Ben Reichert

Term Project: *ChocAn*

Design Document

Table of Contents

Table of Contents

Introduction

This document is a supplement of the Requirements document previously published. This document serves to outline the design requirements of the ChocAn system. The design document contains specific design decisions made by Ben Reichert Software, LLC in order to help the implementation of the ChocAn system. Detailed design decisions such as how the systems will function together is described later in the document. The general system architecture and interactions between users and the system components will also be described.

1.1 Purpose and Scope

The purpose of this document is to provide clarity to the formal requirements and uses of the ChocAn software for reference by the customer, and developers of the system software. The system has many parts, this document's scope excludes the Provider Terminal hardware, Email delivery system, and EFT payment processing system.

1.2 Target Audience

The target audience of this document is the ChocAn representatives responsible for software input, as well as employees of Ben Reichert Software, LLC, not limited to developers and system architects.

1.3 Terms and Definitions

1.3.1 EFT

Electronic Funds Transfers: used to keep track of payments to be made between providers and ChocAn.

1.3.2 DC

ChocAn Data Center, responsible for keeping records and service transactions.

1.3.3 Terminal

The specially designed ChocAn terminal, which will include a PC along with a magnetic card reader for reading member cards. This hardware design is not the responsibility of Ben Reichert Software, LLC, and thus will not be discussed. The software that runs on this Terminal is within the scope of this project and document.

1.3.4 Provider

A provider is an institution registered with Chocoholics Anonymous that provides treatments and consultations with health care professionals, dietitians, internist, and exercise specialists.

1.3.5 MC

MC: Member Card: A plastic card embossed with the ChocAn member's name and member ID. This card has a magnetic strip with the previous information encoded on it for reading through the Terminal.

1.3.6 System

The ChocAn system as a whole, in all of it's functionality.

1.3.7 MID

Member IDentification number, a 9 digit sequential number.

2 Design Considerations

The purpose of the design considerations section is to outline the restrictions and considerations of Ben Reichert Software, LLC when designing the ChocAn software systems. The following are descriptions of constraints of the software, and dependencies of the software.

2.1 Constraints and Dependencies

Functional requirements that were required in the design and implementation state were a way to keep track of a provider directory, display fees, keep track of transactions, add/delete users, creation of member reports, filenames, EFT log file, computer hardware (specifically ubuntu 14.04LTS), and a place to store records (sqlite). All of these required that a specific menu be build for the application, with correct control flow. Specifically with the database, filenames, and data fields, these had to be followed exactly from the requirements document in order to make sure the application was reading and storing data efficiently.

2.2 Methodology

The main methodology that the team will be using to design the software is the caffiene-into-code development method. This is a fast, efficient way of creating a software application on limited resources, such as a student's time and income. Formal definition below:

pro·gram·mer (n) An organism capable of converting caffeine into code.

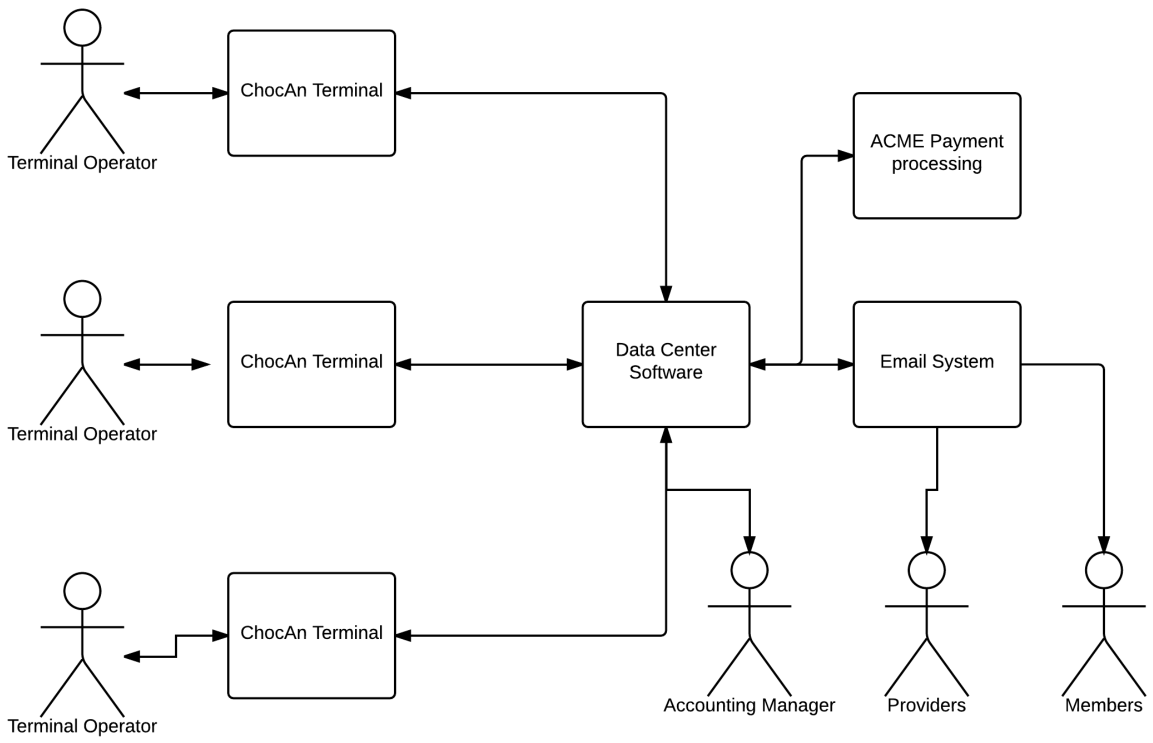
Ben Reichert Software, LLC chose the caffeine development method as our workforce consumes irrational amounts of stimulants regardless. We aided this development methodology through the purchase of large amounts of soft and energy drinks, along with coffee.

As for software development processes, Ben Reichert Software, LLC has chosen to implement the software using Python and a SQLite database backend. This process was chosen because the developers are familiar with Python, and it's modular design. Since the DC application is not a RTOS, it is not critical that it functions as fast as possible, so a slower interpreted language such as Python can be used in place of a more complicated C derivative. This benefits the company since the developers can design higher quality systems without learning new languages, and efficiently with respect to time and costs. From a customer side, the product will be available sooner as development is more rapid, and rapid prototyping is available for testing.

In his case the development methodology process we follow has been waterfall, as this is what our company was tasked with. We are dealing with this method by elaborately designing the software in waterfall increments following the general design process. Once the requirements document was done, we started on this design document, and then will begin implementation, testing, and then publishing. This is a rigid process, and while the general waterfall process does not allow design changes, say in the implementation process, our contract allows us to go back to previous steps and modify requirements or design accordingly if needed.

3 System Overview

The system contains terminals that are operated by terminal operators in each ChocAn provider's office space. These terminals talk over an internet connection to the data center software and perform operations. If the terminals are offline, they store a local cache of data and upon connection to the DC software will upload their logs and transactions. The DC software then talks to the email system (to be implemented by another company) to deal with emailing reports generate by the DC software. EFT logs are send to the ACME payment processing system over a network connection (also being implemented by another company). The Accounting Manager has their own terminal view of the DC application for financial information. The email system is tasked with sending reports to providers and members for various reasons described in the Requirements document.



4 System Architecture

The ChocAn data center software will be implemented without the networking portion. The architecture will be divided into components based on terminal input needs, such as having different components for displaying fees, keeping track of transactions, adding/deleting users, creating member reports, creating EFT data. The architecture will have components for talking to outside services, such as the email, and ACME payment systems.

4.1 User Interactions

The user actions component of the software deals with user input and output from the various pieces of hardware and software. The software will have two menus when started, user interactions, and one for running automated processes. From these menus, sub-component will also have menus and prompts accordingly.

4.1.1 Managing Member Accounts

Managing member accounts will be two separate menus, one for deletion of members, and one for addition of members.

4.1.1.1 Adding Member

The Provider Terminal Operator will follow the terminal prompts for input, and once a member is added, will receive a confirmation after the member is added to the database. Can only be done during member modification hours.

4.1.1.2 Deleting Member

The Provider Terminal Operator will follow terminal prompts for input regarding the MID, and a confirmation to delete page, then the member will be deleted from the database. Can only be done during member modification hours.

4.1.1.3 Look-up Member

When the Provider Terminal Operator scans a MID, they enter this menu which will pull up member information and submenus based on the MID. From here, they can edit a member's information. Modification can only be done during member modification hours.

4.1.1.4 Displaying Fees

The Provider Terminal Operator will follow the terminal prompts for input once the display fees menu is selected after a user is looked up in the database system. This is a submenu to the look-up member system.

4.2 Data Center Automated Processes

These processes are ones that are either time schedule based (ie, it runs on Friday night at 9pm) or automated processes that run on cron times every X minutes or hours or days. These processes need little to no user interaction, other than the occasional "button click" to start the process. Examples may include a manager starting the email system, or asking for a generated report. In this case, they make an initial request, and the automated software does the rest and eventually reports back to them when the process is complete.

4.2.1 Member reports

Member reports will be generate daily through an automated process requiring no user interaction. The reports will be saved to disk as specified in the requirements document. The reports will be sent via email to members through another system outside the scope of this project.

4.2.2 Transaction Logs

The system will keep an automated log saved to disk of every interaction with the system. This will be an automated process requiring no user input. If an individual wishes to see the log, they must contact the DC support team to obtain the secure log as necessary.

4.2.3 EFT Data

Every transaction requiring transfer of funds from a member to ChocAn or ChocAn to a provider will automatically be appended to a EFT Data log every day, and send to ACME to be processes nightly. This process requires no user input.

5 Detailed System Design

5.1 User Interactions

The User interactions menu will be implemented through a Python class structure. The class will have methods for storing user inputed data, and displaying output data to a pseudo-terminal. This will be the main class that has submenus, that will have their own class objects that are part of the UINT(user interaction) object. This way there is one hierarchical structure to deal with that has many submodules. The submodules will be dynamically loaded at runtime for submenu components. This will have a UNIX shell like structure for menus and prompts. Each submenu will have commands to run that will prompt the user for input, and then generate appropriate output.

5.1.1 Managing Member Accounts

Managing member accounts will be two separate menus, one for deletion of members, and one for addition of members. Member data will be stored as temporary strings in a dictionary object, and then be written to the DB. This is the case for both reading and writing data. If the user account needs to be updated the data will be loaded from the DB to a dict (a method will handle this) and then modified as a dictionary, and then loaded back into the DB. If a member needs to be added, a dictionary will be created and then used the dict_save() method to write the dict to the DB.

5.1.1.1 Adding Member

The Provider Terminal Operator will drop into the adding member submenu and be prompted for each requirement as specified in the requirements document for adding a user. This submenu will be a set of prompt methods, that ultimately call a DB set function to add the user to the database system. Information will be stored as a dictionary.

5.1.1.2 Deleting Member

The Provider Terminal Operator will drop into the deleting member submenu and be prompted for each requirement as specified in the requirements document for deleting a user. This submenu will be a set of prompt methods, that ultimately call a DB set function to delete the user to the database system. Verification functions will be called to verify the Provider Members ID, as well as the MID and Members name and personal information for verification purposes. One method will be responsible for input and output, while another does checking against the DB to ensure authentication is valid. Once valid, a set function will remove the member from the database, and only can be added again through the adding members submenu.

5.1.1.3 Look-up Member

When the Provider Terminal Operator scans a MID, they enter this menu which will pull up member information and submenus based on the MID. From here, they can edit a member's information. The Members information will be loaded into a temporary Python dictionary which can be edited, and then passed to a function that will create a SQLite query to update the database accordingly from the given dictionary.

5.1.1.4 Displaying Fees

The Provider Terminal Operator will follow the terminal prompts for input once the display fees menu is selected after a user is looked up in the database system. This is a submenu to the look-up member system. The fee system will be a class object, that will have handler functions for user IO. Since it only needs read abilities, there is no need to write to the database, but a reading method will need to query the database to display fees.

5.2 Data Center Automated Processes

These processes are ones that are either time schedule based (ie, it runs on Friday night at 9pm) or automated processes that run on cron times every X minutes or hours or days. These processes need little to no user interaction, other than the occasional "button click" to start the process. Examples may include a manager starting the email system, or asking for a generated report. In this case, they make an initial request, and the automated software does the rest and eventually reports back to them when the process is complete. These systems will not be based on the terminal system, and will be implemented in a separate set of classes and objects that are created at runtime of the application. Some user input will be necessary from the terminal system to “kick start” these systems, but they will largely required no user input as they are automated.

5.2.1 Member reports

Member reports will be generate daily through an automated process requiring no user interaction. The reports will be saved to disk as specified in the requirements document. A part of the reports class will be a file IO handler to deal with reading and writing files to and from disk. There will need to be a DB handler as well to get information to generate the reports from. Once the reports are created, the application will wait until the specified time to send them to the email system, at which time the reports will be sent via email to members. This will all be contained in a python class object, with helper functions that are based off of the Time module for automated run times. User interaction is not required.

5.2.2 Transaction Logs

The system will keep an automated log saved to disk of every interaction with the system. This will be an automated process requiring no user input. If an individual wishes to see the log, they must contact the DC support team to obtain the secure log as necessary. Every action a terminal based user does, will call a specific log function in this object that records their submissions through the application. This will be a relatively simple class with one or two methods that are just responsible for taking a user input value (as a string) and giving that to a file handler to append the string to an existing log file.

5.2.3 EFT Data

Every transaction requiring transfer of funds from a member to ChocAn or ChocAn to a provider will automatically be appended to a EFT Data log every day, and send to ACME to be processes nightly. This process requires no user input. This class will require a function handler that deals with file IO, as well as DB interactions to keep track of what is stored in the database for EFT data. Each line of the EFT file will be appended by a write function. Another function will handle the format string to prepare to write the appropriate EFT data to the file.