Design Document

Contents

1 Introduction

This design document mainly introduces the system and software design of this project. After analyzing all the requirements of this project, our team set out to establish the overall architecture of the entire system based on the requirements. After discussions between the teams, we chose this structure and added enough details in this design document to have a reference basis in the future development process.This design document can effectively improve the development efficiency of the team and ensure the feasibility and integrity of the final product.

1.1 Purpose and Scope

The purpose of this design document is to explain the design of the entire system in detail so that developers can understand the business logic of each part of the system, and to check the correctness of the development process at any time during the development process. The contents of this design document include the design of the entire system, details of each part of the system, and the relationship between each part of the system. This document serves as the basis for the data management system required by the ChocoAn organization. You can read this document to understand the basic functions of the entire data management system and how our team implements them. This data management system meets the management needs of the ChocAn organization by collecting the data of each member, and does some actions like adding, reducing, deleting, searching the data.

1.2 Target Audience

This design document is applicable to all developers of this project and maintenance personnel who need to maintain the system in the future. After reading this document, anyone can have a clear understanding of the entire project and be able to make some changes and maintenance to the project.

1.3 Terms and Definitions

- ChocAn — Chocoholics Anonymous.
- OOP — Object Oriented Programming.
- Functional Requirements — A Functional Requirement (FR) is a description of the service that the software must offer.
- Software Design Patterns — Software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design.
- UI — User Interface.
- API — Application Programming Interface.
- Client/Server model — For the usage of this document “Client” will refer to requests from outside of this software and “Server” will refer to internal operation. Client software includes but is not limited to UI, EFT, accounting, and communications.

2 Design Considerations

This section details the choices about the design of the software. Constraints and dependencies investigates limitations and requirements of the software. Methodology explains the general design approach and programming paradigms used to develop the software.

2.1 Constraints and Dependencies

What constrains, either functional or non-functional were you required to adhere to in designing and implementing your system? Concurrency Control - This system will be accessed and modified by different groups simultaneously. In order to maintain data coherency, concurrency protection is practiced whenever writing to disk. Data formatting - Data is recorded and reported in a consistent manner across all modules of the software.

2.2 Methodology

To implement the ChocAn data system the Object Oriented Programming (OOP) paradigm is employed. This is a suitable approach for this project as each subsystem will require components that can be represented as loosely coupled objects. OOP concepts such as composition and inheritance will be used to create a safe and functional type system that guides the structure of the software. Discrete top level systems will be broken into modules usings C++ namespaces to provide safety and program segmentation. This will facilitate evolution and maintenance of the product.

3 System Overview

The data processing system will be divided into three primary modules. Each of these modules are for the different modes that the software provides. These modes are provider, manager, and interactive.

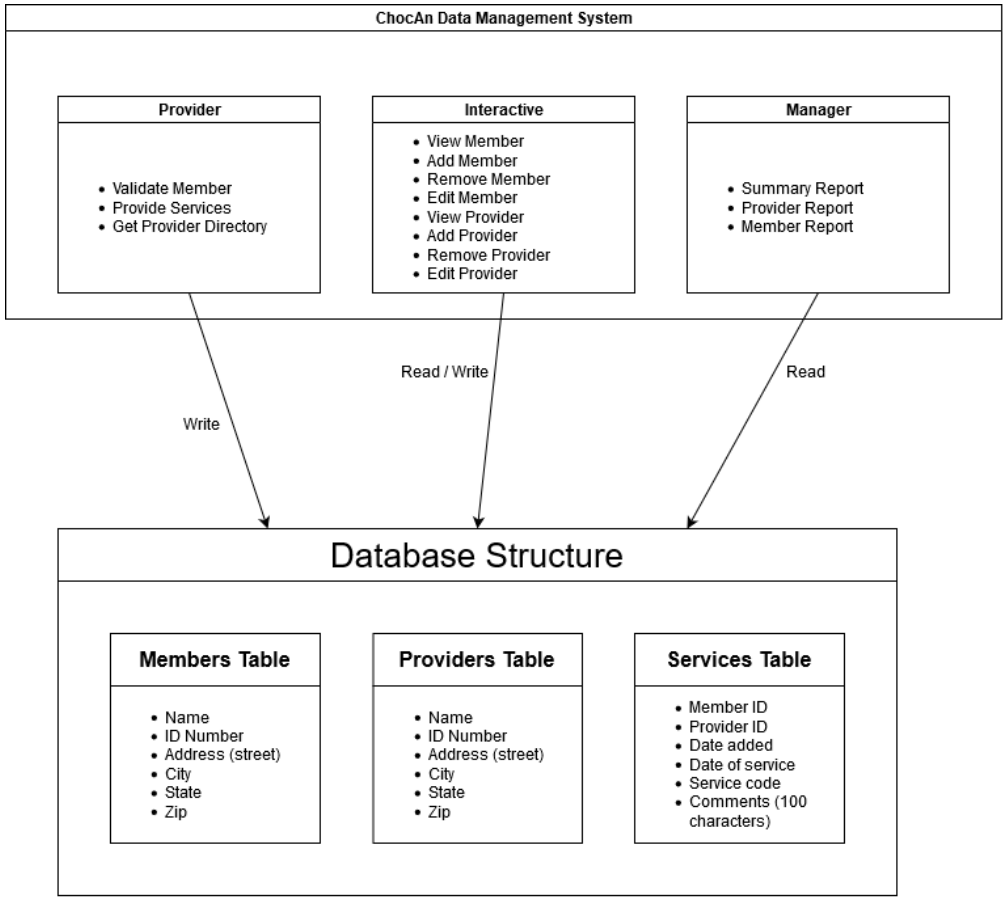

Figure 1: System Overview

Each of these modules will be represented as C++ class objects. The terminal used to simulate the usage

of these API's will ask which mode the user would like to simulate and then provide text based menus to access each of the functionalities.
Data management will be handled with three database tables that will be queried to produce the reports and calculate billing. These tables will be stored as external files on disk in CSV format.
The following objects will be used:

- Person — A base class for Member and Provider
- Member — A member, the members information
- Provider — A provider, the providers information
- Service — An individual service, the service information
- ProviderDirectory — Table to generate services directory to look up service codes
- InteractiveModule — Contains the API to manage database contents (members and providers)
- ManagerModule — Contains the API to run manager reports
- ProviderModule — Contains the API to provide services and request service directory

4 System Architecture

The system architecture will be split into three modules, the provider, interactive, and manager module. The subsystem includes several objects. Some of these objects will be included in multiple of the software modules. The only objects with client side APIs are the ProviderModule, InteractiveModule, and ManagerModule. The rest of the objects API will be for internal usage only. The following chart shows the relationship between these objects. Dotted lines demonstrate multiple systems use the subsystem.

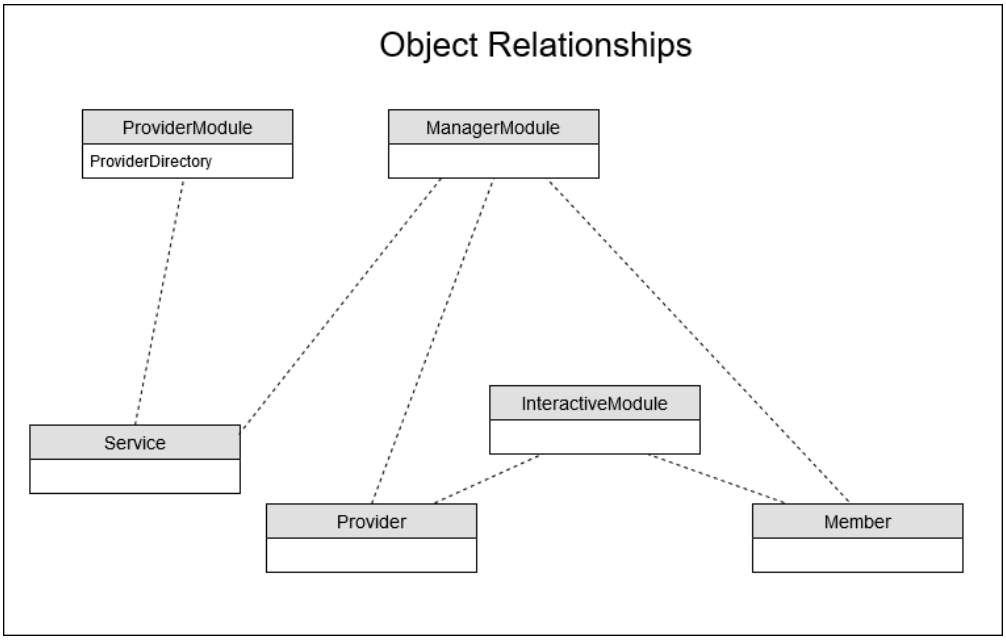

Figure 2: System Architecture

4.1 Person

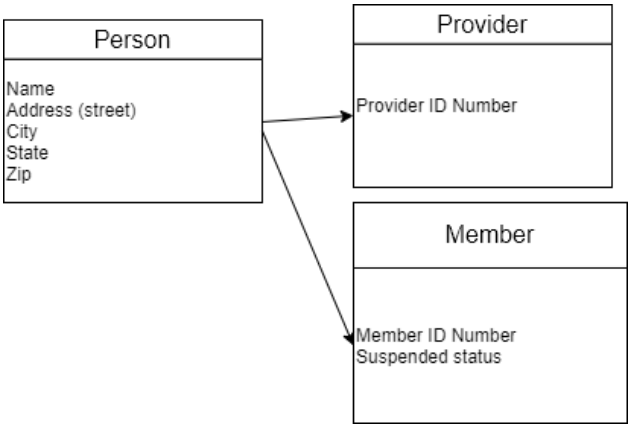

Figure 3: Person Object Architecture

4.1.1 Member

Derived from Person class but has an additional field for status.

4.1.2 Provider

Derived from Person class. Has no additional functionality but is a seperate class to anticipate changes to the Provider in the future.

4.2 Service

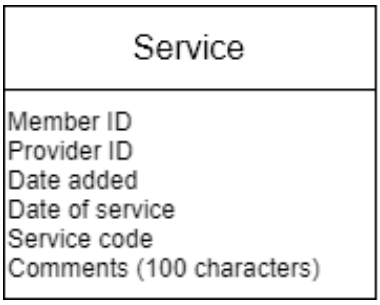

Figure 4: Service

Service - An individual service, the service information

4.3 InteractiveModule

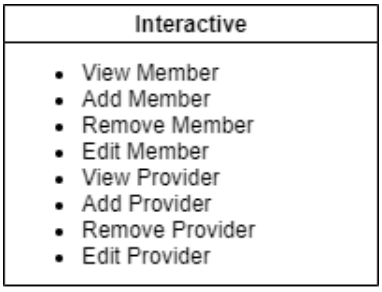

Figure 5: Interactive Module

InteractiveModule - Contains the API to manage database contents (members and providers)

4.4 ManagerModule

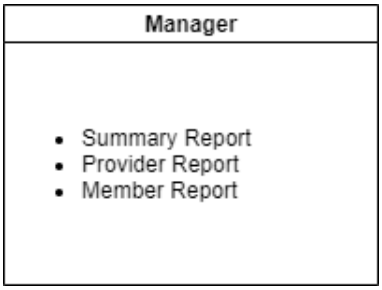

Figure 6: Manager Module

ManagerModule- Contains the API to run manager reports

4.5 ProviderModule

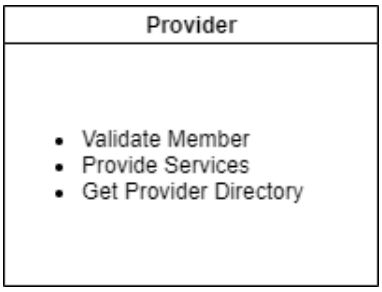

Figure 7: Provider Module

ProviderModule - Contains the API to provide services and request service directory

4.5.1 ProviderDirectory

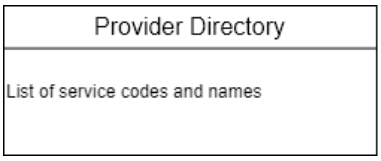

Figure 8: Provider Directory

ProviderDirectory - Table to generate services directory to look up service codes.

5 Detailed System Design

5.1 Person

Data structures:

- string name, city, state
- uint ID, zip

Functions:

- string getName()
- string getCity()
- string getState()
- uint getZip()
- uint getID()

The constructor will require all of the fields up front to verify that the data is valid. The object provides methods to set and get data fields. These setters and getters are used to update and view member data as well as generate reports. Most of the fields will be stored in C++ strings. ID and zip will be stored as unsigned integers. ID's are 9 digits long so a standard 32 bit unsigned int will be sufficient.

5.1.1 Member: public Person

structures:

- bool suspended

Functions:

- Member(string name, uint memberID, string city, string state, uint zip)

The Member object is primarily a container for data associated with a ChocAn member. Member status is stored as a bool (true means suspended).

5.1.2 Provider: public Person

The Provider object is a container for data associated with a specific provider. Currently it doesn't have any data or functionality beyond what is available in Person but it is a seperate class to anticipate possible change to the Provider abstraction in the future.

5.2 Service

Data Structures:

- uint memberID, providerID, serviceCode
- time_t timeOfService, timeRecorded
- string comments

Functions:

- uint getMemberID()
- uint getProvderID()
- uint getServiceCode()
- time_t getTimeOfService()
- time_t getTimeRecorded
- string getComments()

The service object is an abstraction for a service provided by a ChocAn provider to a member. The fields of the service will include the member ID, provider ID, and the service code as unsigned ints. The time of service and time of entry will be saved using the time_t type from the C standard ¡ctime¿ header. The functions will provide a way to access this private data. The constructor will require all of the data needed to create the service object.

5.3 InteractiveModule

Data Structures:

- unordered_map¡uint, Member¿ //hashmap keyed on ID
- unordered_map¡uint, Provider¿ //hashmap keyed on ID

Functions:

- int init()
- int displayMember(uint memberID)
- int addMember()
- int removeMember(unit memberID)
- int editMember(uint memberID)
- int displayProvider(uint memberID)
- int addProvider()
- int removeProvider()
- int editProvider(uint memberID)
- int writeOut()

The InteractiveModule provides an interface for ChocAn data center employees to manage records of members and providers. To start interactive mode the init() function is called from the object. This will display a menu that provides options to add, remove, or edit providers and members or save and exit. When the changes are made the save and exit option is selected and the database is updated with the changes. The members and providers will be stored in an unordered_map (hashmap) C++ data structure. This gives constant time access and insertion to members and providers keyed on ID.

5.4 ManagerModule

Data Structures:

- map¡time_t, Service¿ services
- unordered_map¡uint, Provider¿ providers
- unordered_map¡uint, Member¿ members

Functions:

- int generateWeeklyReports()
- If this fails, errors will be in the ¡date¿.log file
- int generateSummaryReport()
- string generatePersonReportBase(Person& person)
- int generateProviderReport(uint providerID)
- int generateMemberReport(uint memberID)

The ManagerModule is where all the reports will be run from. It will provide functions to run reports on specific providers or members as well as running a weekly summary report. The generateWeeklyReports() method will run each Friday and run a report on every member and provider as well as the summary for that week. All the services will be read in from the database into a map C++ standard library data structure keyed on time added. Because map is implemented as a balanced tree, this allows efficient access to any given time frame and preserves order that the services were added to the system. Providers and members are loaded in from the database into the unordered_map so any Person can be retrieved in constant time. The algorithms to generate the reports are as follows: Provider Report - Find the provider information from the providers map and pass this person object to the base report method to generate a header for the report. Then filter the services tree to only include services from the last week and that are associated with the provider in question. Member Report - Same a Provider. Summary Report - Filter the services map to only include services from the last week. Create a local struct called Total with two fields, an int for service count and a float for total billing. Create a local unordered_map¡uint, Total¿ where the key in the provider ID and the value is the running sum of the providers data. This map will be used to keep track of each unique provider encountered. Iterate over the filtered service map and for each service: get the provider ID try to find the provider in the unordered_map if found: dereference the iterator returned and increment the count and add the price for the current service to the totalBilling field in the struct else: create a new Total struct and set its count variable to one and its totalBilling field to the billing value. Add the key value pair of the providerID and the new Total struct to the unordered_map Then iterate over the (key, value) pairs in the unordered_map. For each: Look up the provider's name from the key and write it to the file Write the count and totalBilling variables to the file from the associated key Weekly Report - Run the Provider Report and Member Report for every provider and member in the unordered_maps. Then run the Summary Report.

5.5 ProviderModule

Data Structures:

- ProviderDirectory directory
- uint providerID

Functions:

- int init(uint providerID)
- int validateMember(uint memberID)

- int provideService(Service& service)
- int getProviderDirectory()

The ProviderModule provides an API for the providers to verify members and provide services. The init() function will run a simulated menu so these functionalities can be tested. ValidateMember will access the members file and search for a member with a matching ID and check if the Member is suspended or not. provideServices() will take a constructed Service object and then display back the details and ask for confirmation that all the information is correct. If so, it will send the service to the data file. getProviderDirectory() will write all the services with codes to a file called directory.txt.

5.5.1 ProviderDirectory

Data Structures:

- map¡service, uint¿ services

Functions:

- int generateDirectory()

The ProviderDirectory object will be used to lookup service names and to display the preview when a service is being provided. The map data structure is chosen because the c++ standard library implements maps using a balanced red/black tree. This allows us to sort the services by name as stated in the requirements document and allows code lookup in log(n) time complexity. If the program requires in the future to add to this structure it will also provide insertion at the same time complexity. GenerateDirectory will write the directory to a file called directory.txt.