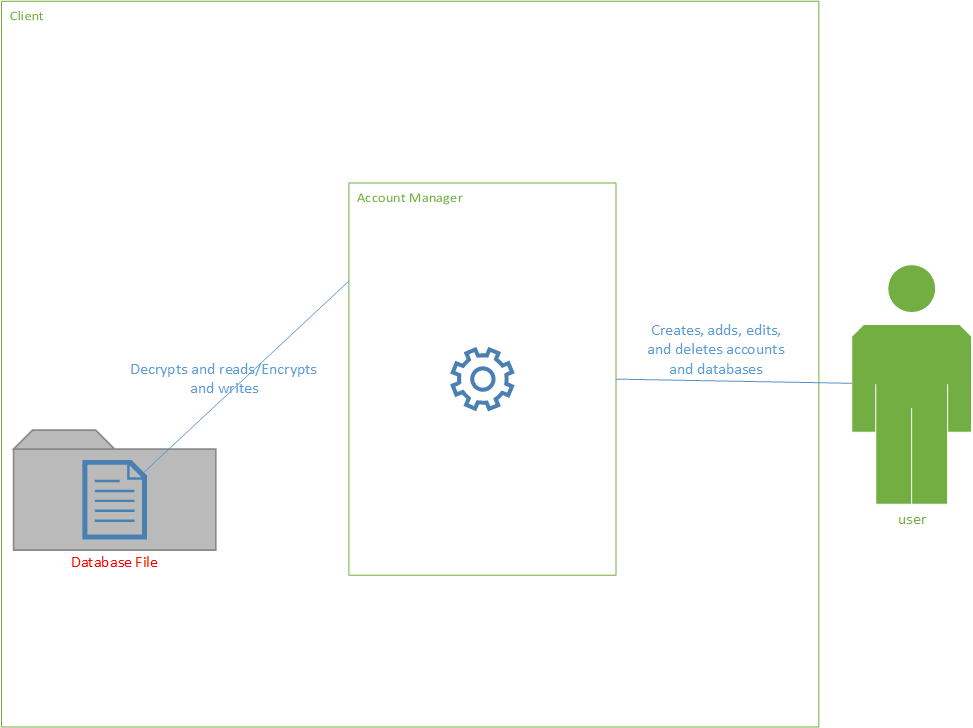
It is important to note; the GSON component is open source and is provided as is. This means support for the library if bugs are found will not be possible.

# Interfaces

The Account Manager does not use any external interfaces.

# Architecture

The architecture is a simple client based account manager program. The user interacts with the account manager and the account manager stores and reads account information to and from a file in any local directory securely using encryption



## User

The user is the person who is using interacting with the client to use the Account Manager.

## Client

The client is the user’s local machine. This is where the Account Manager will be launched and where the local files and directories are located.

## Account Manager

The software described in this document. The Account Manager is responsible for the creation of the account database and getting the master password for the database by getting information input from the user. It will also be responsible for the adding, deleting, and editing of the accounts in the database file itself per the user’s input. The program will be storing the database file in a JSON file and utilize GSON to accomplish these tasks. Also it will do encryption and decryption of the file via the Blowfish encryption algorithm, using a key that is the master password for the database.

The program will also be responsible for finding the file, with the user’s help of getting to the location and giving the name. Then ask the user for the master password and see if it matches the master password needed to get into the database. If it is a match then it will display all the accounts and their information, with some information like security questions and passwords per account being hidden unless a button is pressed to reveal them. This will also allow the user to edit, delete, or add accounts to the currently open database.

## Database File

The file in which the master password for the database to be accessed, as well as the accounts, are stored in. The file will be in the JSON format and will be encrypted via Blowfish and decrypted via Blowfish as well by using a key that is the master password for the database.

The information is stored as follows:

* Master Password – The main password for the database in order to access it. Used as key for encryption and decryption
* Account – The collection of information that is tied to a specific user for a service or login screen like a social media or email website or a computer login screen. A database can contain zero or many accounts.
* Label – Name given to the specific account to identify it.
* Username – The username used to login into the account.
* Password – The password used to login into the account.
* URL – The website address or location to go to in order to login to the account.
* Description – A description for the account.
* Category – Category for the account in which it will be sorted in the Account Manager. If it’s blank the account will go into a “General” category.
* Security Question – This is an optional property, but if used this will be where the user puts in a security question is used for the account. If no Security questions are added that means there are no security questions. There can be multiple security questions per account.
* Question – The question for the security question property.
* Answer – The answer for the security question property.
* Additional Properties – This is an optional property, but if used, it will be where the user puts in a property of their own and to give a “value” for it. If no Additional Properties are added then that means there are no additional properties. There can be multiple additional properties per account.
* Property – The custom property the user would put in.
* Property Value – The “Value” or response to what the initial custom property is for.

The entire database is encrypted as a single string in JSON format via Blowfish. The JSON structure of an example database file is shown below.

{   
   "masterPassword":"password123",  
   "account":[   
      {   
         "label":"Gmail account",  
         "username":"barry.h.jr@gmail.com",  
         "password":"321wordpass",  
         "url":"https://www.google.com/gmail/",  
         "description":"Personal email for Gmail",  
         "category":"Email",  
         "SecurityQuestions":[   
            {   
               "Question":"",  
               "Answer":""  
            }  
         ],  
         "AdditionalProperties":[   
            {   
               "property":"",  
               "propertyAnswer":""  
            }  
         ]  
      },  
      {   
         "label":"Facebook",  
         "username":"barry\_hoinacki\_jr",  
         "password":"123456789password",  
         "url":" https://www.facebook.com/",  
         "description":"Personal facebook account",  
         "category":"Social Media",  
         "SecurityQuestions":[   
            {   
               "Question":"First car model.",  
               "Answer":"Volkswagen"  
            },

            {   
               "Question":"Mother’s maiden name.",  
               "Answer":"Mary"  
            }  
         ],  
         "AdditionalProperties":[   
            {   
               "property":"Work or personal use?",  
               "propertyAnswer":"Personal"  
            }  
         ]  
      }  
  
   ]  
}

# Data Model

There are no actual databases used in the Account Manager. The database is simply a JSON file that is read into the account manager and turned into a database object.

# Appendix A: Issues List

* Edit menu in menu bar doesn’t seem to work correctly. Copy, Paste, and Cut from the menu bar are not working correctly at the moment and effectively disabled.
* The database does not currently keep track of when the account was created. As such, no reminder list for expiring accounts was included in the software
* No help menu was included in the software. This might make figuring out how to use it difficult. For help, please contact the development team for instructions