P.M.S Perl - Messaging System

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Einleitung

Aufgabenstellung

Im Rahmen der Technickerausbildung im Schuljahr 2011/12 in der Klasse ITT 7/8 wurde die Aufgabe gestellt, eine in Perl realisierte Anwendung mit Datenbankanbindung zu planen, entwickeln und vorzustellen. Als Thema der Projektarbeit haben wir uns das Perl-Messaging-System, kurz P.M.S. ,einen Chatserver mit Client und Administrationsinterface, ausgesucht.

Dieses Dokument beschreibt die Funktionsweise und Implementation des Perl Messaging Systems, also den Core-Server, die Erweiterung durch Module, das Admin Interface und den Chat Client.

Die Arbeiten an der Software wurden gemeinschaftlich von Lukas Michalski, Benjamin Zeller und Thorsten Schwalb durchgeführt.

Zielsetzung Core-System

Das meisste Augenmerk lag hierbei auf dem Herzstück des PMS , dem Core.

Die Hauptziele bei der Impementation des Cores waren wie folgt:

1. Implementation eines Multi-Channel Multi-User Chatservers
2. Quasi parallele Verarbeitung mehrerer Connections
3. Schnelle Asynchrone Verarbeitung der Commands, durch Eventbasierte Programmierung
4. Erweiterbarkeit durch Modularisierung
    a. Erweiterbarkeit der Userfunktionen durch Plugins
    b. Erweiterbarkeit der Connectivity durch sogenannte Connection Provider
5. Implementation eines einfachen Chatprotokolls
6. Möglichst lose Objektkopplung zwischen den Plugins und dem Core

Alle dieser Ziele wurden bis zum Projektende in die Tat umgesetzt und liegen in der aktuellen Softwareversion vor.

Zielsetzung Erweiterungs-Module

Es sollten mehrere Erweiterungsmodule entwickelt werden um zu demonstrieren auf welche Weise es möglich ist den P.M.S. Server zu erweitern. Vor allem das Security Modul ist hierbei hervorzuheben, da es fast die komplette Palette der Möglichkeiten ausnutzt.

Die Ziele waren wie folgt:

1. Security

   ```
   Rechteverwaltungssystem und registrierte User,
   einteilung der User in Gruppen und zuweisung von Rechten
   an die Gruppen
   ```

2. Statistik Modul

   ```
   Statistiken zB über die Aufenthaltsdauer eines
   Users, wieviele Messages er schreibt usw.
   ```

3. Backlog Modul

   ```
   Die Messages eines Channels werden in der Datenbank hinterlegt
   und werden an einen User gesendet der den Channel betritt, damit
   dieser weiss was vorher gesprochen wurde.
   ```

Im Rahmen des Projekts wurden die Ziele wie folgt umgesetzt:

1. Security Modul

   ```
   Bis auf die Möglichkeit Gruppen anuzlegen und zu verwalten
   wurde das Modul vollständig umgesetzt
   ```

2. Backlog Modul

   ```
   Wurde komplett umgesetzt
   ```

3. Statistik Modul

   ```
   Konnte im Zeitrahmen des Projektes nicht umgesetzt werden
   ```

Zielsetzung HTML5 - Chat Client

Der Chat Client ist die Referenzimplementation eines Clients, basierend auf einer Kombination von Html und Javascript.

Die Ziele waren wie folgt:

1. Komplette Referenzimplementation des P.M.S. Chatprotokolls
2. Die Möglichkeit in mehreren Channeln gleichzeitig chatten zu können
3. Moderne und ansprechende HTML5 basierte Oberfläche

Alle dieser Ziele wurden bis zum Projektende in die Tat umgesetzt und liegen in der aktuellen Softwareversion vor.

Zielsetzung Admin-Interface

Das Admin Interface wird benötigt, um auf die von Modulen bereitgestellten Daten zugreifen zu können, oder diese zu konfigurieren. Die Implementierung erfolgte in Perl, Javascript und Html.

Die Ziele waren wie folgt:

1. HTML und Javascript basierte Oberfläche
2. Modularer Aufbau
3. Chatserver Security Modul Verwaltung
4. Chatserver Backlog Modul Verwaltung
5. Chatserver Statistik Modul Verwaltung

Im Rahmen des Projekts wurden die Ziele wie folgt umgesetzt:

1. HTML und Javascript basierte Oberfläche

   ```
   Komplett implementiert
   ```

2. Modularer Aufbau

   ```
   Wurde erfolgreich verwirklicht
   ```

3. Chatserver Security Modul Verwaltung

   ```
   Bis auf die Gruppenverwaltung implementiert
   ```

4. Chatserver Backlog Modul Verwaltung

   ```
   Konnte im Zeitrahmen des Projektes nicht umgesetzt werden
   ```

5. Chatserver Statistik Modul Verwaltung

   ```
   Konnte im Zeitrahmen des Projektes nicht umgesetzt werden
   ```

Probleme

Einige der Probleme auf die wir beim erstellen dieser Software hatten waren folgende:

- Probleme mit dem Handshake des Websocket Protokolls, dies war anfangs schwer in den AnyEvent Ablauf einzufügen.
- Wenig oder unübersichtliche Dokumentation der CPAN Module. Informationen sind hier schwer zu finden wenn man nicht weiss wonach man sucht.
- Probleme mit Stored Procedures mit MySQL Version < 5.5
- Probleme mit Standardwerten der Datenbank-API.
- Daten aus einer gespeicherten Session konnten nicht ge- laden werden. Problem war ein falsch gesetzter Parameter beim Zuweisen des Session-Objektes.
- Handling des HTML-Layouts im Browser ist manchmal nicht logisch vorherzusehen
- Schwierigkeiten bei der Koordination gemeinsamer arbeiten, da jeder unterschiedliche Termine und Pflichten hatte
- Engpässe in der Tonerversorgung beim drucken des Quellcodes.
- Schlafmangel

Online Testmöglichkeit und Online Dokumentation

Die Software und Dokumentation wurde von uns auf einen Server im Internet gestellt.

Folgende Links stehen zur Verfügung:

- PMS - Chatclient: http://pms.muhla-solutions.de/client/
  Zum testen des Clients einfach Verbinden und einen Channel erstellen oder einem beitreten.
  Für eine Kommandoreferenz bitte im Kaptiel Externe Schnittstellen/User-Schnittstelle (Kommandos) nachlesen.
  Für eine Kommandoreferenz des Security Moduls bitte im Kaptiel Security Modul/User-Schnittstelle (Kommandos) nachlesen.
- PMS - Admin Interface: http://pms.muhla-solutions.de/admin/
  Es wurde ein Testuser angelegt (testuser/passwort)
- Diese Dokumentation im HTML Format: http://pms.muhla-solutions.de/docs/handbook
- Perl API Dokumentation der Server Software: http://pms.muhla-solutions.de/docs/server
- Perl API Dokumentation der Admin Software: http://pms.muhla-solutions.de/docs/admin
- PhpMyAdmin zum beobachten der Datenbank: http://pms.muhla-solutions.de/phpmyadmin (testuser/passwort)

Es wird empfohlen, diese Dokumentation direkt im PDF oder HTML Format zu lesen, da wir diverse Links und Querverweise zu den API-Dokumentation eingebaut haben um das Verständnis zu erleichtern

Chapter 1

Implementation, Funktionsweise und Design des Core-Systems

1.1 Allgemein

Die komplette Implementation des Cores erfolgte in Perl und wurde als reine objektorienterte, eventbasierte Software konzipiert. Der Core ist die zentrale Einheit des P.M.S. es benötigt bis auf einen ConnectionProvider keine Erweiterungsmodule um eine Basis Chat Funktionalität zu Verfügung zu stellen.

Bei der Implementation wurde auf ein modulares Design wert gelegt. Der Server ist somit beliebig erweiterbar. Die Module werden in der Konfigurationsdatei hinterlegt, und vom Server zur Laufzeit geladen und initialisiert.

Es gibt hierbei zwei Arten von Modulen:

ConnectionProvider	Der ConnectionProvider macht es möglich, den Core über beliebige Kommunikationswege zu verbinden, wie zB TCP-Sockets oder Websockets. Selbst ausgefallene Möglichkeiten wie zB HTTP-Server Push wären denkbar.
Plugins	Plugins sind eine Möglichkeit den Server um diverse Funktionalitäten zu erweitern wie z.B. einen Backlog oder selbst ein Chat-Bot (maschineller User) sind möglich.

Es wäre denkbar selbst das Chat Protokoll als Modul zu implementieren, dies war jedoch im Rahmen des Projekts nicht möglich.

1.1.1 Verwendete Fremdsoftware im Core

- AnyEvent (6.14)

```
Das AnyEvent Modul stellt den vom P.M.S. verwendeten Event-Loop
und einige Helferklassen bereit.Es ermöglicht eine sogenannte asynchrone-Programmierung,
diese versetzt den P.M.S. Core in die Lage mehrere Operationen quasi-parallel auszuführen.
So muss man zum Beispiel nicht darauf warten, bis eine Schreiboperation auf
einen Dateideskriptor beendet ist, sondern das System kümmert sich um dessen
Fertigstellung. Dabei kann man jedem asynchronen Aufruf diverse Callback Funktionen
mitgeben, die nach erfolgreicher Fertigstellung des Befehls, oder sogar im Fehlerfall
aufgerufen werden.
```

- Object::Event (1.23)

```
Dieses Modul stellt das sogenannte Observer-Pattern zur Verfügung,
das es uns ermöglicht interne Signale (Events) zu verschicken und zu verarbeiten.
```

```
Es ermöglicht dem Core komplett ohne das Wissen über eventuell geladene Module,
oder deren Voraussetzungen zu operieren.
```

1.2 Interne Schnittstellen

1.2.1 Signale

Signale und Callbacks werden von uns verwendet um Objekten eine Möglichkeit zu geben sich untereinander zu Verständigen, ohne jedoch zuviel vom eigentlichen Kommunikationspartner wissen zu müssen. Diese Schnittstellen-Technik ist auch als Observer-Pattern bekannt [1];.

Als Ausgangspunkt wird von uns das auch dem CPAN nachinstallierte Modul Object::Event verwendet. Da dies aber keinerlei Möglichkeiten hatte zu testen ob ein bestimmtes Signal eigentlich existiert oder nicht, musste die Klasse noch erweitert werden.

Alle Objekte die Signale versenden wollen, müssen vom Objekt Pms::Core::Object ableiten
und in einem globalen Hash **%pmsEvents** alle von ihm versendeten Signale hinterlegen:

```
#!/usr/bin/perl -w

package MyPackage;
use Pms::Core::Object;
use strict;
use utf8;

our @ISA = qw(Pms::Core::Object);

our %PmsEvents = (
  'signal1' => 1,
  'signal2' => 1
);
```

Diese Signale können mit beliebigen Callbacks verknüpft und es können auch mehrere Callbacks pro Signal registriert werden. Die Callbacks werden in der gleichen Reihenfolge aufgerufen, wie sie beim Sender-Objekt registriert werden. Es ist ausserdem möglich Argumente mit an die Callbacks zu übergeben.

Beispiel

```
#Code im Sender Objekt:
sub anyOperation{
  #aulösen des Signals:
  $self->emitSignal('some_signal',"arg1","arg2");
}

#Code im Receiver Objekt:
sub createConnections{
  my $self = shift;
  my $signalSender = shift;

  $signalSender->connect(
    some_signal => $self->_callback()
  );
}

sub _callback{
  my $self = shift;
  return sub{
    my $eventChain = shift; #Die Event Chain über die das Event abgebrochen werden kann
    my $arg1 = shift;
```

[1] http://de.wikipedia.org/wiki/Observer_(Entwurfsmuster)

```
    my $arg2 = shift;
    #do something
  }
}
```

Beim connecten der Signale, wir automatisch in der kompletten Hirarchie des Objekts nach der Definition eines Signals im PmsEvents Hash gesucht. Somit können auch Signale von übergeordneten Klassen verwendet werden, ohne diese noch einmal im Hash angeben zu müssen.

1.2.2 Events

Jedes Modul kann sich für bestimmte Server Events registrieren, es hinterlegt dabei eine Callback-Funktionen die beim auftreten dieses Events ausgeführt wird. Server Events sind hierbei nichts weiter als Signale die von der Application Klasse versendet werden, mit der Besonderheit, das sie immer ein Event Objekt als ersten und einzigen Parameter mit schicken.

Beispiel

```
sub createConnections{
  my $self = shift;
  #Weist den Server an das Event client_connect_success mit dem Callback zu verbinden
  $self->{m_parent}->connect(
    client_connect_success => $self->_callback()
  );
}

sub _callback{
  my $self = shift;
  return sub{
    my $eventChain = shift; #Die Event Chain über die das Event abgebrochen werden kann
    my $eventType  = shift; #Das Event Objekt selbst ,das diverse Informationen enthält

    #Eine Message an den User schicken
    $eventType->connection()->postMessage(
      Pms::Prot::Messages::serverMessage("default","Hallo vom Callback")
    );
  }
}
```

Folgende Events stehen zur Verfügung:

Name	Beschreibung
client_connect_request	Ein neuer Client versucht sich zu verbinden
client_connect_success	Ein neuer Client hat eine Verbindung aufgebaut
client_disconnect_success	Ein Client hat die Verbindung geschlossen
message_send_request	Ein Client versucht eine Message zu senden
message_send_success	Ein Client hat eine Message gesendet
join_channel_request	Ein Client möchte einen Channel betreten
join_channel_success	Ein Client hat einen Channel betreten
leave_channel_success	Ein Client hat einen Channel verlassen
create_channel_request	Ein Client möchte einen Channel erstellen
create_channel_success	Ein Client hat einen Channel erstellt
channel_close_success	Ein Channel wurde geschlossen
change_nick_request	Ein Client versucht seinen Nickname zu ändern
change_nick_success	Ein Client hat seinen Nickname geändert
change_topic_request	Ein Client versucht ein Channel Topic zu ändern
change_topic_success	Ein Client hat ein Channel Topic zu geänderd
execute_command_request	Ein Client versucht ein Command auszuführen

Jedes Event, das über ein "_request" Postfix verfügt, kann in der Callback Funktion unter Angabe eines Grundes zurückgewiesen werden, die weitere Verarbeitung wird somit abgebrochen und der User bekommt eine entsprechende Meldung darüber. Hiermit wird es nicht nur ermöglicht, auf Events zu reagieren sondern diese sogar von ausserhalb des Cores zu steuern. Diese Funktionalität wird zum Beispiel im Security Modul verwendet um Userrechte zu überprüfen, allerdings könnte man damit auch einfachere Wortfilter erstellen, die zB Messages mit Schimpfwörtern herausfiltern.

Zurückweisung eines Events im Callback

```
sub _callback{
  my $self = shift;
  return sub{
    my $eventChain = shift; #Die Event Chain über die das Event abgebrochen werden kann
    my $eventType  = shift; #Das Event Objekt selbst ,das diverse Informationen enthält

    #Setzt die Error Message die an den Client gesendet wird
    $eventType->reject("Eine beliebige Error Message");

    #Stoppt das Event
    $eventChain->stop_event;
  }
}
```

1.3 Externe Schnittstellen

1.3.1 Allgemein

Wie in der Einleitung schon kurz erwähnt, ist die komplette Kommunikation im Core modular aufgebaut. Hierfür sind zwei abstrakte Klassen im Core vorhanden, die die Implementationsdetails der jeweiligen Kommunikationsbackends "verstecken".

- Pms::Core::ConnectionProvider

  Der ConnectionProvider ist die Abstrakte Form eines Server-Sockets, wie man ihn von der normalen TCP Programmierung kennt: Er nimmt Verbindungsversuche an und informiert den Server über ein Signal (connectionAvailable) darüber, das neue Connections vorliegen. Hierbei wird komplett abstrahiert wie dieser Ablauf im Hintergrund funktioniert. Ob dies nun ein HTTP Requst oder ein normaler TCP-Socket ist, ist für den Core komplett irrelevant.

- Pms::Core::Connection

  Die Connection Klasse hat zwei Aufgaben:

  1. Sie abstrahiert die darunterliegende Kommunikation
  2. Sie identifiziert den verbundenen User und weiss dessen Nicknamen

  Am besten kann man sich die Connection Klasse als einen Socket vorstellen, sie empfängt und schickt Daten entweder synchron oder asynchron, je nachdem welche Funktion verwendet wurde. Auch hier ist das darunterliegende Protokoll irrelevant. Unter Zuhilfenahme einer SessionId und eines Mailservers könnte man sogar, eine Kommunikation per EMail über diese Klasse realisieren.

Die verwendeten Backends werden über die Konfigurationsdatei festgelegt und zur Laufzeit geladen und initialisiert. Standardmässig sollten sie im Packet Pms::Net untergebracht sein, können aber durch Verwendung des FQN in der Konfigurationsdatei von überall geladen werden (solange sie sich im Include Pfad befinden)

1.3.2 Backend Implementation Websocket

Die Referenzimplementation des Backends ist auf dem Modul AnyEvent::Handle und dem WebSocket Protokoll aufgebaut, das es dem Server ermöglicht HTML5-WebSocket Connections entgegenzunehmen, somit kann jeder mit einem HTML5-fähigen Browser, ohne etwas installieren zu müssen, eine Verbindung zum Chatserver aufbauen.

Note
Im Rahmen der Projektarbeit, wurden nur der Chromium und der Firefox Browser getestet.

1.3.2.1 Verwendete Fremdsoftware

- AnyEvent::Handle

```
Ein Teil des AnyEvent Frameworks, das es ermöglicht Socket Handles mit unserer asynchronen
Programmierung zu verbinden. Anstatt in einem blockierenden System-Call auf neue Daten
zu warten, führt der EventLoop unsere Callbacks aus wenn neue Daten ankommen.
```

- AnyEvent::Socket

```
Ein Teil des AnyEvent Frameworks, das es ermöglicht einen TCP-Server mit unserer  ↩
    asynchronen
Programmierung zu verbinden. Anstatt in einem blockierenden System-Call auf neue  ↩
    Verbindungen
zu warten, führt der EventLoop unsere Callbacks aus wenn neue Daten ankommen.
```

- Protocol::WebSocket (0.00906)

```
Ein aus dem CPAN installiertes Modul, implementiert das Websocket Protokoll.
Hierbei wird ein einfacher HTTP-Handshake an den Server gesendet:

GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13

Der Server antwortet dementsprechend und wechselt von der HTTP
in die Socket Kommunikation. Danach kann man die Verbindung wie eine
normale TCP Verbindung verwenden.

Weiterführende Informationen:
http://de.wikipedia.org/wiki/WebSocket
```

1.3.2.2 Line Protokoll

Da man, wie bei jeder Stream basierten Verbindung (wie zB Tcp oder Websocket), nicht weiss wann eine Nachricht komplett angekommen ist oder ob man noch auf Daten warten muss, braucht man ein Protokoll das einem hilft Anfang und Ende der Nachrichten zu erkennen. Hierbei haben wir uns für Netstring, ein einfaches Textbasiertes Protokoll entschieden. Eine Nachricht im Netstring Format hat diese Darstellung:

16 : Das ist ein Text ,

Die Zahl am Beginn der Nachricht, beschreibt wieviel **Zeichen** sich zwischen dem beiden Delimitern : und , befinden. Hierbei ist vollkommen egal welche Zeichen, sogar ein weiterer Netstring könnte eingebettet werden oder sogar binäre Daten. Letzteres ist aber aufgrund von Einschränkungen durch das WebSocket Protokoll nicht möglich, da dies nur UTF-8 Character zulässt. Nun ist es ein leichtes für den Server zu erkennen wann eine Nachricht endet und wann er noch warten muss:

Weitere Informationen zu Netstring, findet man in der offiziellen Dokumentation: http://cr.yp.to/proto/netstrings.txt

1.3.3 Chat und Message Protokoll

Damit Client und Server richtig miteinander kommunizieren können, haben wir ein Chatprotokoll festgelegt, und einen eigenen handgeschriebenen Parser dafür entwickelt. Die Implementierung befindet sich im Pms::Prot::Parser Modul.

Der Aufbau einer Message hat immer den gleichen Aufbau:

```
/command "Hallo ich bin ein String" 12345
```

Folgende Regeln müssen zutreffen:

- Eine Message startet immer mit einem /
- Nach dem / muss sich ein Token befinden (Kommandoname),
- Ein Token startet immer mit einem Buchstaben und kann nur aus Buchstaben und Zahlen bestehen
- Alle Argumente sind duch Leerzeichen getrennt
- Ein Argument kann entweder ein String oder eine Nummer sein
- Ein String startet entweder mit einem " oder einem ’ , das gleiche Zeichen muss verwendet werden um den String zu beenden
- Kommt das Escape Zeichen im String noch einmal vor muss es mit einem \ escaped werden.
- Eine Nummer kann mit folgenden Zeichen beginnen: +,-,. oder einer Nummer

Wenn der Parser keinen Fehler findet, wird er einen Hash zurückgeben der den Kommandonamen (das Token nach dem /) und die Argumente als Array beinhält. Dieses wird dann an das Server-Objekt(Pms::Application) zur weiteren Verarbeitung weitergegeben.

1.4 Das Server Object

Das Herzstück, das Pms::Application Modul verbindet die bisher beschriebenen Funktionalitäten miteinander. Es sorgt dafür, das die ConnectionProvider und Module geladen und initialisiert werden.

Der Server ist ausserdem dafür zuständig, die einzelnen Connections anzunehmen und zu verwalten, Channels zu erstellen und zu löschen und die Messages richtig zu verteilen.

1.4.1 Empfang und Abarbeitung von Messages

Nachdem, der Server von einer der bestehenden Connections ein dataAvailable Signal erhalten ,die Daten ausgelesen und sie wie in den Kapiteln "Line Protokoll" und "Chat und Message Protokoll" verarbeitet hat, liegt ein Hash mit folgendem Aufbau vor:

```
(
  name => "Kommandoname",
  args => [
    arg0,
    arg1,
    #,....,
    argn
  ]
);
```

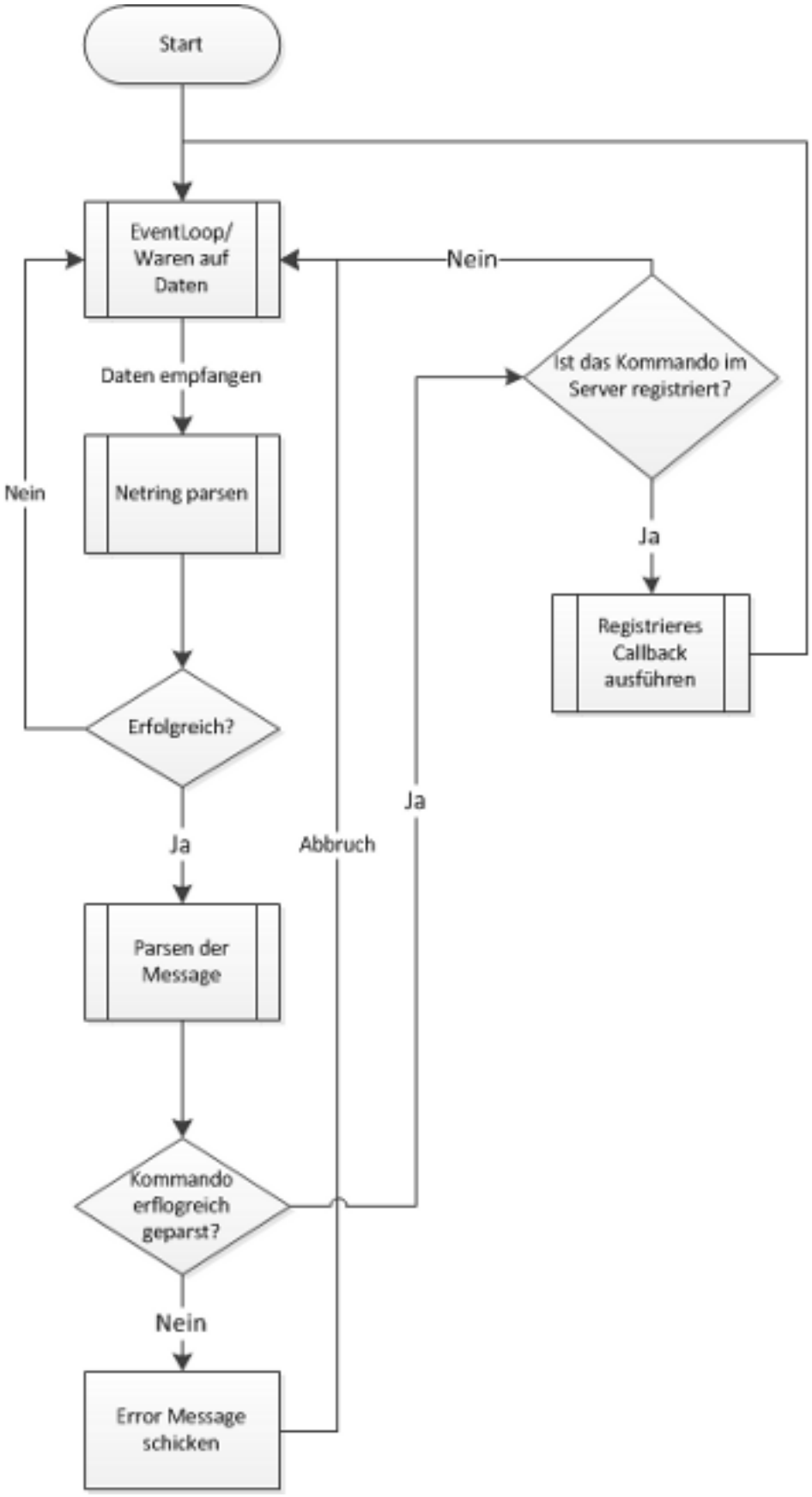
Der Server sucht nun in einem internen Hash, ob ein Callback für dieses Kommando hinterlegt wurde, und führt es aus. Als Parameter für den Callback wird immer als erstes Argument das Connection Objekt übergeben, das das Kommando ausführt, danach das Array mit den Argumenten aus dem Hash.

1.4.1.1 User-Schnittstelle (Kommandos)

Der Core Server kommt von Haus aus mit einigen vordefinierten Kommandos:

- **/send "channel" "message"**
  Eine Chatmessage, die an einen Channel geschickt wird
- **/join "channel"**
  Einen Channel betreten
- **/leave "channel"**
  Einen Channel verlassen
- **/create "channel"**
  Einen neuen Channel erstellen
- **/list**
  Eine Liste von bestehenden Channeln abfragen
- **/nick "newname"**
  Den eigenen Usernamen ändern
- **/users**
  Eine Liste von Usern erhalten (internes Kommando)
- **/topic "channel" "topic"**
  Die Überschrift/ das Thema eines Channels ändern

1.4.1.2 Verarbeitungsschema eines Kommandos



1.4.2 Erweiterungs API für Module

Eine weitere wichtige Aufgabe, ist die Bereitstellung einer API für die Erweiterung durch Module. Dazu gehören nicht nur die Events, die durch die Module beinflusst werden können, sondern auch eine Möglichkeit eigene "Chat-Kommandos" zu registrieren, die vom User oder der Clientapplikation aufgerufen werden können. Ein Modul muss dazu nur ein Kommando mit einer dazugehörigen Callback-Funktion registrieren, den Rest macht das Server Objekt von alleine:

```
sub createCommands{
  my $self = shift;
  my $srv  = shift; #Das Server Objekt

  $srv->registerCommand("ping",$self->_pingCallback());
}

sub _pingCallback{
  my $self = shift;
  return sub{
    my $connection = shift;
    $connection->postMessage(Pms::Prot::Messages::serverMessage("default","Pong"));
  }
}
```

Führt der User jetzt das Kommando **/ping** aus, wird der Server im "default" [2] Channel mit "Pong" antworten.

Das verarbeiten und stoppen von Events wurde im Kapitel Events schon beschrieben. Eigene Events zu registrieren ist jedoch nicht möglich,allerdings kann über das **execute_command_request** Event die Ausführung eines von einem Modul registrierten Kommandos beinflusst werden.

1.4.3 Laden der Module

Das Server Objekt, lädt die Module zur Laufzeit, welche Module der Server lädt, in welcher Reihenfolge und welche Abhängigkeiten die Module untereinander haben, werden in der Server Konfigurationsdatei hinterlegt. Initialisierung und Shutdown der Module muss in den Konstruktor und Destruktor Funktionen durchgeführt werden, ansonsten werden keine Anforderungen an die API der Module erstellt. Der Server übergibt als erstes Argument an den Konstruktor immer eine Referenz auf sich selbst und als zweites Argument den Konfigurationshash des Moduls, dieser wird aus der Konfigurationsdatei gelesen.

Beispiel-Modul

```
#!/usr/bin/perl -w

package Pms::Modules::Ping;

use strict;
use Pms::Application;
use Pms::Prot::Messages;

sub new{
  my $class = shift;
  my $self  = {};
  bless ($self, $class);

  $self->{m_parent} = shift;     ❶
  $self->{m_config} = shift;     ❷
  $self->createCommands();

  return $self;
}

sub DESTROY{
```

[2] Der default Channel ist das Hauptfenster in dem der Serverprozess Nachrichten schickt

```
  my $self = shift;
  $self->shutdown();
}

sub createCommands{
  my $self = shift;
  my $srv  = shift; #Das Server Objekt

  $srv->registerCommand("ping",$self->_pingCallback()); ❸
}

sub _pingCallback{      ❹
  my $self = shift;
  return sub{
    my $connection = shift;
    $connection->postMessage(
      Pms::Prot::Messages::serverMessage("default","Pong")
    );
  }
}

sub shutdown{
  my $self = shift;
  warn "Shutting Down";
}

1;
```

❶ Das ServerObjekt ist immer der erste Parameter

❷ Der Konfigurationshash des Moduls

❸ Registrierung des **ping** Kommandos im Server

❹ Callback Funktion für das **ping** Kommando

Chapter 2

Security Module

2.1 Allgemein

Um eine sinnvolle Anbindung an eine Datenbank und die Chatfunktion um Rechte zu erweitern wurde das Security Modul entwickelt. Das Security Modul führt zwei Arten von Rechten ein:

1. Globale Rechte

   ```
   Diese Art von Rechten legt fest ob ein User z.B. einen Channel erstellen,
   oder einen Channel löschen kann (das löschen wurde nicht mehr implementiert).
   Also Aktionen ausserhalb der Channel.
   ```

2. Channel Rechte

   ```
   Hier wird festgelegt welche Rechte ein User in einem bestimmten Channel hat.
   Es gibt pro User und Channel ein eigenes Rechtepacket, oder falls nichts in
   der Datenbank hinterlegt ist wird ,je nach Art des Channels, ein Default Packet erstel
   ```

2.2 Verwendete Fremdsoftware

- AnyEvent::DBI

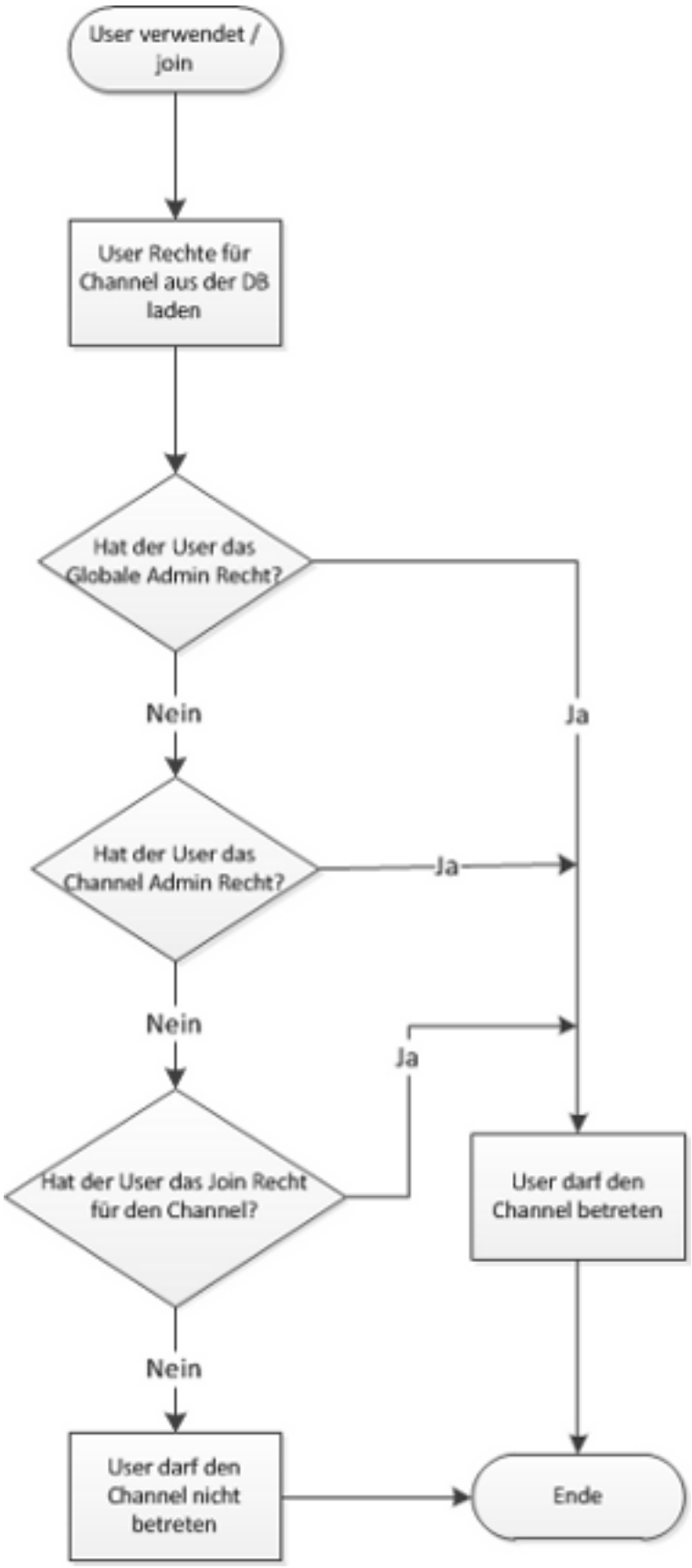
```
Ein aus dem CPAN installiertes Modul. Es ermöglicht eine Asynchrone Verbindung
zur Datenbank aufzubauen. Datenbankoperationen die sehr lange dauern,
können so nicht den Mainloop blockieren und der Server kann eingehende
Events schneller verarbeiten.
Da Perl echte Threads fehlen, startet das AynEvent::DBI Modul kleine Hilfsprozesse,
die die Queries entgegennehmen und ausführen, ist die Datenbankoperation fertig
wird im Hauptprogramm ein mitgegebenes Callback ausgeführt.
```

2.3 Funktionsweise

Das Security Modul nützt alle Möglichkeiten des Core Servers um ihn zu erweitern. Es registriert diverse Befehle und behandelt fast alle Events die der Server schickt. Nachdem ein User erfolgreiche eine Verbindung zum Server aufgebaut hat, bekommt er ein Standard Rechtepacket zugewiesen, das es ihm möglich macht eigene Channels zu erstellen. Hat er einen Channel erstellt, ist er dort Channel-Admin solange er den Channel nicht verlässt, er hat in diesem Channel alle Rechte. User die den Channel nur betreten, bekommen ein Standard Channel Rechtepacket, sie dürfen den Channel standardmässig betreten und darin sprechen.

Das Security Modul führt aber noch eine weitere Art von Channels ein, sogenannte persistente Channel. Diese sind in der Datenbank hinterlegt und werden beim starten des Servers automatisch erzeugt. Um in einen persistenten Channel betreten zu können benötigt ein User extra Rechte, hier wird kein standard Regelpacket erzeugt, hat der User keine Rechte für diesen Channel in der Datenbank ist es auch nicht möglich den Channel zu betreten.

2.4 Überprüfung der Rechte (Schema)



2.5 User-Schnittstelle (Kommandos)

- **/identify "username" "password"**
  Identifiziert den User beim Security Modul, die globalen Rechte des Users werden aus der DB geladen
- **/giveOp "channel" "username"**
  Ein Channeladministrator, hat die Möglichkeit einem anderen User im Channel Administratorenrechte zu geben
- **/takeOp "channel" "username"**
  Einen Channeladministrator kann Admin Rechte eines anderen Users entfernen

Chapter 3

Administrations-Interface

3.1 Allgemein

Um die Module und deren Einstellungen ,wie zB die Rechte des Security Moduls, zu verwalten wurde ein Webinterface auf Basis von Html, Javascript und Perl-CGI Programmierung entwickelt. Als Datenbankbackend kommt natürlich MySQL zum Einsatz.

3.2 Fremdsoftware

- JSON (2.53) und JSON::XS (2.27)

  Aus dem CPAN nachinstallierte Module, sie stellen einen Parser und Serializer für JSON Dokumente bereit. JSON ist ein leichtgewichtiges Datentransferformat, das das Webinterface verwendet um Daten per AJAX vom Server abzuholen.

- HTML::Template

  Aus dem CPAN nachinstalliertes Modul, es ermöglicht die strikte Trennung von HTML und Perl Code. Zu diesem Zweck wird der HTML Code in sogenannte Template Dateien ausgelagert, in denen bestimmte "Tags" hinterlegt sind. Diese "Tags" kann man durch dynamische Daten ersetzen.

- JQuery

  JQuery ist ein Javascript Framework, das es einfach macht Browserübergreifendes Javascript zu schreiben, in unserer Software kommmt es hautpsächlich bei AJAX-Datenabfragen und beim dymamischen ändern des DOM-Trees zum Einsatz.

- Twitter Bootstrap

  Bootstrap ist ein Html, Css und Javascript Framework, das vordefinierte Elemente liefert, um das Styling und Erstellen von HTML basierten Interfaces zu erleichtern. Die komplette Ui des Admin Interfaces wurde mit Bootstrap erstellt.

3.3 Aufbau des Source Codes

```
pms-admin
├── css                                 (Stylesheet Dateien)
├── img                                 (Vom Interface vewendete Images)
├── js                                  (Javascript Code der Grundkonfig)
│   ├── bootstrap.min.js                (Fremdsoftware)
│   ├── helper.js
│   ├── jquery-blockUI.js               (Fremdsoftware)
│   ├── jquery.js                       (Fremdsoftware)
│   ├── login.js
│   └── userview.js
├── Pms
│   ├── BaseModule.pm (Die Basisklasse für alle Module)
│   ├── MainView.pm (Grundgerüst der Oberfläche)
│   ├── Modules
│   │   └── Security
│   │       ├── js (Die Javascript Dateien des Security Moduls)
│   │       │   ├── channel.js
│   │       │   ├── channelRoles.js
│   │       │   ├── user.js
│   │       │   └── userRoles.js
│   │       ├── tmpl                    (Die HTML Templates des Moduls)
│   │       ├── Mod.pm                  (Die Implementation des Moduls selbst)
│   │       ├── Channel.pm              (Die View um Channels anzulegen und zu löschen)
│   │       ├── ChannelRoles.pm         (Die View um Channel-Rechte zu verwalten)
│   │       ├── User.pm                 (Die View um User anzulegen und zu löschen)
│   │       └── UserRoles.pm            (Die View um User-Rechte zu verwalten)
│   ├── Session.pm              (Alle Session bezogenen Funktionalitäten)
│   └── UserView.pm             (Modul, zur Administratoren-Verwaltung)
├── tmpl                        (Enthält die HTML-Templates des Basis Interfaces)
│   ├── addUser.tmpl
│   ├── base.tmpl
│   ├── index.tmpl
│   └── login.tmpl
│
├── index.pl            (siehe Abschnitt Application Entry Scripts)
├── login.pl            (siehe Abschnitt Application Entry Scripts)
├── module.pl           (siehe Abschnitt Application Entry Scripts)
├── PmsConfig.pm        (Die Konfigurationsdatei des Servers)
└── users.pl            (siehe Abschnitt Application Entry Scripts)

9 directories, 42 files
```

3.4 Implementation und Design

Auch das Admin Interface ist wie der Server modular aufgebaut. Server Module, die eine Konfiguration benötigen, die über die Fähigkeiten einer Konfigurationsdatei hinausgehen, müssen nur ein Modul/Plugin für das Admin Interface liefern und einige bestimmte API Vorgaben erfüllen.

3.4.1 Application Entry Scripts

Es gibt insgesamt nur 4 Skripte, die der HTTP-Server direkt ausführen kann, diese leiten die Requests dann an interne Module weiter oder bearbeiten sie direkt selbst:

- **login.pl**

  ```
  Ist für Login und das erstellen der Session zuständig, gibt
  es schon eine bestehende Session, wird der Client auf das index.pl
  Script umgeleitet.
  ```

- **index.pl**

  ```
  Zeigt die Startseite und das Menu an.
  ```

- **module.pl**

  ```
  Dieses Skript behandelt Requests an die Module -> s.a. Module Funktionsweise
  ```

- **users.pl**

  ```
  Ist dafür zuständig neue Admin User über die Oberfläche anzulegen und zu verwalten.
  ```

3.4.2 Funktionsweise von Modulen

Wie schon im vorherigen Abschnitt erwähnt, werden Requests an die Module von Application-Entry Skript module.pl gesteuert. Hierbei wird vorausgesetzt, das es ein Module gibt das folgende Voraussetzungen erfüllt:

- Ein Modul im Verzeichnis Pms/Modules/<PluginName> mit dem Namen Mod.pm

  ```
  Für das Security Modul würde das bedeuten: Pms/Modules/Security/Mod.pm
  ```

- Dieses Module implementiert die Basisklasse Pms::BaseModule
- Es wurden alle abstrakten Funktionen implementiert.

Für genauere Informationen wie ein Modul implementiert wird, beachten Sie bitte die API Dokumentation.

Es gibt zwei Arten von Requests an ein Modul: 1. Die Anforderung einer neuen HTML-Seite, also eine View 2. Ein Datenrequest das per AJAX geschickt wird.

Welche Art von Request durchgeführt wird und an welches Modul es geschickt wird, wird per GET Daten in der URL entschieden:

URL für eine View

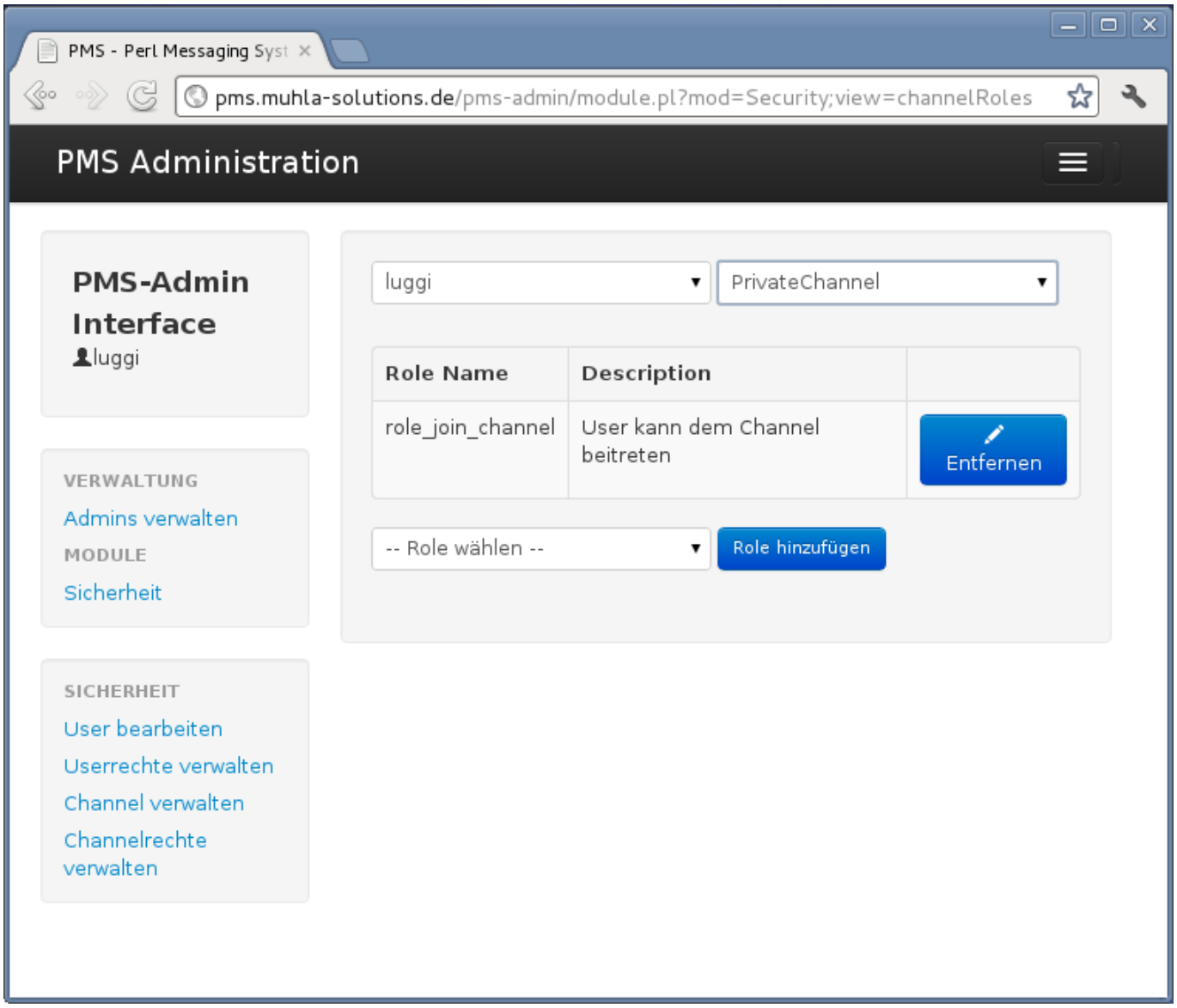
```
http://hostname/module.pl?mod=<ModuleName>;view=<ViewName>
```

Wie man im Beispiel erkennen kann, wird über die **mod** Variable die Bezeichnung des Moduls an das module.pl Skript übergeben, dieses versucht dann zur Laufzeit das Modul zu laden und ruft die Funktion **renderContent($viewName)** auf. Diese Funktion sollte den Inhalt der HTML Seite zurückgeben, dieser Inhalt wird dann in den Datenbereich des Haupt-Templates eingebettet.

URL für einen Daten Request

```
http://hostname/module.pl?mod=<ModuleName>;action=<ActionName>
```

Die Url für einen Daten-Request ist im Aufbau der View-Request ähnlich, allerdings hat sie anstatt dem **view** Parameter, einen **action** Parameter. Das Skript module.pl lädt das Plugin in der gleichen Weise wie bei der View, führt hier jedoch die Funktion **dataRequest($actionName)** aus. Daten, die an die dataRequest Funktion selbst übergeben werden, müssen als JSON-Dokument per POST Request übergeben werden. In der aktuellen Konfiguration werden unbekannte GET-Parameter einfach ignoriert.


PMS - Perl Messaging Syst
pms.muhla-solutions.de/pms-admin/module.pl?mod=Security;view=channelRoles
PMS Administration
PMS-Admin Interface
luggi
VERWALTUNG
Admins verwalten
MODULE
Sicherheit
SICHERHEIT
User bearbeiten
Userrechte verwalten
Channel verwalten
Channelrechte verwalten
luggi
PrivateChannel
Role Name
Description
role_join_channel
User kann dem Channel beitreten
Entfernen
-- Role wählen --
Role hinzufügen

Chapter 4

Graphische Benutzeroberfläche

4.1 Allgemein

Damit Benutzer über den Core miteinander kommunizieren können, wurde dieser Chat Client entwickelt.

4.2 Fremdsoftware

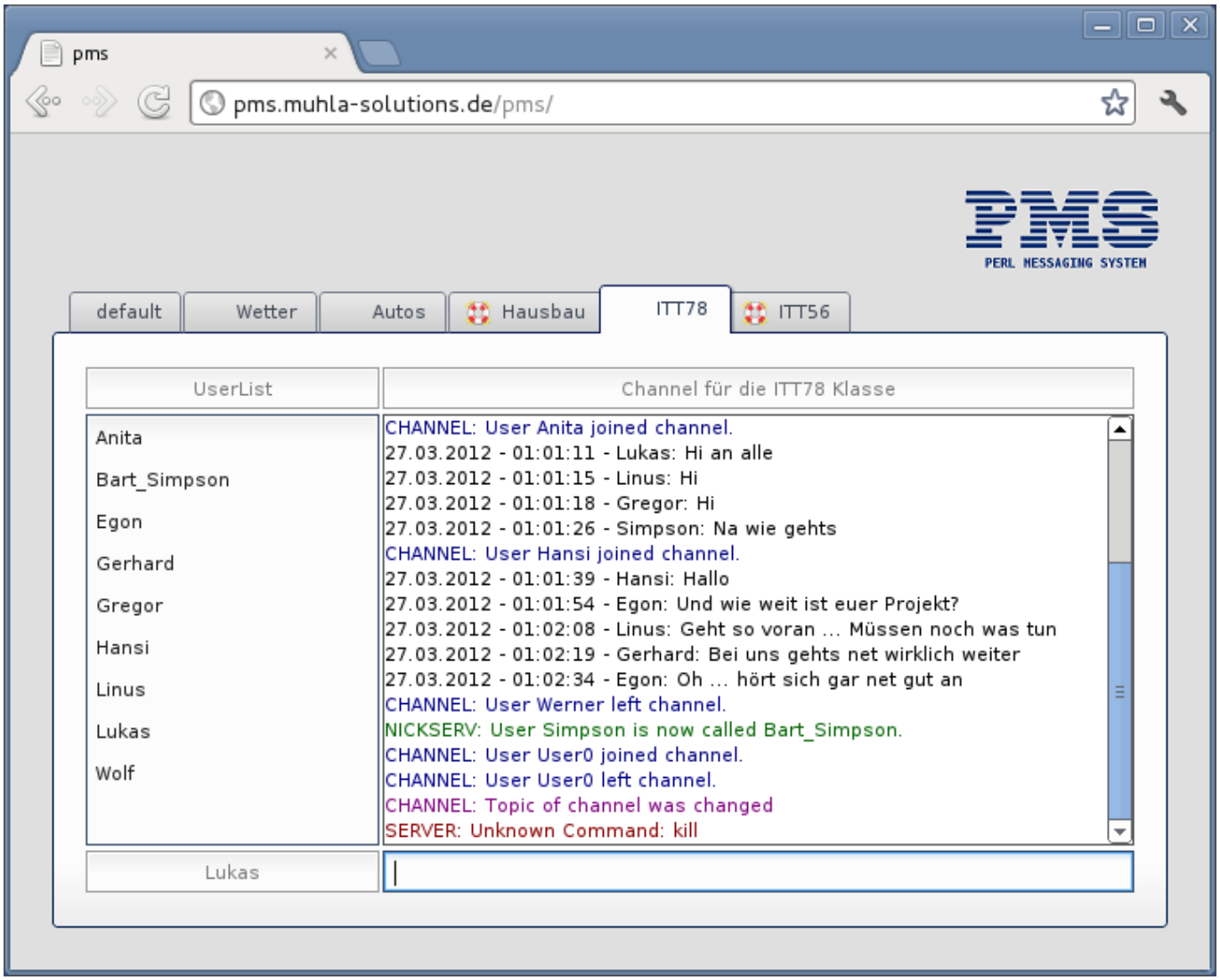
- Qooxdoo JavaScript Framework SDK v1.6

```
qooxdoo ist ein clientseitiges Framework für den Bau grafischer
Benutzeroberflächen für Webanwendungen mit Hilfe des Programmierkonzepts
Ajax. qooxdoo ist ein Rahmenwerk für die Entwicklung von Anwendungen,
die der Benutzer über seinen Webbrowser aufruft und bedient. Es stellt
in einer Programmbibliothek zahlreiche auf der Skriptsprache JavaScript
basierende Komponenten zur Verfügung, die Aussehen und Bedienung von
Webanwendungen an klassische Desktop-Anwendungen angleichen. Dabei
verwendet es das Programmierkonzept Ajax, das es erlaubt, Benutzer-
eingaben zu verarbeiten, ohne die gesamte Webseite neu zu laden.
```

4.3 Implementation und Design

Die komplette Implementation erfolgte in Javascript und als Framework wurde Qooxdoo verwendet. Es wurden alle Commands und Messages des Server implementiert.

Genauere Implementationsdetails des Chatclients werden in diesem Dokument nicht weiter behandelt, da die graphische Oberfläche in JavaScript entwickelt wurde und nicht Gegenstand des Unterrichtes ist.


pms
pms.muhla-solutions.de/pms/
PMS
PERL MESSAGING SYSTEM
default
Wetter
Autos
Hausbau
ITT78
ITT56
UserList
Channel für die ITT78 Klasse
Anita
Bart_Simpson
Egon
Gerhard
Gregor
Hansi
Linus
Lukas
Wolf
CHANNEL: User Anita joined channel.
27.03.2012 - 01:01:11 - Lukas: Hi an alle
27.03.2012 - 01:01:15 - Linus: Hi
27.03.2012 - 01:01:18 - Gregor: Hi
27.03.2012 - 01:01:26 - Simpson: Na wie gehts
CHANNEL: User Hansi joined channel.
27.03.2012 - 01:01:39 - Hansi: Hallo
27.03.2012 - 01:01:54 - Egon: Und wie weit ist euer Projekt?
27.03.2012 - 01:02:08 - Linus: Geht so voran ... Müssen noch was tun
27.03.2012 - 01:02:19 - Gerhard: Bei uns gehts net wirklich weiter
27.03.2012 - 01:02:34 - Egon: Oh ... hört sich gar net gut an
CHANNEL: User Werner left channel.
NICKSERV: User Simpson is now called Bart_Simpson.
CHANNEL: User User0 joined channel.
CHANNEL: User User0 left channel.
CHANNEL: Topic of channel was changed
SERVER: Unknown Command: kill
Lukas

Chapter 5

Systemvoraussetzungen

5.1 Allgemein

Um diese Software nutzen zu können, müssen diverse Voraussetzungen erfüllt sein. Im weiteren wird ein System beschrieben, welches erfolgreich getestet wurde.

5.2 Core - System

Betriebssystem: ArchLinux x64

- Repositories: core-remote
- Paketauswahl: base, standard

5.2.1 Zu installierende Pakete

- base-devel
  (pacman -S base-devel) (all)
- openssh
  (pacman -S openssh)
- git
  (pacman -S git)
- mysql
  (pacman -S mysql)
- perl-dbd-mysql
  (pacman -S perl-dbd-mysql)

5.2.2 Perl Pakete

- AnyEvent
- AnyEvent::DBI
- Object::Event
- Protocol::WebSocket

5.2.3 Quellen

- git://gitorious.org/p-ms/pms-server.git

```
[/srv/project]$ git clone git://gitorious.org/p-ms/pms-server.git
```

5.2.4 Datenbank

- MySQL DB anlegen

```
– mysql –u root –p
  Enter password:

  mysql> create database pms;
  Query OK, 1 row affected (0.00 sec)

  mysql> grant usage on *.* to pms@localhost identified by 'pass';
  Query OK, 0 rows affected (0.00 sec)

  mysql> grant all privileges on pms.* to pms@localhost;
  Query OK, 0 rows affected (0.00 sec)
```

- MySQL Dump importieren
- PmsConfig.pm konfigurieren
  Bereich **%baseDBHash** einrichten

5.3 Admin - Interface

5.3.1 Zu installierende Pakete

- lighttpd
  (pacman -S lighttpd)

5.3.2 Perl Pakete

- HTML::Template
- CGI::Session
- JSON

5.3.3 Quellen

- git://gitorious.org/p-ms/pms-admin.git

```
[/srv/project]$ git clone git://gitorious.org/p-ms/pms-admin.git
```

5.3.4 Datenbank

- PmsConfig.pm konfigurieren

5.3.5 HTTP-Server

- Konfiguration des Servers
  (nano /etc/lighttpd/lighttpd.conf)

```
server.port             = 80
server.username         = "http"
server.groupname        = "http"
server.document-root    = "/srv/http"
server.errorlog         = "/var/log/lighttpd/error.log"
dir-listing.activate    = "enable"
index-file.names        = ( "index.html", "login.pl" )
mimetype.assign         = ( ".html" => "text/html",
                            ".txt" => "text/plain",
                            ".jpg" => "image/jpeg",
                            ".png" => "image/png",
                            ".css" => "text/css",
                            ".js" => "text/javascript",
                            "" => "text/html"
                          )

server.modules = ( "mod_cgi" )

$HTTP["url"] =~ "^/pms-admin/" {
        cgi.assign = ( ".pl" => "/usr/bin/perl" )
}
```

- Standardbenutzer:
  - luggi
  - zbenjamin
- Standardpasswort:
  - passwort

5.4 HTML5 - Chat Client

5.4.1 Zu installierende Pakete

- unzip
  (packer -S unzip)
- python2
  (packer -S python2)

5.4.2 Quellen

- git://gitorious.org/p-ms/pms-webclient-qx.git

```
[/srv/project]$ git clone git://gitorious.org/p-ms/pms-webclient-qx.git
```

5.4.3 Framework

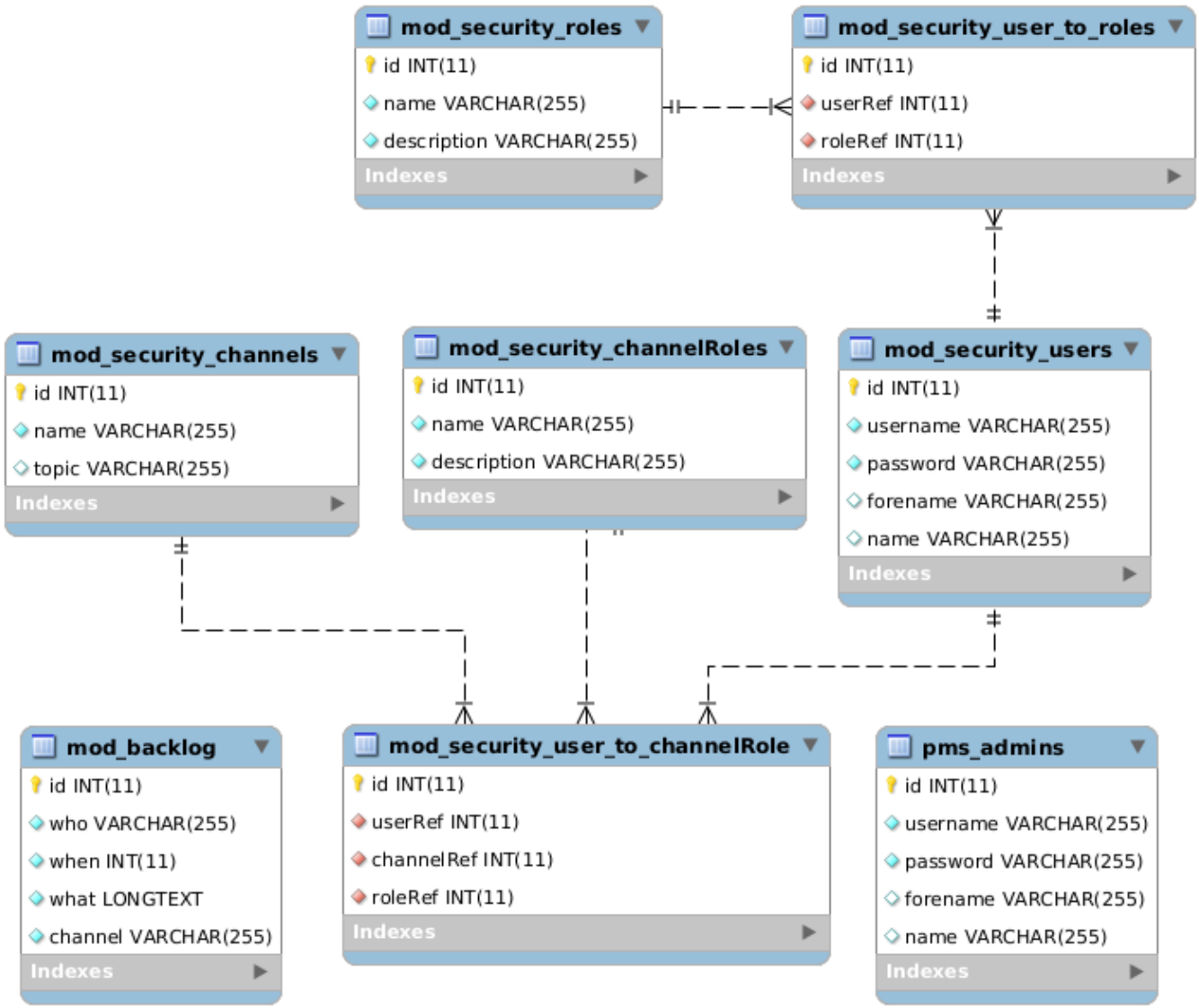
- Installation vom Qooxdoo Framework

```
[/srv/packages]$ wget http://downloads.sourceforge.net/qooxdoo/qooxdoo-1.6-sdk.zip

[/srv/packages]$ unzip qooxdoo-1.6-sdk.zip

[/srv/packages]$ mv qooxdoo-1.6-sdk /opt/

[/srv/packages]$ cd /srv/project

[/srv/project]$ mkdir qooxdoo

[/srv/project]$ cd qooxdoo

[/srv/project/qooxdoo]$ python2 /opt/qooxdoo-1.6-sdk/tool/bin/create-application.py \
                      > --name=pms --out=.

[/srv/project/qooxdoo]$ cp /srv/project/pms-webclient-qx/*.js \
                      > /srv/project/qooxdoo/pms/source/class/pms/

[/srv/project/qooxdoo]$ cp /srv/project/pms-webclient-qx/images/logo_pms.png \
                      >/srv/project/qooxdoo/pms/source/resource/pms

[/srv/project/qooxdoo]$ cd pms

[/srv/project/qooxdoo/pms]$ python2 generate.py build

[/srv/project/qooxdoo/pms]$ cd /srv/http/

[/srv/http]$ ln -s /srv/project/qooxdoo/pms/build/ pms_admin
```

Chapter 6

Datenbank MySQL

6.1 Datenbank Schema



6.2 Primary und Foreign Keys

```
ALTER TABLE `mod_security_user_to_channelRole`
  ADD CONSTRAINT `mod_security_user_to_channelRole_ibfk_1`
    FOREIGN KEY (`userRef`)
    REFERENCES `mod_security_users` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  ADD CONSTRAINT `mod_security_user_to_channelRole_ibfk_2`
    FOREIGN KEY (`channelRef`)
    REFERENCES `mod_security_channels` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  ADD CONSTRAINT `mod_security_user_to_channelRole_ibfk_4`
    FOREIGN KEY (`roleRef`)
    REFERENCES `mod_security_channelRoles` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE;
```

```
ALTER TABLE `mod_security_user_to_roles`
  ADD CONSTRAINT `mod_security_user_to_roles_ibfk_1`
    FOREIGN KEY (`userRef`)
    REFERENCES `mod_security_users` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  ADD CONSTRAINT `mod_security_user_to_roles_ibfk_2`
    FOREIGN KEY (`roleRef`)
    REFERENCES `mod_security_roles` (`id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE;
```

6.3 Normalisierung

Die Datenbank-Struktur wurde von uns so entworfen, das sie so nah wie möglich an die 3. Normalform herankommt wie es in einer praktischen Lösung möglich ist. Es gab hierbei keinen extra Normalisierungsschritt sondern es wurden direkt Tabellen in der Endform entwickelt.

6.4 Logisches Datenbankmodell

- mod_backlog(id,who,when,what,channel)
- mod_security_channelRoles(id,name,description)
- mod_security_channels(id,name,topic)
- mod_security_roles(id,name,description)
- mod_security_users(id,username,password,forename,name)
- mod_security_user_to_channelRole(id,userRef(FK),channelRef(FK),roleRef(FK))
- mod_security_user_to_roles(id,userRef(FK),roleRef(FK))
- pms_admins(id,username,password,forename,name)

6.5 Stored Procedures

6.5.1 mod_backlog_delChannel

Diese Procedure entfernt den kompletten Backlog einens Channels aus der Datenbank. Dies wird verwendet, wenn ein Channel vom Server komplett geschlossen wird.

```
CREATE DEFINER=`root`@`localhost`
  PROCEDURE `mod_backlog_delChannel`( in channelName varchar(255) )

BEGIN
  PREPARE
    stmt
  FROM
    'DELETE FROM mod_backlog where channel=?;';
  SET
    @a = channelName;

  EXECUTE stmt USING @a;
End
```

6.5.2 mod_backlog_emptyTable

Diese Procedure wird beim initialisieren des Servers verwendet, um die Backlogs der Channel zu löschen, die nach dem Neustart des Servers nicht mehr vorhanden sind.

Ein gutes Beispiel dafür, wie man mit stored procedures zusätzliche Logik im "Hintergrund" ausführen kann ohne das der aufrufende Code davon beinflusst wird.

In dem Fall wenn der Server das Security Modul verwendet, ist es möglich persistente Channel zu schaffen, die nach dem Neustart erhalten bleiben. Existiert die *mod_security_channels* Tabelle nimmt die stored procedure darauf Rücksicht und entfernt den Backlog dieser Channels nicht.

```
CREATE DEFINER=`root`@`localhost`
  PROCEDURE `mod_backlog_emptyTable`()

BEGIN
 IF (
  (
    SELECT
      COUNT(*) AS table_exists
    FROM
      information_schema.tables
    WHERE
      TABLE_SCHEMA = DATABASE() AND
      table_name = 'mod_security_channels'
  ) = 0)
  THEN
    DELETE FROM
      `mod_backlog` where 1;
  ELSE
    DELETE FROM
      `mod_backlog` where `channel` not in
      (
        SELECT
          `name`
        FROM
          `mod_security_channels`
      );
  END IF;
```

```
END
```

6.5.3 mod_backlog_get

Diese Procedure ruft den Backlog für einen bestimmten Channel ab um diesen an den Client zu schicken.

```
CREATE DEFINER=`root`@`localhost`
  PROCEDURE `mod_backlog_get`(in channelName varchar(255),in count int)
BEGIN
  SELECT
    who,`when`,what
  FROM (
    SELECT
      who,`when`,what,id
    FROM
      mod_backlog
    WHERE
      channel=channelName
    ORDER BY id DESC LIMIT count
    ) as tbl
  ORDER BY
    tbl.`when`,tbl.id;
END
```

6.5.4 mod_backlog_write

Schreibt eine neue Message in die Backlog Tabelle. Verwendet ein "prepared" Statement.

```
CREATE DEFINER=`root`@`localhost`
  PROCEDURE `mod_backlog_write`(in nickname varchar(255)
                                ,in msgTime int
                                ,in channelName varchar(255)
                                ,in msg longtext)
BEGIN
  PREPARE
    stmt
  FROM
    'INSERT INTO
        mod_backlog (who,`when`,channel,what)
     VALUES
        (?,?,?,?)';

  SET @a = nickname;
  SET @b = msgTime;
  SET @c = channelName;
  SET @d = msg;

  EXECUTE stmt USING @a,@b,@c,@d;
End
```

6.5.5 mod_security_getChannelRoles

Diese Procedure liest die Rechte eines Users für einen bestimmten Channel aus der Datenbank.

```
CREATE DEFINER=`root`@`localhost`
  PROCEDURE `mod_security_getChannelRoles`(in userRefArg int
                                           ,in channelRefArg int )
```

```
BEGIN
    SELECT
      roles.name
    FROM
      mod_security_user_to_channelRole as usrToRoles,
      mod_security_channelRoles as roles
    WHERE
      usrToRoles.userRef = userRefArg AND
      usrToRoles.channelRef = channelRefArg AND
      usrToRoles.roleRef = roles.id;
END
```

6.5.6 mod_security_getChannels

Diese Procedure liest die persistenen Channel aus, wird verwendet um diese beim initialisieren des Servers erstellt.

```
CREATE DEFINER=`root`@`localhost`
  PROCEDURE `mod_security_getChannels`( )
BEGIN
  SELECT
    id,name,topic
  FROM
    mod_security_channels;
END
```

6.5.7 mod_security_getUserWithRoles

Liest userid und roles aus der Datenbank, wenn username und passwort bekannt sind. Es wäre zwar möglich erst zu überprüfen ob username und passwort richtig sind und dann die roles auszulesen. Allerdings würde das zwei Selects benötigen, worauf wir aus Gründen der optimierung verzichtet haben.

```
CREATE DEFINER=`root`@`localhost`
  PROCEDURE `mod_security_getUserWithRoles`(in name VARCHAR(255)
                                           ,in pass VARCHAR(255) )
BEGIN
    SELECT
      users.id as userId,roles.name as roleName
    FROM
      mod_security_users as users ,
      mod_security_user_to_roles as usrToRoles,
      mod_security_roles as roles
    WHERE
      users.id = usrToRoles.userRef AND
      usrToRoles.roleRef = roles.id AND
      users.username = name AND
      users.password = pass;
END
```

Chapter 7

Logbücher

7.1 Logbuch Core-System (git)

Name	Date	Comment
Benjamin Zeller	Fri Apr 27 05:03:03 2012	Docs
Lukas Michalski	Thu Apr 26 21:09:00 2012	Adding comment templates
Benjamin Zeller	Thu Apr 26 19:26:11 2012	Stored Procedures
Benjamin Zeller	Thu Apr 26 19:25:58 2012	Stored Procedures
Benjamin Zeller	Tue Apr 24 00:59:35 2012	Empty message and message with just 0
Benjamin Zeller	Tue Apr 24 00:41:34 2012	Less output
Benjamin Zeller	Tue Apr 24 00:38:05 2012	Bugs
Benjamin Zeller	Mon Apr 23 23:32:01 2012	Restructured Security Module, added a own UserInfo for rights n stuff
Benjamin Zeller	Mon Apr 23 01:33:41 2012	Adding takeOp command, About to split Security.pm in seperate Modules
Benjamin Zeller	Mon Apr 23 00:59:01 2012	Introducing showRights and giveOp commands
Benjamin Zeller	Mon Apr 23 00:04:32 2012	Remove Verbose Output from Backlog.pm
Benjamin Zeller	Sat Apr 21 13:18:37 2012	Utf-8 is a bitch
Benjamin Zeller	Sat Apr 21 11:22:21 2012	Backlog Module
Benjamin Zeller	Sat Apr 21 09:44:09 2012	Use mysql reconnect
Benjamin Zeller	Fri Apr 20 23:42:01 2012	Bugs
PMS Development Server	Fri Apr 20 23:28:31 2012	Bugs
Benjamin Zeller	Fri Apr 20 23:08:13 2012	Bugs
Benjamin Zeller	Fri Apr 20 23:02:56 2012	Wrong Nick Change
Benjamin Zeller	Fri Apr 20 22:47:26 2012	Merge branch *master* of gitorious.org:p-ms/pms-server
Benjamin Zeller	Fri Apr 20 22:47:07 2012	Bugs
PMS Development Server	Fri Apr 20 22:36:23 2012	Bugs
Benjamin Zeller	Fri Apr 20 22:31:29 2012	Send disconnect event
Benjamin Zeller	Fri Apr 20 22:22:41 2012	Disconnect Handling in security module
PMS Development Server	Fri Apr 20 22:17:36 2012	Bugs
Benjamin Zeller	Fri Apr 20 17:04:06 2012	Bugs
Benjamin Zeller	Fri Apr 20 16:17:53 2012	Bugs
Benjamin Zeller	Fri Apr 20 01:58:00 2012	Initial Information about the Users Nick

Name	Date	Comment
Benjamin Zeller	Fri Apr 20 00:06:57 2012	Fixed some bugs, send a userlist when someone joins a channel
Benjamin Zeller	Wed Apr 18 22:46:10 2012	Basic Security Stuff seems to work
Benjamin Zeller	Tue Apr 17 00:36:54 2012	Send nickchange from the right place
Benjamin Zeller	Tue Apr 17 00:03:24 2012	Send nickchange just once
Benjamin Zeller	Mon Apr 16 20:26:17 2012	Bug In Message Packet
Benjamin Zeller	Mon Apr 16 10:37:24 2012	send userlist message
Benjamin Zeller	Mon Apr 16 10:29:38 2012	Sending new Messages, Added a way to change the topic
Benjamin Zeller	Sun Apr 15 01:23:24 2012	Commited test code, reversed
Benjamin Zeller	Sun Apr 15 01:22:43 2012	Ok exit can not print a message, reversed to die
Benjamin Zeller	Sun Apr 15 01:12:57 2012	Use exit in case of wrong arguments, noone should be able to catch th
Benjamin Zeller	Sun Apr 15 00:26:17 2012	Fixing a bug
Benjamin Zeller	Sun Apr 15 00:23:33 2012	Adding a config file
Benjamin Zeller	Sat Apr 14 23:06:16 2012	possible userrights structure
Benjamin Zeller	Sat Apr 14 22:11:02 2012	Removed unused Prototypes, connect to DB and query usernames and pass
Benjamin Zeller	Sat Apr 14 20:50:17 2012	Use FQN for Modules
Benjamin Zeller	Fri Apr 13 00:51:57 2012	fixed bug
Benjamin Zeller	Fri Apr 13 00:50:44 2012	Message generating packet
Benjamin Zeller	Thu Apr 12 21:31:38 2012	Refactoring file structure of the Websocket ConnectionProvider code,
Benjamin Zeller	Mon Apr 9 23:18:36 2012	Introducing the users command, a way to list users
Benjamin Zeller	Mon Apr 9 22:52:04 2012	Working on some global API, created a basic security module, fixed som
Benjamin Zeller	Wed Mar 28 20:38:55 2012	Added handling when a Websocket closes and its still in the handshake
Benjamin Zeller	Wed Mar 28 20:30:44 2012	More Debug checks for debug messages
Benjamin Zeller	Wed Mar 28 20:08:48 2012	Handling the Websocket Netstring correctly, basic events are sent now
Benjamin Zeller	Wed Mar 28 18:44:14 2012	Merge branch *master* of gitorious.org:p-ms/pms-server
Benjamin Zeller	Wed Mar 28 18:43:50 2012	Renaming of signals
Lukas Michalski	Wed Mar 28 18:05:51 2012	Changing /list command
Benjamin Zeller	Tue Mar 27 23:52:54 2012	stuff
Benjamin Zeller	Tue Mar 27 23:24:40 2012	implemented /nick , removed some wrong error handling
Benjamin Zeller	Mon Mar 26 20:48:01 2012	Handle Client disconnects, send messages only to the connections that
Benjamin Zeller	Sun Mar 25 22:02:44 2012	Fixed commit error
Benjamin Zeller	Sun Mar 25 21:59:11 2012	Automatic username stuff
Lukas Michalski	Sun Mar 25 12:29:28 2012	Removing unused lines
Lukas Michalski	Sun Mar 25 12:25:00 2012	Adding some checks and a new command called /list
Benjamin Zeller	Sat Mar 24 10:50:06 2012	Introducing createChannel command, sending Events for client actions,
Benjamin Zeller	Wed Mar 21 18:23:06 2012	Added non quoted strings to the parser
Benjamin Zeller	Mon Mar 12 20:45:33 2012	Refactoring, intitial protocol handling
Benjamin Zeller	Thu Mar 8 00:59:14 2012	Replaced all complex class members with references, basic communicatio
Benjamin Zeller	Wed Mar 7 19:31:32 2012	Lots of refactoring, Introducing the abstract for Connections and Conn

Name	Date	Comment
Benjamin Zeller	Mon Feb 27 18:33:24 2012	added comments to Parser.pm
Benjamin Zeller	Wed Feb 15 18:39:06 2012	Netstring implementation works
Benjamin Zeller	Mon Feb 13 19:28:04 2012	Parser works
Benjamin Zeller	Sat Feb 11 11:17:03 2012	First Parser code, not tested
Benjamin Zeller	Sat Feb 11 09:18:57 2012	Restructuring
Benjamin Zeller	Fri Feb 10 00:33:56 2012	Trivial websocket communication seems to work
Benjamin Zeller	Wed Feb 8 09:43:39 2012	resolved conflict
Benjamin Zeller	Wed Feb 8 09:37:44 2012	Restuctured, added some Event classes, testcode to reject events
Lukas Michalski	Tue Feb 7 00:29:21 2012	Laden mehrerer Module nicht moeglich.
Benjamin Zeller	Mon Feb 6 20:56:38 2012	Basic Module loading implemented
Benjamin Zeller	Mon Feb 6 20:20:56 2012	untabify
Lukas Michalski	Mon Feb 6 20:07:42 2012	Merge branch *master* of gitorious.org:p-ms/pms-server
Lukas Michalski	Mon Feb 6 20:06:54 2012	changing .gitignore
Benjamin Zeller	Mon Feb 6 20:05:31 2012	Getting rid of EventHandler
Benjamin Zeller	Mon Feb 6 20:04:35 2012	Removed Testcode
Benjamin Zeller	Mon Feb 6 20:02:19 2012	Basic Module Structure, use strict
Benjamin Zeller	Mon Feb 6 19:35:03 2012	Basic Application Class

7.2 Logbuch Admin-Interface (git)

Name	Date	Comment
Benjamin Zeller	Fri Apr 27 05:02:18 2012	Merge branch *master* of gitorious.org:p-ms/pms-admin
Benjamin Zeller	Fri Apr 27 05:01:56 2012	Removed old files
Benjamin Zeller	Thu Apr 26 23:43:45 2012	Added configuration, deleted old files
Benjamin Zeller	Wed Apr 25 16:42:16 2012	Last Changes, Feature Freeeze
Benjamin Zeller	Sat Apr 21 10:36:31 2012	Merge branch *master* of gitorious.org:p-ms/pms-admin
Benjamin Zeller	Sat Apr 21 10:36:14 2012	Add Page to manage pms-admins
Benjamin Zeller	Sat Apr 21 00:01:14 2012	Docs belongs in DOCS
Thorsten Schwalb	Fri Apr 20 22:49:47 2012	Merge branch *master* of gitorious.org:p-ms/pms-admin
Thorsten Schwalb	Fri Apr 20 22:48:25 2012	Erster Entwurf
Benjamin Zeller	Tue Apr 17 20:34:32 2012	Finished functionality of Security Admin module
Benjamin Zeller	Tue Apr 17 14:42:23 2012	Add and remove Users completely works
Benjamin Zeller	Tue Apr 17 01:00:37 2012	Add,Remove,Edit users works
Benjamin Zeller	Mon Apr 16 14:46:26 2012	loading users per ajax, show a busy indicator
Benjamin Zeller	Mon Apr 16 00:24:56 2012	Creating a Path for AJAX
Benjamin Zeller	Sun Apr 15 23:17:33 2012	Did not use html5 doctype
Benjamin Zeller	Sun Apr 15 21:48:28 2012	Loading Users from the database
Benjamin Zeller	Sun Apr 15 21:24:12 2012	Usermod page
Benjamin Zeller	Sun Apr 15 20:58:07 2012	Shaping out modularization
Benjamin Zeller	Sun Apr 15 18:24:15 2012	Working on modulaization
Benjamin Zeller	Sun Apr 15 16:02:05 2012	Merged
Benjamin Zeller	Sun Apr 15 16:00:50 2012	Merge branch *master* of gitorious.org:p-ms/pms-admin
Benjamin Zeller	Sun Apr 15 16:00:31 2012	Structuring and templating

Name	Date	Comment
Thorsten Schwalb	Sun Apr 15 13:52:45 2012	Sql
Thorsten Schwalb	Sun Apr 15 12:55:20 2012	Import Work
unknown	Sun Apr 8 21:33:35 2012	Iinitial Import
unknown	Sun Apr 8 21:32:59 2012	Iinitial Import
Lukas Michalski	Wed Feb 29 17:10:22 2012	Initial commit

7.3 Logbuch HTML5 - Chat Client (Qooxdoo) (git)

Name	Date	Comment
Lukas Michalski	Fri Apr 27 00:53:36 2012	smaller logo
Lukas	Thu Apr 26 16:51:59 2012	Adding ignoreCommand
Lukas Michalski	Tue Apr 24 21:23:16 2012	Some changes and other colors
Lukas Michalski	Tue Apr 24 02:00:14 2012	routing serverMessages to activeTab
Lukas Michalski	Tue Apr 24 01:40:16 2012	New Comments, restructure, bugs
Lukas Michalski	Tue Apr 24 00:50:23 2012	Removing comments
Lukas Michalski	Tue Apr 24 00:45:04 2012	Outsource message and fix some bugs
Lukas Michalski	Mon Apr 23 21:05:00 2012	Fixing tab problem and bug fixing
Lukas Michalski	Sun Apr 22 23:26:21 2012	Bug fixing, HtmlWidget now works and scroll down properly
Lukas Michalski	Sat Apr 21 19:01:12 2012	Redesign
Lukas Michalski	Fri Apr 20 23:40:43 2012	Bug fixing
Lukas Michalski	Fri Apr 20 22:46:58 2012	UserList
Lukas Michalski	Fri Apr 20 22:20:56 2012	Bugs fixed
Lukas Michalski	Fri Apr 20 02:12:12 2012	Fixing nickchange problem, adding some checks
Lukas Michalski	Wed Apr 18 18:18:14 2012	Adding white logo
Lukas Michalski	Wed Apr 18 18:12:31 2012	Adding image, change core layout, bugfixes
Lukas Michalski	Wed Apr 18 03:34:40 2012	Adding UserName Functions, adjust output messages, bugfix
Lukas Michalski	Wed Apr 18 03:10:19 2012	Redesign of Layout, Adding Username Field, some fixes
Lukas Michalski	Wed Apr 18 01:13:44 2012	Adding warpMessage, bug fixing, outsource functions
Lukas Michalski	Tue Apr 17 01:54:52 2012	Some bugfixes. Adding setTopic method
Lukas Michalski	Mon Apr 16 20:52:37 2012	Adding message-routing and bugfix. Adding timeFormat class
Lukas Michalski	Mon Apr 16 04:27:55 2012	Fix build errors
Lukas Michalski	Mon Apr 16 04:08:31 2012	Redesign, adding Hash-class und fix send/receive message Problem
Lukas Michalski	Sun Apr 15 19:35:32 2012	Upload missing changed files
Lukas Michalski	Sun Apr 15 19:32:46 2012	Adding Parser and different adjustments
Lukas Michalski	Sun Apr 15 12:05:46 2012	First build
Lukas Michalski	Sat Apr 14 20:17:11 2012	Inital commit

7.4 Logbuch HTML5 - Chat Client (Obsolete) (git)

Name	Date	Comment
Benjamin Zeller	Wed Mar 28 20:31:20 2012	Use pre for chatlines so whitespaces are visible

Name	Date	Comment
Lukas Michalski	Wed Mar 28 18:06:55 2012	Adding timeFormat funktion
Benjamin Zeller	Wed Mar 28 00:01:56 2012	beautifing parser.js
Benjamin Zeller	Tue Mar 27 23:45:13 2012	fixed problem with some default messages
Benjamin Zeller	Mon Mar 26 20:49:10 2012	Handle Client disconnects, send messages only to the connections that
Benjamin Zeller	Sun Mar 25 22:03:57 2012	Protocol updates
Benjamin Zeller	Sat Mar 24 10:50:48 2012	fixed a parser error
Benjamin Zeller	Wed Mar 21 18:23:44 2012	Sending Messages to the right channel
Benjamin Zeller	Mon Mar 12 20:46:19 2012	Initial protocol handling and parsing
Benjamin Zeller	Thu Mar 8 00:59:56 2012	some fixes
Benjamin Zeller	Wed Feb 22 18:35:30 2012	Layout works
Benjamin Zeller	Wed Feb 15 18:40:53 2012	Netstring in webclient works, initial communication works
Lukas Michalski	Wed Feb 15 16:54:53 2012	Renaming file
Lukas Michalski	Wed Feb 15 16:52:21 2012	Initialimport

7.5 Logbuch Dokumentation (git)

Name	Date	Comment
Benjamin Zeller	Fri Apr 27 15:32:17 2012	removed temp files
Benjamin Zeller	Fri Apr 27 15:31:37 2012	some flows and explanations
Lukas Michalski	Fri Apr 27 13:01:49 2012	Correction part 2
Lukas Michalski	Fri Apr 27 12:59:16 2012	Correction
Lukas Michalski	Fri Apr 27 12:53:40 2012	Writing this document
Lukas Michalski	Fri Apr 27 10:07:48 2012	Adding Logbuch of Thorsten Schwalb
Benjamin Zeller	Fri Apr 27 04:52:42 2012	Merge branch *master* of gitorious.org:p-ms/pms-docs
Benjamin Zeller	Fri Apr 27 04:52:24 2012	Recreating docs
Lukas Michalski	Fri Apr 27 04:52:13 2012	adding database stuff
Benjamin Zeller	Fri Apr 27 04:51:41 2012	Recreated docs
Benjamin Zeller	Fri Apr 27 04:51:02 2012	Merge branch *master* of gitorious.org:p-ms/pms-docs
Benjamin Zeller	Fri Apr 27 04:50:57 2012	Finishing Docs
Benjamin Zeller	Fri Apr 27 04:47:18 2012	Finishing docs
Lukas Michalski	Fri Apr 27 03:10:46 2012	More doc
Lukas Michalski	Fri Apr 27 02:39:36 2012	Adding Logbuch
Lukas Michalski	Fri Apr 27 02:21:06 2012	Additional documentation
Benjamin Zeller	Thu Apr 26 23:44:54 2012	Admin docs
Benjamin Zeller	Thu Apr 26 11:14:14 2012	Doku
Benjamin Zeller	Thu Apr 26 00:07:29 2012	Stuff
Benjamin Zeller	Thu Apr 26 00:05:19 2012	Image
Benjamin Zeller	Wed Apr 25 23:56:25 2012	Merge branch *master* of gitorious.org:p-ms/pms-docs
Benjamin Zeller	Wed Apr 25 23:54:55 2012	Anpassung der Doku an die Vorgaben
Thorsten Schwalb	Wed Apr 25 19:18:19 2012	Anmerkungen ergaenzt
Benjamin Zeller	Wed Apr 25 14:10:25 2012	Some Changes
Benjamin Zeller	Wed Apr 25 14:08:18 2012	More docs
Benjamin Zeller	Wed Apr 25 11:21:05 2012	Adding Docs
Benjamin Zeller	Wed Apr 25 02:30:10 2012	Feintunging
Benjamin Zeller	Tue Apr 24 22:08:47 2012	Merge branch *master* of gitorious.org:p-ms/pms-docs
Benjamin Zeller	Tue Apr 24 22:06:01 2012	Stuff
Thorsten Schwalb	Tue Apr 24 21:47:30 2012	Bilder und Aenderungen

Name	Date	Comment
Thorsten Schwalb	Tue Apr 24 20:10:13 2012	Manual
Thorsten Schwalb	Tue Apr 24 18:49:10 2012	Beschreibung Webclient ergaenzt
Benjamin Zeller	Tue Apr 24 14:29:11 2012	More docs
Thorsten Schwalb	Mon Apr 23 16:50:03 2012	Screenshots eingefuegt
Benjamin Zeller	Sun Apr 22 15:56:28 2012	My own thoughts
Benjamin Zeller	Sat Apr 21 00:01:55 2012	Here it belongs
Benjamin Zeller	Tue Apr 10 19:49:09 2012	Merge branch *master* of gitorious.org:p-ms/pms-docs
Benjamin Zeller	Tue Apr 10 19:48:38 2012	Saving Ideas
Lukas Michalski	Sat Feb 11 11:22:08 2012	PreAlpha Database structure
Benjamin Zeller	Sat Feb 4 11:04:54 2012	Multiple Requests
Benjamin Zeller	Wed Feb 1 21:08:12 2012	Websocket Test
Benjamin Zeller	Wed Feb 1 14:11:49 2012	Research work
Benjamin Zeller	Mon Jan 30 20:55:43 2012	logfiles
Benjamin Zeller	Mon Jan 30 20:53:27 2012	Initial Import

7.6 Logbuch Thorsten Schwalb

Name	Date	Comment
Thorsten Schwalb	Thu Feb 23 2012	Erstellung von admin.html
Thorsten Schwalb	Thu Feb 23 2012	login_success.html
Thorsten Schwalb	Fri Feb 24 2012	post.html
Thorsten Schwalb	Sat Feb 25 2012	Bearbeitung von admin.html
Thorsten Schwalb	Sun Feb 26 2012	Bearbeitung von fill_database.pl
Thorsten Schwalb	Wed Feb 29 2012	Oberfläche Tk
Thorsten Schwalb	Wed Feb 29 2012	Erstellung von admin_css.html
Thorsten Schwalb	Mon Mar 12 2012	Erstellung von user_delete.cgi
Thorsten Schwalb	Wed Mar 14 2012	Backlog
Thorsten Schwalb	Wed Mar 21 2012	Bearbeitung backlog.cgi, style.css
Thorsten Schwalb	Sat Mar 31 2012	Einarbeitung in die Verwendung des CSS-Frameworks Bootstrap
Thorsten Schwalb	Sat Apr 14 2012	login_false.html
Thorsten Schwalb	Sun Apr 15 2012	Erstellung Datenbank Dump und versch. Stored Procedures,GIT
Thorsten Schwalb	Wed Apr 18 2012	Dokumentation
Thorsten Schwalb	Fri Apr 20 2012	Dokumentation
Thorsten Schwalb	Mon Apr 23 2012	Dokumentation
Thorsten Schwalb	Wed Apr 25 2012	Dokumentation

7.7 Logbuch Enviroment

Name	Date	Comment
Lukas Michalski	Thu Apr 26 2012	Install new virtual Maschine
Lukas Michalski	Web Apr 25 2012	Decided to change distribution
Lukas Michalski	Mon Apr 23 2012	Further errors on system
Lukas Michalski	Wed Apr 11 2012	Troubleshooting errors
Lukas Michalski	Fri Mar 23 2012	Examine different Webservers
Lukas Michalski	Sat Mar 3 2012	Restructure folders
Lukas Michalski	Tue Feb 14 2012	Create folder structure on Host
Lukas Michalski	Thu Feb 09 2012	Reconfigure and optimize system
Lukas Michalski	Fri Feb 3 2012	Install virtual Maschine for Project

7.8 Logbuch MySQL

Name	Date	Comment
Lukas Michalski	Wed Apr 26 2012	Final database design
Lukas Michalski	Wed Apr 26 2012	Fix stored procedure error on import
Lukas Michalski	Wed Apr 25 2012	Work on stored procedure
Benjamin Zeller	Wed Apr 25 2012	Work on stored procedure
Lukas Michalski	Wed Apr 15 2012	Create SQL Statements for PMS-Admin
Benjamin Zeller	Wed Apr 15 2012	Create SQL Statements for PMS-Admin
Lukas Michalski	Wed Mar 14 2012	Next database design
Lukas Michalski	Wed Jan 13 2012	First database design
Benjamin Zeller	Wed Jan 30 2012	Collecting initial ideas
Lukas Michalski	Wed Jan 30 2012	Collecting initial ideas

Chapter 8

Quellenverzeichnis

8.1 Perl Pakete

- AnyEvent
- AnyEvent::DBI
- CGI::Session
- JSON
- HTML::Template
- Object::Event
- Protocol::WebSocket

8.2 Fremdsoftware

- Qooxdoo JavaScript Framework SDK 1.6
- Bootstrap CSS & JavaScript
- JQuery
- MySQL
- Lighttpd

8.3 Versionsverwaltung

- git
- gitorious

8.4 Dokumentation

- naturalDocs
- asciiDoc
- dblatex
- gimp

8.5 IDE

- kate