CMPSC122-hw4-bank Documentation

Eric Roeum

Mar 14, 2018

CONTENTS:

0.1 Introduction

This document discusses how to use this program and how it operates

0.1.1 Setup

This program only requires the python standard library and must be at least version 3.6.4. To develop any more documentation, sphinx must be installed. On a linux machine perform the following to setup:

1. apt-get update
2. apt-get -y upgrade
3. apt-get -y install build-essential python3-pip python3-dev
4. pip install sphinx (Optional for documentation generation
5. git clone https://github.com/eroeum/CMPSC122-hw4-bank.git

0.1.2 Running the Program

To run the program perform the following:

1. Go to CMPSC122-hw4-bank directory
2. Run the program by running **bank/core.py -t** (Note: the -t is to run the program in terminal)

An example of running in a Linux shell is below:

1. cd CMPSC122-hw4-bank
2. python3 bank/core.py -t

0.2 Example

To help go through the program, an example of its functionallity is given below. Note that anything italicized is entered after each punctuation mark and anything after # are notes. Anything in "[]" should be also entered but the value depends after each run. This can be used as a rudamentary systems acceptance test.

- # Make a Manger & set-up bank
- :*man*
- :*pass*
- # Make 3 Customers
- :*new*
- :*new*
- :*cust1*
- :*pass*
- :*new*
- :*new*
- :*cust2*
- :*pass*
- :*new*
- :*new*
- :*cust3*
- :*pass*
- # Login as the manger
- :*man*
- :*pass*
- # Make an assistant and bank teller
- »*make assi1 pass*
- »*make tell1 pass*
- »*exit*
- # Login as a customer
- :*cust1*
- :*pass*
- # Create some transactions
- »*balance*# The balance should be 0
- »*deposit 100*
- »*withdrawal 40*
- »*deposit 10*
- »*balance*# The balance should still be 0 until the transaction is approved

- »*exit*
- # Login as a bankteller and approve a transaction
- :*assi1*
- :*pass*
- »*ls* # A list of all the customers should be shown
- »*requests* [Cust1 userid] # get the userid by copying the 8-character code that comes first in the ls feature. This is the hashed user id for security purposes
- »*accept* [Cust1 user ID] *1* # Accepts the first customer's first transaction
- »*exit*
- # Check that the transaction went through
- :*cust1*
- :*pass*
- »*balance* # The balance should be 100 now
- »*exit*
- # Exit out of the program
- :*exit*
- :*exit*

This list goes over the fundamental functions of this program. Please continue to explore by creating an account and typing "help" for a full list of description and things to do. For more help, type "help" plus the function name.

0.3 Caveats & Fundamentals

The purpose of this program is to simulate a bank. Of course, because of the level of programming that this was programmed with, the security measures are not industry-worthy and this bank does not, in any shape, accept any real monetary value and only deals with hypothetical values. As a result, a user can deposit and withdrawal as much money as he/she wants with no penalty. The users' balance may even drop below 0 to simulate debt, but because of the nature of this bank, there is no penalty for doing such.

The fundamentals of the hierarchy is as follows:

- Manager: Head of the bank, can do anything. There is only one manager since this person/group should share the same account while assistants help the manager. The only real addition that the manager has opposed to an assistant is creating assistants.
- Assistant: Second in command. There can be as many assistants that the manager wants. The assistant can create new bank tellers and do anything a bank teller can do.
- Bank Teller: Lowest administrative user. The bank teller can only accept requests that a customer has submitted when completing a transaction.
- Customer: No real power in the bank. Can only submit requests to complete transactions

Additional caveats include that there was no consideration for loans or subaccounts or shared accounts or things of those nature. Since this is a simply a simulation of a bank with no penalty, constructing a bank with these features requires investigation into penalty and economic theory to do so. As a result, it may be a future work. Further when running the program, a "-t" is needed because if a GUI implementation would be created, that would be the main way of running the program. Lastly there was no consideration for malicious code insertion, since crypotgraphy is beyond our scope.

The password authentication system relies on SHA512 hashing which may not be the best, but is still better than nothing since it allows a one-way authentication system that is blind for both the user and the bank administrators.

CHAPTER

ONE

CMPSC122-HW4-BANK

1.1 bank package

1.1.1 Submodules

1.1.2 bank.assistant module

class bank.assistant.**Assistant**(*userID*, *customers*, *bankTellers*)
Bases: bankTeller.BankTeller

Represents a user with mid-level priveledges Users having these priveledges are usually bank tellers

createBankTeller(*userID*, *customers*)
Creates a Bank Teller under this account

Parameters

- **userID** (*string*) – the userID of the assistant
- **customers** (*list*) – all customers that assitant can access
- **bankTellers** (*list*) – all bank tellers that assistant can access

viewBankTellers()
Returns the bank tellers under this account

1.1.3 bank.bankTeller module

class bank.bankTeller.**BankTeller**(*userID*, *customers*)
Bases: customer.Customer

Represents a user with mid-level priveledges Users having these priveledges are usually bank tellers

acceptRequest(*customer*, *reqNum*)
Accepts a request for a customer

Parameters

- **customer** (Customer) – Customer with the request
- **reqNum** (*int*) – Request number that wishes to be resolved

addHistory()
Bank tellers do no actually have any history All history is written in customers

getHistory()
Bank tellers do no actually have history All history is written in customers

getRequests()
Bank tellers do no actually have requests

getSubaccounts()
Bank tellers do no actually have a sub accounts

getValue()
Bank tellers do no actually have a value

removeRequest()
Bank tellers do no actually have any requests

requestDelta()
Bank tellers do no actually have a value

viewCustomers()
Returns the customers under this account

viewRequests(*customer*)
Retrieves the requests held within the customer

Parameters **customer** (Customer) – Customer with requests

1.1.4 bank.core module

bank.core.**main**(*arg*)
Runs the core of the program Integrates all other python scripts/modules Use the core.py to run the Bank python script

Parameters **arg** (*list*) – list of arguments in terminal

1.1.5 bank.customer module

class bank.customer.**Customer**(*value*, *userID_owner*, *userID_users=[]*)
Bases: object

Represents a user with low-level priveledges Users only having these priveledges are usually customers

addHistory(*event*)
Adds an event to the history

Parameters **event** (*string*) – Event that has take place (only accepted requests)

getHistory()
Retrieves history of this account

getOwner()
Retrieves owner of the bank

getRequests()
Retrieves request of account

getSubaccounts()
Retrives accounts owned by this account

getUsers()
Retrieves priveledged users of the bank

getValue()
Retrieves value held within bank

removeRequest(*reqNum*)
Removes request from request list

Parameters **reqNum** (*int*) – Index of request

requestDelta(*delta*)
Rquests the withdrawl or deposit into customer's account

Parameters **delta** (*float*) – Amount to request funds

1.1.6 bank.manager module

class bank.manager.**Manager**(*userID*, *customers*, *bankTellers*, *assistants*)
Bases: assistant.Assistant

Represents a user with high-level priveledges Users having these priveledges are usually managers

createAssistant(*userID*, *customers*, *bankTellers*)
Creates an assistant object

Parameters

- **userID** (*string*) – the userID of the assistant
- **customers** (*list*) – all customers that assistant can access
- **bankTellers** (*list*) – all bank tellers that assistant can access

viewAssistants()
Returns the assistants under this class

1.1.7 bank.passwordAuthentication module

class bank.passwordAuthentication.**Password**(*approvedUsers*)
Bases: object

addAutheticatedUser(*userid*, *password*)
Adds autheticated user using UUID and SHA512 password hashing

Parameters

- **userid** (*string*) – User ID that user has chosen
- **password** (*string*) – Password to attempt to authenticate userid

authenticate_password(*userid*, *password*)
Autheticates Passwords using UUID and SHA512 password hashing

Parameters

- **userid** (*string*) – User ID that user has chosen
- **password** (*string*) – Password to attempt to authenticate userid

hashUsername(*userID*)
Hashes provided username for hased user id

Parameters **userid** – User ID that user has chosen

readEncrypted(*filename*, *dest='./'*)
Imports all usernames and passwords from a txt file File must be formatted using _writeEncrypted function
NOTE: All previous passwords will be deleted and replaced

Parameters

- **filename** (*string*) – filename of txt file
- **dest** (*string*) – location to write txt file

writeEncrypted(*filename*, *dest='./'*)
Writes all usernames and passwords to a txt file

Parameters

- **filename** (*string*) – filename of txt file
- **dest** (*string*) – location to write txt file

1.1.8 bank.terminalFunctions module

bank.terminalFunctions.**acceptRequest**(*user*, *customer*, *reqNum*)
Accepts a customer's request

Parameters

- **user** (*class*) – The current user
- **customer** (*customer*) – Customer that is being accepted
- **reqNum** (*int*) – Request number correponding to the request

bank.terminalFunctions.**balance**(*person*)
Shows balance in the account

bank.terminalFunctions.**clear**(*platf*)

bank.terminalFunctions.**deposit**(*person*, *value_to_add*)
Deposits value into the account

Parameters **value_to_add** (*float*) – Value to be added in funds

bank.terminalFunctions.**exit**()
Exits the program This is a "fake" function that only confirms exit Clears screen after exit

bank.terminalFunctions.**help**(*accountType*, *helpFunc=''*)
Help menu for giving users all commands with short description

Parameters

- **accountType** (*string*) – Type of account
- **helpFunc** (*string*) – Option functional to get more info

bank.terminalFunctions.**history**(*customer*)
Prints the history of a customer

Parameters **customer** (*Class*) – customer with desired history

bank.terminalFunctions.**ls**(*accountType*, *account*, *customers*)
Lists all accounts accessible

Parameters

- **accountType** (*string*) – type of account
- **account** (Customer) – The account that is being inspected

bank.terminalFunctions.**make**(*genesis*, *accountType*, *userID*, *users*)
Create an account

Parameters

- **genesis** (Customer) – Original account
- **accountType** (*string*) – type of account
- **userID** (*string*) – user id of the account

bank.terminalFunctions.**requests**(*user*, *customer*)
Lists all requests of the customer

Parameters

- **user** (*class*) – The current user
- **customer** (*customer*) – Customer that will be viewed

bank.terminalFunctions.**whoami**(*person*)
Return the owner's ID

bank.terminalFunctions.**withdrawal**(*person*, *value_to_deduct*)
Withdrawals value from the account

Parameters **value_to_deduct** (*float*) – Value to be withdrawled in funds

1.1.9 bank.terminalInterface module

bank.terminalInterface.**displayInterface**()
Terminal-Based banking application

Parameters **customers** (*dict*) – dictionary of all created customers

Params users Class with all developed user-password authentication

1.1.10 Module contents

1.2 tests package

1.2.1 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

t

INDEX