

**R.V. COLLEGE OF ENGINEERING  
BANGALORE**

**DEPARTMENT OF COMPUTER SCIENCE  
AND ENGINEERING**

---

**INTERNET PROGRAMMING LAB  
PROJECT REPORT**

---

**PERLCHAT – A CGI APPLICATION  
FOR HTML BASED CHAT**

**DEVELOPED BY**

**SRIVAS N. CHENNU 1RV98CS086**

**VISHWAS N. 1RV98CS104**

**R.V. COLLEGE OF ENGINEERING  
BANGALORE**

**DEPARTMENT OF COMPUTER SCIENCE  
AND ENGINEERING**

---

**CERTIFICATE**

---

This is to certify that

**Srivasa N. Chennu (1RV98CS086)**

studying in 7th semester Comp. Sc. and Engg. has satisfactorily  
completed the project work on

**Web Development using CGI/Perl**

in partial fulfillment of the Internet Programming Lab syllabus, as  
prescribed by the VTU for the academic year 2001-2002.

INTERNAL EXAMINER

EXTERNAL EXAMINER

PROF. B. I. KHODANPUR

H.O.D. CSE Dept.

---

# TABLE OF CONTENTS

---

1	Introduction.....	5
1.1	The Web And Html .....	5
1.2	The Common Gateway Interface ....	6
1.3	Forms .....	6
1.4	Gateways .....	7
1.5	Cgi Internals.....	7
2	Software Requirements Specification	8
3	Synopsis.....	9
4	Design .....	10
4.1	The User Interface .....	11
	The Client Browser .....	12
4.2	The Web Server .....	13
4.3	The Perlchat Scripts .....	13
	Perlchat Object Modules.....	14
4.4	The Perlchat Database.....	15
5	Implementation.....	18
5.1	Source Code.....	19
	Channel.Pm.....	19
	Channel.Pm.....	23

User.Pm.....	27
Message.Pm.....	29
Privatemessage.Pm.....	30
Login.Pl.....	31
Perlchat.Pl.....	33
6      Testing.....	40
7      Conclusion.....	40
The End .....	40

---

# 1 INTRODUCTION

---

PerlChat is a web application, which utilizes the functionality, provided by the Common Gateway Interface or CGI to provide users with a fast and easy-to-use chat based interaction facility. PerlChat uses HTML based interaction sequences to provide its users with chat rooms which they can subscribe to, and after proper authentication, can chat with other users currently in the channel via instantly delivered messages. Given below is a brief introduction to the concepts involved in web development.

## 1.1 THE WEB AND HTML

World Wide Web (WWW) programming deals with the development of hypertext document interaction mechanisms, which provide the client with a rich and intuitive interface to the information that he or she desires to view. Web development heavily utilizes the functionality of the Hypertext Markup Language, commonly known as HTML. HTML is a simple scripting language that is interpreted within a web browser. It provides functionality to identify and specify how information is presented to the user. Some of the important features of HTML that make it ideal for online representation of information are –

- ❑ Easy of Use – HTML constructs are very easy to comprehend, and can be used effectively by anybody.
- ❑ Machine Independence – The methodology used by HTML to mark up information is independent of its representation on a particular hardware or software architecture.
- ❑ Standardization – HTML syntax is a worldwide standard, developed by the W3C
- ❑ Flexible – HTML has been extended in many forms to provide additional functionality.

## 1.2 THE COMMON GATEWAY INTERFACE

The WWW works due to the process by which web servers generate and provide HTML encoded information, as and when requested by client browsers. CGI is the part of the Web server that can communicate with other programs running on the server. With CGI, the Web server can call up a program, while passing user-specific data to the program such as what host the user is connecting from, or input the user has supplied using HTML form syntax etc. The program then processes that data and the server passes the program's response back to the Web browser. CGI turns the Web from a simple collection of static hypermedia documents into a whole new interactive medium, in which users can ask questions and run applications.

## 1.3 FORMS

One of the most prominent uses of CGI is in processing forms. Forms are a subset of HTML that allows the user to supply information. The forms interface makes Web browsing an interactive process for the user and the provider.

A number of graphical widgets are available for form creation, such as radio buttons, text fields, checkboxes, and selection lists. When the user completes the form, the Submit button is used to send the information to the server, which executes the program associated with the particular form to "decode" the data.

Generally, forms are used for two main purposes. At their simplest, forms can be used to collect information from the user. But they can also be used in a more complex manner to provide back-and-forth interaction. For example, the user can be presented with a form listing the various documents available on the server, as well as an option to search for particular information within these documents. A CGI program can process this information and return document(s) that match the user's selection criteria.

## 1.4 GATEWAYS

Web gateways are programs or scripts used to access information that is not directly readable by the client. CGI provides a solution to the problem of representing encoded information in the form of a gateway. A CGI program serves as a gateway to a collection of information like a database, and provides a convenient view of the desired data.

Virtual, or dynamic, document creation is at the heart of CGI. Virtual documents are created on the fly in response to a user's information request. Programs that use a combination of graphics libraries, gateways, and forms can be used to create complex virtual documents.

## 1.5 CGI INTERNALS

Most servers expect CGI programs and scripts to reside in a special directory, usually called **cgi-bin**, and/or to have a certain file extension. When a user opens a URL associated with a CGI program, the client sends a request to the server asking for the file.

For the most part, the request for a CGI program looks the same as it does for all Web documents. The difference is that when a server recognizes that the address being requested is a CGI program, the server does not return the file contents verbatim. In case of CGI programs, the server understands that it should execute the program instead of relaying it directly to the browser. The client request to the server specifies the CGI script name, the request method, the accepted data formats etc. The server executes the CGI script with input being derived from the standard input or environment variables, depending on the request method. The CGI program produces output in the form of dynamically generated HTML content, which is received by the server. The server parses this output for relevant information and then passes it to the client browser for display.

---

## 2 SOFTWARE REQUIREMENTS SPECIFICATION

---

Listed below are some of the software requirements of the PerlChat application.

- It should be implemented using CGI/Perl as the scripting language.
- It should be platform independent.
- It should be browser independent.
- It should incorporate a portable design.
- It should deliver rich and well structured content to the user.
- It should lightweight in its implementation.

---

## 3 SYNOPSIS

---

PerlChat provides HTML based chat room functionality to its users. It implements the following functionality.

- Users access the PerlChat facility by first signing up for the service. Once users have registered themselves, they are assigned a username and a password.
- PerlChat provides its users with a large collection of chat rooms on a variety of themes, and the user can add one or more chat rooms to his or her personal favorites.
- Using the assigned password, the user can access a collection of his or her favorite chat rooms.
- Once a user enters a chat room he can send interact with other users currently in the chat room, via instant messages.
- PerlChat keeps track of all the users currently logged into the service. It ensures password secrecy, so that other users cannot access a user's password or profile information.

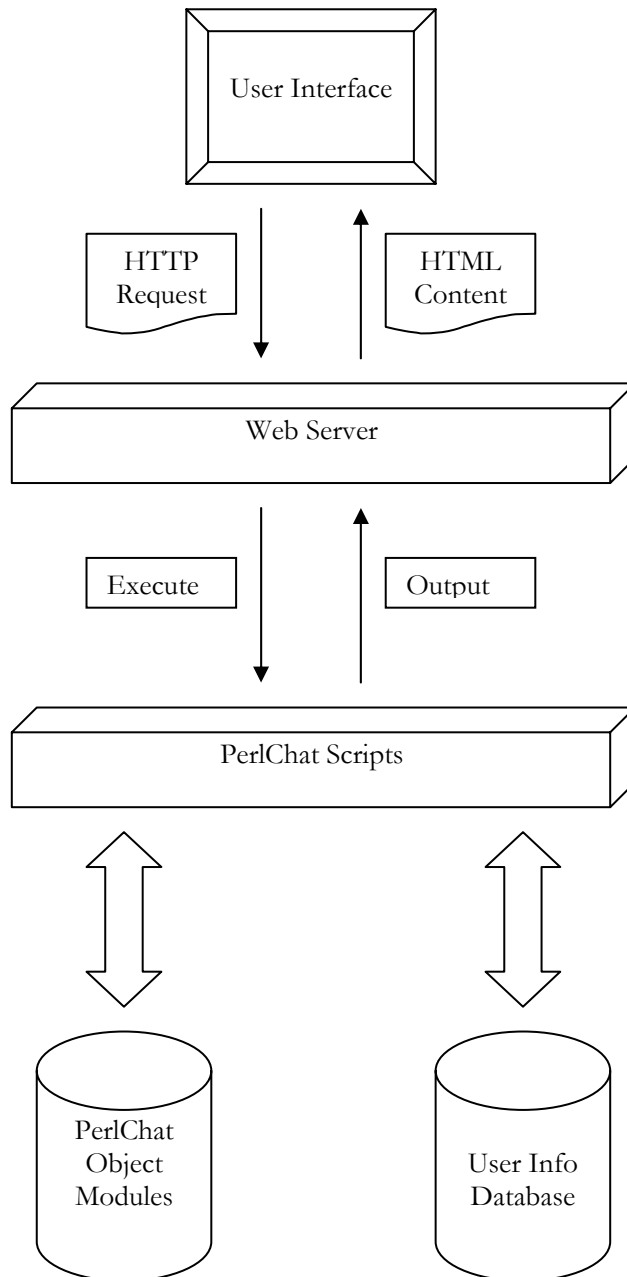
PerlChat has been implemented using Perl, short for Practical Extraction and Reporting Language. It uses the facilities of a CGI enabled web server to generate dynamic content, which is viewed by the user. Also, it provides for user authentication, and maintains information about its users in a database. This allows for customized content to be delivered to the user.

---

## 4 DESIGN

---

The architecture of the PerlChat system is illustrated in the block diagram below.



PerlChat System Architecture

## 4.1 THE USER INTERFACE

Listed below are some of the HTML pages served to the PerlChat user.

- ❑ **LOGIN** – This page provides the user with a mechanism to login into the PerlChat using his username and password. It also provides him with an option to sign up with PerlChat if he does not have a username and password, through the signup page.
- ❑ **SIGNUP** – This page is accessed when a potential user wants to sign up for PerlChat. It is basically a form which requires the user to fill in important details like his personal particulars, and a suitable username and password.
- ❑ **FAVORITES** – This page allows the user to select the chat rooms he or she would like to add to his or her favorites list. This list will be provided to the user when he or she logs into PerlChat.
- ❑ **WELCOME** – The welcome page is shown immediately after the user successfully logs into the system. Here, the list of favorite chat rooms is provided, with a link on which the user can click on, to enter the corresponding room. Also provided, is a menu bar where he or she can alter his PerlChat account profile and other settings.
- ❑ **CHATROOM** – This page is shown once the user has entered a chat room of his or her choice. It contains the list of currently posted messages, private messages to the user, and a list of all the users currently in the chat room. The user can now chat with other users by sending them instant messages.
- ❑ **PROFILE** – This page allows the user to edit his or her personal profile, modify the list of favorite chatrooms, create new rooms etc.
- ❑ **LOGOUT** – When the user logs out of the PerlChat system, this page is shown, indicating that he or she has successfully closed the PerlChat session.

## THE CLIENT BROWSER

The client browser displays the HTML content served by the PerlChat CGI scripts. The HTML browser sends HTTP requests to the web server, which contain the information regarding the requested document, Browser information, accepted MIME types etc.

The web server responds the query and sends a HTTP response back to the browser, along with a status code, and the requested document.

PerlChat supports most of the commonly used browsers. These are listed below.

- ✓ Microsoft Internet Explorer 4.0 and above.
- ✓ Netscape Navigator 3.0 and Mozilla
- ✓ Opera 4.0 and above
- ✓ Konqueror

PerlChat requires the browser to provide the following features.

- ❑ Graphical display
- ❑ Frame based HTML formatting
- ❑ Embedded Perlscript processing.
- ❑ Client side Cookies

## 4.2 THE WEB SERVER

PerlChat requires a web server which supports the following constructs.

- ❑ Full support for the HTTP protocol
- ❑ Server Side Includes
- ❑ Support for CGI gateways

The web server used for developing the PerlChat system is the Apache HTTP server. It is one of the most popular web servers used on the Internet and supports a multitude of features other than the ones listed above.

## 4.3 THE PERLCHAT SCRIPTS

The PerlChat system has been coded using the Perl scripting language. The advantages of Perl include -

- ❑ It is highly portable and readily available.
- ❑ It contains extremely powerful string manipulation operators, as well as functions to deal with binary data.
- ❑ It contains very simple and concise constructs.
- ❑ It makes calling shell commands very easy, and provides some useful equivalents of certain UNIX system functions.
- ❑ There are numerous extensions built on top of Perl for specialized functions.

Because of these overwhelming advantages, Perl is one of the most popular languages for developing web scripts.

The PerlChat scripts are accessed by the HTML pages listed above. These scripts generate the dynamic HTML content viewed by the user. For doing so, the scripts rely on a collection of objects, which are a part of the PerlChat object module library.

## PERLCHAT OBJECT MODULES

The design and implementation of the PerlChat system is Object Oriented in nature. The design ensures that data abstraction and encapsulation is inherent in the functioning of PerlChat. The PerlChat object modules contain the class definitions for the objects used in the PerlChat system. The classes which define the functionality of the PerlChat objects are-

- ❑ **Channels** – This class contains an aggregate collection of all the rooms in the PerlChat system, along with methods to manipulate the collection.
- ❑ **Channel** – This class encapsulates the functionality of a single chatroom. It in turn contains a collection of all the users currently in the chatroom, and the public and private messages posted in the chatroom.
- ❑ **User** – This class encapsulates the properties of a PerlChat user. It contains his username, current chatroom information etc.
- ❑ **Message** – This class represents a public message posted in a chatroom. It contains information regarding the sender and content of the message.
- ❑ **PrivateMessage** – This class represents a private message sent from one user to another, both of who are in the same chatroom. It inherits from the **Message** class, and additionally contains information regarding the recipient of the message.

## 4.4 THE PERLCHAT DATABASE

The PerlChat system uses a database to store the following information about its users.

- ❑ The unique username and the password.
- ❑ The user profile, including his username full name, date of birth and contact information.
- ❑ Demographic information about the user.
- ❑ The complete list of chatrooms in PerlChat.
- ❑ The list of favorite chatrooms for every user.

The **Relation Schema** for the PerlChat database is shown below.

Shown below is the **Entity – Relationship** Diagram for the PerlChat System.

PerlChat employs **PostgreSQL** as the DBMS of choice for implementing the database described above. A brief introduction to the PostgreSQL system is given below.

The PostgreSQL RDBMS is a Object Relational Database Management system. It is been developed and improved for the last 15 years, and is a full-fledged industry-strength DBMS, deployed in a large number of web-based solutions.

Some of the important features offered by PostgreSQL are given below.

- ✓ Fully ACID compliant.
- ✓ Highly robust and secure.
- ✓ Full fledged Transaction support.
- ✓ Stored Procedures.
- ✓ Update, insert and delete triggers.
- ✓ Object oriented databases.
- ✓ Advanced locking systems.
- ✓ Concurrency management under multi-user, mutli-transaction environments.
- ✓ Sub-queries.
- ✓ Server-side cursors.
- ✓ Query caching.
- ✓ Locking of databases.
- ✓ Better table join supports.

---

## 5 IMPLEMENTATION

---

The tables description for the PerlChat database is given below.

```
create table profiles (  
    userid text,  
    fname text not null,  
    mname text,  
    lname text not null,  
    bdate text,  
    gender text,  
    addr text,  
    city text,  
    zip text,  
    phone text,  
    email text not null,  
    job text,  
    primary key(userid)  
);  
  
create table passwords (  
    userid text,  
    passwd text not null,  
    primary key(userid),  
    foreign key(userid) references profiles(userid)  
);  
  
create table rooms (  
    rname text,  
    primary key(rname)  
);  
  
create table favorites (  
    userid text,  
    rname text,  
    primary key(userid, rname),  
    foreign key(userid) references passwords(userid),  
    foreign key(rname) references rooms(rname)  
);
```

## 5.1 SOURCE CODE

Detailed below are the PerlChat object library modules.

### CHANNEL.PM

```

package Channels;

$channels = undef;
$DEFAULT_URL = "http://localhost";
$SCRIPT_ALIAS = "http://localhost/cgi-bin/";
$MAX_USERS_PER_CHANNEL = 15;
$MAX_MESSAGES_PER_CHANNEL = 20;
$MAX_PVT_MESSAGES_PER_CHANNEL = 10;
$MAX_LATENT_TIME = 5;

@EXPORT = qw($MAX_LATENT_TIME);

sub new {
    my $class = shift;
    my $self = {};
    $self->{CHANNELS} = [];
    bless($self, $class);
    return $self;
}

sub addChannel {
    my $self = shift;
    if (@_) {
        push(@{$self->{CHANNELS}}, shift);
    }
    return $self;
}

sub getChannel {
    my $self = shift;
    my $name = shift;
    my @channels = @{$self->{CHANNELS}};
    my $count = @channels;
    my $i = 0;
    while ($i < $count) {
        if ($channels[$i]->getName eq $name) {
            return $channels[$i];
        }
        $i++;
    }
    return 0;
}

```

```

sub removeChannel {
    my $self = shift;
    my $name = shift;
    my @channels = @{$self->{CHANNELS}};
    my $count = @channels;
    my $i = 0;
    while ($i < $count) {
        if ($channels[$i]->getName eq $name) {
            splice(@channels, $i, 1);
            return;
        }
        $i++;
    }
}

sub getChannelCount {
    my $self = shift;
    my @channels = @{$self->{CHANNELS}};
    my $count = @channels;
    return $count;
}

sub getUser {
    my $self = shift;
    my $name = shift;
    my @channels = @{$self->{CHANNELS}};
    my $count = @channels;
    my $i = 0;
    my $user = 0;
    while ($i < $count) {
        $user = $channels[$i]->getUser($name);
        if ($user != 0) {
            return $user;
        }
        $i++;
    }
    return 0;
}

sub getUserCount {
    my $self = shift;
    my $user_count = 0;
    my @channels = @{$self->{CHANNELS}};
    my $count = @channels;
    my $i = 0;
    while ($i < $count) {
        $user_count += $channels[$i]->size;
        $i++;
    }
    return $user_count;
}

sub write {
    my $self = shift;

```

```

open DUMP, ">dump";

my @channels = @{$self->{CHANNELS}};
print DUMP scalar(@channels), "\n";

foreach my $channel (@channels) {

    print DUMP $channel->getName, "\n";

    my @users = $channel->getUsers;
    print DUMP scalar(@users), "\n";
    foreach my $user (@users) {
        print DUMP $user->getName, $user->getChannelName, $user->
        >getLastAction, "\n";
    }

    my @messages = $channel->getMessages;
    print DUMP scalar(@messages), "\n";
    foreach my $message (@messages) {
        print DUMP $message->getSender->getName, "\n";
        print DUMP $message->getContent, "\n";
    }

    my @pmessages = $channel->getAllPrivateMessages;
    print DUMP scalar(@pmessages), "\n";
    foreach my $pmessage (@pmessages) {
        print DUMP $pmessage->getSender->getName, "\n";
        print DUMP $pmessage->getContent, "\n";
        print DUMP $pmessage->getRecipient->getName, "\n";
    }
}
close DUMP;
}

sub read {
    open DUMP, "<dump";
    my $channel_count = <DUMP>;
    chomp $channel_count;
    $channel_count = atoi($channel_count);
    my @channels = @{$self->{CHANNELS}};

    for (my $i = 0; $i < $channel_count; $i++) {

        my $channel_name = <DUMP>;
        chomp $channel_name;
        my $channel = Channel->new($channel_name);

        my @users = $channel->getUsers;
        my $user_count = <DUMP>;
        chomp $user_count;
        $user_count = atoi($user_count);
        for (my $j = 0; $j < $user_count; $j++) {
            my $user_name = <DUMP>;
            chomp $user_name;
            my $user = User->new($user_name);

```

```

my $chname = <DUMP>;
chomp $chname;
$user->setChannelName($chname);
my $last_action = <DUMP>;
chomp $last_action;
$last_action = atoi($last_action);
$user->setLastAction($last_action);
push(@users, $user);
}
$channel->setUsers(@users);

my @messages = $channel->getMessages;
my $message_count = <DUMP>;
chomp $message_count;
$message_count = atoi($message_count);
for (my $j = 0; $j < $message_count; $j++) {
    my $sender_name = <DUMP>;
    chomp $sender_name;
    my $content = <DUMP>;
    chomp $content;
    my $message = Message->new($channel->getUser($sender_name),
$content);
    push(@messages, $message);
}
$channel->setMessages(@messages);

my @pmessages = $channel->getAllPrivateMessages;
my $pmessage_count = <DUMP>;
chomp $pmessage_count;
$pmessage_count = atoi($pmessage_count);
for (my $j = 0; $j < $pmessage_count; $j++) {
    my $sender_name = <DUMP>;
    chomp $sender_name;
    my $content = <DUMP>;
    chomp $content;
    my $recipient_name = <DUMP>;
    chomp $recipient_name;
    my $pmessage = PrivateMessage->new($channel->getUser($sender_name),
$content, $channel->getUser($recipient_name));
    push(@pmessages, $pmessage);
}
$channel->setPrivateMessages(@pmessages);

push(@channels, $channel);
}
@{$self->{CHANNELS}} = @channels;
}
1;

```

## CHANNEL.PM

```

package Channel;
use Channels;

sub new {
    my $class = shift;
    my $self = {};
    $self->{NAME} = shift;
    $self->{USERS} = [];
    $self->{MESSAGES} = [];
    $self->{PVT_MESSAGES} = [];
    bless($self, $class);
    return $self;
}

sub hasMaxUsers {
    my $self = shift;
    my $flag = 0;
    my @users = @{$self->{USERS}};
    my $count = @users;
    if ($count == $Channels::MAX_USERS_PER_CHANNEL) {
        $flag = 1;
    }
    return $flag;
}

sub hasMaxMessages {
    my $self = shift;
    my $flag = 0;
    my @messages = @{$self->{MESSAGES}};
    my $count = @messages;
    if ($count == $Channels::MAX_MESSAGES_PER_CHANNEL) {
        $flag = 1;
    }
    return $flag;
}

sub hasMaxPvtMessages {
    my $self = shift;
    my $flag = 0;
    my @pvtmsgs = @{$self->{PVT_MESSAGES}};
    my $count = @pvtmsgs;
    if ($count == $Channels::MAX_PVT_MESSAGES_PER_CHANNEL) {
        $flag = 1;
    }
    return $flag;
}

sub addMessage {
    my $self = shift;

```

```

$self->refreshUsers;
if ($self->hasMaxMessages) {
    shift(@{$self->{MESSAGES}});
}
push(@{$self->{MESSAGES}}, shift);
}

sub addPrivateMessage {
    my $self = shift;
    if ($self->hasMaxPvtMessages) {
        shift(@{$self->{PVT_MESSAGES}});
    }
    push(@{$self->{PVT_MESSAGES}}, shift);
}

sub getMessages {
    my $self = shift;
    $self->refreshUsers;
    return @{$self->{MESSAGES}};
}

sub setMessages {
    my $self = shift;
    @{$self->{MESSAGES}} = @_;
}

sub getAllPrivateMessages {
    my $self = shift;
    return @{$self->{PVT_MESSAGES}};
}

sub setPrivateMessages {
    my $self = shift;
    @{$self->{PRIVATE_MESSAGES}} = @_;
}

sub getPrivateMessages {
    my $self = shift;
    my $user = shift;
    my @pvtmsg = [];
    $self->refreshUsers;
    my @pvtmsgs = @{$self->{PVT_MESSAGES}};
    my $pmcount = @pvtmsgs;
    my $i = 0;
    while($i < $pmcount) {
        $pmref = $pvtmsgs[$i];
        $rcpt = $pmref->getRecipient;
        $send = $pmref->getSender;
        if ($user->getId == $rcpt->getId || $user->getId == $send->getId) {
            push(@pvtmsg, $pmref);
            if ($user->getId == $rcpt->getId) {
                splice(@{$self->{PVT_MESSAGES}}, $i, 1);
            }
        }
        $i++;
    }
}

```

```

    }
    return @pvtmsg;
}

sub removePrivateMessages {
    my $self = shift;
    my $user = shift;
    my $removed = 0;
    my @pvtmsgs = @{$self->{PVT_MESSAGES}};
    my $pmcount = @pvtmsgs;
    my $i = 0;
    while($i < $pmcount) {
        $pmref = $pvtmsgs[$i];
        $rcpt = $pmref->getRecipient;
        if ($user->getId == $rcpt->getId) {
            splice(@{$self->{PVT_MESSAGES}}, $i, 1);
            $removed++;
        }
        $i++;
    }
    return $removed;
}

sub removeAllPrivateMessages {
    my $self = shift;
    splice(@{$self->{PVT_MESSAGES}}, 0);
}

sub addUser {
    my $self = shift;
    my $user = shift;
    if ($self->hasMaxUsers) {
        die("Room is full");
    }
    push(@{$self->{USERS}}, $user);
}

sub removeUser {
    my $self = shift;
    my $user = shift;
    my @users = @{$self->{USERS}};
    my $usercount = @users;
    my $i = 0;
    while ($i < $usercount) {
        if ($users[$i]->getName eq $user->getName) {
            splice(@{$self->{USERS}}, $i, 1);
        }
        $i++;
    }
}

sub getUsers {
    my $self = shift;
    $self->refreshUsers;
    return @{$self->{USERS}};
}

```

```

}

sub setUsers {
    my $self = shift;
    @{$self->{USERS}} = @_;
}

sub logout {
    my $self = shift;
    my $user = shift;
    my $default = $Channels::channels->getChannel("default");

    $self->removeUser($user);
    $default->addUser($user);
    $user->setChannelName("default");
}

sub refreshUsers {
    my $self = shift;
    my @users = @{$self->{USERS}};
    my $usercount = @users;
    my $i = 0;
    while ($i < $usercount) {
        if (!$users[$i]->refresh) {
            addMessage(Message->new($users[$i], "REMOVED DUE TO INACTIVITY"));
            $self->logout($users[$i]);
        }
        $i++;
    }
}

sub size {
    my $self = shift;
    my @users = @{$self->{USERS}};
    my $usercount = @users;
    return $usercount;
}

sub getUser {
    my $self = shift;
    my $user = shift;
    my @users = @{$self->{USERS}};
    my $usercount = @users;
    my $i = 0;
    while ($i < $usercount) {
        if ($users[$i]->getName eq $user) {
            return $users[$i];
        }
        $i++;
    }
    return 0;
}

sub getName {
    my $self = shift;

```

```

    return $self->{NAME};
}

sub save {
    my $self = shift;
    open DUMP, ">>dump";
    print DUMP $self->{NAME}, "\n";
    close DUMP;
    my @users = @{$self->{USERS}};
    my $ucount = @users;
    my @messages = @{$self->{MESSAGES}};
    my $mcount = @messages;
    my @private_messages = @{$self->{PRIVATE_MESSAGES}};
    my $pcount = @private_messages;
    my $i = 0;
    while ($i < $ucount) {
        $users[$i]->save;
        $i++;
    }
    $i = 0;
    while ($i < $mcount) {
        $messages[$i]->save;
        $i++;
    }
    $i = 0;
    while ($i < $pcount) {
        $private_messages[$i]->save;
        $i++;
    }
}
1;

```

## USER.PM

```

package PerlChat::User;

sub new {
    $class = shift;
    my $self = {};
    $self->{NAME} = shift;
    $self->{CHANNEL} = "default";
    $self->{LAST_ACTION} = time;
}

sub refresh {
    my $self = shift;
    $current_action = time;
    if (((current_action - $self->{LAST_ACTION}) / 60) > $MAX_LATENT_TIME)
    {
        return 0;
    }
    else {

```

```
    return 1;
}
}

sub getName {
    my $self = shift;
    return $self->{NAME};
}

sub getChannelName {
    my $self = shift;
    return $self->{CHANNEL};
}

sub setChannelName {
    my $self = shift;
    $self->{CHANNEL} = shift;
}

sub getLastAction {
    my $self = shift;
    return $self->{LAST_ACTION};
}

sub setLastAction {
    my $self = shift;
    $self->{LAST_ACTION} = shift;
}

sub save {
    my $self = shift;
    open DUMP, ">>dump";
    print DUMP $self->{NAME};
    print DUMP $self->{CHANNEL};
    print DUMP $self->{LAST_ACTION};
    close DUMP;
}

sub read {
    my $self = shift;
    1;
}
```

**MESSAGE.PM**

```
package Message;

sub new {
    my $class = shift;
    my $self = {};
    $self->{SENDER} = shift;
    $self->{CONTENT} = shift;
    bless($self, $class);
    return $self;
}

sub getSender {
    my $self = shift;
    return $self->{SENDER};
}

sub getSenderId {
    my $self = shift;
    return $self->{SENDER}->getId;
}

sub getSenderName {
    my $self = shift;
    my $sender = $self->{SENDER};
    return $sender->getName;
}

sub getContent {
    my $self = shift;
    return $self->{CONTENT};
}

sub save {
    my $self = shift;
    $self->{SENDER}->save;
    open(DUMP, ">>dump");
    print DUMP $self->{CONTENT};
    close(DUMP);
}
1;
```

**PRIVATEMESSAGE.PM**

```
package PrivateMessage;
use Message;
@ISA = ("Message");

sub new {
    my $class = shift;
    my $sender = shift;
    my $content = shift;
    my $self = $class->SUPER::new($sender, $content);
    $self->{RECIPIENT} = shift;
    bless($self, $class);
    return $self;
}

sub getRecipient {
    my $self = shift;
    return $self->{RECIPIENT};
}

sub getRecipientName {
    my $self = shift;
    return ${$self->{RECIPIENT}}->getName;
}

sub save {
    my $self = shift;
    $self->SUPER::save;
    $self->{RECIPIENT}->save;
}
1;
```

Detailed below are the PerlChat scripts which use the PerlChat object library to generate dynamic HTML content. These scripts also access the PerlChat database to obtain the stored user information.

### LOGIN.PL

```
#!/usr/bin/perl

use CGI;
use DBI;
use Message;
use PrivateMessage;
use Channel;
use Channels;
use User;

$dbname = "perlchat";
$dbuser = "postgres";
$MAX_LATENT_TIME = 10;

$dbh = DBI->connect("dbi:Pg:dbname=$dbname", $dbuser, "", { RaiseError
=> 1, AutoCommit => 0 });

$Channels::channels = Channels->new;

$sth = $dbh->prepare("select rname from rooms");
$sth->execute();
while ( $row = $sth->fetchrow_arrayref ) {
    $Channels::channels->addChannel(Channel->new(${$row}[0]));
}

$q = new CGI;

print $q->header;

if ($q->request_method eq 'GET') {
    open(LOGIN, "<./html/Login.html") || die("File open error");
    while(<LOGIN>) {
        print;
    }
    close(LOGIN);
}
else {

    $userid = $q->param("userid");
    $password = $q->param("password");
    $sth = $dbh->prepare("select count(*) from passwords where userid='" .
$userid . "'");
    $sth->execute;
```

```

@row = $sth->fetchrow_array;
if ($row[0] == 0) {
    open(LOGINERR1, "<./html/Loginerr1.html") || die("File open error");
    while(<LOGINERR1>) {
        print;
    }
    exit 1;
}
$sth = $dbh->prepare("select passwd from passwords where userid=' " .
$userid . "'");
$sth->execute;
@row = $sth->fetchrow_array;
if ($row[0] ne $password) {
    open(LOGINERR2, "<./html/Loginerr2.html") || die("File open error");
    while(<LOGINERR2>) {
        print;
    }
    exit 2;
}

$user = User->new($userid);
$channel = $Channels::channels->getChannel("default");
$channel->addUser($user);

print << "MAIN";
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
<HEAD>
<TITLE>PerlChat Chat Rooms</TITLE>
</HEAD>
<BODY bgcolor=#FFFFFF>
<form name="jchat" method=post action="">
<table border=0 width="891" height="77">
<tr>
<td width="875" height="73">
    <p align="center"><font face="Verdana" size="2"><b>Select a Chat Room
to Enter</b></font></p>
<table border=0 cellspacing=0 cellpadding=0 width=100% height=262>
<tr>
<td align=center valign=middle height="216">
<table border=0 cellspacing=0 cellpadding=3 width=389 bgcolor=brown>
<tr>
<th width="391">
    <p align="center"><font size="-1" color="white" face="Verdana">Your
favorite Chat Rooms</font></p>
</th>
</tr>
<tr><td align=center valign=middle width="391"><table border=0
cellspacing=0 cellpadding=3 bgcolor=silver width=621 height="1">
MAIN

    $uname = $user->getName;
    $sth = $dbh->prepare("select rname from favorites where userid = ' " .
$uname . "'");
    $sth->execute;

```

```

while (@row = $sth->fetchrow_array) {
    $channel = $Channels::channels->getChannel($row[0]);
    $usercount = $channel->size;
    print "<tr>",
        "<td align=left width=\"605\" height=\"1\">",
        "<a href=\"" . $Channels::SCRIPT_ALIAS . "perlchat.pl?userid=" .
$username . "&channel=" . $row[0] . ">" . $row[0] . "</a></td>",
        "<td align=left width=\"405\" height=\"1\">",
        $usercount,
        " User(s)</td>",
        "</tr>\n",
    }

    print << "MAIN";
</table></td>
</tr>
</table>
<p>&nbsp;</p>
</td>
</tr>
</table>
</td>
</table>
</form>
</BODY>
MAIN

    $dbh->disconnect;
}
1;

```

## PERLCHAT.PL

```

#!/usr/bin/perl

use CGI;
use DBI;
use Message;
use PrivateMessage;
use Channel;
use Channels;
use User;

$DEFAULT_LOGIN_MESSAGE = "<b>HAS JOINED";
$DEFAULT_LOGOUT_MESSAGE = "<b>HAS LEFT";

$q = new CGI;

print $q->header;

if ($q->request_method eq 'GET') {

    $userid = $q->param("userid");
    $channel_name = $q->param("channel");

```

```

if ($userid == undef || $channel_name == undef) {
    print "hello";
    exit(1);
};

$default = $Channels::channels->getChannel("default");
$user = $default->getUser($userid);

if ($user == 0) {
    print "Your session has expired. Please re login.";
    exit(2);
}

$channel = $Channels::channels->getChannel($channel_name);
login($user, $channel);

show_header($user, $channel);
show_messages($user, $channel);
show_private_messages($user, $channel);
show_footer($user, $channel);
}

else {

    $logout = $q->param("btn_logout");
    $post = $q->param("btn_post");
    $userid = $q->param("str_user_name");
    $channel_name = $q->param("str_channel");
    $message = $q->param("str_message");
    $recipientid = $q->param("str_recipient_id");

    $user = $Channels::channels->getUser($userid);

    if ($user == undef) {
        print "The user name <i><b>" . $userid . "</b></i> is not logged in.
Please relogin.";
        exit(3);
    }

    $channel = $Channels::channels->getChannel($channel_name);

    if ($logout != undef) {
        logout($user, $channel);
        showmain($user);
        exit(0);
    }

    if ($user->refresh() == 0) {
        logout($user, $channel);
        print "You have been logged out due to prolonged latency.\nMaximum
allowable latency: ",
        $Channels::MAX_LATENT_TIME,
        " minute(s)";
        exit(4);
    }
}

```

```

if ($post != undef && ($message != undef && $message ne "")) ||
($message != undef && $message ne "") {
    if ($recipientid ne "ALL USERS") {
        $recipient = $channel->getUser($recipientid);
        $channel->addPrivateMessage(PrivateMessage->new($user, $message,
$recipient));
    }
    else {
        $channel->addMessage(Message->new($user, $message));
    }
}

show_header($user, $channel);
show_messages($user, $channel);
show_private_messages($user, $channel);
show_footer($user, $channel);
exit(0);
}

sub show_header {
    my $user = shift;
    my $channel = shift;

    print "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">",
        "<HTML>",
        "<HEAD>",
        "<TITLE>PerlChat: ", $user->getChannelName(), "</TITLE>",
        "<script language=Javascript>",
        "    function setFocus(){",
        "        document.jchat.str_message.focus();",
        "        return true;",
        "    }",
        "</script>",
        "</HEAD>",
        "<BODY bgcolor=white onload=\"return setFocus()\"><center>",
        "<form name=\"jchat\" method=post action=\" . $Channels::SCRIPT_ALIAS
. \"perlchat.pl>",
        "<input type=hidden name=\"str_user_name\" value=\"\" . $user->
getName . \">",
        "<input type=hidden name=\"str_channel\" value=\"\" . $user->
getChannelName . \">",
        "<font face=\"tahoma,verdana,arial\" size=-1><br>",
        "Be Sure To Logout When You Are Finished!";
}

sub show_footer {
    my $user = shift;
    my $channel = shift;
    my @users = $channel->getUsers;
    my $usercount = @users;

    print "<P><table border=1 cellpadding=3 cellspacing=0 width=550>",
        "<tr bgcolor=brown>",

```

```

    "<th colspan=2><font color=white face=\"tahoma,verdana,arial\"
size=-1>",
    "Control Panel: ", $user->getName, "@PerlChat.", $channel->getName,
"</th>",
    "</tr>",
    "<tr bgcolor=silver>",
    "<th width=450><font face=\"tahoma,verdana,arial\" size=-1>",
    "Message</th>",
    "<th width=100><font face=\"tahoma,verdana,arial\" size=-1>",
    "Send To</th>",
    "</tr>",
    "<tr>",
    "<td nowrap>",
    "<nobr><input type=text name=str_message size=45 maxlength=255>
<input type=submit name=btn_post value=\"Send\">",
    "</td>",
    "<td align=center>",
    "<select name=str_recipient_id>",
    "<option selected>ALL USERS";
my $i = 0;
while ($i < $usercount) {
    print "<option value=\"", $users[$i]->getName, "\">", $user->getName;
    $i++;
}
print "</select>",
    "</td>",
    "</tr>",
    "</table><P>",
    "<input type=submit name=\"btn_logout\" value=\" Logout \",
    "<br><font face=\"tahoma,verdana,arial\" size=-2>",
    "</center>";
}

sub show_messages {
    my $user = shift;
    my $channel = shift;
    my @messages = $channel->getMessages;
    my $message_count = @messages;

    if ($message_count == 0) {
        return;
    }

    print "<br><table border=1 cellpadding=3 cellspacing=0 width=550>",
        "<tr bgcolor=brown>",
        "<th colspan=2><font color=white face=\"tahoma,verdana,arial\"
size=-1>",
        "Channel: ", $user->getChannelName, "</th>",
        "</tr>",
        "<tr bgcolor=silver>",
        "<th width=100><font face=\"tahoma,verdana,arial\" size=-1>",
        "User</th>",
        "<th width=450><font face=\"tahoma,verdana,arial\" size=-1>",
        "Message</th>",
        "</tr>";

```

```

my $i = 0;
while ($i < $message_count) {
    if (($i % 2) == 0) {
        print "<tr bgcolor=ffffca>";
    }
    else {
        print "<tr>";
    }
    print "<td align=right><font face=\"tahoma,verdana,arial\" size=-1>";

#           if (msg.getSenderEmail() != null) {
#               out.println("<a href=\"mailto:\" + msg.getSenderEmail()
+ "\">");
#           }

    print $messages[$i]->getSenderName, "</a></td>",
        "<td align=left>",
        "<font face=\"tahoma,verdana,arial\" size=-1>",
        $messages[$i]->getContent,
        "</td>";
    print "</tr>";
    $i++;
}
print "</table>";
}

sub show_private_messages {
    my $user = shift;
    my $channel = shift;
    my @private_messages = $channel->getPrivateMessages;
    my $pmessage_count = @private_messages;

    if ($pmessage_count == 0) {
        return;
    }

    print "<P><table border=1 cellpadding=3 cellspacing=0 width=550>",
        "<tr bgcolor=brown>",
        "<th colspan=2><font color=white face=\"tahoma,verdana,arial\"
size=-1>",
        "Private Messages</th>",
        "</tr>",
        "<tr bgcolor=silver>",
        "<th width=100><font face=\"tahoma,verdana,arial\" size=-1>",
        "User</th>",
        "<th width=450><font face=\"tahoma,verdana,arial\" size=-1>",
        "Message</th>",
        "</tr>";

    my $i = 0;
    while ($i < $pmessage_count) {
        print "<tr>",
            "<td align=right><font face=\"tahoma,verdana,arial\" size=-2>";
    }
}

```

```

#             if (msg.getSenderEmail() != null) {
#                 out.println("<a href=\"mailto:" + msg.getSenderEmail()
+ "\">");
#             }

    print $private_messages[$i]->getSenderName, "</a> -&gt; ";

#             if (msg.getRecipientEmail() != null) {
#                 out.println("<a href=\"mailto:" +
msg.getRecipientEmail() + "\">");
#             }

    print $private_messages[$i]->getRecipientName, "</a></td>",
        "<td align=left>",
        "<font face=\"tahoma,verdana,arial\" size=-2>",
        $private_messages[$i]->getContent,
        "</td>",
        "</tr>";

    $i++;
}
print "</table>";
}

sub show_main {
    my $user = shift;
    my $dbname = "perlchat";
    my $dbuser = "postgres";
    my $dbh = DBI->connect("dbi:Pg:dbname=$dbname", $dbuser, "", {
RaiseError => 1, AutoCommit => 0 });

    print << "MAIN";
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
<HEAD>
<TITLE>PerlChat Chat Rooms</TITLE>
</HEAD>
<BODY bgcolor=#FFFFFF>
<form name="jchat" method=post action="">
<table border=0 width="891" height="77">
<tr>
<td width="875" height="73">
    <p align="center"><font face="Verdana" size="2"><b>Select a Chat Room
to Enter</b></font></p>
<table border=0 cellspacing=0 cellpadding=0 width=100% height=262>
<tr>
<td align=center valign=middle height="216">
<table border=0 cellspacing=0 cellpadding=3 width=389 bgcolor=brown>
<tr>
<th width="391">
    <p align="center"><font size="-1" color="white" face="Verdana">Your
favorite Chat Rooms</font></p>
</th>
</tr>
<tr><td align=center valign=middle width="391"><table border=0
cellspacing=0 cellpadding=3 bgcolor=silver width=621 height="1">

```

MAIN

```

my $uname = $user->getName;
my $sth = $dbh->prepare("select rname from favorites where userid = ' "
. $uname . "'");
$sth->execute;
while (my @row = $sth->fetchrow_array) {
    my $channel = $Channels::channels->getChannel($row[0]);
    my $usercount = $channel->size;
    print "<tr>",
        "<td align=left width=\"605\" height=\"1\">",
        "<a href=" . $Channels::SCRIPT_ALIAS . "perlchat.pl?userid=" .
$uname . "&channel=" . $row[0] . ">" . $row[0] . "</a></td>",
        "<td align=left width=\"405\" height=\"1\">",
        $usercount,
        " User(s)</td>",
        "</tr>\n",
    }

    print << "MAIN";
</table></td>
</tr>
</table>
<p>&nbsp;
</td>
</tr>
</table>
</td>
</table>
</form>
</BODY>
MAIN
$dbh->disconnect;
}

sub login {
    my $user = shift;
    my $channel = shift;
    my $default = $Channels::channels->getChannel("default");
    $default->removeUser($user);
    $channel->addUser($user);
    $user->setChannelName($channel->getName);
    $channel->addMessage(Message->new($user, $DEFAULT_LOGIN_MESSAGE));
}

sub logout {
    my $user = shift;
    my $channel = shift;
    my $default = $Channels::channels->getChannel("default");
    $channel->removeUser($user);
    $default->addUser($user);
    $channel->addMessage(Message->new($user, $DEFAULT_LOGOUT_MESSAGE));
    $user->setChannelName("default");
}

```

---

## 6 TESTING

---

The PerlChat system has been completely tested under the Linux Operating system. The version of the Linux OS used for the development of PerlChat is 2.2.4-2 (Version 7.1). The system configuration used for testing PerlChat is given below.

- ✓ Pentium III 700 MHz Processor
- ✓ 128 MB RAM
- ✓ 20 GB HDD
- ✓ Linux OS with the X Windows System and Apache server preinstalled.

Under these test conditions, the PerlChat system was able to deliver rich HTML content to the user, with low response and page loading times, resulting in a enriched user experience.

---

## 7 CONCLUSION

---

The PerlChat HTML based chat application has been successfully designed and implemented for the Linux Operating System. Its capabilities have been fully tested in this environment. A positive response provided by the users of the system demonstrates that the PerlChat system has achieved its design goals.

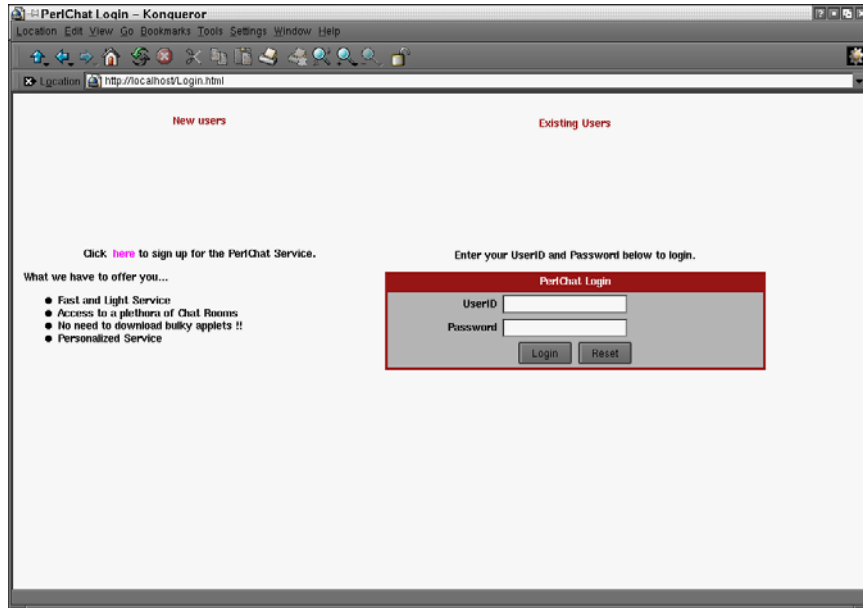
**THE END**

---

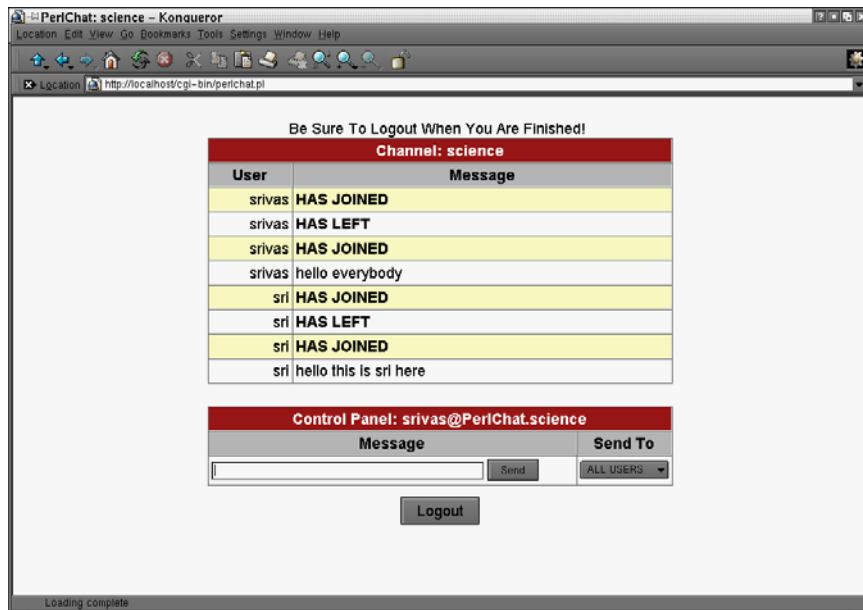
# APPENDIX

---

Below are some screenshots of the PerlChat system.



PerlChat Login Screen



PerlChat Main Screen