[CPSC 319]

Software Design Specification

First Draft

Date: February 21, 2005

[Team 4]

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Aaron Hui, Sunae Kim, Andy Kim, Carter Lukman, John Wong, Yow-Hann Lee, Charles Krzysik	First Draft	02/21/2005

Contents

1. INTRODUCTION ……..……..…….….4

1.1 SYSTEM OVERVIEW ……..……....……4
1.2 DEFINISTIONS AND ACRONYMS ……..……..……4

2. DESIGN CONSIDERATIONS ……..……..……..…5

2.1 ASSUMPTIONS ……...….5
2.2 CONSTRAINTS ……...……..5
2.3 SYSTEM ENVIRONMENT ……..….….….….6

3. ARCHITECTURE …….…...……..7

3.1 OVERVIEW ……..…….7
3.2 RATIONALE ……..……7

4. HIGH LEVEL DESIGN ……..……..8

4.1 CONCEPTUAL VIEW …….…….8

5. LOW LEVEL DESIGN …….….…...9

5.1 MODULE 1..N …….……..9

6. USER INTERFACE DESIGN …….……50

6.1 APPLICATION CONTROL …….…50
6.2 SCREEN 1..N ……....51

1 Introduction

1.1 System Overview

This purpose of this project is to design and implement an online chat program. The program is intended to be used as part of an interactive live help system where a customer enters into a chat session with a customer service representative (CSR) and may ask questions in regards to a product or service.

The system will have three primary users, the customer who asks the questions, the CSR who answers the questions, and the manager who oversees the CSRs and monitors their overall performance in regards to their ability to answer questions and solve problems for the customer. After each chat session a customer may give a single digit rating to the CSR reflecting the customer's opinion on the CSR's ability to help and if they choose, write a comment expanding on their opinions.

Each user of the system will have and account and access the system through a web page and be identified by their unique login ID and password. In order for a customer to gain access to the system they will be required to register by filling out a web application with required information. This information will be used to create a new account for the new customer.

All account and chat information will be stored on a database. The system will be developed using client/server architecture which will allow the system to be scaled up as the number of users increase.

1.2 Definitions and Acronyms

Applet	- Java based code embedded within a webpage
Manager	- the person who is in charge of the CSRs. A manager may operate as a CSR within the system, taking on all the responsibilities and task that a CSR may perform
Customer	- the customer who engages in real time chat
Database	- SQL based database containing customer information
GUI	- Graphical User Interface
JDK	- Java Development Kit used for compiling Java code.
JDBC Driver	- A requirement for Java Database Connectivity to communicate with the database using the database's native networking protocols.
JVM	- Java Virtual Machine used during execution of Java code.
OS	- Operating System
Server	- A computer hosting the chat software
SQL	- Structured Query Language; used for querying and updating a database
Interface	- the graphical user interface or the display that encapsulates functionality
Scalable	- the ability to upgrade or to reuse existing material to satisfy future requirements

2 Design Considerations

2.1 Assumptions

The system is to be run on UNIX and Windows machines. The systems are assumed to have web browsers pre-installed. The list includes: Firefox, Internet Explorer 5.5 and later and Mozilla. Customers are assumed to all be connected via high speed internet. Therefore, the consumption of bandwidth by the system will not be an issue. We also assume that hardware and server components can be upgraded to satisfy the criteria for withstanding increasing demands for live customer support. Lastly, managers and CSRs are assumed to have sufficient technical training to manage the system. The manager should be trained to maintain the MySQL database of the system. Therefore, they have an understanding in database administration and how to write SQL queries.

2.2 Constraints

2.2.1 Regulatory policies

The manager is allowed to monitor all chats. CSRs will have different interfaces from customer interfaces. Also, the manager's interface must be slightly altered from the interface of CSRs. Manager's interface includes extended functionality. Managers are allowed to view previous chat log histories and to view current dialogues between CSRs and customers. CSRs have the ability to place hidden comments on the customer's account for other CSRs to view. Due to the nature of a chat support system, customers are assumed to be at a novice level in computer use. Therefore, the customer interface will remain uncluttered and contain basic functionality. The customers only have the ability to view the current chat log and to send messages.

2.2.2 Reliability Requirements

The customer will be able to activate the live chat support system with the expectation that the MTBF or Mean Time between Failure is one month. In addition, the live chat support system is to be deployable from a web browser without the necessary installation of it as a standalone application. In other words, the customer should not have to download an executable file in while to install a program that provides chat support. The system shall be deployable at any time during the course of a day; this excludes periods where the system may undergo maintenance or where there may be hardware failure.

2.2.3 Criticality of the System

The chat support system provides help for customers who have questions regarding services or goods. Therefore, it is not essential for the system to be operable 100% of the time after deployment. It is not intended for circumstances in which lives depend on the proper functionality of the system.

2.2 Constraints (Continued…)

2.2.4 Safety and Security Considerations

Security regarding internet chat logs is not a priority. However, one precautionary measure should be taken. The IP addresses of customers seeking CSR's support must be tracked. The purpose of this implementation is that CSRs do not create fraudulent customers and reap the benefits from providing more customer support. Since CSRs are to be compensated on a per customer basis, this feature is particularly vital to the integrity of the chat support system. The system connections are not required to have encryption in place. The security of the chat dialogues between CSRs and customers are projected to be minimal.

2.3 System Environment

The system will be run on UNIX and Windows machines. Also, it will be able to run on a 2.53 GHz Pentium IV with 512MB of memory and a broadband Internet connection. The server will run on a server machine, and the customers will run on individual customer machines. The system must be written in Java 1.5.xx using SWING to meet GUI requirements. And for network communication, the Java RMI protocol will be used.

3 Architecture

3.1 Overview

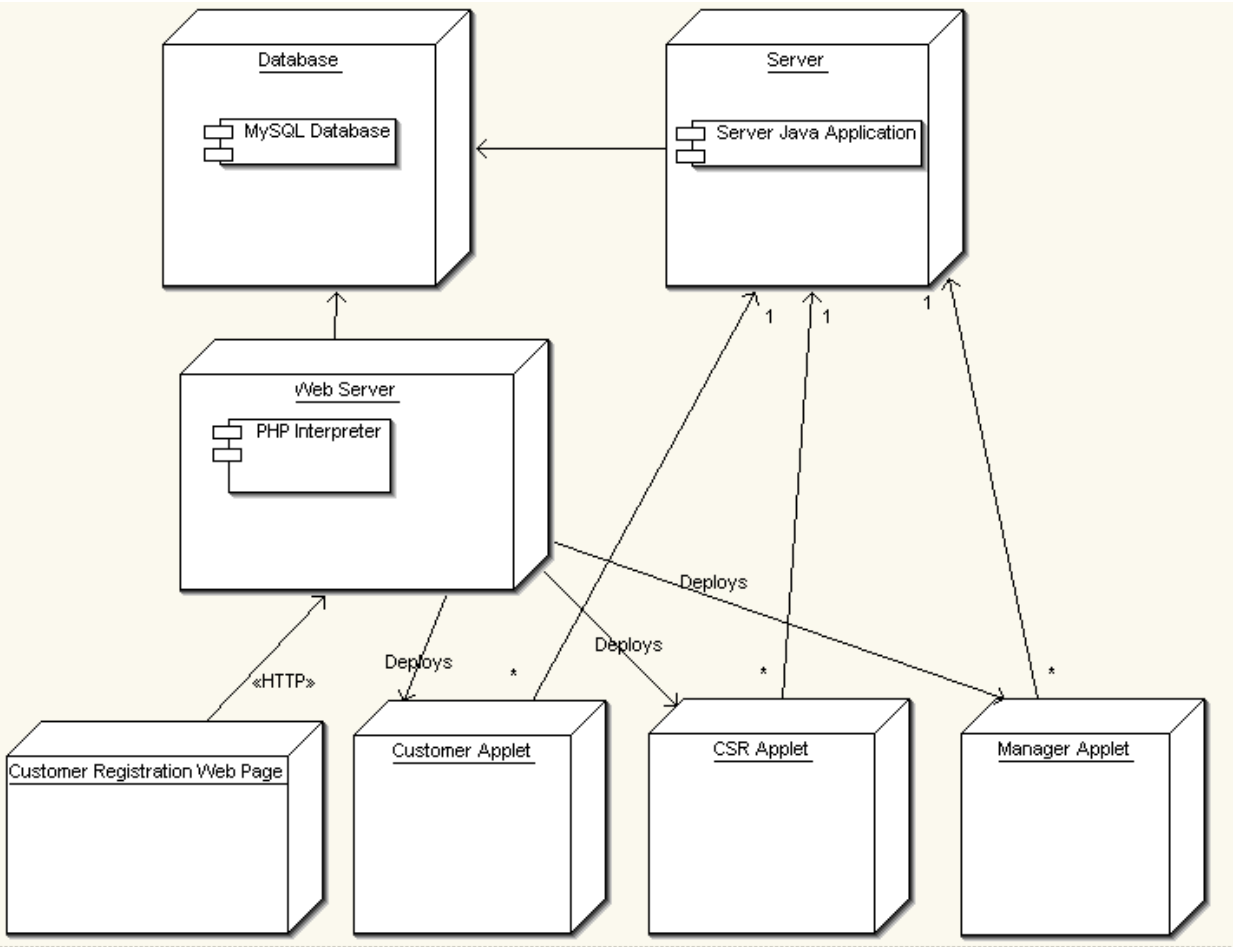
The software architecture for online chat support system is based on the Client/Server architecture style. Client and Server will communicate with each other using the Java RMI protocol. The client program uses Java SWING for the UI. Clients will communicate their requests to the Server and the Server will carry them out. The system is decomposed in customer, CSR, manager, database subsystems, and all subsystems communicate through the server.

3.2 Rationale

The Client/Server architecture style is logical choice because it allows multiple users accessing on the server and centralized data such as access control, synchronization, concurrency control, and data backup.

4 High Level Design

4.1 Conceptual View



The Web Server, Database, and Server machines can all be separate or in fact be running on one machine. The web server delivers the applets via Java WebStart, launch from a web site. The customer registers new accounts from a webpage.

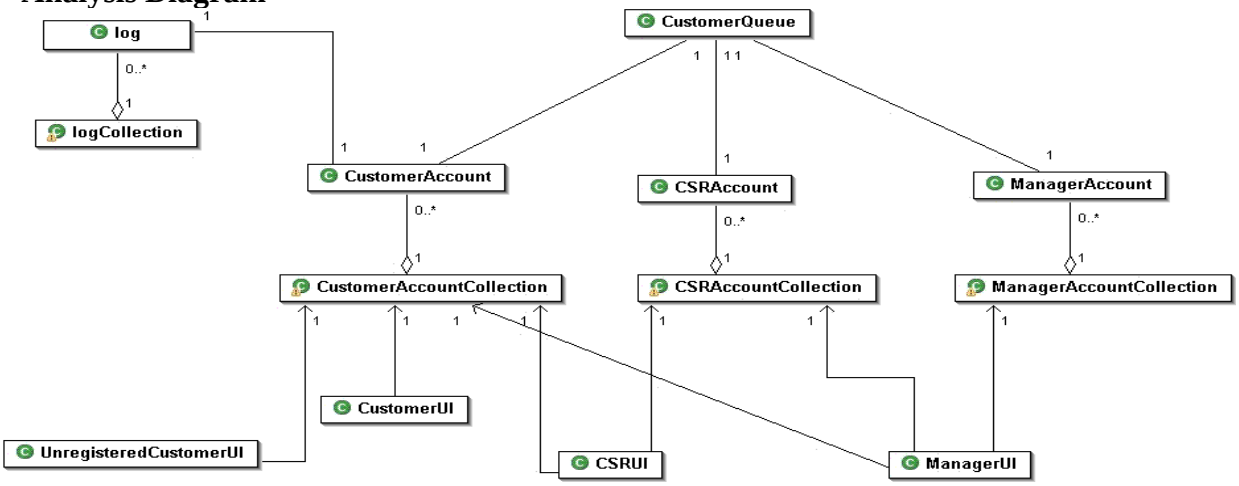
The Customer, CSR and Manager applets after launch establish a connection with the Server Java Application on the Server machine. The Customer Registration Web Page interacts with the Database without having to access the Server Java Application.

5 Low Level Design

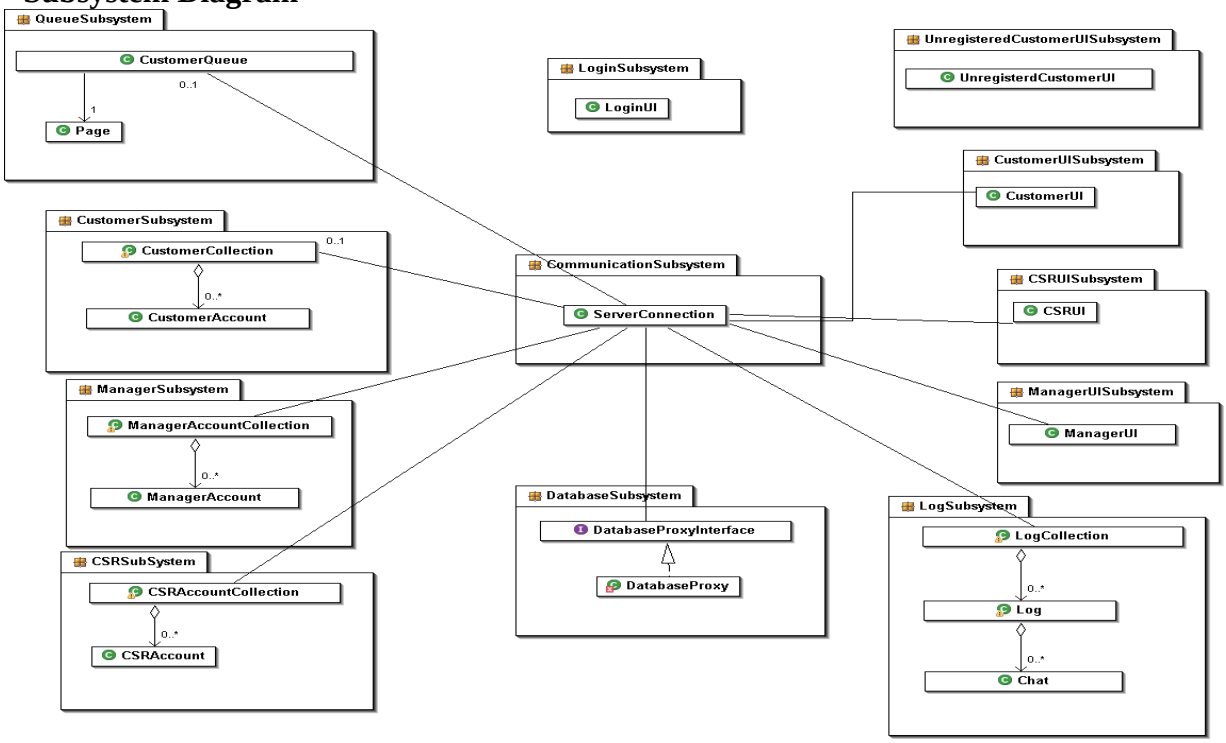
This section is gives a description of the low-level design of the system. It contains the break down of the over all system into it's individual subsystems and the classes that compose each subsystem. Each class and its methods are described fully and the interactions between class methods are shown in terms of subsystem interactions shown in the subsystem diagram. In addition the interaction between classes are shown through sequence diagrams that represent the primary interactions of users with the system and which classes take part in the particular interaction.

5.1 Module 1..n

- Analysis Diagram



- Subsystem Diagram



5 Low Level Design

- Data Dictionary

<u>LogSubsystem</u>

Entity Class: LogCollection

Responsibilities	**Collaborators**	**Attributes**	**Operations**
Stores and maintains the logs of all the registered customers	- Log - Chat	- logCollection	isEmpty - returns true if there are no logs in the collection addLog - enables a client to add a Log to the collection deleteLog - enables the client to delete a Log from the collection doesLogExist - returns true if a Log associated with a given customer id exist in the collection getLog - returns the Log that is associated with the give customerId getNumberOfLogs - returns the number of Logs in the collection

<u>LogCollection</u>
Attributes:

Array CustomerLogsArray [ ]
//Stored in alphabetical order by customer account Id ??

Methods:

public boolean isEmpty();
//pre**:** none
//post**:** returns true if there are no customer chat logs in the collection

5 Low Level Design

LogCollection (Continued…)

Methods:

public void addLog(Log aCustomerChatLog)
//pre**:** none
//post**:** aCustomerChatLog has been added to the collection of all //customer chat logs

public void deleteLog(String: customerId)
//pre**:** doesLogExist(customerId) == true;
//post**:** Log that belongs to a customer with id == customerId has been deleted

public boolean doesLogExist(String: customerId)
//pre**:** none
//post**:** returns true if a Log in the log collection is associated with a
//customer with id == customerId
public Log getLog(String: customerId)
//pre**:** doesLogExist(customerId) == true;
//post**:** the Log associated with customerId has been returned

public integer getNumberOfLogs()
//pre**:** none
//post**:** the number of Logs in Log Collection has been returned

Entity Class: Log

Responsibilities	Collaborators	Attributes	Operations
Stores and maintains all the chats that a single registered user has made.	- LogCollection - Chat	- chatsCollection - customerId	isEmpty - returns true if there are no Chats in this log getChats - returns all Chats that were created in between the given dates getCustomerComments - returns all the customer comments associated with all the Chats in this Log getCSRComments - returns all the CSR comments associated with all the Chats in this Log

5 Low Level Design

Entity Class: Log (Continued…)

Responsibilities	Collaborators	Attributes	Operations
Stores and maintains all the chats that a single registered user has made.	- LogCollection - Chat	- chatsCollection - customerId	getCustomerId - returns the Chat associated with the given chat number getNumberOfChatSessions() - returns the number of Chats in this log getCustomerId - returns the customer id associated with this log setCustomerId - sets the Id of the customer associated with this Log addChat - adds a chat to this Log

<u>Log</u>

Attributes:

Array individualChatsArray [ ]
//Stored chronologically from earliest time to latest
String customerId //id of the customer this Log belongs to

Methods:

public boolean isEmpty()
//pre**:** none
//post**:** returns true if the this Log contains no chats

public Array[ ] getChats(Date: startDate, endDate)
//pre**:** startDate <= endDate
//post**:** an array that contains all chats in this Log where
// startDate <= aChat.getDate() <= endDate

public Array[ ] getCustomerComments()
//pre**:** isEmpty() == false
//post**:** an array customer comment strings arereturned

public Array[ ] getCSRComents()
//pre**:** isEmpty() == false
//post**:** an array CSR comment strings is returned

Log (Continued...)

Methods:

public int getNumberOfChatSessions()
//pre**:** none
//post**:** the number of chat sessions in this Log has been returned

public void addChat(Chat aChat)
//pre**:** none
//post**:** aChat has been added to this Log

public Chat getChat(Int: chatNumber)
//pre**:** 1 <= chatNumber <= getNumberOfChatSessions
//post**:** the chat session with Chat.getChatNumber() == chatNumber has been returned

public String getCustomerId()
//pre**:** setCustomerId(customerId)
//post**:** customerId has been returned

public void setCustomerId(String customerId)
//pre: none
//post: the owner of this Log has been set to customerId

Entity Class: Chat

Responsibilities	**Collaborators**	**Attributes**	**Operations**
Stores the information for a single chat session	- LogCollection - Log	- moderator - chatNumber - date - CustomerComment - numberCustComm - CSRComment - NumberCSRComm - ChatText - chatStartTime - chatEndTme	getModerator - returns the Id of the moderator for this Chat setModerator - sets the Id for the moderator for this chat to mId getCustomer - gets the customerId for this chat setCustomer - sets the customeId setChatNumber - sets the unique chat number for this Chat

Entity Class: Chat (Continued…)

Responsibilities	Collaborators	Attributes	Operations
Stores the information for a single chat session	- LogCollection - Log	- moderator - chatNumber - date - CustomerComment - numberCustComm - CSRComment - NumberCSRComm - ChatText - chatStartTime - chatEndTme	getChatNumber - returns the Chat Id for this Chat setDate - sets the Date that this chat was created getDate - returns the Date that this chat was created setCustomerComment - sets the customer comment for this Chat getCustComment - returns the customer comment associated with this Chat setCSRComment - sets the CSR comment for this Chat getCSRComment - returns the CSR comment for this Chat addChatLine - adds a Line of text for this Chat getChatLine - returns the text of a given line number editChatLine - sets the given chat line to be a new string S

5 Low Level Design

Entity Class: Chat (Continued…)

Responsibilities	Collaborators	Attributes	Operations
Stores the information for a single chat session	- LogCollection - Log	- moderator - chatNumber - date - CustomerComment - numberCustComm - CSRComment - NumberCSRComm - ChatText - chatStartTime - chatEndTme	getNumberOfChatLines - returns the number of lines of text in this Chat setStartTime - sets the starting time of this Chat getStartTime - returns the start time of this Chat setEndTime - sets the end time of this Chat getEndTime - returns the end time of this Chat setCSRRating - sets the rating given to the CSR in this Chat getCSRRating - returns the rating given to the CSR in this Chat

<u>ChatInterface</u>

Attributes:

String moderatorId
String customerId
int chatNumber
Date chatDate
String customerComment
String CSRComment
Array[ ] chatText

Methods:

public void setModerator(String moderatorId)
//pre**:** none
//post**:** the moderator for this chat has been set to moderatorId

ChatInterface (Continued…)

Methods:

public String getModerator()
//pre**:** setModerator(moderatorId)
//post**:** moderatorId has been returned

public void setCustomer(String customerId)
//pre**:** none
//post**:** the customer for this chat has been set to customerId

public String getCustomer()
//pre**:** setCustomer(customerId)
//post**:** customerId has been returned

public void setChatNumber(int: chatNumber)
//pre**:** none
//post**:** the id number of this Chat has been set to chat Number
public int getChatNumber()
//pre**:** setChatNumber(chatNumber)
//post**:** chatNumber has been returned

public void setDate(Date: theDate)
//pre**:** none
//post**:** the date of this Chat has been set to theDate

public Date getDate()
//pre**:** setDate(theDate)
//post**:** theDate has been returned

public void setStartTime(Stirng: sTime)
//pre**:** none
//post**:** start time for the chat has been set to sTime

public String getStartTime()
//pre**:** setStartTime(sTime)
//post**:** sTime has been returned

public void setEndTime(Stirng: eTime)
//pre**:** none
//post**:** end time for the chat has been set to eTime

public String getEndTime()
//pre**:** setEndTime(eTime)
//post**:** eTime has been returned

ChatInterface (Continued…)

Methods:

public void setCustomerComment(String: theCustComment)
//pre**:** none
//post**:** this Chat's customer comment has been set to theCustComment

public String getCustomerComment()
//pre**:** setCustomerComment(theCustComment)
//post**:** theCustComment has been returned

public void setCSRComment(String: theCSRComment)
//pre**:** none
//post**:** this Chat's CSR comment has been set to theCSRComment

public String getCSRComment()
//pre**:** setCustomerComment(theCSRComment)
//post**:** theCSRComment has been returned

public void addChatLine(String: chatLine)
//pre**:** none
//post**:** a line of text == chatLine has been added to this chat

public void editChatLine(int lineNumber, String newText)
//pre**:** 0 < lineNumber<= getNumberOfChatLines()
//post**:** the text on lineNumber has been changed to newText

public String getChatLine(int lineNumber,)
//pre**:** 0 < lineNumber<= getNumberOfChatLines()
//post**:** the text on lineNumber has been returned

public int getNumberOfChatLines()
//pre**:** none
//post**:** the number of lines of text in this Chat has been returned

public void setCSRChatRating(int: csrRating)
//pre**:** none
//post**:** the CSR's rating for this Chat has been set to csrRating

public int getCSRChatRating()
//pre**:** setCSRChatRating(csrRating)
//post**:** csrRating has been returned for this Chat.

5 Low Level Design

CustomerSubsystem

Entity Class: CustomerUI

Responsibilities	Collaborators	Attributes	Operations
To provide a registered customer with an interface to access and utilize the system	- CustomerAccountCollection - CustomerAccount custAcc	- UserName - Password - Email - FirstName - LastName	loginUserName - logs into the system enterCustomerQueue - enters the customer into the chat queue to speak to a CSR or manager sendMessage - send message to CSR or manager deleteChatComment - delete comment of a chat session viewCurrChatLog - view current chat log viewChatHistory - view chat history of customer with CSRs editOwnAccount - change own customer account

CustomerUI

Attributes:

```
String password;
String loginUserName;
String msg;
Chat theChat;
Log theLog;
CustomerAccount custAcc
```

<u>CustomerUI (Continued…)</u>

Methods:

```
public void loginUserName(String UserId, String password);
// pre: none
// post: customer with the associated account is connected to the server

public void enterCustomerQueue();
// pre: none
// post: customer has been entered into the chat queue

public String sendMessage(String msg);
// pre: none
// post: customer's typed input is sent to the server

public String viewCurrChatLog();
// pre: none
// post: customer's current chat dialogue with CSR is displayed

public Log viewChatHistory();
// pre: none
// post: customer's chat history/log is returned

public void editOwnAccount(CustomerAccount custAcc)
// pre: none
// post: customer's own account has been edited
```

5 Low Level Design

Entity Class: CustomerAccount

Responsibilities	Collaborators	Attributes	Operations
To store information on the customer	- Log - CustomerAccountCollection - CustomerQueue	- userId - firstName - lastName - phone - email	checkUser - authorizes user with valid userId and password getUserID - get customer userID getFirstName - get customer first name getLastName - get customer first name getTelephoneNum - get customer phone number getEmailAddress - get customer email address getPassword - get manager password editUserID - change manager user ID editFirstName - change manager first name editLastName - change manager last name editTelephoneNum - change manager phone number editEmailAddress - change manager email address editPassword - change manager password

5 Low Level Design

<u>CustomerAccount</u>

Attributes:

String userID;
String firstName;
String lastName;
String telephoneNum;
String emailAddress;
String password;

Methods:

public Bool checkManager(String password);
// pre: none
// post: authenticates customer if the given password matches the password in the customer account

public String getUserID();
//pre**:** none
//post**:** customer userID has been returned

public String getFirstName();
//pre**:** none
//post: customer first name has been returned

public String getLastName();
//pre: none
//post: customer last name has been returned

public String getTelephoneNum();
//pre: none
//post: customer phone number has been returned

public String getEmailAddress();
//pre: none
//post: customer email address has been returned

public String getPassword();
//pre: none
//post: customer password has been returned

public String editUserID(String userID);
//pre: none
//post: customer user name has been changed

<u>CustomerAccount (Continued…)</u>

Methods:

```
public String editFirstName(String firstName);
//pre: none
//post: customer first name has been changed

public String editLastName(String lastName);
//pre: none
//post: customer last name has been changed

public String editTelephoneNum(String telephoneNum);
//pre: none
//post: customer phone number has been changed

public String editEmail(String email);
//pre: none
//post: customer email has been changed

public String editPassword(String password);
//pre: none
//post: customer password has been changed
```

5 Low Level Design

Entity Class: CustomerAccountCollection

Responsibilities	Collaborators	Attributes	Operations
To maintain information regarding registered customers of the system.	- CustomerAccount - CustomerUI	- userId - email - password - firstName - lastName - CustomerAccountColl	addUser - adds a new user into CustomerAccount collection checkCustomerAccount - check for the existence of userName isExist - check if there is a manager account with the associated email in the collection AddManager - add a new manager account into the Manager AccountCollection deleteManager - remove an existing manager account from the ManagerccountCollection getAccount - retrieve the ManagerAccount associated with the userID from the collection getSize - calculate the number of manager account in the collection getArray - return the array of all managers account in the system

5 Low Level Design

<u>CustomerAccountCollection</u>

Attributes:

Array CustomerAccountCollection [];
// stored in alphabetical order by Customer account userName;
String userName;
String email;
String password;
String firstName;
String lastName;

Methods:

public void addUser((String FirstName, String LastName, String phone, String email, String password);
// pre: none
// post: inserts essential customer account information

public bool checkCustomerAccount();
// pre: none
// post: returns true/false regarding the existence of the customer account

public ManagerAccount getArray();
// pre: none
// post: return the array of all customer accounts in the system

Entity Class: CustomerQueue

Responsibilities	**Collaborators**	**Attributes**	**Operations**
To maintain the order of the customers requesting to speak with a CSR or manager	- CustomerAccount - CSRAccount - ManagerAccount	- QueueSize (same as numberInQueue)	isEmpty - checks to see if the queue is empty of customers enqueue - adds a customer into the queue for chat request dequeue - pops a customer off the queue to speak with a CSR or Manager

5 Low Level Design

CustomerQueue

Attributes:

int QueueSize;
String UserId;

Methods:

public bool isEmpty();
// pre: none
// post: returns true/false stating whether the queue is empty or not

public void enqueue(String UserId);
// pre: none
// post: enters a customer with their corresponding userId into the chat queue

public String dequeue()
// pre: queue is not empty
// post: pops off a customer off of the chat queue

CSRSubsystem

Entity Class: CSRUI

Responsibilities	Collaborators	Attributes	Operations
To provide a registered CSR with an interface in order to access the system	- CustomerAccountCollection - CSRAccountCollection	- userId - password - name - phone - email - userClass	Login - log in the system with userId and password Register - a new CSR account will be created. ViewCustomerAccount - get all the registered customers and display DeleteCustomer - delete the registered customer UpdateCustomer - update a list of the registered customer

5 Low Level Design

CSRUI

Attributes:

String userId;
String password;
String name;
String phone;
String email;
int userClass;

Methods:

public void Login(String userId, String password)
// pre: none
// post: CSR account with the associated userID and password is connected to the server

public void Register(string userId, string password, string name, string phone, string email, int userClass)
// pre: the userId doesn't exit in the collection of user account
// post: register a new CSR in the collection of user account

public void ViewCustomerAccount()
// pre: none
// post: display all customer account

public void DeleteCustomer(string userId)
// pre: customer userID exits in the collection of user account
// post: the chosen customer account has been deleted from the collection of user account

public void UpdateCustomer(string userId)
// pre: customer userId exits in the collection of user account
// post: the chosen customer account has been updated from the collection of user account

5 Low Level Design

Entity Class: CSRAccount

Responsibilities	**Collaborators**	**Attributes**	**Operations**
To maintain a registered CSR's information	- CustomerQueue - CSRAccountCollection	- CSRId - password	getCSRId - get CSR Id getPassword - get CSR password getName - get CSR name getEmail - get CSR email getPhone - get CSR phone editPassword - change CSR password editName - change CSR name editPhone - change CSR phone editEmail - checkCSR - authorize the CSR with a valid CSRId and password.

<u>CSRAccount</u>
Attributes:

String userId;
String password;
String name;
String email;
String phone;

CSRAccount (Continued…)

Methods:

```
public string getCSRId()
// pre: none
// post: return CSR userId

public string getPassword()
// pre: none
// post: return CSR password

public string getName()
// pre: none
// post: return the name of CSR

public string getPhone()
// pre: none
// post: return CSR phone

public string getEmail()
// pre: none
// post: return CSR email

public string editPassword()
// pre: CSR usrID already exits in the collection of CSR
// post: return changed CSR password

public string editName()
// pre: CSR usrID already exits in the collection of CSR
// post: return changed CSR name

public void editPhone()
// pre: CSR usrID already exits in the collection of CSR
// post: return changed CSR phone

public void editEmail()
// pre: CSR usrID already exits in the collection of CSR
// post: return changed email

public void checkCSR()
// pre: none
// post: authenticate if the given userId and password match in the CSR account collection
```

5 Low Level Design

Entity Class: CSRAccountCollection

Responsibilities	Collaborators	Attributes	Operations
To maintain information about registered CSR of the system.	- CSRAccount - CSRUI	- CSRId	addCSR - add a new CSR into the CSRAccountCollection deleteCSR - delete a new CSR from the CSRAccountCollection getCSRAccount - retrieve the CSR account associated with CSR userId from the collection of CSR account checkAccount - check whether the CSRId exists or not

<u>CSRAccountCollection</u>

Attributes:

```
Array CSRAccountArray[]    // stored in alphabetical order by CSR account name
String userId;
String password;
String name, phone, email;
int userClass;             // 1-customer, 2-CSR, 3-manager
```

Methods:

public void AddCSR(string userId, string password, string name, string phone, string email, int userClass)
// pre: CSR account with the given userId doesn't exit in the collection of CSR account.
// post: a new CSR account has been added to the collection of all CSR

public void DeleteCSR(string userId)
// pre: CSR account with the given userId exits in the collection of CSR
// post: a new CSR account has been deleted from the collection of all CSR

public CSRAccount getCSRAccount(string userId)
// pre: CSR account with the given userId exits in the collection of CSR
// post: a chosen CSR account has been returned.

public void checkAccount()
// pre: none
// post: return true if a CSR account with the given userId exits in the collection

5 Low Level Design

Entity Class: CustomerQueue

Responsibilities	Collaborators	Attributes	Operations
To maintain information about customers in queue.	- CSRAccount	- customerId	getCustomer - get next requested customer from the queue. getNumCustomer - get the number of customer in the queue. enqueue - add a customer in queue. dequeue - delete a customer from queue. checkQueue - return true if a customer in queue, otherwise return false.

<u>CustomerQueue</u>

Attributes:

String userId;

Methods:

public string getCustomer(userId)
// pre: none
// post: get the customer with the given userId.

public int getNumCustomer()
// pre: none
// post: get the number of customers in the queue.

public void enqueue(string userId)
// pre: none
// post: enqueue a new customer in the queue.

public void dequeue(string userId)
// pre: none
// post: dequeue an existed customer in the queue.

public boolean checkQueue()
// pre: none
// post: return true if a customer in queue, otherwise return false.

5 Low Level Design

ManagerSubsystem

Entity Class: ManagerUI

Responsibilities	Collaborators	Attributes	Operations
To provide the manager with an interface in order to access the system	- ServerConnection	- emailAddress - password - customerAccount - csrAccount - managerAccount - theChat - theLog	Login - log in the system selectCustomer -select a customer from a chat queue startChat - invoke a chat session endChat - end a chat session addChatComment - add a comment to a chat session deleteChatComment - delete comment of a chat session addCustomerComment - add a comment to a customer account editCustomerComment - edit comment in a customer account viewCustomerComment - view comment in a customer account deleteCustomerAccount - remove customeraccount from the customeraccountcollection editCustomerAccount - change customer account information

Entity Class: ManagerUI (Continued…)

Responsibilities	Collaborators	Attributes	Operations
To provide the manager with an interface in order to access the system	- ServerConnection	- emailAddress - password customerAccount - csrAccount managerAccount - theChat - theLog	viewChatLog - open a chat log createCSRAccount - create a new CSR account and add it to the CSRAccountCollection deleteCSRAccount -remove CSR account from the csraccountcollection editCSRAccount - change CSR account information createManagerAccount - add a new manager account to manageraccountcollection editManagerAccount - change manager account information deleteManagerAccount - remove manager account from the manageraccountcollection monitorChat - join an ongoing chat generateReport - create a statistic report and open it viewOnlineCSR - open the list of CSR that is currently online

5 Low Level Design

ManagerUI

Attributes:

```
String emailAddress;
String password;
String customerAccount;
String csrAccount;
String customerAccount;
String csrAccount;
Chat theChat;
Log theLog;
```

Methods:

```
public void createCSRAccount(CSRAccount csrAccount);
//pre: none
//post: csrAccount is created and added to the csrAccountCollection

public void deleteCSRAccount(CSRAccount csrAccount);
//pre: none
//post: csrAccount is deleted from the csrAccountCollection

public void editCSRAccount(CSRAccount csrAccount)
//pre: none
//post: csrAccount is changed.

 public void createManagerAccount(ManagerAccount managerAccount);
//pre: none
//post: managerAccount is created and added to the managerAccountCollection

public void editManagerAccount(ManagerAccount managerAccount)
//pre: none
//post: managerAccount is changed.

public void deleteManagerAccount(ManagerAccount managerAccount)
//pre: none
//post: managerAccount is deleted from the managerAccountCollection

public String generateReport();
//pre: chats exist
//post: return the statistic report of all chats

public void monitorChat(Chat theChat);
//pre: chat is ongoing
//post: manager join a chat session in invisible mode
```

ManagerUI (Continued…)

Methods:

public CSRAccountarray viewOnlineCSR();
//pre: none
//post: return the array of CSR that are currently active in the system

---

This method is inherited from CSR class (according to the use case diagram)

public void login(string emailAddress, string password);
//pre: none
//post: manager account with the associated email is connected to the server

public void selectCustomer();
//pre: none
//post: a customer from the chat queue is selected

public void startChat();
//pre: none
//post: a chat session in invoked

public void endChat();
//pre: none
//post: a chat session terminated

public void addChatComment(Chat theChat, String theComment);
//pre: none
//post: comment is added to the log of a chat

public void deleteChatComment(Chat theChat);
//pre: none
//post: comment is deleted from the log of a chat

public void addCustomerComment(Chat theChat, String theComment);
//pre: none
//post: comment is added to the log of a chat

public void editCustomerComment(Chat theChat);
//pre: none
//post: comment is deleted from the log of a chat

public string viewCustomerComment(Chat theChat);
//pre: none
//post: return the comment of both CSR and customer on a particular chat

ManagerUI (Continued…)

Methods:

```
public string editCustomerAccount(CustomerAccount customerAccount);
//pre: none
//post: customerAccount is changed
public void deleteCustomerAccount(CustomerAccount customerAccount);
//pre: none
//post: customerAccount is deleted from the customerAccountCollection

public string viewChatLog(Log theLog);
//pre: none
//post: return the chat log.
```

Entity Class: ManagerAccount

Responsibilities	Collaborators	Attributes	Operations
To maintain a manager information	- ManagerAccountColl	- userID - firstName - lastName - telephoneNum - emailAddress - password	ManagerAccount - a constructor to create a new manager account getUserID - get manager userID getFirstName - get manager first name getLastName - get manager first name getTelephoneNum - get manager phone number getEmailAddress - get manager email address getPassword - get manager password editUserID - change manager user ID editFirstName - change manager first name

5 Low Level Design

Entity Class: ManagerAccount

Responsibilities	Collaborators	Attributes	Operations
To maintain a manager information	- ManagerAccountColl	- userID - firstName - lastName - telephoneNum - emailAddress - password	getLastName - get manager first name getTelephoneNum - get manager phone number getEmailAddress - get manager email address getPassword - get manager password editUserID - change manager user ID editFirstName - change manager first name editLastName - change manager last name editTelephoneNum - change manager phone number editEmailAddress - change manager email editPassword - change manager password checkManager - authorize the manager with a valid userID and password

<u>ManagerAccount</u>
Attributes:

String userID;
String firstName;
String lastName;
String telephoneNum;

ManagerAccount (Continued…)

Attributes:

String emailAddress;
String password;

Methods:

public ManagerAccount(String UserID, String firstName, String lastName, String telephoneNum, String emailAddress, String password)
//pre**:** none
//post: a new manager account is returned

public String getUserID();
//pre: none
//post: manager userID has been returned

public String getFirstName();
//pre: none
//post: manager first name has been returned

public String getLastName();
//pre: none
//post: manager last name has been returned

public String getTelephoneNum();
//pre: none
//post: manager phone number has been returned

public String getEmailAddress();
//pre: none
//post: manager email address has been returned

public String getPassword();
//pre: none
//post: manager password has been returned

public String editUserID(String userID);
//pre: none
//post: manager user name has been changed

public String editFirstName(String firstName);
//pre: none
//post: manager first name has been changed

ManagerAccount (Continued…)

Methods:

```
public String editLastName(String lastName);
//pre: none
//post: manager last name has been changed

public String editTelephoneNum(String telephoneNum);
//pre: none
//post: manager phone number has been changed

public String editEmail(String email);
//pre: none
//post: manager email has been changed

public String editPassword(String password);
//pre: none
//post: manager password has been changed

public String checkManager(string emailAddress, string password);
//pre: none
//post: authenticate if the given password match the password of the manager account
associated with the given email
```

5 Low Level Design

Entity Class: ManagerAccountCollection

Responsibilities	Collaborators	Attributes	Operations
To maintain information about registered manager of the system.	- ServerConnection	- ManagerAccountArray	ManagerAccountCollection - check if there is a manager account in the collection isEmpty - check if there is a manager account in the collection isExist - check if there is a manager account with the associated email in the collection AddManager - add a new manager account into the Manager AccountCollection deleteManager - remove an existing manager account from the ManagerccountCollection getAccount - retrieve the ManagerAccount associated with the userID from the collection getSize - calculate the number of manager account in the collection getArray - return the array of all managers account in the system

5 Low Level Design

CSRAccountCollection

Attributes:

ManagerAccount ManagerAccountArray [ ]
//Stored in alphabetical order by manager account name

Methods:

public ManagerAccountCollection()
//pre: none
//post: returns a new empty ManagerAccountCollection

public boolean isEmpty();
//pre: none
//post: returns true if there are no manager account in the collection

public boolean isExist(String userID)
//pre: none
//post: returns true if a manager account with the given userID exist in the //collection

public void addAccount(ManagerAccount newAccount)
//pre: none
//post: a new manager account has been added to the collection of all //manager

public void deleteAccount(String userID)
//pre: manager account with the given userID already exist in the collection
//post: the chosen manager account has been deleted from the collection //of all manager

public ManagerAccount getAccount(String userID)
//pre: a manager with the given userID exist in the collection
// post: the chosen manager account has been returned

public int getSize()
//pre: none
//post: the number of manager account in Manager account Collection has been returned

public ManagerAccountArray getArray()
//pre: none
//post: return the array of all manager account in the system

5 Low Level Design

Database System

The database structure will be created as specified by the following standard SQL queries. The database name is 'chatsystem' in the MySQL database.

Users table specifies all users in the system, and the role type is identified by the class column. The class can have a value of 1,2 or 3. Where 1 = Customer, 2 = CSR, and 3 = Manager.

```
CREATE TABLE Users (
  userid       char(16),
  password     char(16),
  lastname     char(20),
  firstname    char(20),
  email        char(40),
  phone        char(16),
  class        tinyint,
  PRIMARY KEY  (userid)
);
```

The ChatLog contains each individual chat session identified by a Chat Session Number (CSN)

```
CREATE TABLE ChatLog (
  CSN          int unsigned NOT NULL auto_increment,
  datetime     timestamp,
  custuserid   char(16),
  csruserid    char(16),
  duration     smallint unsigned,
  waittime     smallint unsigned,
  custrating   tinyint(1),
  custcomment  varchar(255),
  csrcomment   varchar(255),
  PRIMARY KEY  (CSN)
);
```

All lines of chat in the system are recorded as entries in the ChatMessages table, which are identified by a unique combination of CSN and line values.

```
CREATE TABLE ChatMessages (
  CSN          int unsigned,
  line         int unsigned NOT NULL,
  userid       char(16),
  datetime     timestamp,
  textmessage  varchar(255)
);
```

5 Low Level Design

The following subsystems are done entirely with PHP code on a website

Customer Registration Subsystem

Attributes:

userid
password
confirmPassword
lastname
firstname
email
phone

Methods:

Submit
// the form will post the information and the php code will update the database with the new data and insert a new user of class = 1 (Customer). The user will be notified if he provided an inappropriate password or email or phone number.
// pre: All fields are filled, password and confirmPassword are equal
// post: If the userid is not already used in the database, then the new account is added. else, the user will be displayed a notification that the userid is already in use.

Login Subsystem

Attributes:

userid
password

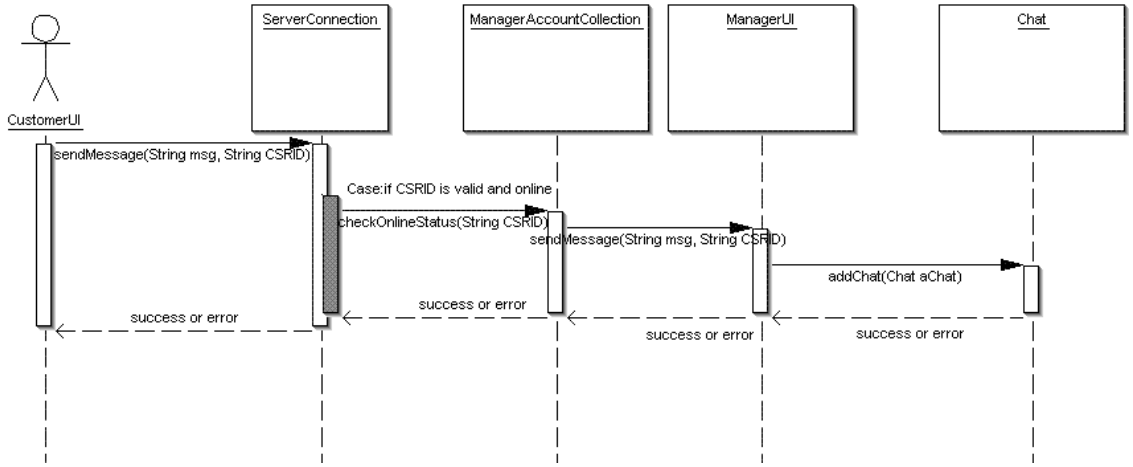
Methods:

Submit
// the form will post the information and the php code will query the database to determine if the userid and password is matches. If it does, then the class of the matching tuple will determine which applet is sent to the client.
// pre: All fields are filled
// post: If the user name and password is correct, the appropriate Applet will be launched on the clients machine.

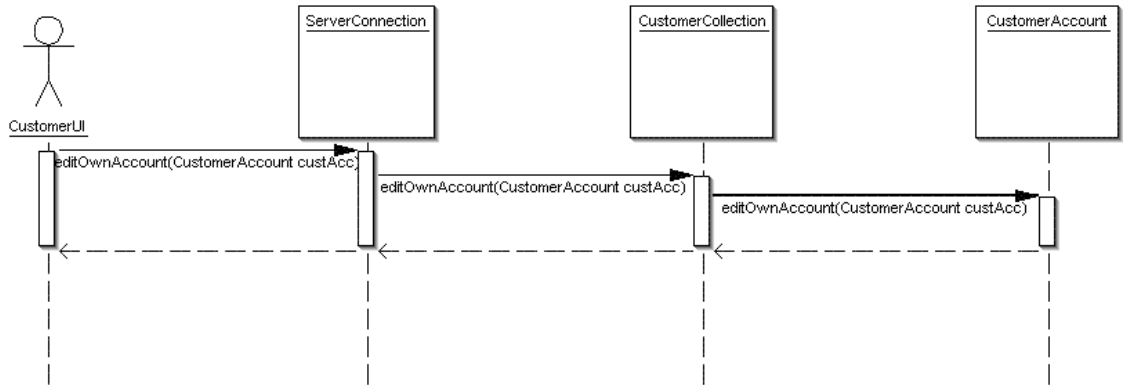
5 Low Level Design

Customer Sequence Diagram

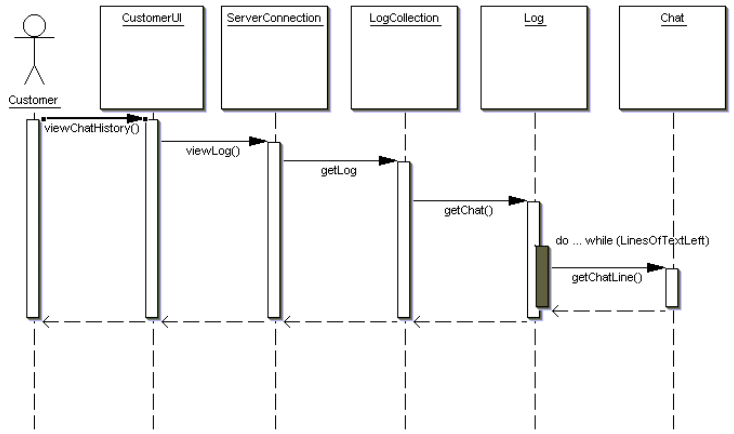
- chat with CSR



- edit own account information

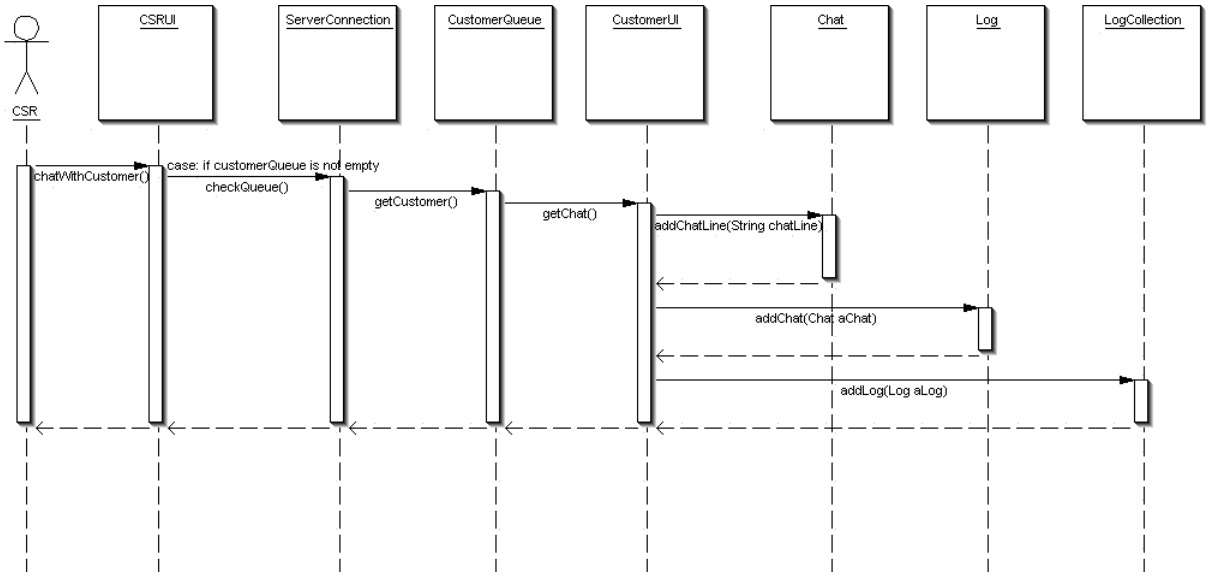


- view own chat log



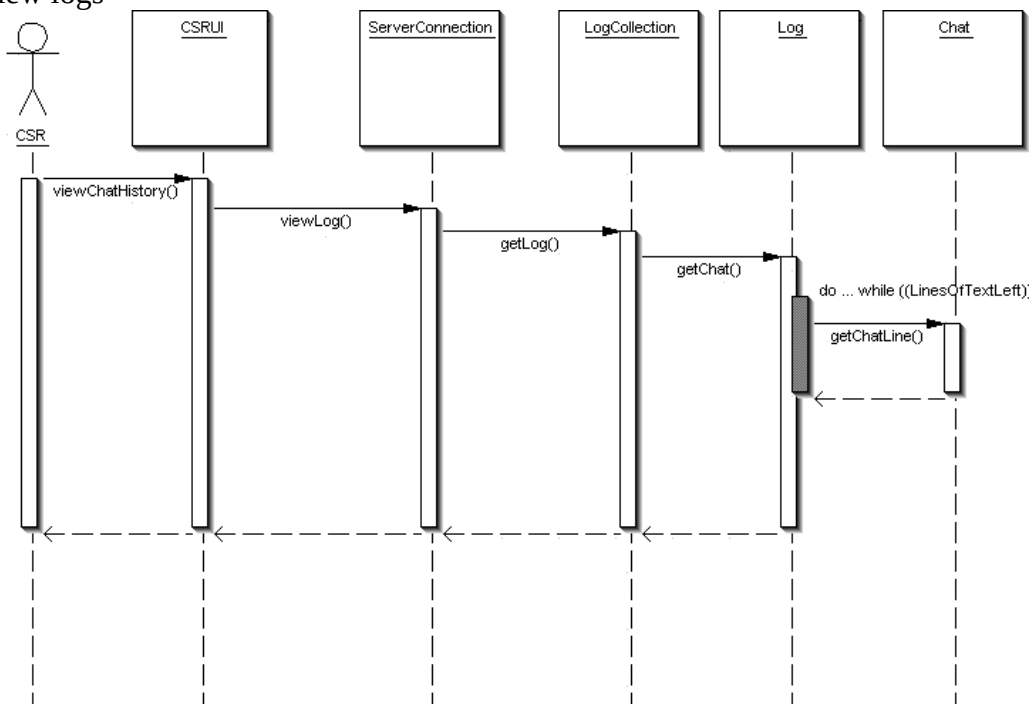
CSR Sequence Diagram

- chat with customer

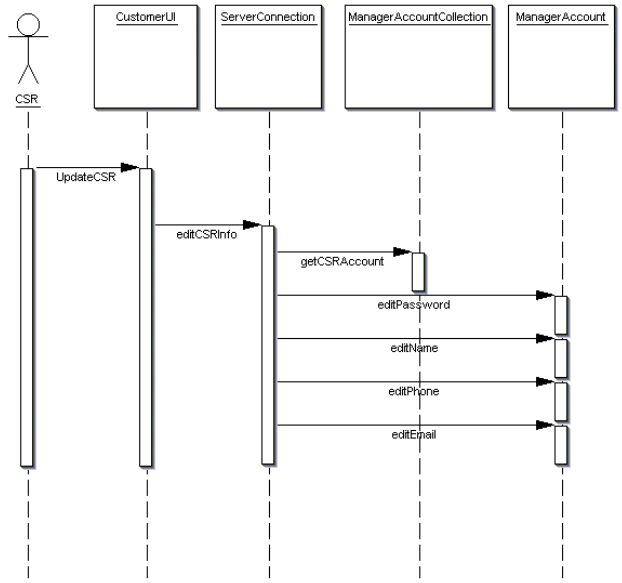


5 Low Level Design

- view logs



- edit own account

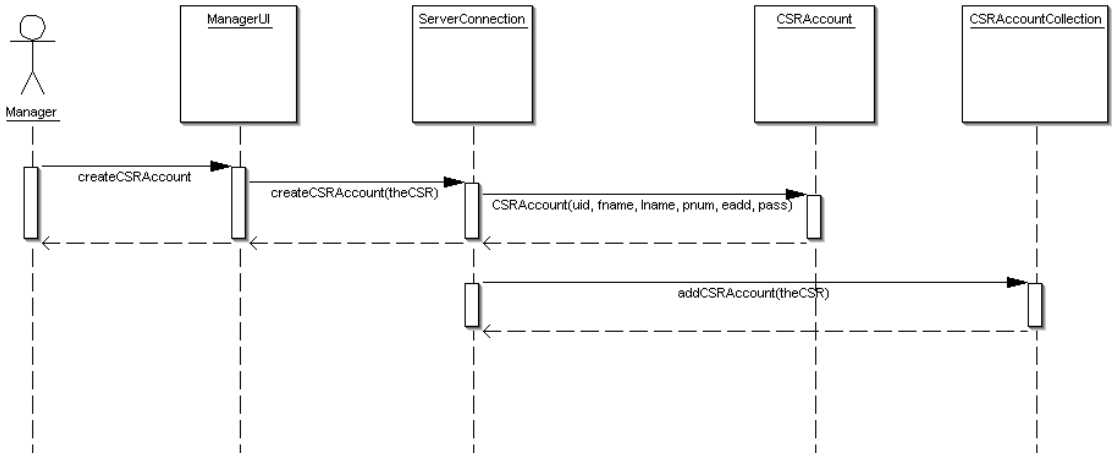


5 Low Level Design

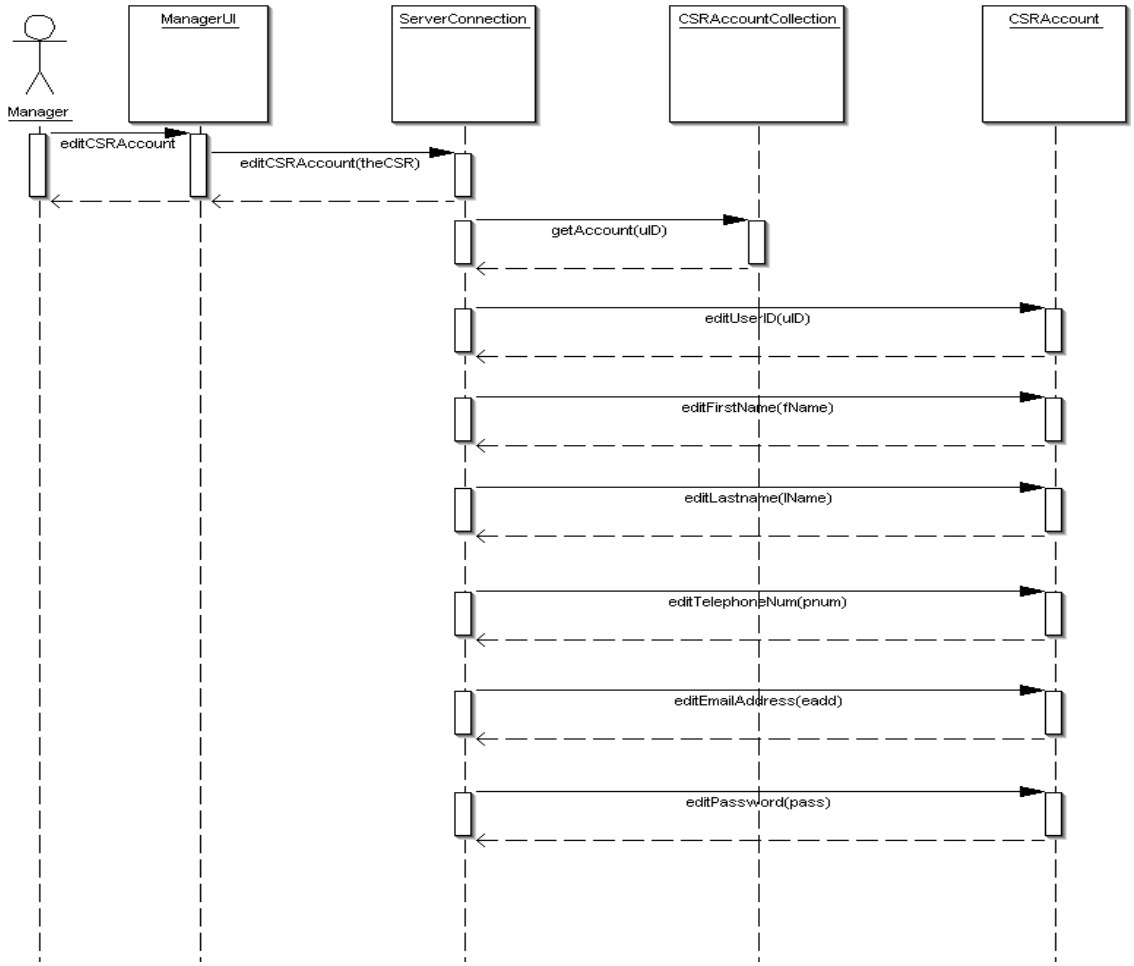
Manager Sequence Diagram

- create/edit/delete CSR

. create CSR

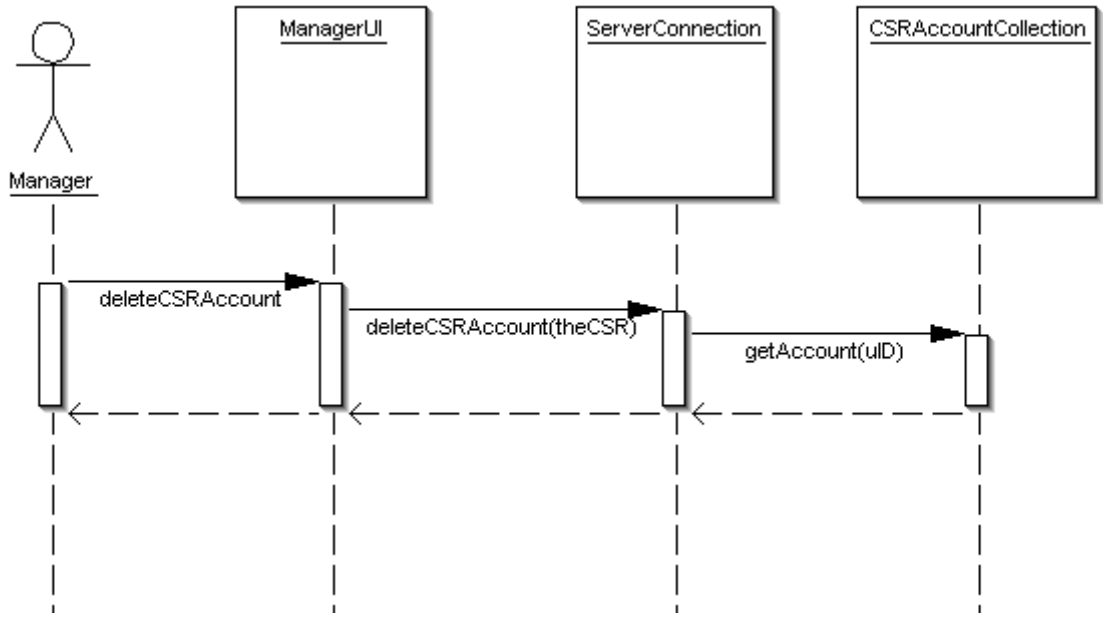


. edit CSR



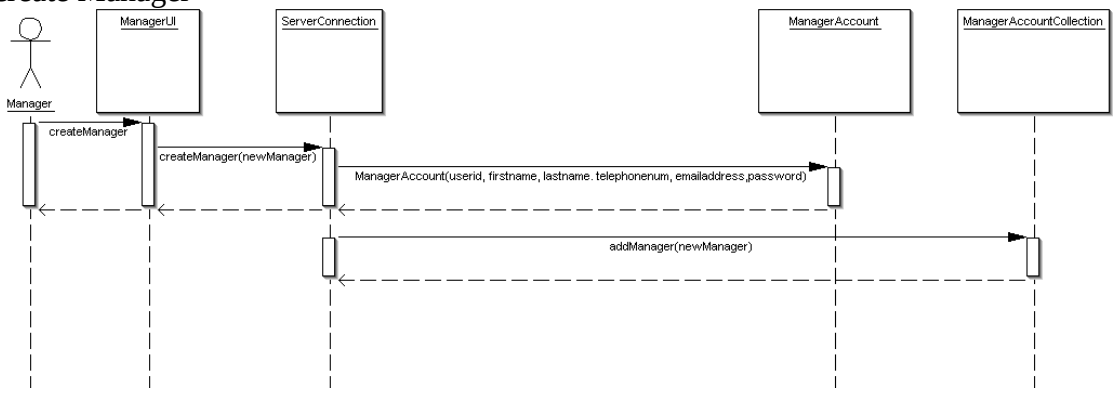
5 Low Level Design

. delete CSR



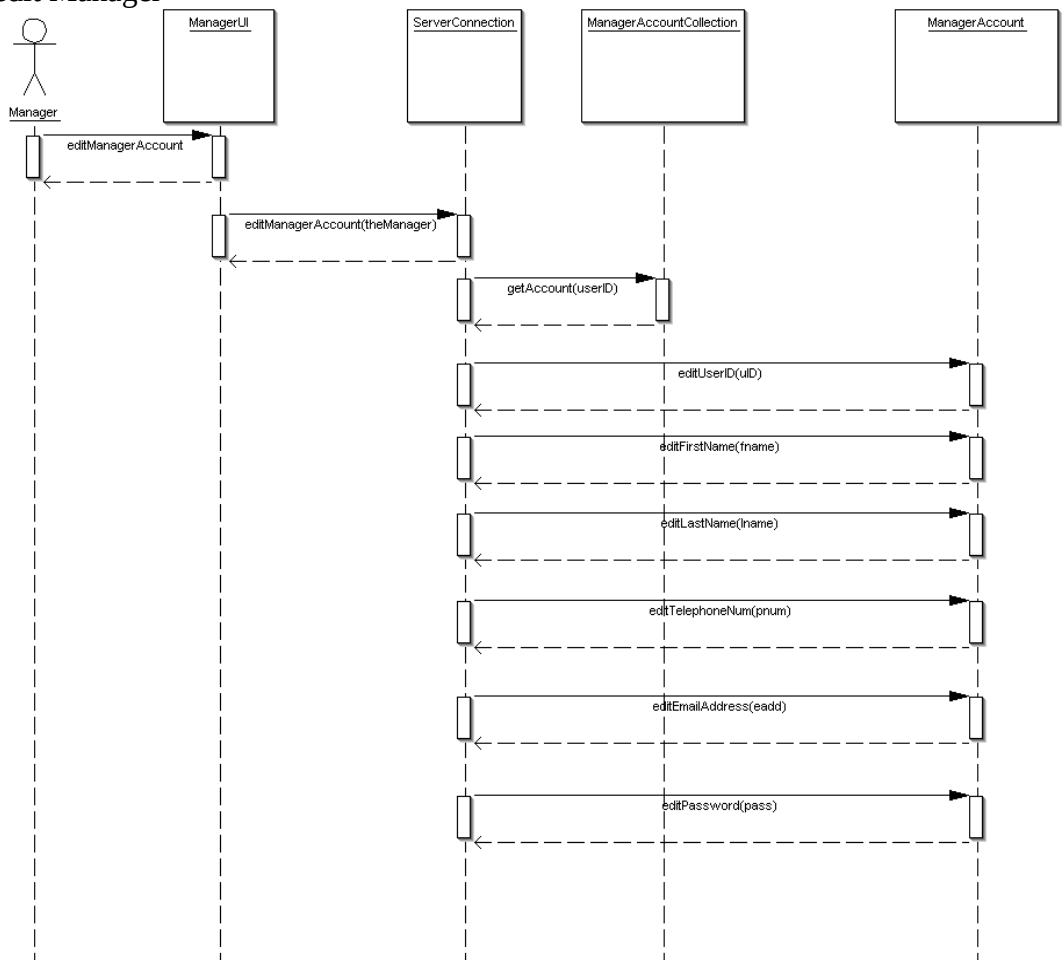
- create/edit/delete Manager

. create Manager

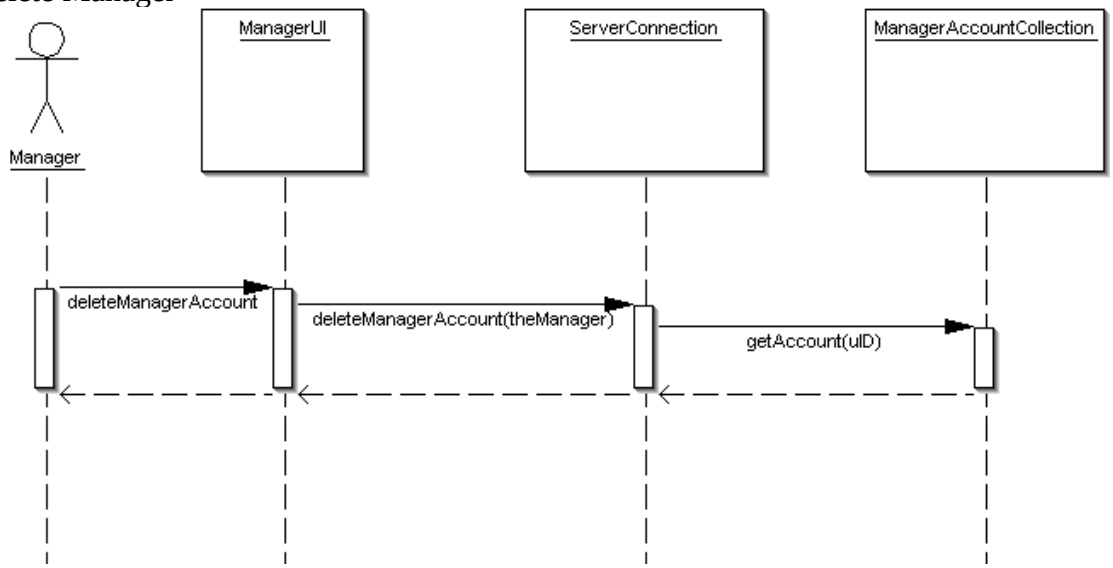


5 Low Level Design

. edit Manager

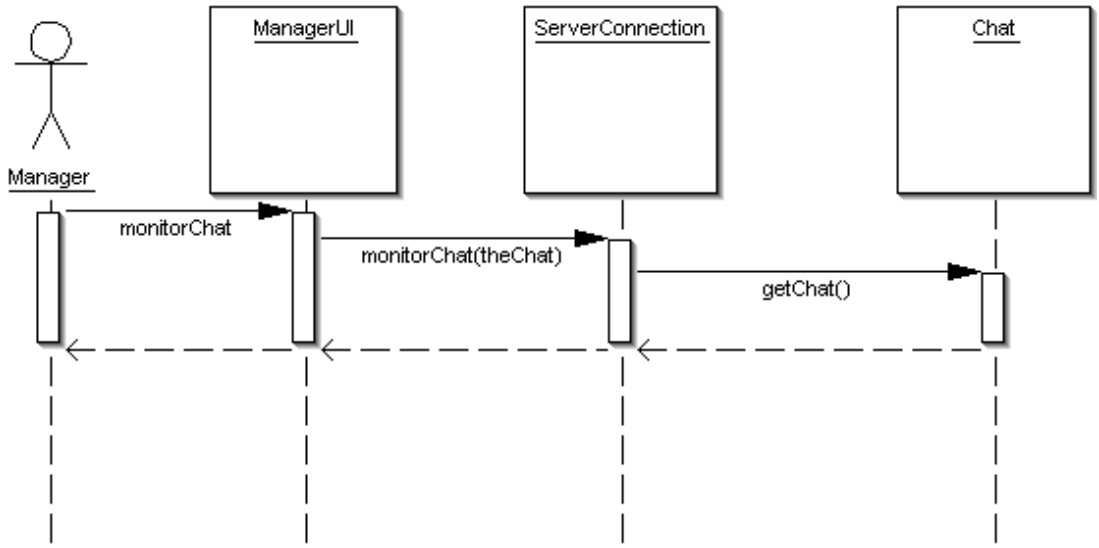


. delete Manager

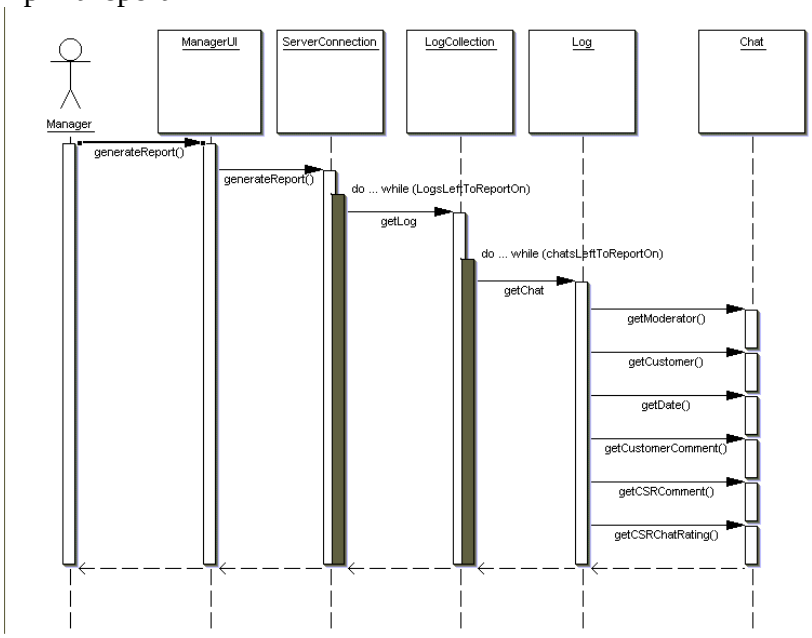


5 Low Level Design

- monitor chat



- print report



6 User Interface Design

6.1 Application Control

The Application will be designed with ease of use for end user in mind. With this view it was decided that all user classes will share the same basic design with differing functionality. So the starting login webpage will be shared by all users. The Chat application will be divided into three GUIs according to class the user belongs to (Customer/CSR/Manager).

All three GUIs will share the same menu system:

--File Menu
- -Customer :: Quit
- -CSR :: Accept Chat, Quit
- -Manager :: Accept Chat, Monitor Chat, Quit

-Chat History
- -Customer :: View own Chat Logs
- -CSR :: View own or Customer (enter userid) Chat logs
- -Manager :: View any Chat logs (enter userid)

-Account Management
- -Customer :: View/Edit own Account information
- -CSR :: View/Edit own or Customer (enter userid) Account information
- -Manager :: View/Edit (enter userid) any Account Information

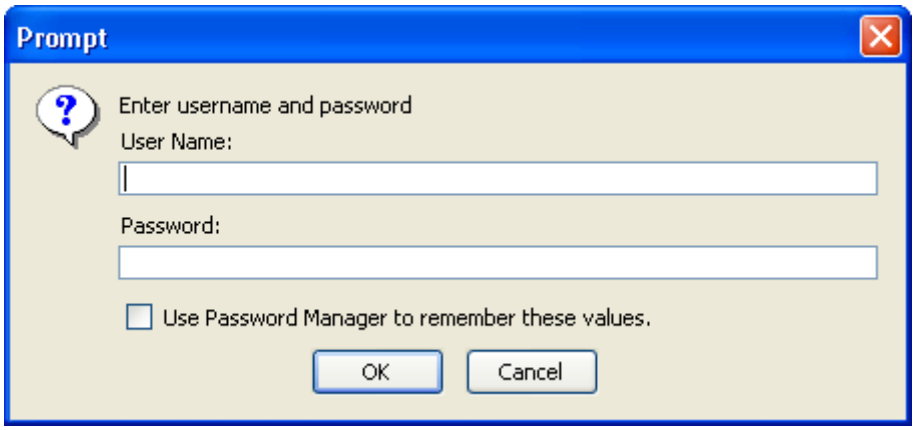
Beneath the Menu a Chat Control Panel will be provided. This panel will be shared by all three GUIs. It will contain a text box for message, Send and LogOut buttons.

Under The Chat Control Panel the main messaging window will be located. For CSR and Managerial GUIs this window will contain an additional panel on left side. The additional panel will provide a list of user in queue (CSR/Manager) and list of ongoing chats with ability to monitor (Manager). The Customers GUI will provide only the messaging window.

6.2 Screen 1..n

Login GUI:

-any of the three classes of users and enters their user name and password. The server will determine the class user belongs to and send appropriate GUI.

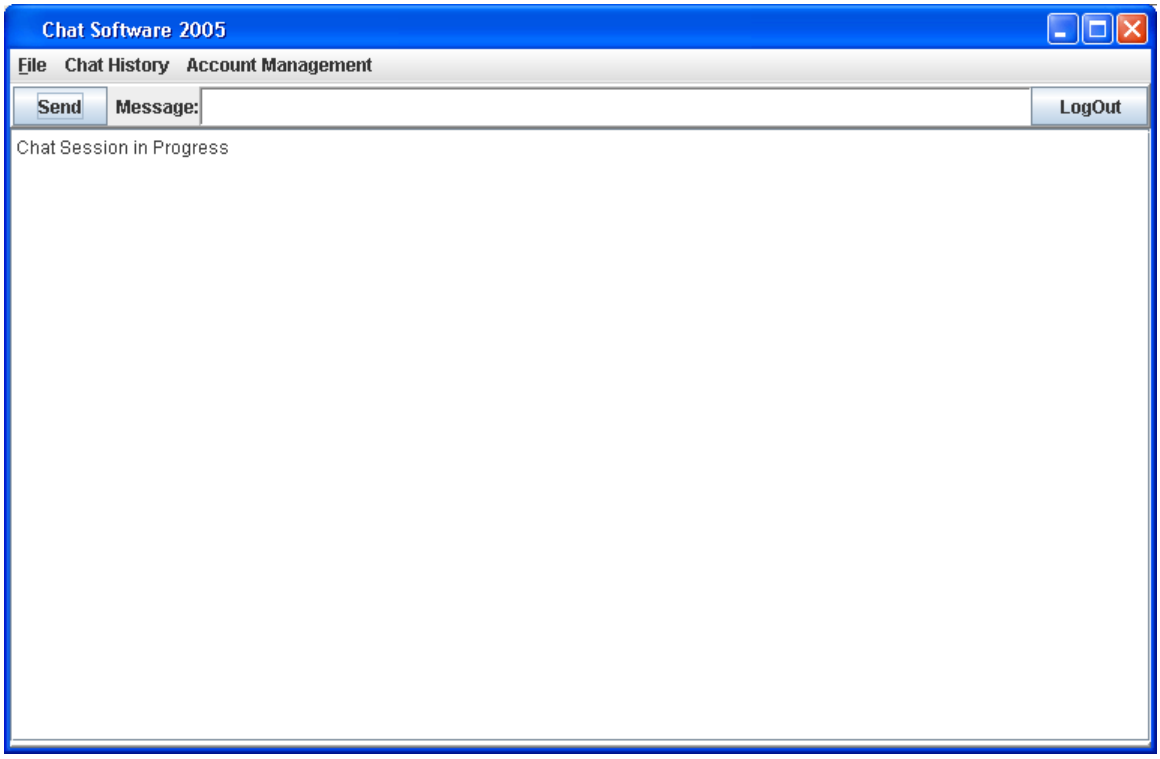


Upon successful authentication one of the following GUIs will be displayed:

6 User Interface Design

Customer GUI:

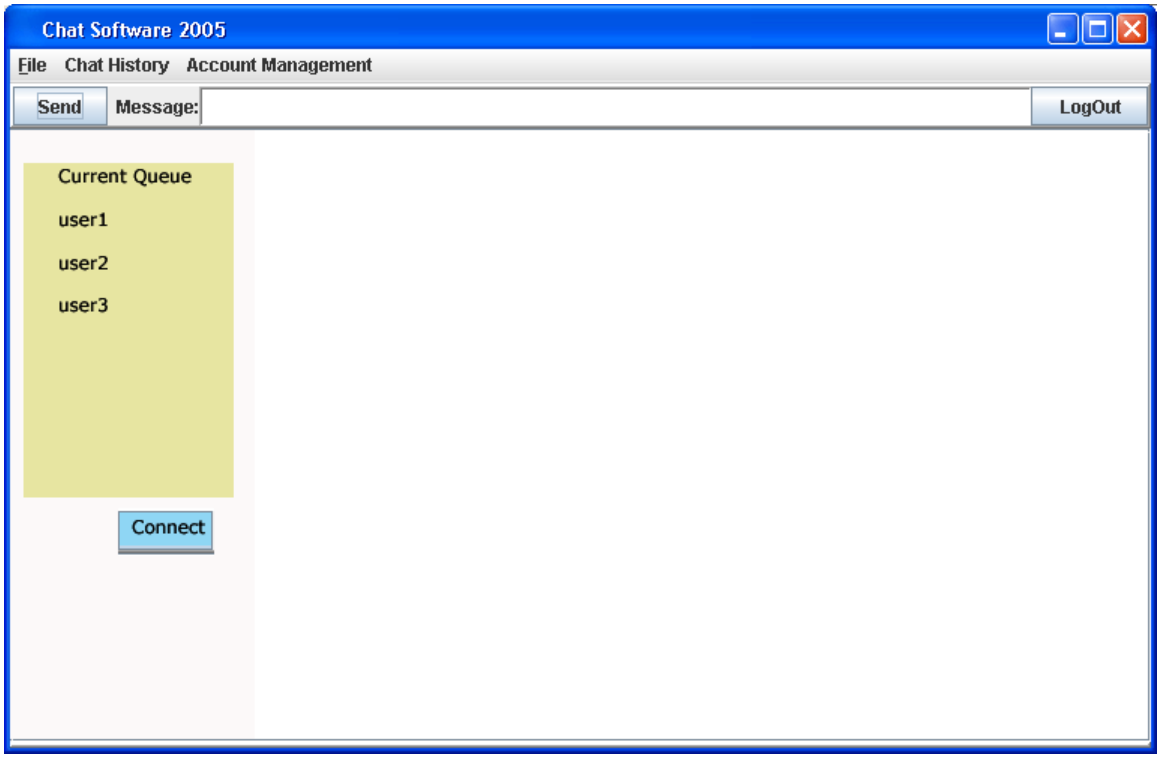
-basic chat functionality is provided
-the customer has the ability to view/edit Customer's own account information (Account Management Menu)
-can also view their own Chat Logs (Chat History Menu)



6 User Interface Design

CSR GUI:

- in addition to regular Customer user interface the CSR GUI provides the ability to monitor ongoing Customer Queue and initiate a chat with a customer (next in queue) (click Connect), the ability to edit a Customer's Account (Account Management Menu and enter userid)



6 User Interface Design

Manager GUI:

-in addition to entire functionality provided by the CSR User Interface, the manager has the ability to monitor live chat conducted by other CSRs, and the manager can edit both CSR and Customer accounts.

-to monitor chat a current chat session between a customer and CSR is selected and monitor clicked.

-to edit CSR/customer accounts the account management menu is selected and user name entered.