Term Project: *ChocAn*

Design Document

Rudd Johnson
Justin Moore
Daniel Eynis
Rohan Elukurthy
Timothy Handojo

Table of Contents

1. Introduction

This document delineates the design for the data processing software commissioned by Chocoholics Anonymous (ChocAn) for the system developers of this software and all other stakeholders. ChocAn, a for profit health care entity that provides chocolate addiction treatment to its clients with the aim of freeing them from chocolate dependence and maintaining long term abstinence. The following document outlines the design considerations as they pertain to software development methodologies and constraints and dependencies thereof. Additionally, the document will profile, in detail, the system architecture. The system architecture includes a general description of all systems and subsystems in addition to a detailed description of system design form the data bases and data structures to the UI/UX and and all the functionality linking the front and back end.

1.1. Purpose and Scope

The purpose of this document is to outline, for the software stakeholders, the ChocAn data processing software design and implementation. This includes the software development methodology utilized, the programming language, data structures, and front end UI/UX the users of the software will use to interface with the aforementioned data structures. This document will identify the constraints and dependencies of the software and delineate overall system architecture, broken down into subsystems and components, and provide a detailed system design of each subsystem and component.

1.2. Target Audience

The target audience of the documents are the system developers for the data processing software to be rendered for ChocAn (Rudd Johnson, Justin Moore, Daniel Eynis, Rohan Elukurthy, and Timothy Handojo), ChocAn themselves, and IT professionals utilizing this software for ChocAn. This document will serve as an official reference for the planned design and implementation of the commissioned software as well as a guide for the way in which system requirements outlined by ChocAn stakeholders is translated into the software architecture.

1.3. Terms and Definitions

ChocAn: Chocoholics Anonymous

UI: User Interface. The competent of the system the user interacts with

UX: User Experience. The experience the user has while interfacing with the system from usability and functionality of the system to their overall satisfaction using the software.

UML: Unified Modeling Language. Modeling language used as a tool for developers to document the structure of software systems and subsystems.

GUI:Graphical User Interface. Interface allowing users to interact with software through graphical icons as oppose to command line

IT: Information Technology. Use of computers and the software running on them as infrastructure to store, retrieve, transmit, or display information.

EFT: Electronic Funds Transfer. Electronic transfer of money from one individual, business, government or financial institution to another.

2. Design Considerations

In this section, the design and implementation constraints and dependencies as well as the software development methodologies will be described. The constraints and limitations of the software pertain to functional and non-functional system requirements that have placed restrictions on the type of design that can be implemented. System dependencies refer to the subsystems in the ChocAn software that are requisite for meeting functional and non functional system requirements. Finally the methodology section will discuss the software development methodology that the development team is using to implement the ChocAn data processing software.

2.1. Constraints and Dependencies

The functional requirements for the ChocAn data processing software are a way to store provider information, patient information, services provided, and weekly billing information. Additionally there must be input/output between the user and the aforementioned information such that it can be displayed either through a terminal or sent via e-mail at request by a provider, member, manager or other operator. Furthermore, this information must be modifiable so that at any point if a member is no longer active, a provider leaves or changes their services, or a patients billing record need to be appended, the appropriate changes can be made. Non-functional requirements include a system that is reliable, secure, and efficient.

The system constraints that result from the aforementioned functional and non-functional requirements are a need for databases, implemented as external files for patient and provider records as well as services provided and weekly billing data. Functions will be implemented to both parse the external file for patient/provider/service name and number and remove or modify appropriate values. Regarding the previously mentioned non functional requirements, reliability and efficiency can be maximized by implementing data structures and unit tests with these attributes in mind. In a system that is scaled to a larger size, patient and provider information can be password protected (using hashed password) and input/output between the provider terminal and the ChocAn data center would occur via secure shell.

The primary internal dependencies are databases storing provider information, patient information, metrics on total weekly billings, and services provided. The databases will be implemented as external files read in and read out as as memberships are validated, at the request of providers for a list of services provided, at the request of managers for up to date billing amounts in total or per provider, and at midnight each Friday when billing information is sent to patients, providers and managers alike. Without these databases, no other subsystem would be functional as there wouldn't be a source of information with which to verify or store members, providers, or services. External dependencies are Java libraries used to implement data structures (linear linked list and hash tables), and system hardware that the software is being executed on. The ChocAn billing software is implemented in Java and will work on Linux, UNIX, Mac OS, and Windows operating systems.

2.2. Methodology

The software development methodology used is an agile and plan driven approach blend. The approach follows elements of the waterfall method, a plan driven approach, in that requirements definition and system and software design have occurred before the the implementation of any code, whereas in a classically agile approach there is inter-leaving between the planning and development steps. Unlike the waterfall method, the software is being developed incrementally, with each milestone established in the requirements document acting as a deliverable to ChocAn. Rather than having a single functional version developed and delivered on June 1st, A functional version of the software will be written that addresses each requirement individually, once that requirement is fulfilled, the software will be built upon to include additional requirements until all functional and non-functional requirements are accounted for. Finally, each member in the group is focusing on their own particular programming strength and being given the opportunity to approach their respective problem in a manner that best suits them, which is in keeping with the “people not process” principle of agile development.

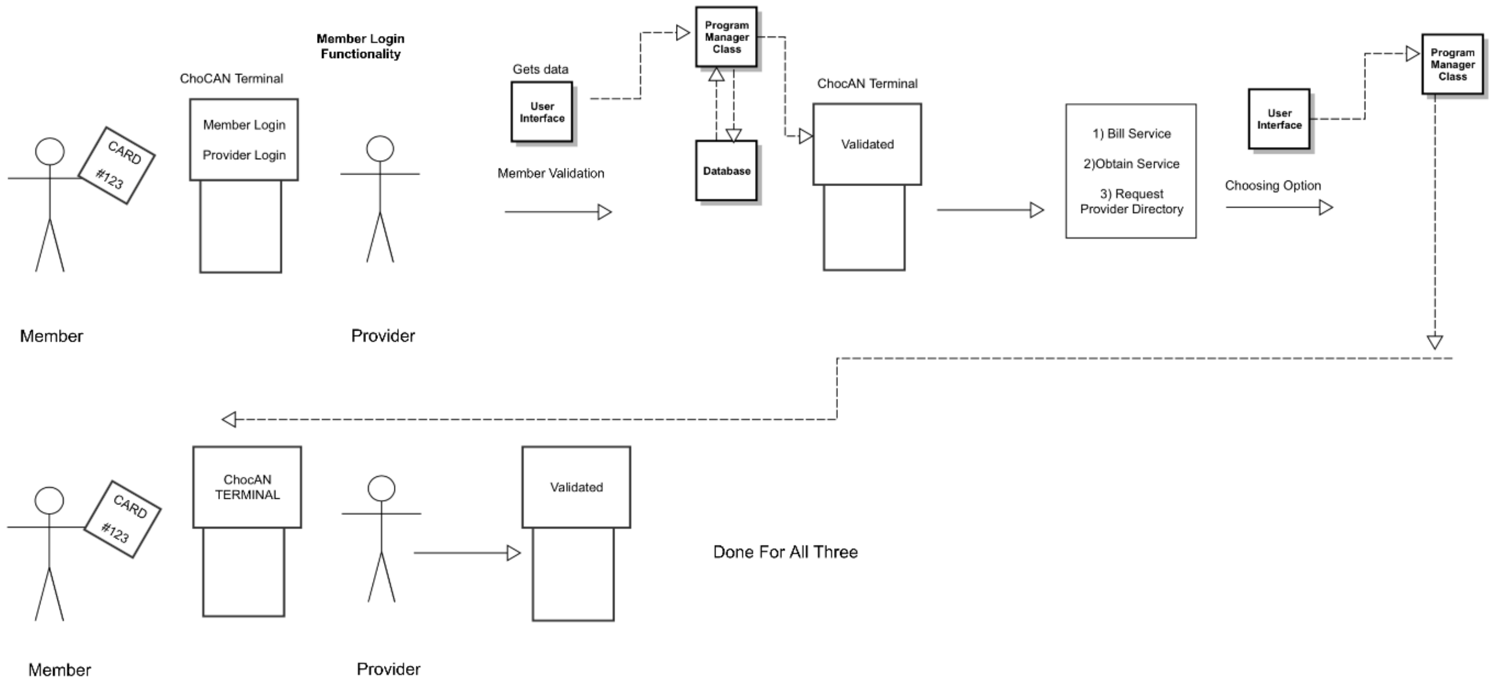
3. System Overview

ChocAn data processing software begins with the the member seeking services being validated as a current ChocAn member who's up to date on their account, by a provider or operator with a card swipe in the ChocAn terminal. Once member validation has taken place, the member is provided a menu where they can choose to Bill a service, Obtain a Service. At this point, the Members can choose to obtain services. In order to Bill a Service, the provider again swipes the member's card and bills the members account.

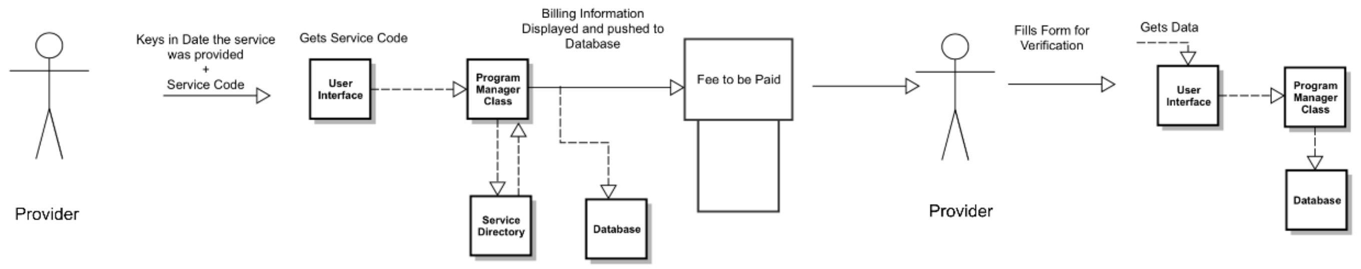
At any time a provider can request an email with an alphabetized list of provider services. During the day, the software at the ChocAn Data Center is run in interactive mode. Any provider, manager or terminal operator can append to, remove information from, or otherwise edit the member directory, provider directory, and provider services directory.

At any time a manager can login and request an email containing the week's billings to date and the amount each provider has billed up to that point in the week. At midnight every Friday, the main accounting procedure runs, which prints a report for the manager that contains the weeks total billings for all providers, and the amount of billing each provider at ChocAn has done for the week, which specifies who has billed and who hasn't. Finally an EFT record is kept holding every billing transaction made by ChocAn members.

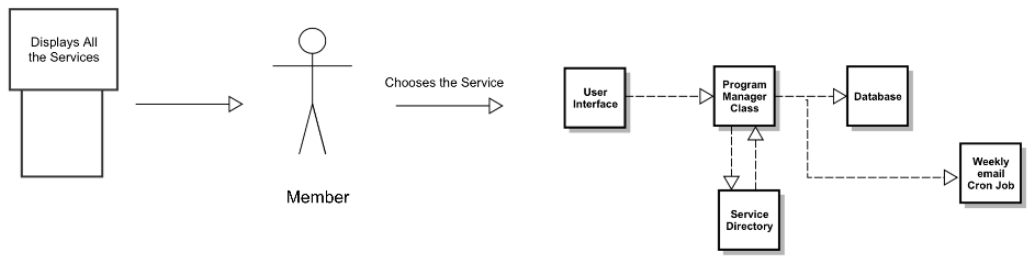
System Overview



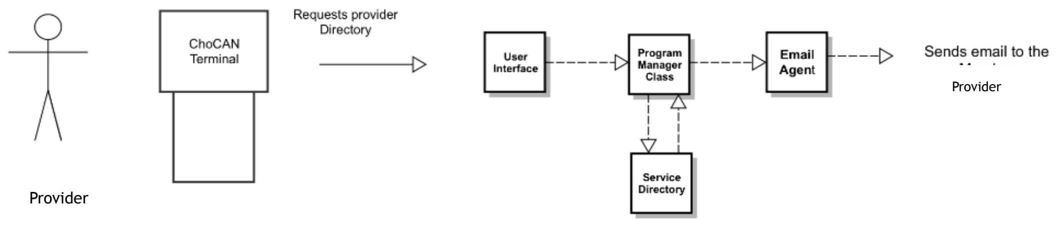
Bill Service

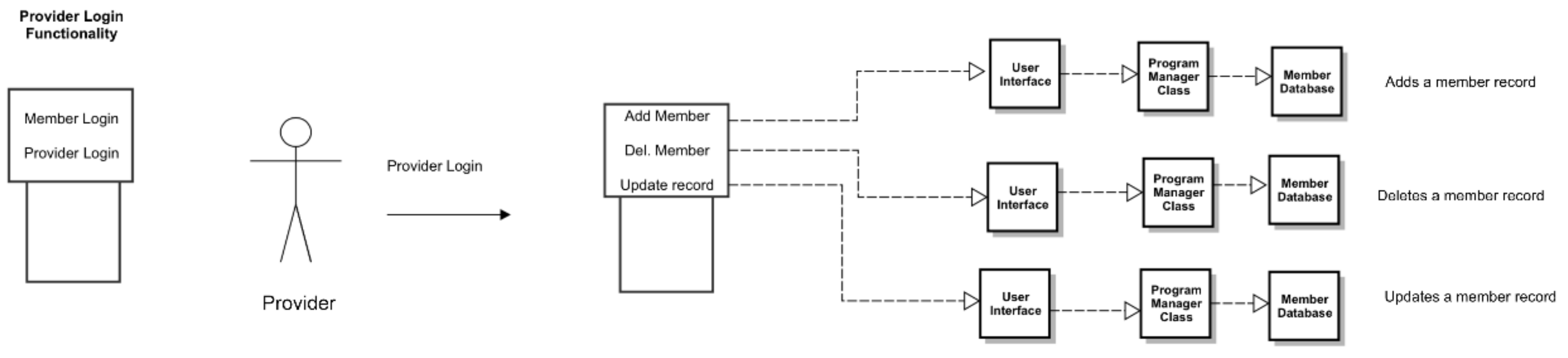


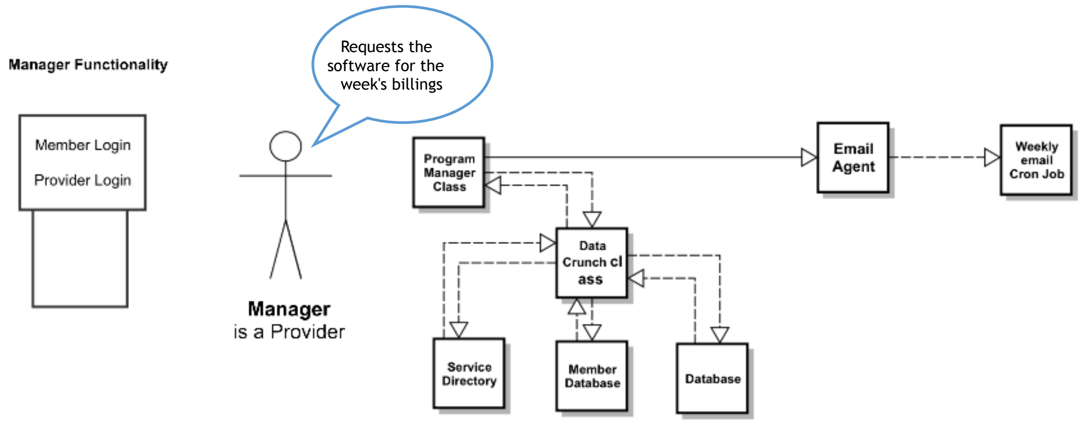
Obtain Service



Request Provider Directory







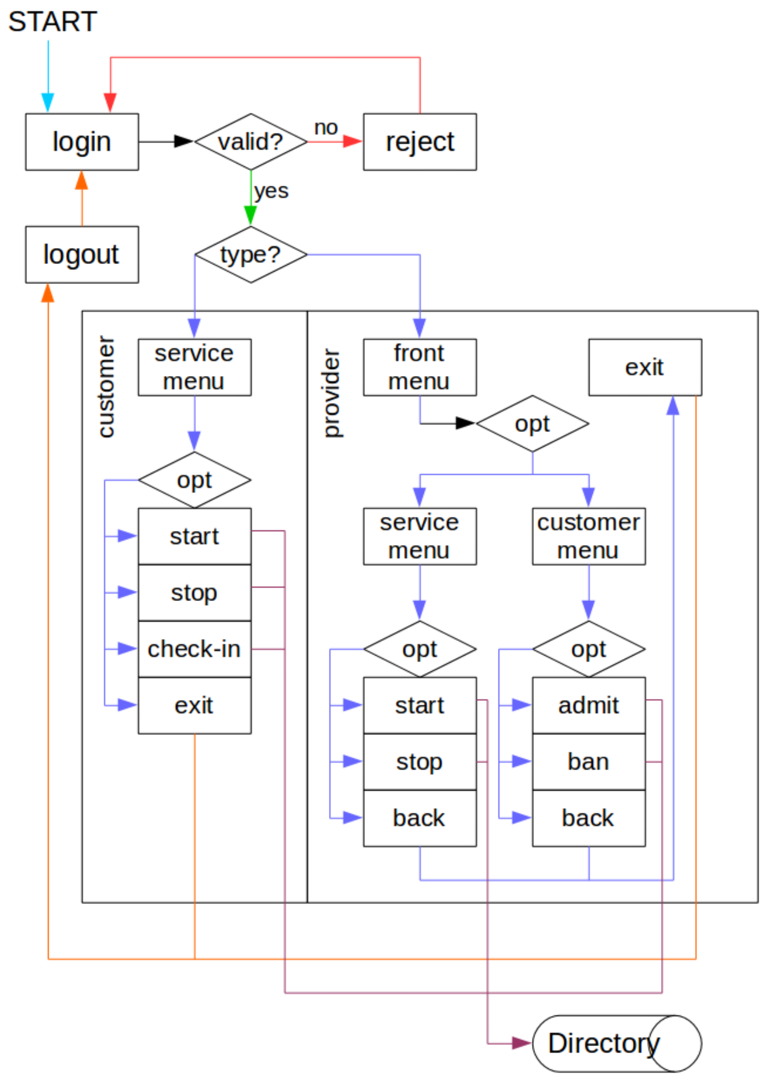
* The System Overview diagram also contains a high level overview of the data flow through the program as the end application users interact with it.

4. System Architecture

The system-level architecture descriptions of the project will include multiple sections. These sections are the client interface, service directory, provider directory, patient and customer directory, and file input and output. The directories will manage the patient and provider information. The file input and output will interact with the file database to save and load up patient and provider data. These systems will be instrumental for the data processing side of the program. This section will identify and discuss these sub-systems and provide an overview of how they will work.

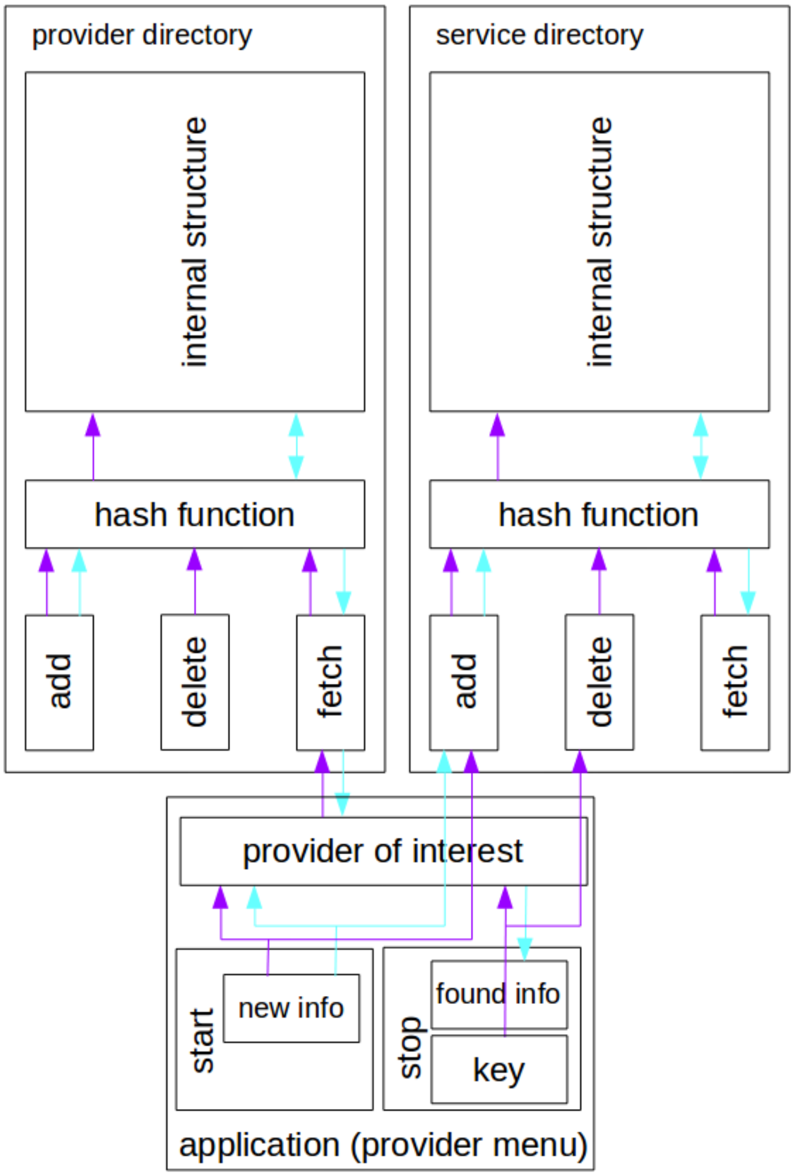
4.1. Client Interface and Commands

The client interface sub-system will be the way that operators will be able to communicate with the other sub-systems that control the information stored on the ChocAn data processing software. The client interface will include commands like adding, deleting, displaying, and billing members. The client interface will display available commands depending on the context of current time in the program. The client interface will receive commands from an operator through a keyboard. The commands that are received by the program will be sent to other sub-systems to be executed. The client interface will not have access to the information being processed and will only be able to communicate with sub-systems and send them commands. This separation is necessary to create a design that includes security and encapsulation.


START
login
valid?
no
reject
yes
logout
type?
customer
service menu
opt
start
stop
check-in
exit
provider
front menu
exit
opt
service menu
customer menu
opt
start
stop
back
opt
admit
ban
back
Directory

4.2. Service Directory

The service directory will be a sub-system that will manage all the possible services provided by ChocAn providers. This sub-system will keep track to any changes that operators will make to the services that are offered by ChocAn. The service directory will not be the main sub-system so it will act as a supporting sub-system. The main functions of the service directory will include adding, deleting, and fetching provider services. There usually will not be direct interactions between the client and the service directory. The service directory will be able to display all the services that are provided by ChocAn. When a service needs to be deleted or edited the service ID shall be entered into the client program. Once the ID number is entered, it will be sent to the service directory to validate that such a service with the provided ID number exists. The System shall hash the ID number and will or will not return the full information of a particular service. When a service is to be added, the directory shall hash the service based on the ID and insert it into the directory alongside the other services.

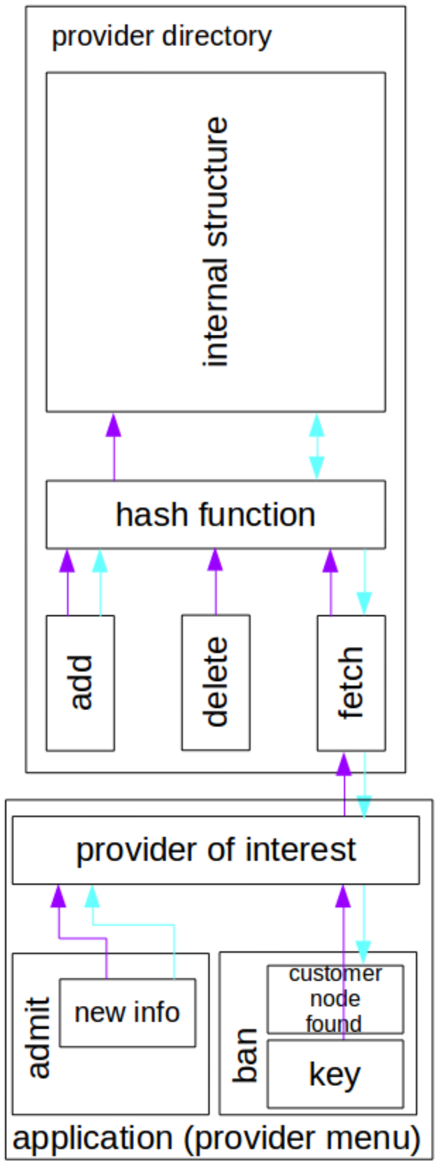

provider directory
internal structure
hash function
add
delete
fetch
service directory
internal structure
hash function
add
delete
fetch
provider of interest
start
new info
stop
found info
key
application (provider menu)

4.3. Provider Directory

The provider directory sub-system will manage all the provider information along with some service and patient information that pertains to certain providers. The provider directory will receive commands from the operator through the client interface. The provider directory will be able to be accessed by providing a provider ID. The provider directory will then hash the provider ID and return the appropriate provider information if it exists. Otherwise, the program will throw an error. The provider directory will be able to add, delete, or edit provider information. The provider directory will include options to add a provider and add patients who are using the providers' services. Each provider within the provider directory will also manage a list of services and corresponding list of patients. The provider directory will be able to search through the list of services as they pertain to the patients that used those services. This system will also keep track of the number of times certain patients have used a certain service of a provider.

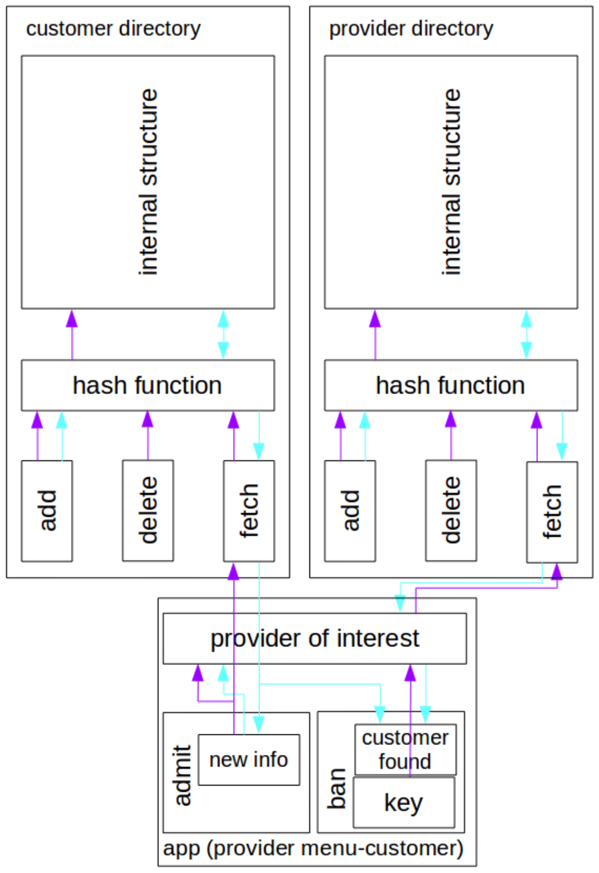
4.3.1 Provider Directory – Services and Patients

The services and patients is a sub-system of the provider directory. This system is responsible to processing and managing the service and patient information as it pertains to a certain provider. This system will keep track of basic service information. A service will be written in this system if a patient uses that particular service. This system will not include all the service information for each provider. That is not necessary in the implementation of this project and is handled by the service directory system. Each service written will include a list of basic patient information that have used that service and the amount of times the patient has used that service. This sub-system will handle the adding, deleting, and editing of services and their corresponding patient information.


provider directory
internal structure
hash function
add
delete
fetch
provider of interest
admit
new info
ban
customer node found
key
application (provider menu)

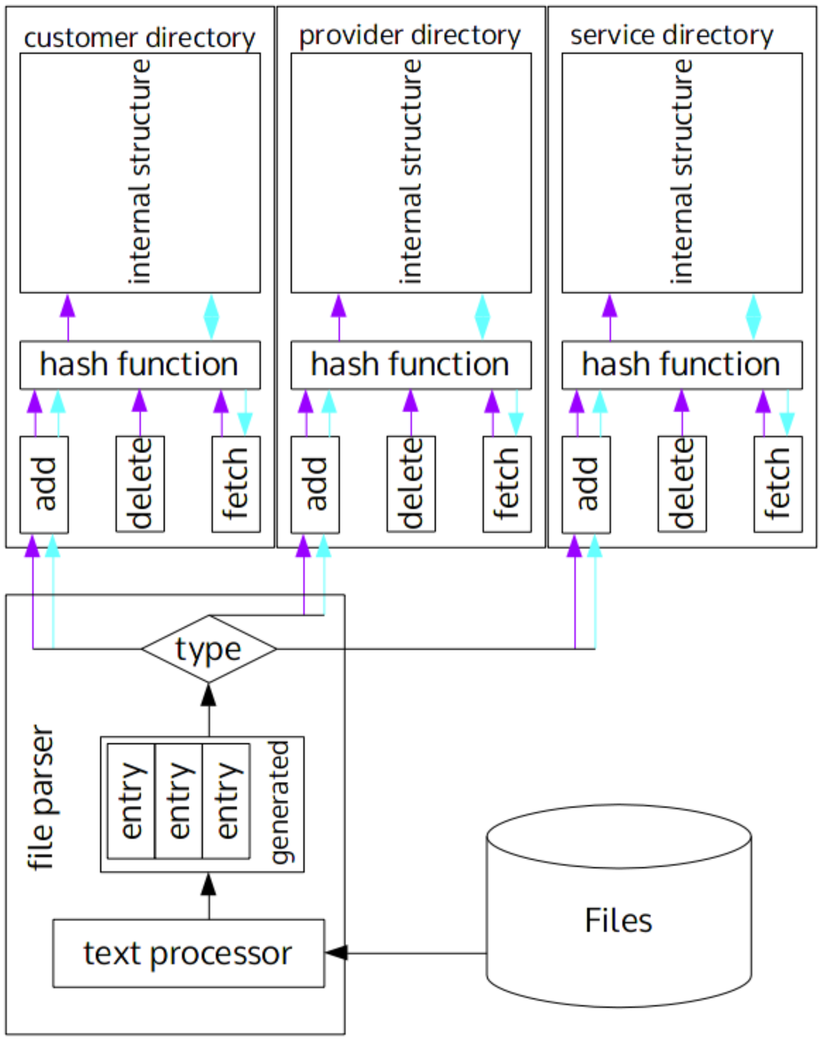
4.4. Customer and Patient Directory

The customer and patient directory sub-system will handle everything that is patient related and customer related. The directory will be accessed by providing a customer ID number that will be hashed by the system and the appropriate patient information will be retrieved. The system will allow for the adding, deletion, editing, displaying, and billing of patients at ChocAn. This system will include a list of patients. Each patient will also have a basic list of the services that they have used throughout their history at ChocAn. Every time a patient used a particular service, the list of services will either grow or stay the same. This system will also keep track of how many times a certain service was used by a particular patient and the amount of fees owed by the same patient. This subsystem will interact with the client interface through commands. The operators will not have direct access to the system as to ensure security. This system will be abstracted out and shall use encapsulation to protect patient and service information.


customer directory
internal structure
hash function
add
delete
fetch
provider directory
internal structure
hash function
add
delete
fetch
provider of interest
admit
new info
ban
customer found
key
app (provider menu-customer)

4.5. File Input and Output

The file input and output sub-system will be responsible for handling of the file database. This system will have the capabilities to load and save to a file database. It will assign standard names to files and manage directories. The system will ensure that each patient and provider will have their own files and directories where data shall be stored. The data will be stored locally and not transmitted over Internet or other means. This system will directly interact with the raw patient and provider data. When the program is started, the system will be signaled to start reading and loading in information. It will read the data and send it to other subsystems to be inserted into the data structures that are being created as the program processes and inserts data. Once the program is to be shut down, it will signal this system to start saving the data to the file system. It will request the information to be written from other systems and will update the information in the file system properly. This system will mainly be invoked in the beginning and end of running the ChocAn data processing software.


customer directory
internal structure
hash function
add
delete
fetch
provider directory
internal structure
hash function
add
delete
fetch
service directory
internal structure
hash function
add
delete
fetch
type
file parser
entry
entry
entry
generated
text processor
Files

5. Detailed System Design

This section will cover a detailed level overview of the system design that will be used in this project. This section will go over how the sub-systems that were discussed in the previous sections will be implemented and what data structures and libraries shall be used.

5.1 Client Interface

The client Interface will consist of a terminal that allows the client to input commands. These commands will be used to navigate the client to the appropriate directory or output data that the client is searching for. The implementation for the client interface and terminal will be written in Java.

Class name: ClientInterface	
Summary: ClientInterface class will be used as a controller class that abstracts to other classes. This class will be made up of small functions that will send the user into more specific classes or display basic information based on the user input.	

Example method name:
 	
PatientSearch();
PatientFind();
Patientlocate();
	A method that will allow the client to search and display the patients services previously used, Balance, and last visit.
	Syntax Description
	public PatientSearch(int SearchID){
	//Check that search type is correct
	//if invalid type return error messages
	if Patient ID is found {
	return “found”
	//it is possible that the information will be display from this if statement
	return data or true
	}
	else if Patient ID not found{
	//ID was not found, inform the user that this ID does not exist in the directory
	return “error message”}
	}

Example method name:
 	
ProviderDirectorySearch();
LocateProvider();
FindProvider();
	A method that can be used to search for providers and display information regarding providers.
	Syntax Description
	Public Void ProviderDirectorySearch(){
	    //Ask user if they wish to search a
	    provider by name or display all
	    //If the user searchers by name
	    provider.retrieve(SearchName);
	    //if the user searches all
	    provider.DisplayProviders();

Examples of method names:
 	
ServiceDirectorySearch();
Servicelookup();
FindService();
	This method will be used to search for services that are offered by providers.
	Syntax Description
	public Void ServiceSearch(){
	    //Ask the user if they want to look up a
	    service by index number or search for
	    a complete list of the service provided
	    //If the user searches by index
	    service.retrieve(indexFind);
	    //If the user wants to search for all
	    of the users
	    service.displayServices();

5.2 Provider Directory

The Provider Directory will consist of methods that will allow for the addition,deletion and update ability for a new provider. This class will also be implementing a hash table or hash map of sorts from the Java library. The Implementation for the Provider Directory will be written in Java.

Class name: ProviderDirectory
Summary: The providerDirectory class will be implemented using a Hash Table. The Hash table will hash based on the Provider ID number. The class will have methods that can add new providers, delete providers, update providers, display providers, search for a provider and hash on provider .

Example method name:
 	
addNewProvider();
InsertProvider();
CreateProvider();
	A method that will add a new provider to the directory.
	Syntax Description
	`public bool addNewProvider(Provider toAdd){`
	`    //This method will take a provider object send the toAdd.ID to a hash function hash(toAdd.ID). Once the ID has been mapped to a hash value the new provider will be stored in the directory`
	`    //if this the new provider is added without error the method will`
	`    return true;`
	`}`

Example method name:
 	
DeleteProvider();
RemoveProvider():
	This method that will be used to delete providers from the directory.
	Syntax Description
	`Public bool DeleteProvider(){`
	`    //allow the user to delete a provider`
	`    //If the provider is found then delete the provider and`
	`    return true;`
	`    //else if{the service is not found`
	`    //display a message indicating that the service does not exist in the directory`
	`    return false; }`
	`}`

Example method name:
 	
UpdateProvider();
	A method that will be used to update general provider information.
	Syntax Description
	public void UpdateProvider(){
	    //this method will have multiple options that the user can go through to edit general provider information.
	    //this method has no return type but will always ask the user to verify if there update is what they intended.
	}

Example method name:
 	
DisplayProviders();
ShowProviders();
OutputProviders();
	This method will allow for the provider directory to be searched by the user
	Syntax Description
	Public void DisplayProviders(){
	    //If there exist a directory then there is a search
	    //If there is no directory this the method sends an error and returns out of the method
	    //this method will display a list of providers within this directory
	    //this method has no return type
	}

Example method name:
 	
RetrieveProvider();
FindProvider();
LookUpProvider();
	This method will allow for the search of a specific provider.
	Syntax Description
	public void RetrieveProvider(string Provider){
	//This method will receive a Provider as the argument and look through the directory.
	//If{ the search comes back and there is a match in the directory for the provider being search hen that provider and there info is outputted to the screen or returned to the client interface}
	//else if{ the search comes back null or not found the method will output an error or return an error to the client interface informing the user that the provider is not in the directory}
	//method has no return type
	}

5.3 Service Directory

The service directory will consist of methods that will allow for the addition,deletion and update ability for a new service. This class will also be implementing a hash table or hash map of sorts from the Java library. The Implementation for the Provider Directory will be written in Java.

Class name: serviceDirectory
Summary: The serviceDirectory class will be implemented using a Hash Table. The Hash table will hash based on the service type. The class will have methods that can add new services, delete services, update services, display services, search for a services and hash on service .

Example method name:
 	
addNewService();
AddService();
CreateNewService();
InsertService();
MakeService();
	This method will be used to create a Service and add it to the existing directory.
	Syntax Description
	public bool addNewService(Service toAdd){
	    //This method will take a provider object
	    send the toAdd.ID to a hash function
	    hash(toAdd.ID). Once the service type has
	    been mapped to a hash value the new
	    service will be stored in the directory
	   //if this the new provider is added without
	   error the method will
	   return true;
	   else if{ there was in error with this process
	   return false; }
	}

Example method name:
 	
DeleteService();
RemoveService();
	This method will be used to delete unused or unneeded services in the services directory.
	Syntax Description
	Public bool DeleteService(){
	    //allow the user to delete a service
	    //If the service is found then delete
	    the service and
	    return true;
	    //else if {the service is not found
	    //display a message indicating that the
	    service does not exist in the directory
	    return false; }
	}

Example method name:
 	
UpdateService();
	A method that will be used to update a service that is currently stored inside the service directory
	Syntax Description
	public void UpdateService(){
	//this method will have multiple options that the user can go through to edit general service information.
	//this method has no return type but will always ask the user to verify if there update is what they intended.
	}

Example method name:
 	
DisplayServices();
ShowService();
lookUpService();
	This method will allow for the service directory to be search.
	Syntax Description
	Public void DisplayServices(){
	//If there exist a directory then there is a search
	//If there is no directory this method sends an error and returns out of the method
	//this method will display a list of the services being offered in the directory
	//this method has no return type
	}

Example method name:
 	
RetrieveService();
FindService();
LookUpService();
LocateService():
	This method will allow the user to search for a specific service in the directory
	Syntax Description
	public void RetrieveService(String ServiceType){
	//This method will receive a Service type as the argument and look through the directory.
	//If{ the search comes back and there is a match in the directory for the service being searched for then that service and its info is outputted to the screen or returned to the client interface}
	//else if{ the search comes back null or not found the method will output an error or return an error to the client interface informing the user that the service is not in the directory}
	//method has no return type
	}

5.4 Patient Directory

The Patient directory will consist of methods that will allow for the addition of a patient,deletion, ability to update for patients, display, and search. This class will also be implementing a hash table or hash map of sorts from the Java library. The implementation for the Patient Directory will be written in Java.

Class name: patientDirectory
Summary: The patientDirectory class will be implemented using a Hash table. The Hash table will hash based on the patient ID number. The class will have methods that can add a new patient, delete a patient, update a patients information, display a list of patients, or search for a specific patient in the directory.

Example method name:
 	
addNewPatient();
AddPatient();
CreateNewPatientProfile();
	This method will allow the user to create/add/insert a new patient into the directory
	Syntax Description
	public bool addNewPatient(Patient toAdd){
	    //This method will take a patient object
	    send the toAdd.ID to a hash function hash(toAdd.ID). Once the patient type has been mapped to a hash value the new patientwill be stored in the directory
	    //if this the new patient is added without error the method will
	    return true;
	    else if{ there was in error with this process
	    return false; }
	}

Example method name:
 	
DeletePatient();
RemovePatient();
	This method will allow the user to remove a patient from the directory.
	Syntax Description
	public bool DeletePatient(){
	    //allow the user to delete a patient in order to perform this
	    //If the patient is found then delete the patient and
	    return true;
	    //else if {the patient is not found
	    //display a message indicating that the patient does not exist in the directory
	    return false; }
	}

Example method name:
 	
UpdateService();
	This method will allow for updating the information that exist in the specific patients directory.
	Syntax Description
	public void UpdateService(String toFind){
	    //this method takes in the patients name
	    as a argument
	      //this method will have multiple options
	      that the user can go through to edit general
	    service information.
	    //this method has no return type but will
	    always ask the user to verify if there
	    update is what they intended.
	}

Example method name:
 	
DisplayPatient();
ShowPatient();
LookUpPatient();
	This method will allow for a list of patients to be displayed
	Syntax Description
	public void DisplayPatient(){
	    //If there exist a directory then
	    there is a search
	    //If there is no directory this method
	    sends an error and returns out of the
	    method
	    //this method will output a list of the
	    patients contained in the directory
	    //this method has no return type
	}

Example method name:
 	
RetrievePatient();
LookUpPatient();
	This methods purpose is to find a specific patient in the directory and output there information
	Syntax Description
	public void RetrievePatient(int PatientID){
	//This method will receive a PatientID as the argument and look through the directory.
	//If{ the search comes back and there is a match in the directory for the ID being searched for then that patient and its info is outputted to the screen or returned to the client interface}
	//else if{ the search comes back null or not found the method will output an error or return an error to the client interface informing the user that the patient is not in the directory}
	//method has no return type
	}

5.5 File Input and Output

The file reader will be used to read in and read out files from our database to the terminal screen. The methods contained in this class will be used as helpers to ensure that data is read in and out correctly. This portion of the program will be written in Java.

Class name: FileIO
Summary: This class will be used to read in data to the database and store it. This class will also be responsible for reading out filed to the terminal or returning information to the client interface screen.

Example method name:
 	
ReadIn();
ReadInFIle();
	This method will be used for reading in files
	Syntax Description
	Int ReadInFIle(){
	    //method will be responsible for reading in
	    text from a file
	    //if there any errors while reading in this
	    method will return a 0
	   //if there is nothing to be read this method will
	    also return a 0
	   //else, this method will go into a loop that will
	   read in the file until the end of the file.
	   While(!readeof())
	   {
	     …
	   return 1;
	   }
	   //this method should return a 0 if it fails
	   and a 1 for success

Example method name:
 	
ReadOut();
ReadOutFIle();
	This method will be used to read data to text file or database.
	Syntax Description
	Int ReadOutFile(){
	      //this method will create a file to write to
	       if{ there is nothing to write to then
	       this method will return a 0 }
	       //this method will use a loop to iterate
	       through the list and write the data
	       to a file
	       while()
	       {
	         …
	        return 1;}
	        //this method returns a 0 for fail and a 1
	       for success
	}