CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Nuclear Sciences and Physical Engineering
Department of Mathematics

Modular Content Management System for Creation and Operation www/intranet

Bachelor's Degree Project

Author:	Marek Faltýsek
Advisor:	Ing. Petr Vokáč
Academic year:	2004/2005

Prague – 2005

Declaration

I hereby declare that I evolved this Bachelor's Degree Project by myself and that all sources used are listed as references.

Prague 31th May, 2005

____

Acknowledgements

I wish to thank Ing. Petr Vokáč for his advice, suggestions and comments regarding the subject matter, and Mrs. Alena Faltýsková and Mr. R. Conor O'Neill, M.A. for their support regarding the linguistic aspects of my work.

Abstrakt

Cílem této práce je vytvořit modulárni, jednoduše rozšiřitelný nástroj pro tvorbu, distribuci, správu a administraci informací pomocí www prohlížeče, fungující různými způsoby pro různé uživatele a skupiny. Po první části věnující se úvodu do problematiky a popisující vlastnosti použitých technologí, je v druhé části pojednáváno o jádru systému, datovém modelu, přístupu k datovému zdroji a popisují se zde podrobněji i jednotlivé části zdrojového kódu. Třetí část je dokumentace pro administrátora a dokumentace k modulu starající se o prezentaci www stránek.

Abstract

The aim of this work was to create a modular, easily scaleable tool for generating, distributing and administering information by means of a www browser, functioning in different ways for different individual users and user groups. The first part includes introduction to the problem and description of the properties of the technologies used, the second deals with the core of the system, the data model, access to the data source, and provides more detailed information about the individual parts of the source code. The third and final part consists of documentation for the administrator and documentation relating to the module which caters for the presentation of the www pages.

Contents

List of Figures

Part I

Introduction

Chapter 1

Introduction

1.1 Content Management System

With the expansion of information technologies, especially of the internet, it has become more important to share various kinds of information among all its users. One of the ways of making this possible is a *Content Management System.*
Content is defined as any type of digital information, such as, text, graphics, archives, PDF files, etc.
Management System is defined as a system which allows a user to effectively manage something.

There are plenty of Content Management Systems available for download nowadays. They are run on many platforms, are programmed in different languages, and use different data sources. Examples include PHPNuke, OpenCMS, and Plone, for example. All of these systems exist to make creating web sites easier even for the amateur.

1.2 Modular CMS

Now it is quite clear what the Content Management System really is. The difference I feel between "normal" content management system and "modular" content management system is that the modular system should be somehow logically divided. For example, the system should be divided into one main part - the *Core of the system* and everything else in the system, particular operations, should act as stand-alone modules.

The main advantages of the system written like this are

- Easier maintainance
  It is much easier to trace errors, repair them, etc. if the system is divided into modules rather than in contained in a single complete unit.

- Easier upgrading
  Separate upgrades rather than upgrading the whole system each time

It is also much easier for collaborative development to take place in the future. Person A can be responsible for developing the part of the system responsible for communication with a database, for example, while person B can be responsible for formatting outputs for end users.

1.3 Goals of this project

The purpose of this project is to create such modular content management system. The system core should be stored as files on a www server, while the modules, including their source codes, should be stored in the database server. This markedly speeds up their development and makes administration more comfortable. The system should also make it possible to display to one group of users modules which others do not see, while showing another group completely different modules, which are in turn concealed to the first group. A system designed in this manner will make it posible easily to create a countless number of different applications working by means of shared connection to the database, and at the same time use the same differentiation/division of users into groups.

1.4 Core of the system

This part of the system should be responsible for all properties and actions common to all the modules. One of the most important functions of the core is to link communication between the particular modules and the database. A given module should be unconcerned about what exact data source it is using, what exact manufacturer produced the db server software, what exact version is currently running, etc. The given module is just supposed to give simple order that it wants to save data to the database or that it wants to get certain results.
This is job for a component called a *DB Access Component.* If it is decided that a DB server should be replaced by another in the future, the only interference to the code is the replacement of this component. Authentication and role-based access is also often needed. Therefore, another function of the core should be some kind of mechanism to let a user log in, log off, and to allow particular users to access elements which are hidden, i.e., not accessible to other ones.

1.5 Modules

Nearly anything could act as a module. There could be a Module for sending short messages between users, a module for creating and maintaining forums, module for user management, module for maintaining bug database, etc. These are just a few examples of the functionality of modules.
Simply put, anything that is dependent on the system while the system is not dependent on it, should be considered a module.

1.6 Technology Used

As has been stated, the described architetcure will need some Relational Database Management System (RDBMS) and some hypertext preprocessor of HTML outputs. There are a few alternatives how to consider for this purpose. I have chosen PHP server side hypertext preprocessor and MySQL with InnoDB Engine as a data storage, because PHP has excellent web documentation, and both (PHP and MySQL) are open sourced software. Because all the system will be managed and displayed through web browser, a few more technologies will be used. HTML used for formating outputs, CSS for making these outputs attractive and flexible, and JavaScript often used for automation HTML forms. A few following sections which are taken over from various encyclopaedia on the internet and abbreviated tries to describe what PHP, MySQL, CSS, HTML and JavaScript is.

1.6.1 PHP 5

PHP is a server-side Hypertext Preprocessor. It was created sometime in 1994. PHP script is enclosed in an HTML document within special PHP tags. This script is executed after a particular file is opened, processed, and the output is sent to HTTP output. Because the script is executed by the server, there is no possibility for a client to view the source code. Thus, it is secure. Nowadays, there is a large number of PHP libraries available for communication with other protocols (IMAP, SNMP, NNTP, POP3, FTP) or with various RDBMS (MySQL, PosgreSQL, Oracle, Sybase). Its syntax is similar to Perl or C. Many improvements including object oriented programming were made with the release of version 5. PHP is written as Open Source.
More information can be found on http://www.php.net

1.6.2 MySQL with InnoDB engine

MySQL is a relational database management system (RDBMS). RDBMS means that the data are stored in separate tables rather than in one big area. These data are created, modified, and acquired by Structured Query Language (SQL), which is the most common

language used to access databases. MySQL is also open sourced as PHP is and is the most commonly used open sourced RDBMS in the world.
InnoDB provides MySQL with a transaction-safe ACID (Atomicity, Consistency, Isolation, Durability) compliant storage engine with commit, rollback, and crash recovery capabilities. Another helpful feature which InnoDB supports are the foreign keys constraints.
More information is at http://www.mysql.com

1.6.3 HTML

HTML - HyperText Markup Language is the coding language used to create Hypertext documents for use on the WWW. Basically it is a set of <markup> symbols or codes inserted in a file that tells the Web browser how to display a Web page's words and images for the user. The individual mark up codes are referred to as elements.

1.6.4 CSS

A cascading style sheet (CSS) is used to define how web pages are displayed. The purpose is to provide more control over the fonts, colours, layout, etc. that go into the web page than could be provided by raw HTML. Also, since the cascading style sheet file is separate from the HTML files, it can be shared (or even inherited; a little outside the scope of this document) by multiple web pages to help provide a consistent look-and-feel across a web site.

1.6.5 JavaScript

JavaScript (in spite of its name) has nothing whatsoever to do with Java. JavaScript is an interpreted language built into a browser to provide a relatively simple means of adding interactivity to web pages. It is only supported on a few different browsers, and tends not to work in exactly the same way on different versions. Thus its use on the Internet is somewhat restricted to fairly simple programs. On intranets where there are usually fewer browser versions in use, JavaScript has been used to implement much more complex and impressive programms.

Part II

Analysis

Chapter 2

Core of the System

The Core of the system consists of three parts. The first is responsible for communication between the modules and the database. It is called DB Access Component.
The second is responsible for running all the modules, i. e. creating the so called forms, mapping them with the database, determining which action is supposed to be executed, and after that its processing, and acquiring data from the db and their parsing using a template in a particular view.
The last part is the so called Administration Module and is responsible for managing forms with their actions, and views. This module is also responsible for granting access to particular elements of the system.

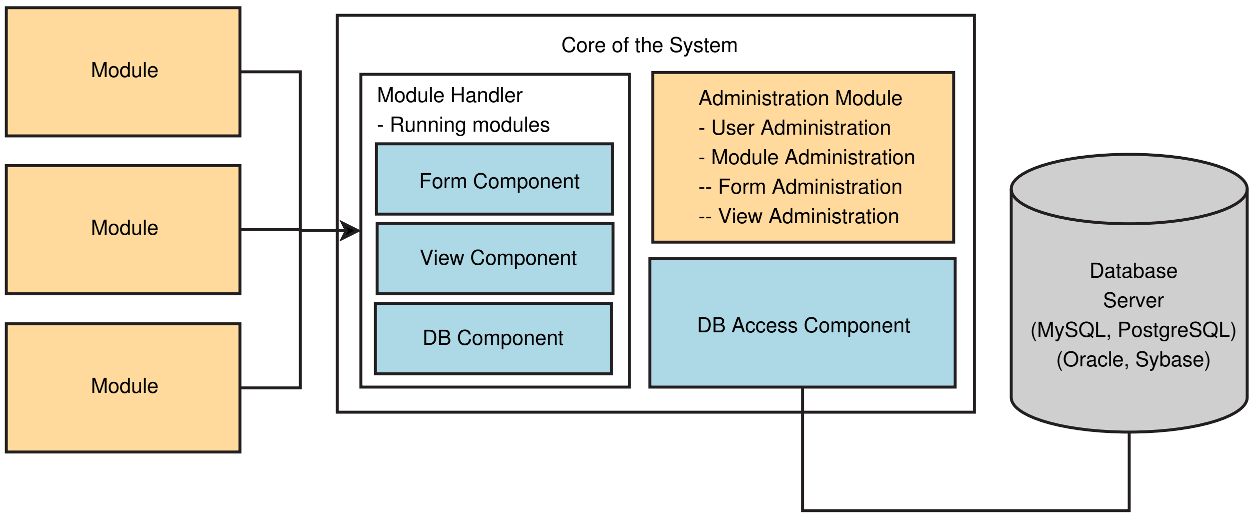


Figure 2.1: Core of the system - Logical view

2.1 DB Access component

As written above, the DB access component is responsible for linking the Database Server with the modules. This component should be designed in such a way as to make it easily replaceable by another db component, in case the Relational Database Management System changes. For instance, PostgreSQL instead of MySQL. Even though the majority of today's RDBMS SQLs are more or less standardized to comply with SQL92 (ANSI/ISO SQL92, ISO/IEC 9075:1992(E) Information Technology), there are still some differences between them. Therefore it is neccessary to create such interface for communication between the modules and the core which will be included in all these DB Access Components (component for MySQL, component for PostgreSQL, component for Oracle, etc.)
Every particular component consists of three classes. The first is used for direct communication with the server, the second is used for parsing given results, and the third for active mapping of rows in the DB table with the particular object in memory.

2.1.1 DB_Request class

To create this class, several attributes: host - hostname or IP address of the computer where the RDBMS is run, username, password and the name of the particular database where all the system tables are saved. Having this, one directive is simply placed into the code. The best way is to put it at one place from where it is distributed throughout the whole code by means of several includes. Configuration file called *config_mysql.inc.php* located in */config* directory would be appropriate.

```
$dbh = new DB_Request(String host, String username,String password,String dbname);
```

This directive creates the so called handle, with the aid of which all the operations are later executed.

Variables

Every instance of the DB_Access class has 5 protected variables.

- **Username**
- **Password**
- **Host name**
- **DB Name**
- **Database Handle**

Methods

- **public __construct(Username, Password, Hostname, DBName)**
  Constructor of the class. All that happens inside the class when it is being created is allocation of the respective values to the above mentioned variables.

- **protected connect()**
  This method is called when connection to the database is required. In case of error, an exception is thrown.

- **public execute(query)**
  By means of this method, an SQL query is run. Firstly it is found whether the DB handle exists. The handle is created only once per each HTTP session. If there have been any SQL executions earlier, the handle is already created. If not, a new handle is created. Once we have a handle, the given query is executed. The obtained results are saved in the so called associative array, i. e. array([line][array([attribute][value]])

2.1.2 DB_Result class

DB result class is used for parsing results given from *execute()* function of the DB_Request class. It is simple to access particular documents through this class.

Variables

There are 4 variables within this class.

- **lines** - results formated as an array of DB_Lines
- **fetch_array** - results formated as associative array
- **pos** - current position in the result
- **count** - number of lines in the result

Methods

- **Constructor(fetch_array)**
  The only parameter of this method is the associative array returned by the *execute* function of the DB_Request class. First of all it is neccessary to decide whether *fetch_array* results from SQL SELECT, i.e., has at least one element (row), or from SQL INSERT, SQL UPDATE or SQL DELETE. (or, of course, from SQL SELECT with no returned results). If it comes from SELECT, all rows of the result are processed while single DB_line instances are being created of which each belongs to a particular row of the returned result.

The next step is to adjust the count variable to the number of returned results. Finally, the results are saved in the form of associative array in the variable called *fetch_array*, and the *position* is set at 0.

- **getnext()**
  This method is used to return next DB_Line object from the result. If there is no next, the method sets *position* to zero and returns FALSE.

- **reset()**
  This method just resets position counter to zero.

- **__get()**
  This function is used for easier access to the first row of the result. It works through operator overloading. When it recieves a request for a non-existing attribute of the DB_Result class, the function tries to return the value of the given attribute from the first row in the result. For instance, if the result is an array of all users in the system, it is easy to get the username of the first user just by calling for DB_Result.username. This is very helpful when looking for a single document in the database.

2.1.3 DB_Line class

DB_Line class is designed to represent a row from a table as an object instance. For example, if there is a table called users in which the users with their attributes are saved, the instance of this class has similar structure as the particular row in the table, i. e. particular user. We can easily get attributes such as username by simple call $user.username, etc. This technique is called active mapping.

Variables

- **data** - Array of attributes and their values representing one row in the table.

Methods

- **Constructor(line_array)**
  Given array looks like array[attribute]=value

- **__set(attribute, value)**
  Method called when overloading the class - sets value of the attribute

- **__get(attribute)**
  Method called when overloading the class - returns value of the attribute

- **delete(table, key_name, key_value)**
  Tries to find a row in the *table* with *key_name* having *key_value* and deletes the document. *Key_name* and *key_value* can acquire multiple values. If this happens, these attributes must have the form of array.

- **save(table,key_name,key_value)**
  Firstly, this method obtains description of the table from the database, assigns existing values of the current class instance to real, possible attributes of the row in the table, and after discovering whether a particular row already exists, it determines between insertion and updating. Again, key_name and key_values can acquire multiple values.

All the classes cooperate with one another. DB_Request is needed for creating DB_Result, which is in turn needed for creating DB_Line. There is one exception. There is no need to create the instance using an array in the constructor when wanting to create a new document.

2.1.4 Class diagrams

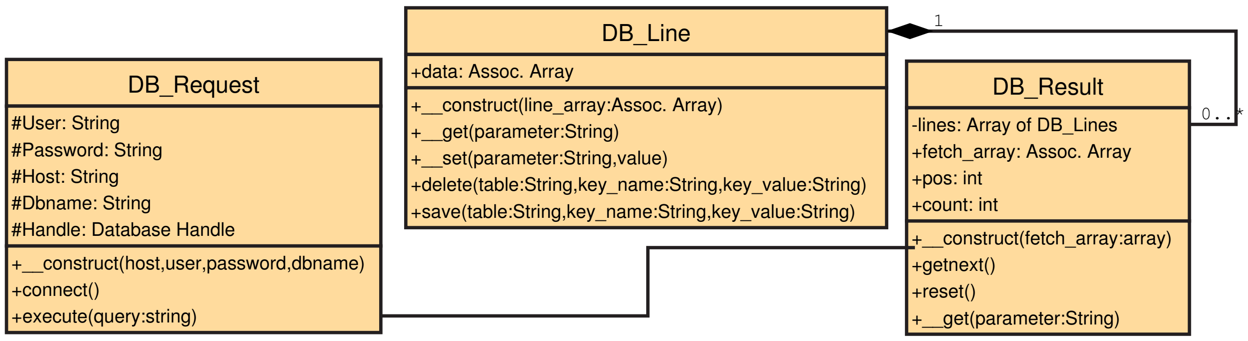

DB_Request
#User: String
#Password: String
#Host: String
#Dbname: String
#Handle: Database Handle
+__construct(host,user,password,dbname)
+connect()
+execute(query:string)
DB_Line
+data: Assoc. Array
+__construct(line_array:Assoc. Array)
+__get(parameter:String)
+__set(parameter:String,value)
+delete(table:String,key_name:String,key_value:String)
+save(table:String,key_name:String,key_value:String)
DB_Result
-lines: Array of DB_Lines
+fetch_array: Assoc. Array
+pos: int
+count: int
+__construct(fetch_array:array)
+getnext()
+reset()
+__get(parameter:String)
1
0..*

Figure 2.2: DB Access Component - Classes

2.2 Data model

2.2.1 Entity Relationship Model

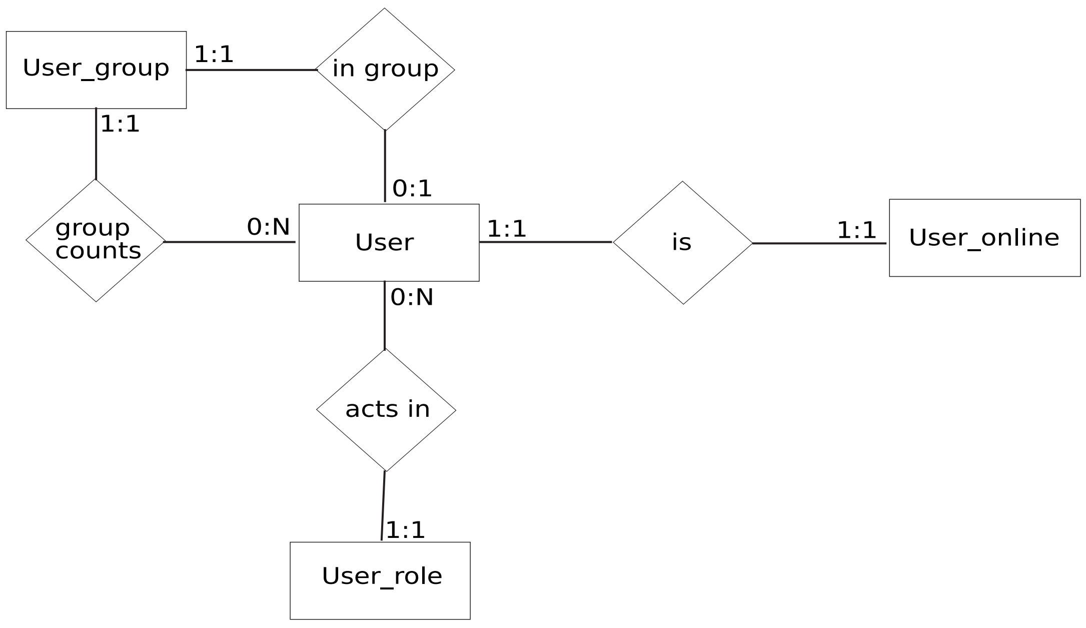


Figure 2.3: Core of the system - ER diagram - User, group, user roles

2.2.2 Tables

User

Field ID	**Type**	**PK**	**FK**	**Unique**
user_id	INT(3)	Y		
username	CHAR(20)			Y
password	CHAR(20)			
name	CHAR(30)			
email	CHAR(64)			
icq	INT(10)			
hidden	TINYINT(1)			

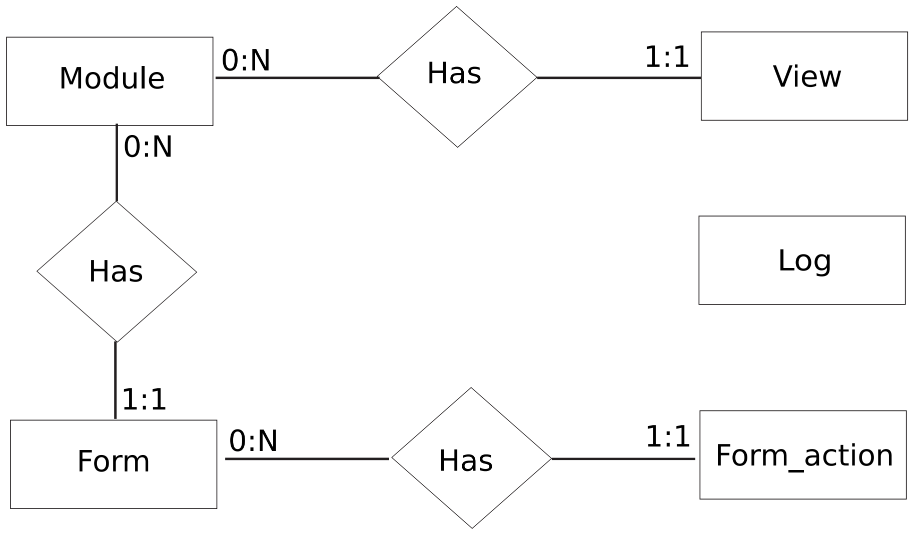


Figure 2.4: Core of the system - ER diagram - Module, Form, View

User_group

Field ID	Type	PK	FK	Unique
group_id	INT(3)	Y		
username	CHAR(20)		User.username	
group_name	CHAR(20)		User.username	

User_online

Field ID	Type	PK	FK	Unique
username	CHAR(20)	Y	User.username	
last_operation	INT(10)			

User_role

Field ID	Type	PK	FK	Unique
role_id	INT(4)	Y		
role_name	CHAR(20)			
actor	CHAR(20)		User.username	
modue_name	INT(3)		Module.module_name	

Module

Field ID	Type	PK	FK	Unique
module_name	VARCHAR(32)	Y		
module_code	TEXT			
module_display_text	VARCHAR(12)			
module_additional_code	TEXT			
module_icon	VARCHAR(64)			

Form

Field ID	Type	PK	FK	Unique
form_name	VARCHAR(32)	Y		
module_name	VARCHAR(32)	Y	Module.module_name	
form_template	TEXT			
action_parameter	VARCHAR(8)			

Form_action

Field ID	Type	PK	FK	Unique
form_name	VARCHAR(32)	Y	Form.form_name	
action_name	VARCHAR(32)	Y		
action_code	TEXT			
action_role	VARCHAR(20)			

View

Field ID	Type	PK	FK	Unique
view_name	VARCHAR(32)	Y		
module_name	VARCHAR(32)	Y	Module.module_name	
select	TEXT			
view_key	VARCHAR(32)			
view_header	TEXT			
view_template	TEXT			
view_footer	TEXT			
num_display_elements	INT(2)			
roll_parameter	VARCHAR(8)			

2.3 Module Handler

To follow the description of Module Handler, which is responsible for running the modules, it would be usefull to understand what the module is. As mentioned above, everything that depends on the system core while the system core is not dependent on it should be a module.

Every module has its own source code, which is written in PHP. This code is executed when the given module is run. Even though this code can be stored in a file in modules directory of the directory tree, the most of the modules are stored in the database. This is very useful, for example, when

- **Upgrading** - There is no need to change files on the web server. New modules can be also installed much more easily.
- **Administration** - Administrator can change the code from any computer having a web browser
- **Access control** - Every module is bound with a certain role. The identifier of this role is automatically set to the name of the module.

Every module refers to particular record in database table called *Module*. This record consists of module name (unique identifier), module code, text displayed in the listing of all modules (in the menu, for example), additional code (code which is executed in listing all the modules) and module icon.
To clarify what is the additional code, let us use an example. There is a module for sending and receiving short messages among users. All the modules are displayed as a menu on the top of the page. If a such user has unread messages in his mailbox it would be usefull to inform him about this fact. Therefore, the additional code of the mail module contains a PHP code which obtains the number of unread messages in current user's mailbox and prints it out.

The particular module which should be run is determined on the basis of URL parameter *go*. For example, when the value of URL is *index.php?go=Admin*, a module identified as *Admin* is called.

Apart from name, code, additional code, etc. every module consists of its tables and also two special elements.

Forms

These elements are used to

- Display HTML forms
- Process sent data (create, update, delete documents)
- Acquire data (display one particular document)

Each form corresponds to a particular DB_Line object which is mapped on the database. Every form has its own template which determines what it looks like in HTML browser. (for more details about templates see section 2.4) Each form has its own actions. By means of these actions, it is possible to work with the document quite easily. The identifier of particular action is given in URL. All information about the form, its template, codes of its actions, etc.is stored in database.

Views

Views are used for formatting the results returned from the query. Templates are also very important for understanding what views really are. (see section 2.4). In the same way as forms, all the information about views is stored in database.

2.3.1 Module Class

Each module in the system is run by a class called *Module*.

Attributes

- **Module Name** - Name and identifier of the module
- **Module Code** - PHP source evaluated when running the module
- **Database Handle** - Handle used for communication with database

Methods

- **Constructor(Module name)**
  The first thing done is assigning database handle which is stored in HTTP session. Then the class attempts to find the module of the given name. In case it is not found, the constructor throws a message and halts the evaluation. If the module is found, attributes such as *module_code* and *default_access* are assigned. At the end of creation, method *run()* is called.
- **run()**
  First of all it is determined whether the current user has enough permissions. This is done by means of attribute *default_access*. If the attribute equals 0, function *notRestricted()* is called, in other cases, the *isAllowed()* function is used. If everything regarding permissions is found in order, the code in *Module_code* is evaluated.
- **notRestricted(Element)**
  Returns TRUE if the element is not restricted for the current user. This is done by means of method *access()* which is in the *Authentification* class.
- **isAllowed()**
  Very similar to *notRestricted()* with only one difference. If there is no record about the specified element in *Access* table, FALSE is returned, i. e. the user is not allowed to

access the particular element. On the other hand, if this situation occurs when calling the method *notRestricted()* TRUE is returned.

- **includeForm(Form name)**
  Used for including form into module code.
- **includeView(View name)**
  Including View in module code
- **insertView(View name)**
  Similar to *includeView()*, with the difference that the output of the view is not printed out immediatelly, but only returned as String. This is essential when inserting a particular view into the Form.
  This method is often used in templates. To understand it properly, please see the section 2.4 about *Template data type*

2.3.2 Form Class

Each form is represented by a class called *Form*. One of form variables is a handle to the particular DB_Line object which is mapped on the database. Thus, variables of the related DB_Line element can be easily obtained. Meta information about each form is saved in the database, in a table called *Form*.

Attributes

- **Form Name** - Name and also an identifier of the form
- **Module Name** - Name of the module to which the current form belongs
- **Action Parameter** - URL parameter used for determining which action should be used
- **DB Keyname** - Unique Key (or array) by means of which a particular line can be found
- **DB Keyvalue** - Value (or array of values) of the above mentioned key
- **DB Table** - Name of the table in which a particular row should be located
- **Form Action** - Multiple actions associated with current form; acquired from *Form_action* table
- **Form Template** - Source of the HTML form with embedded values in { }

Methods

- **Constructor(Form name, Module)**
  Firstly, several variables such as *DB_handle*, *Module* and *Module name* are allocated. Another step of this method is to load *HTML form template* and *action parameter* from the database. Finally, all related form actions are inserted into *Form Action* array.

- **__get(Attribute)**
  Method using operator overloading. Returns value of given attribute of bound DB_Line object.

- **__set(Attribute, Value)**
  Again, uses operator overloading. Adjusts given value to the given attribute in bound DB_Line object

- **process()**
  This method determines which of the form actions should be used. It tries to evaluate the action whose identifier corresponds to the value of the previously defined action parameter.

- **drawForm**
  Prints out the source of the HTML form.

- **createElement(Table, Keyname, Wheretogo)**
  This method is used for creating a new entry in the table. It is used when there is no equivalent stored in the database and needs to be create it. After the operation is finished, browser is redirected to the specified URL (*wheretogo* attribute)

- **findElement(Table, Keyname, Keyvalue)**
  Tries to map existing row in the table with current instance of the object. Keyname and Keyvalue can acquire multiple values.

- **updateElement(Table, Keyname, Wheretogo)**
  This method is used for updating already mapped DB_line with attributes held in the current form instance object. If the appropriate document is found in the database, it is updated, otherwise a new row in the table is created.

- **deleteElement()**
  Deletes mapped row in the table. If there is no *Confirm* parameter in URL, it automatically requires confirmation.

2.3.3 View Class

Attributes

- **View name** - Name and also identifier of the view
- **Module** - Pointer to the parent module
- **Select** - SQL Select
- **DB Result** - Returned results from the query formated as DB_Result
- **View header** - Template of the view header
- **View Template** - Template of each row of the view
- **View Footer** - Template of the footer of the view
- **num_display_elements** - Default number of displayed rows in a view
- **pos_display_elements** - Temporary variable - current position in a view
- **roll_parameter** - URL parameter used for scrolling pages

Methods

- **Constructor(View_Name, Module)**
  Assigning view variables and acquiring view information from the database. This method tries to find a particular view from a table called *View*. If there is one, it assigns other attributes to the instance and afterwards executes SQL query stored in the database. These results are then saved in *DB_Result* attribute.
- **getView()**
  This function returns formatted results of the view. Every result starts with *View header* and ends with *View footer*. Rows in-between are processed by means of a template stored in *View template*.
- **drawView()**
  Runs preceding function *getView()* and prints out the result.
- **drawSimple()**
  Used for printing out a view which has no template. This function firstly gets the structure of the results with all possible fields, and then it prints it out.
- **drawListing()**
  This method is used for printing out a bar with the *Previous* and *Next* buttons. The name of the URL parameter is stored in *roll_parameter* variable, default number of

displayed documents is stored in a variable called *num_display_elements* and starting position is obtained from a variable called *pos_display_elements*. After having these, SQL query is executed with limitation consisting in starting position and the number of documents.

2.4 Templates

Quite often, there is a need to display something a number of times when changing only data. This is the reason for the existence of the Template data type. Template is a fragment of HTML code with embedded identifiers of variables which contains data. These identifiers are enclosed in {} marks.
The fragment is then passed to function called *evalTemplate()* which is located in */functions/globals.inc.php*.
By means of this function the given fragment is split into HTML code and PHP code, which is embedded in {}. After this, everything that is inside { and } is processed by means of PHP function *eval* and the original expressions are replaced by the given results. The outcome is then returned as a string.

Exaple of a template

Let's imagine there is a HTML template stored in $template variable.

```
<table>
   <tr>
      <td> Username </td>
      <td> {$username} </td>
   </tr>
   <tr>
      <td> E-mail </td>
      <td> {$email} </td>
   </tr>
</table>
```

if there is a PHP code which contains

```
$username = "Bob";
$email = "bob@yahoo.com";
echo evalTemplate($template);
```

The result in a web browser of this code will be

Username	Bob
E-mail	bob@yahoo.com

However, in this example the context in which function *evalTemplate()* is called is clear. Very important for comprehending the correct use of using templates in forms and views is to understand the context in which the function is being run. When the function is processing a template for a form or a view, the only variables which can be accessed "directly" are those within this function. Therefore, function *evalTemplate()* needs a few more arguments. The first is a pointer to a current module whose attributes and functions can be accessed via $mod (for example `$mod->module_name)` and the second is a pointer to the currently processed document. Document properties can be accessed via $doc variable (for example `$doc->username).`

Embedding conditions

It is possible to process embedded conditions. The syntax is `(Condition)?(TRUE):(FALSE)` In the next example, if the variable $name is empty, string "Name is empty" is printed out.

```
{ ((!$name)?"Name is empty":$name) }
```

Template in views

Templates in views are used for displaying a list of rows (documents) which were returned by some SQL select. Every view has its own header - fragment of HTML code which is displayed just before all the particular documents, its footer - a fragment of HTML code which is displayed just after all documents have been printed out, and the most important fragment stored in a field, called *View_template*. This template is processed for each row in the returned result.

Example There is a request to display all users and their e-mails in the database in a HTML table. There is a view with suitable SQL select which obtains the results. We want the system to print out the header with names of the fields. Therefore, *View_header* will look like this:

```
<table>
   <tr>
      <td> Username </td>
      <td> E-mail </td>
   </tr>
```

To format every row of the result the template should look like this:

```
   <tr>
      <td> {$doc->username} </td>
      <td> {$doc->email} </td>
   </tr>
```

Every HTML table has to be also closed. Variable *View_footer* will contain only

```
</table>
```

An example of the result of these templates when having two users (Bob, John) will look like this:

```
<table>
   <tr>
      <td> Username </td>
      <td> E-mail </td>
   </tr>
   <tr>
      <td> Bob </td>
      <td> bob@yahoo.com </td>
   </tr>
   <tr>
      <td> John </td>
      <td> john@hotmail.com </td>
   </tr>
</table>
```

Template in forms

Templates are also used in forms. The reason is that one form can be used in multiple situations. One form can be used for creating a new user while the same one is also suitable for editing the user. The only difference is in the presence of data. As in view, module and document variables can be accessed. A simple example will illustrate the usage of a template in a form.

```
<form method="POST" action="./index.php?go={ ($doc->doc_id=="")?"create":"update" }
<input type="HIDDEN" name="doc_id" value="{$doc->doc_id}">
Document name: <input type="text" name="doc_name" value="{$doc->doc_name}"><br>
Document author" <input type="text" name="author" value="{$doc->author}"><br>
<input type="submit" value="Save">
</form>
```

If this form is used for creating a new document, action url will contain *go=create* instead of *go=update*. Let's assume that attribute *$doc_id* is assigned during the first processing of the form (after submitting the form). Because of this, if the same document is opened with the same form next time, attribute *$doc_id* is already assigned and the above condition in action returns string ”update”. The rest of the correspondent fields will be predefined by means of the HTML attribute VALUE.

Inserting view into the form template

Sometimes it is required to insert a view into the form template.
Everything enclosed in { and } should only return some value. This is done by method *InsertView(View_name)* which is included in Module class. Let's imagine, a view called *User_select* which displays all the users in the system in the form of HTML SELECT field. This view has the following attributes.

View_header:

```
<select id="recipient">
```

View_footer:

```
</select>
```

View_template:

```
<option value="{$doc->username}"> {$doc->username} </option>
```

Then there is a form for sending short messages to a specified user, and it is more comfortable to select among all users than to write down their username manually. A template for a form which sends these messages will contain the following:

```
<form method="post" action="./send.php?action=send">
   Recipient: { $mod->insertView("User_select") }
   <hr>
   <textarea name="message" rows="5" cols="40">
      Write your message here
   </textarea>
</form>
```

The final result in case there are two users will look like

```
<form method="post" action="./send.php?action=send">
   Recipient:
      <select name="recipient">
```

```
         <option value="bob"> bob </option>
         <option value="john"> john </option>
      </select>
   <hr>
   <textarea name="message" rows="5" cols="40">
      Write your message here
   </textarea>
</form>
```

2.5 Authentication

2.5.1 Users and Groups

The overwhelming majority of the system should be accessible only to registered users. Thus, there has to be a functionality which allows users to log in and log off; it should also be responsible for granting particular users access to particular elements. To simplify administration, users can be divided into groups.

The class responsible for carrying out authentication is called *Authentification* class. Although this class will be described later, it is necessary to mention it now. The reason is that it needs to be considered which of several alternatives will be used to store data in the database.
In this class, there is one important method called *access()* to which username/group and identifier of accessed element are given as an argument. For example, user/group called "ABC" asks for permission to access element called "forum5".

The question is how to store information about users and groups, whether divided into two tables or kept together in just one. If the data are stored in two separate tables (one for individual users, one for groups), it is quite difficult to detect in the source code whether the type of query is user-element or group-element. A table with user roles should also include certain mechanism designed to inform the above mentioned *access()* function about the type of query (if it is role-user or role-group query). Another reason for keeping users and groups in a single table is that in such case, it is possible to have in the table with roles a "database integrity constraint foreign key", which provides some functionality for free. For example, when a particular user is deleted, all his roles are deleted as well. The reason is that it is not possible to define two different FOREIGN KEYS from two different tables for a single field in a single table. The only con of this solution is impossibility of storing additional information about particular groups, e.g. *group_description*
Hence, entry about group and entry about user is stored in one table. The difference between entry representing user and entry representing group is that in the latter case, the first letter of the username equals symbol "@". The relation between particular users and particular groups is stored in a table called *User_group*.

Example Case

We want to create users *ABC*, *DEF*, and *XYZ*, and group called *@GROUP1*. Group *@GROUP1* should include users *ABC* and *DEF*. We will need to insert four entries into table *User* and two entries into table *User_group*.

User table entries

Username	Password	Name, email, icq ...
ABC	abc's password	anything
DEF	def's password	anything
XYZ	xyz's password	anything
@GROUP1		

User_group table entries

group_name	username
@GROUP1	ABC
@GROUP1	DEF

2.5.2 User Roles

Role is the relation between the so called *element* and the so called *actor*. An *element* can be for example access to a particular module, access to its certain part (permission to edit and delete documents), permission to administer users, visibility of a certain document, etc. *Actor* can mean either a user or a group. Every role is uniquely defined by module name, user or group, and by role name. If access to the whole module is required, role name must be empty. On the other hand, if access is required to a particular function within a module, for example permission to edit or permission to delete a document, the role inside a module must be uniquely defined by *role_name*. Roles are stored in a table called *User_role*.

Example Case

Supposing there is a module for reporting bugs. There is a group called *@editors*, which includes users *ABC* and *DEF*, and another two users called *MNO* and *XYZ*, who are not members of any other group. The task is to grant access to all users except user *XYZ*, and additionally, to allow users who are members of group *@editors* to edit and delete the reports. In the source code of the bug reporting module, the identifier of the role, consisting in permission to edit, equals string "edit", and the identifier of the role consisting in permission to delete equals string "delete".
The solution relies upon insertion of 4 entries into *User_role* table. The first two will represent the relationship between general access to the module on the one hand, and user *MNO* and group *@editors* on the other. Another two entries will connect group *@editors* with role *edit* and role *delete*.

User_role table entries

Actor	Module name	Role name
MNO	Bug_reports	
@editors	Bug_reports	
@editors	Bug_reports	edit
@editors	Bug_reports	delete

2.5.3 Authentication Class

Authentication class is used to manage authentication.

Attributes

- **access_elements** - Array with all elements which the particular user is allowed to access
- **dbh** - Database Handle

Methods

- **__construct(Username, Password)**
  First of all, this method tests whether *Username* and *Password* have been given. If not, *drawForm()* function is called, in the opposite case *login()* is called.
- **login(Username, Password)**
  This method assigns db handle and queries the DB server about the user using the given username and password. After successfull reply, the rest of attributes is assigned, and via function *createSession()*, a HTTP session is created. The last step is to update information about online users.
- **updateTimeStamp()**
  This method updates information about online users and last logins.
- **logout()**
  Method used for logging off. HTTP session is destroyed.
- **access(Default Action, Module Name, Element ID)**
  Method for finding out current user's permissions to access the given *Element ID*. The first step is to get appropriate records from *User_role* table. *Default action* parameter is used to choose between two methods of access. If this parameter is set to 0 and there are no corresponding records in *User_role* table, access is granted. Conversely, if it is set to 1 and no corresponding records are found, access is denied. If any matching records in *User_role* table were found, the so called ACL (Access Control List) is created. The

next step is to find out whether ACL contains an entry connecting the element with the current username. Returns true (access granted) if it does. The last step is to discover whether the particular user comes under any of the groups permitted access. Consequently, all the groups the particular user comes under are loaded and one after another they are compared with groups which have records in *User_role* table. If any match is found, access is granted. In the opposite case, it is denied.

- **createSession**
  This method creates HTTP Session, assigns username and user id to corresponding parameters in the session, and "hangs" the whole current *Authentification* class on the session.

2.6 Other classes

2.6.1 Message Class

Sometimes, the system needs to inform a current user what exactly is happening in the system at the moment or to ask him what he wants to be done. The message class is responsible for doing all this, i. e. displaying various messages, warnings, questions, and errors. For example, information about the fact that the current user has not enough permissions to access the element he requires.
A message can be of three types:

- **Information message** - This type is used when the system needs to display information meant only for the current user, such as *Mail message has been sent.* Every message of this type may be important for the current user but it is definitelly not important for the administrator of the system.

- **Error message** - This message is used when there appears any critical error. Most of such messages relate to problems with connection to the database, missing views, missing forms, or an attempt to access something which is restricted.
  The specified view has not been found
  or
  You have not enough permissions to access this element
  are the typical examples of this type of a message

- **Warning** - This type of message is used when something goes wrong during execution but it is not necessary to stop the subsequent processing of the code.
  Message you are trying to delete has been already deleted
  is an example of this type of a message

Hovewer, apart from this typology, it is necessary to determine whether the processing of the rest of the page should be continued or stopped. For example, there is a difference between an error which occurs when a user has not enough permissions, and an error occuring when a particular view cannot be found. If someone is trying to access something that is restricted for him, the system should consider it as a security violation. On the other hand, absence of a particular view is not an error so serious as to stop the system from drawing the footer of the page. Therefore, there is a number in every message call which, if it is odd, makes the system stop. Conversely, if the number is even, the system continues the processing of the rest of the page.

All the messages are displayed to the user. Hovewer, it may be useful for the administrator to have a possibility to browse through the list of all the messages. (Except information messages). Thus, every message call of non-information type is stored into the system table called *System_log*. Every record consists of date and time, caller (from where in the code it was called), username, type, and the text of the message.

Attributes

- **err** - Identifier of the message
- **errstr** - Text of the message
- **errlnk** - Return URL

Methods

- **__construct(err,errstr,errlnk,caller)**
  Firstly it determines which of the four types of message is involved. If *err* equals 0 or 1: the message is an error, 2 or 3: information message, 4 or 5: warning, 6 or 7: question. After this, if err is 0,2,4 or 6, the system continues in evaluating the code, if not, the system halts.

2.7 Additional functions

Additional functions are saved in a file with filename */functions/globals.inc.php.*

2.8 Views which are not part of any module

Views which are not part of any of the modules.

Menu_module

Prints out all modules as horizontal menu. It is displayed at the top of the page.

Module_select

HTML SELECT of modules installed in the system

Chapter 3

Modules

3.1 Home module

Home module act as entry page to the system. It only prints information about users who are online and simple history of their logins.

Tables:

User_online
This table is redundant and its information could be stored directly in *User* table, however, because of plans of including a user-tracking system, it was inserted into the system.

Field ID	**Type**	**PK**	**FK**	**Unique**	**Description**
username	VARCHAR(20)	Y	User.username		
last_operation	INT(10)				

Views:

Users_online

A view displaying users who are on-line at the moment.

User_logins

A view displaying all users and their last logins.

3.2 Mail module

The mail module is used for sending short messages among the users of the system. Every message contains sender username, recipient username, and plain text which represents the subject of the message.

Tables:

Mail_message

Field ID	Type	PK	FK	Unique	Description
message_id	INT(6)	Y			
date	INT(10)				
is_read	TINYINT(1)				
text	text				
from	VARCHAR(20)		User.username		
to	VARCHAR(20)		User.username		

Forms:

Mail_message
This form is responsible for sending messages and deleting them. It has 3 actions:

- Default - Used when no action is called, only draws HTML form.
- Send - This action is used for sending messages.
  It assigns appropriate values from HTTP POST and creates a message
- Delete - This action is used for deleting messages.
  It recieves message_id, finds appropriate document, and deletes it

Views:

Mailbox
This view displays the mailbox. In mailbox either sent or received messages are shown. Beside every message there is a link containing url for deleting the message.
Users_select
This view is included in form *Mail_message* and displays all visible users in the system. It is printed in a form of HTML select and represents recipient field.

3.3 Forums Module

This module is used for creating, editing, deleting, and listing all forums in the system. Particular forums are then maintained by means of a different module called *Forum module*. Hovewer, these modules cooperate.

Tables:

Mail_message

Field ID	**Type**	**PK**	**FK**	**Unique**	**Description**
forum_id	INT(3)	Y			
forum_name	VARCHAR(30)				
forum_desc	VARCHAR(255)				
messages	INT(4)				
owner	VARCHAR(20)		User.username		
forum_category	VARCHAR(64)	Y			

Forms:

Forum
This form is used for creating, editing, and deleting particular forums. It has following the actions:

- Default - HTML form is drawn
- create - Determines whether forum allready exists (updating) or not (creating). Assigns appropriate values from HTTP POST and creates/updates record in the table with forums.
- edit - This action gets forum identifier from URL, finds relevant record in database, and displays HTML form

Views:

Forums
This view lists all forums in the system. It does not print the forums which are not accessible for the current user

User roles:

It is possible to restrict access to particular forums. The only way to do this is to create record in *User_role* table containing user/group and as role_name, string "f_"+[id of the forum] (*forum_id* attribute). For example, if there is a forum having *forum_id* = 15, the corresponding role_name in *User_roles* will be "f_15".

3.4 Forum module

Forum module is used for displaying a particular forum, writing messages, and deleting them. It relates to the *Forums module* described in previous section.

Tables:

Forum_message

Field ID	Type	PK	FK	Unique	Description
message_id	INT(6)	Y			
date_id	INT(10)				
text	TEXT				
username	VARCHAR(20)		User.username		
forum_id	INT(6)		Forums.forum_id		

Forms:

Forum_message
This form is used for writing plain text messages into particular forums. It has 3 actions:

- Default - Draws HTML form
- send - Creates a message in the forum. This action firstly assigns appropriate attributes from HTTP POST. The next step is to modify the number of all messages in the forum.
- delete - finds corresponding document and deletes it.

Views:

Forum
View is displaying a certain number of latest documents written to forum (10 by default). Beside each message written by the current user there is a link for deleting the message.

3.5 Profile module

This module is used for changing information about current user.

Tables:

User
This module works with *User* table, which is a core table - see section 2.2

Forms:

Profile
Form used for updating current user's profile. There are 2 actions:

- Default - Draws HTML form
- update - Assigns appropriate values from HTTP POST and stores the document in the database.

3.6 Bug reporting module

This module is used for simple reporting of bugs, accepting the reports, and recording information that the bug has been solved. There are 3 stages in the lifetime of each bug. The bug is reported by any user of the system who has relevant access to the module. Then, another user who has corresponding role takes over the problem and if he finds the solution, he marks the bug as closed and writes down the solution. Every bug can be deleted in any of these stages by the user who has the relevant role. This process is illustrated by the next diagram.

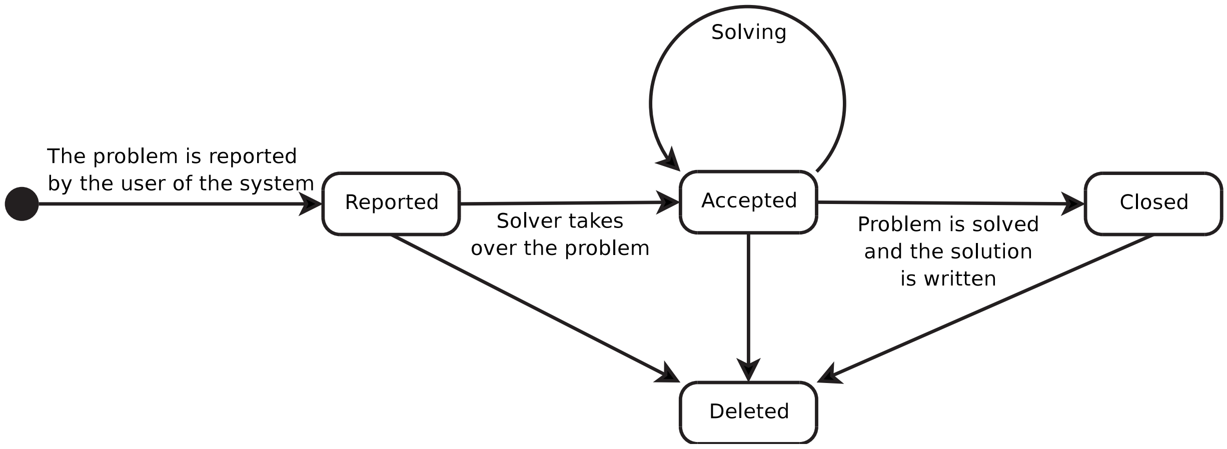

Solving
The problem is reported by the user of the system
Reported
Solver takes over the problem
Accepted
Problem is solved and the solution is written
Closed
Deleted

Figure 3.1: Bug reporting module - state diagram

Tables:

Requirement

Field ID	**Type**	**PK**	**FK**	**Unique**	**Description**
requirement_id	INT(3)	Y			
date_created	INT(10)				
date_closed	INT(10)				
description	TEXT				
solution	TEXT				
creator	VARCHAR(20)		User.username		
state_num	INT(1)				
subject	VARCHAR(64)				

Forms:

Requirement

This form is used for creating, editing, accepting, closing, and deleting bug reports. It has 6 actions:

- Default - Draws HTML form

- create - Assigns data from HTTP POST, set status to 1 (reported), and creates a document
- edit - Locates the document in database and draws HTML form
- accept - Changes status to 2 (accepted) and saves the document
- close - Locates the document, saves the solution sent through HTTP POST, changes state to 2 (closed), and saves the document in the database.
- delete - Locates the document and deletes it.

Requirement_view
The form called *Requirement_view* is used just to display information about the "bug". It has 2 actions, however, default action is empty.

- view - Locates the document and draws HTML output.

Views:

Requirements
This is a view which displays all the bug reports sorted out by their state. Beside every report are two links. The first is used for deleting the bug and it is visible only to users with the relevant role. The second is for opening the report for editing. Editing can also be done only by the user with the relevant role.

User roles:

edit - Role of editing, accepting, and closing reports
delete - Role of deleting reports

3.7 Users module

This module is used either for displaying information about users in the system or administering it.

Tables:

User
This module works with *User* table, which is a core table - see section 2.6

Forms:

User_admin
Form used for creating, editing, and deleting users. It has the following actions:

- Default - Draws HTML form
- edit - Locates the document and draws HTML form
- update - Creating/updating user information. The fist step is to decide whether the user already exists or not. The next step is assignation of attributes from HTTP POST. Document is then created or updated.
- delete - Locates the document and deletes it

Views:

User_admin
This view is displayed only to users who have the corresponding role. Beside every user there are three links - one for edit, second for delete, and third for sending short messages.

Users
If a current user has no permissions to have the view *User_admin* displayed, the *Users* view is drawn. Beside every user there is a link for sending short messages.

User roles:

maintain
- role allowing creation, editing, and deletion users

3.8 Administration Module

By means of this module, the administrator or another user who has administration role maintain the system. The module is divided into several submodules: administration of forms, views, modules, and groups.

Tables:

This module works with the following system tables:
User, User_group, User_role, Module, Form, Form_action, View
See section 2.2 for more information.

Forms:

Admin_form
Form used to manage all forms in the system. It has the following actions:

- Default - Prints out HTML form
- create - allocates relevant variables from HTTP POST and creates document
- edit - locates the document and draws HTML form
- update - locates the document and updates it
- delete - locates the document and deletes it.

Admin_form_action
Administering form actions in the system. It has the following actions:

- Default - Draws HTML form
- create - Assigns relevant data from HTTP POST and creates document
- edit - Locates the document and draws HTML form
- update - Locates the document and updates it
- delete - Locates the document and deletes it

Admin_module
Used for administration of modules, with the following actions:

- Default - draws HTML form

- create - Allocates data from HTTP POST and creates document
- edit - Finds relevant document and draws HTML form
- update - Finds relevant document, assigns data from HTTP POST, and updates the document
- delete - Finds relevant document and deletes it.

Admin_role
Administration of user roles. This form is shown in module administration

- Default - draws HTML form
- edit - locates the document and prints HTML form
- update - creates or updates role. Assigns relevant data and if the document already has its id (*role_id*), it is updated. If not, a new role is created.
- delete - locates the document and deletes it

Admin_user_group
Administration of groups. It has the following actions:

- Default - Does nothing.
- edit - Draws HTML form
- insert - Assigns corresponding attributes of HTTP POST and creates a new document
- delete - locates the document and deletes it

Admin_view
Administration of views

- Default - Draws HTML form
- create - Assigns corresponding data from HTTP POST and creates a new document
- edit - Locates the document and draws the form
- update - Locates the document, assigns relevant data, and updates them
- delete - Locates the document and deletes it

Views:

Actor_select
This view is used in module administration in *Admin_role* form for selecting either users or groups

Admin_form_actions
Admin_form_actions view is displayed in form administration submodule. Beside every form action are links to edit and delete

Admin_forms
View displayed in form administration submodules. It contains links to edit and delete.

Admin_group
Displayed in User/group submodule. It is used for listing groups. It contains a link by means of which a particular group is selected.

Admin_module
Falls into module administration submodule. This view lists all the modules installed in the system. There are two links - edit and delete.

Admin_roles
Falls into submodule which is responsible for administering modules. By means of this view, all roles related to the selected module are displayed.

Admin_user_group
Included in user/group submodule. It displays all the users included in the selected group. The view contains link for removing user from group.

Admin_views
Listing of all views in the system. It contains links for editing and deleting the view.

User_select
HTML SELECT of all users.

3.9 Web administration module

Web administration module is used for creating, editing, and subsequently approving and publishing documents which are then visible to anonymous users of the internet/intranet. Information contained in each document may have one of three different forms:

- HTML source
- Plain text which is then converted by means of PHPMarkdown filter to HTML
- Blog

Each document can also be linked with one or more files which are then displayed as attachments. There are 2 roles acting in this module: editor - user, allowed to create and edit documents, and Approver - responsible for approving, i. e. publishing. The approving process is shown in the next diagram.
The web admin module is divided into five submodules. The first is for editors, second for approvers and the rest for editing header, footer, and CSS style.

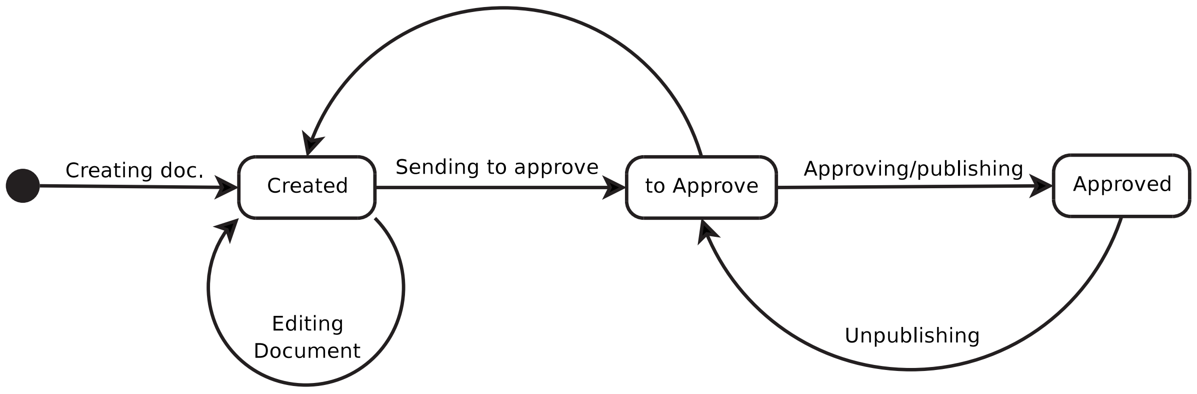


Figure 3.2: Web administration module - state diagram

Tables:

Asset

Field ID	Type	PK	FK	Unique	Description
file_name	VARCHAR(64)	Y			
doc_id	INT(4)	Y			
data	MEDIUMBLOB				
type	VARCHAR(32)				

Web_profile

Field ID	Type	PK	FK	Unique	Description
name	VARCHAR(32)	Y			
value	TEXT	Y			

Document

Field ID	**Type**	**PK**	**FK**	**Unique**	**Description**
doc_id	INT(4)	Y			
doc_name	VARCHAR(32)				
abstract	TEXT				
created	INT(10)				
updated	INT(10)				
sortstr	VARCHAR(30)				
type	INT(1)				
content	LONGTEXT				
fullcat	VARCHAR(255)				
title	VARCHAR(64)				
parent	VARCHAR(20)				
state_num	TINYINT(2)				
author	VARCHAR(20)INT(6)		User.username		

Forms:

Document_edit
Form used for creating and editing document. It has the following actions:

- Default - Draws HTML form
- create - Creates a document
- new - Used for creating new document with pre-defined parent document. Parent field is set and the form is printed out.
- edit - Locates the document and draws HTML form
- update - Locates the document, assigns relevant parameters from HTTP POST, and updates the document.
- delete - Locates the document and deletes it after confirmation.

Document_view
This form displays the document. There is a button which sends the document for approval. It has following actions:

- Default - Does nothing
- view - Locates the document and prints it out
- toapprove - Locates the document and sets state_num to 10 (to approve)

Profile_edit
Form used for editing web profile variables such as header, footer, or CSS. It has the following actions:

- Default - Locates the document in *Web_profile* table and displays the form
- update - Locates the document and updates it

Approval
Form for displaying the document and changing its status to *Approved*, *to Approve*, and *Unpublish*.

- Default - Does nothing
- approve - Locates the document and change its state to 20 - approved
- return - changes documents state to 0 - created, i. e. returns the document to editor
- unapprove - sets state to 10, i. e. unpublishes the document
- view - Locates the document and displays it.

Asset
Asset form is used for attaching files to documents.

- Default - if document (web page) is selected, the form is drawn
- create - assigns relevant attributes and uploads the file into the table called Asset. If there is any attachment with the same name, it is replaced by the new one.
- delete - finds relevant attachment in database and deletes it.

Views:

Web_documentedit
This view displays all the documents with links to edit, delete, and send document to approve.
Web_approval
This view is shown to approvers. It contains links for returning document to the editor, approving the document, and unpublishing it.
Web_parent_select
View in a form of HTML SELECT. Used in each document when choosing parent document.

Part III

Manuals

Chapter 4

Installation

4.1 Requirements

PHP version 5 and MySQL with InnoDB

PHP sources and binaries are available for free download on http://www.php.net. MySQL sources and binaries are available on http://www.mysql.com. However, if you are using any linux distribution which supports any kind of packages, it is highly recommended to install php and mysql from a package.
Open Community (OC) should work on any operating system having a http server, MySQL, and support of PHP5.
Nevertheless, this guide describes installation of Linux/Unix system.

4.2 Installing essential files

Although the majority of the code is stored in a database there are a few files which have to be saved on the disk of WWW server. Most of these files are classes responsible for communication with database and running the modules, processing views, etc. The rest is used for displaying outputs of web module to anonymous users.

- Download the latest version from http://kmlinux.fjfi.cvut.cz/˜faltysekm1/oc/
  Then copy it to the document-root of your WWW server.
- Unzip the file into the document-root of your WWW server, i. e.
  when considering /var/www/htdocs as document-root:

  ```
  cd /var/www/htdocs & gunzip -c oc-[version].tar.gz | tar -xvf -
  ```

  In this case, /var/www/htdocs/oc directory should be created

4.3 Configuration files

4.3.1 Main configuration file

The main OC configuration file is called config.inc.php and it is located in the config/ directory. The only variable you have to change is $cms_root which represents total path to the root of open community directory. Considering our previous guide, the variable has to equal "/var/www/htdocs/oc".

4.3.2 Database access configuration

Now it is time to configure variables determining how to connect to the MySQL database. All these are defined in a file called config_mysql.inc.php in the same directory as the main configuration file. Change $cms_mysql_host, $cms_mysql_username, $cms_mysql_password, and $cms_mysql_database to corresponding values.

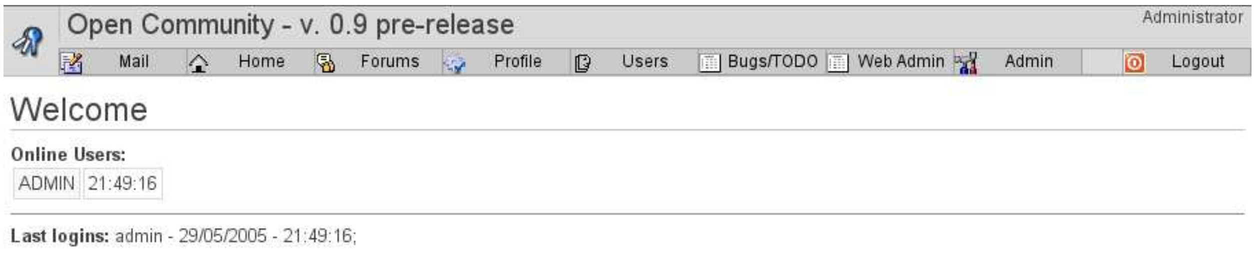
4.4 Uploading the system to a database

The basics have been done. Now it is necessary to record the basic modules, views, forms, admin user, and some groups. MySQL create script (or scripts) are saved in install/ directory. Upload data by means of any MySQL client. The next example shows how to upload data by means of native mysql client. Replace db_name with the appropriate value.

```
mysql [db_name] -p < db_ver-[version].sql
```

4.5 Logging in the system

If everything is set up properly, the system should be alive. Open OC in the web browser. Address should have the shape "http://YOUR-WEB-SERVER/oc". The default home page should be displayed. There is a link to login to the system. If you follow it, login window will appears. Use "admin" for username and "admin" for password.



Welcome

Online Users:

ADMIN 21:49:16

Last logins: admin - 29/05/2005 - 21:49:16;

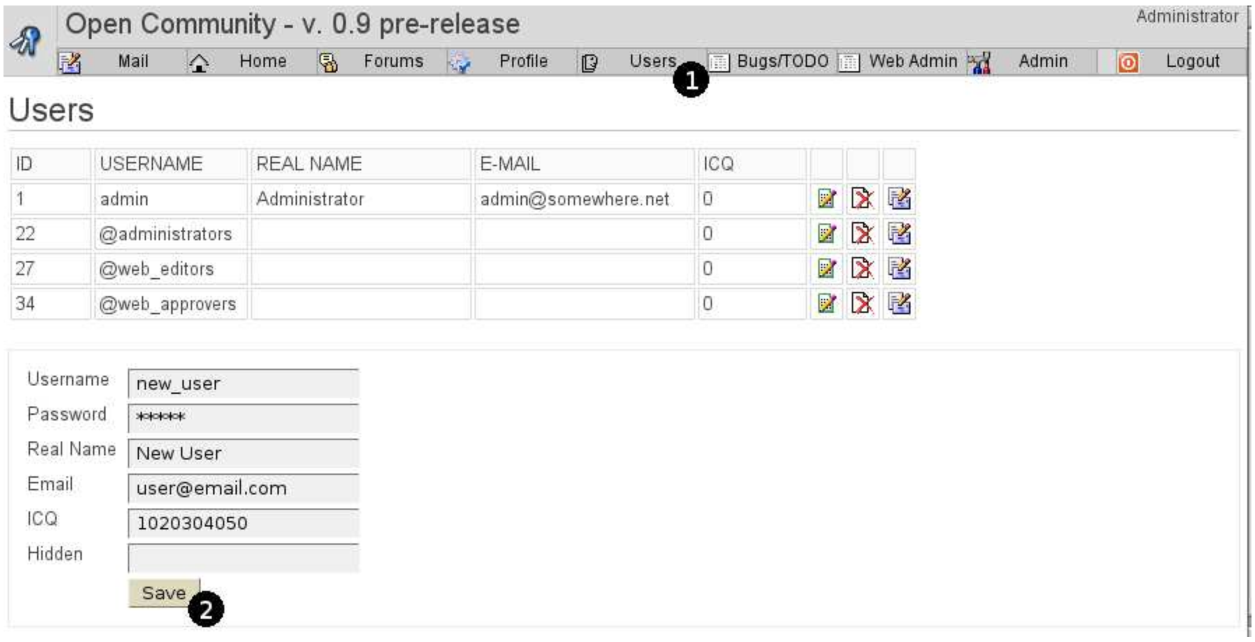
Figure 4.1: Screenshot - Entering the system

Chapter 5

Administration

5.1 Adding a user

Adding a new user is done in Users module. Follow the link on the top of the page. In the users section, there is a list with all the users in the system. Below the list there is a form which, when filled in and submitted, creates a new user.



ID	USERNAME	REAL NAME	E-MAIL	ICQ
1	admin	Administrator	admin@somewhere.net	0
22	@administrators			0
27	@web_editors			0
34	@web_approvers			0

Username	new_user
Password	*****
Real Name	New User
Email	user@email.com
ICQ	1020304050
Hidden	

Save

Figure 5.1: Screenshot - Adding a new user

5.2 Adding a group

Group is added in the same way as the user. The only difference is the need to use @ as the first character of the username. For example, if the username field equals "@users", the created record will be considered as a group "users".

5.3 Including user into a group

Including user into a group is done in Admin section. There is a link called ”Group Administration”. Follow it and you will see a list with all groups in the system. Select the group into which you want to include the user. After the group is selected, the select field containing all the users in the system will appear at the side. Choose a particular user and press the button on the right. As a result, the user should become included in the group.

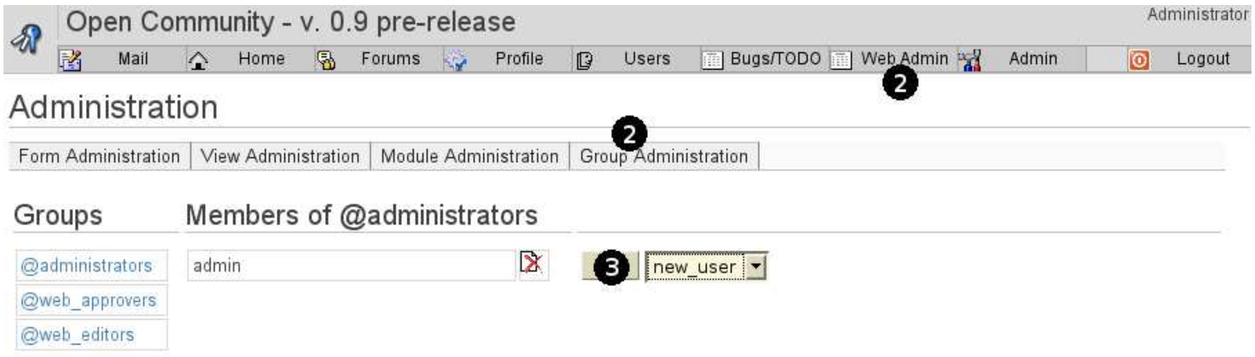


Figure 5.2: Screenshot - Including user into group

5.4 Editing modules

The system allows the administrator to change the attributes of each module. Although it is not recommended, the administrator can change either module code or additional code of the module. All this can be done in Module administration submodule. Edit the values and simply press the submit button.

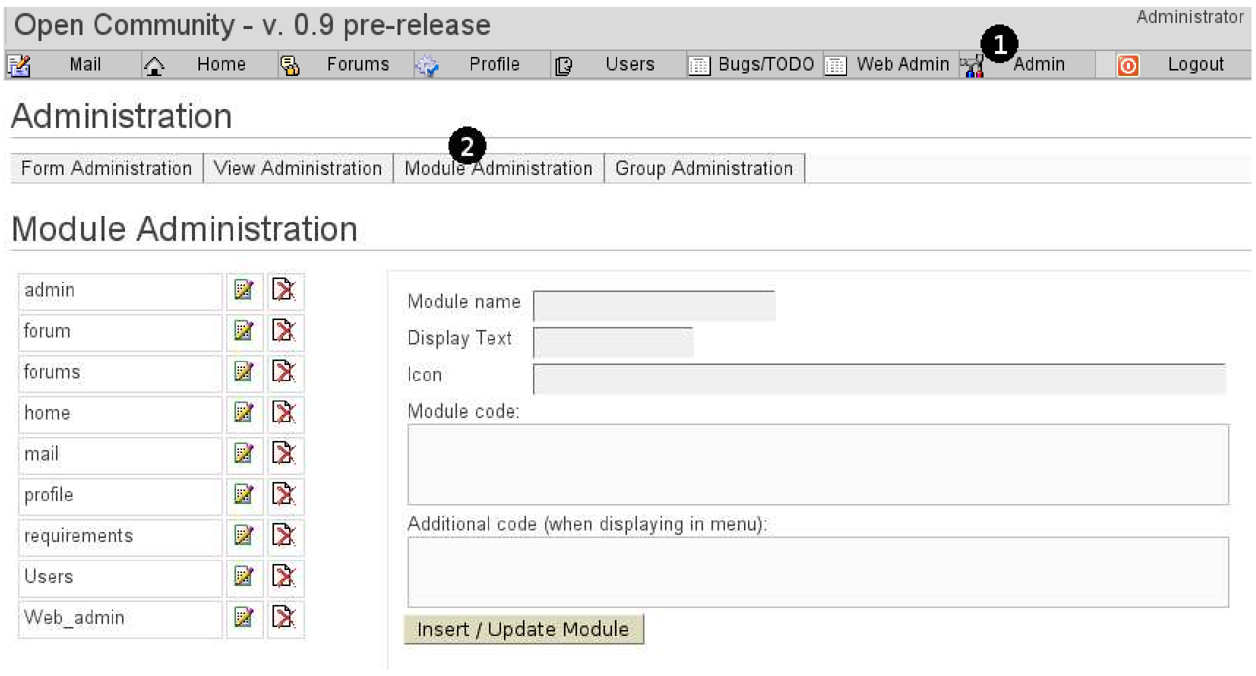


Figure 5.3: Screenshot - Editing a module

5.5 Adding a user role

User role is relation between an element in module and a user. This is why role administration is located in the module administration submodule. Below module attributes, there is a simple form with field for selecting so called "actor" (user or group acting in a role) and the name of the role.

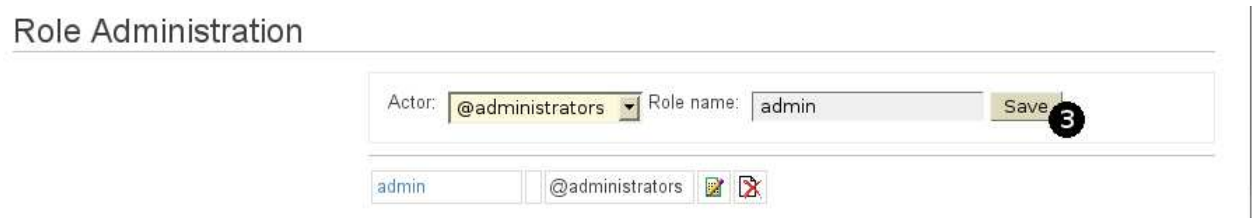
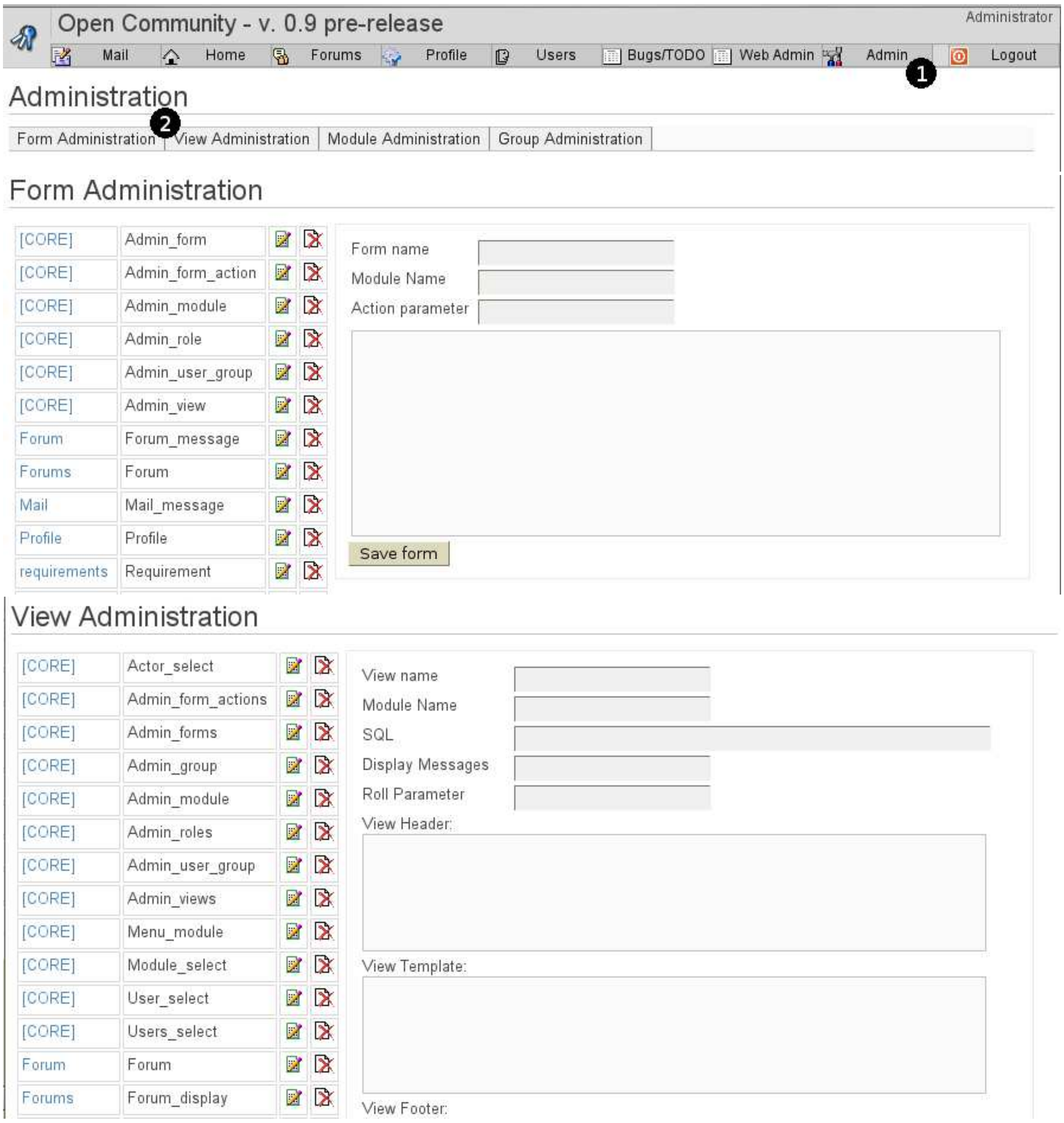


Figure 5.4: Screenshot - Adding a user role

5.6 Editing Forms/Views

Just as in case of modules, it is not recommended to edit forms and views of the core modules. However, it can be done by means of admin module, submodule Views or submodule Forms.



Administration

Form Administration

[CORE]	Admin_form
[CORE]	Admin_form_action
[CORE]	Admin_module
[CORE]	Admin_role
[CORE]	Admin_user_group
[CORE]	Admin_view
Forum	Forum_message
Forums	Forum
Mail	Mail_message
Profile	Profile
requirements	Requirement

Form name

Module Name

Action parameter

View Administration

[CORE]	Actor_select
[CORE]	Admin_form_actions
[CORE]	Admin_forms
[CORE]	Admin_group
[CORE]	Admin_module
[CORE]	Admin_roles
[CORE]	Admin_user_group
[CORE]	Admin_views
[CORE]	Menu_module
[CORE]	Module_select
[CORE]	User_select
[CORE]	Users_select
Forum	Forum
Forums	Forum_display

View name

Module Name

SQL

Display Messages

Roll Parameter

View Header:

View Template:

View Footer:

Figure 5.5: Screenshot - Editing a form or a view

Chapter 6

Running the web

6.1 Introduction

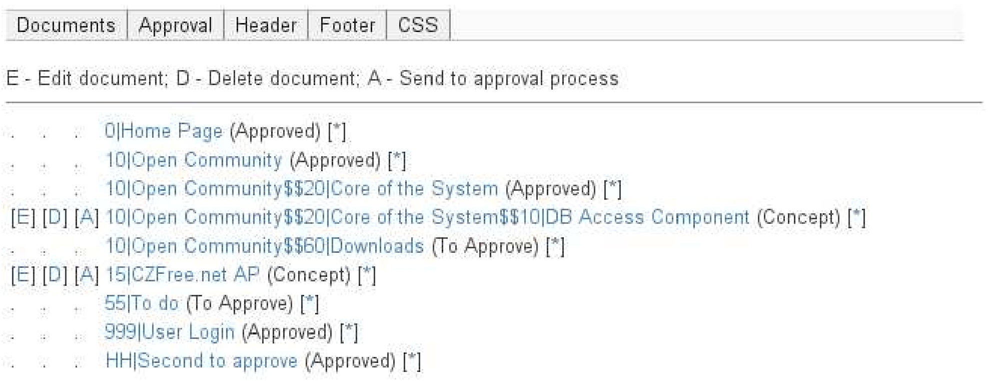
One of the most important parts in the system is the one responsible for presenting web pages. This part is bound with a module called *Web admin*, which is used for creating and approving particular documents which are to be published.
Any user, if he has appropriate permissions, is able to create web pages. These documents have to be approved first in order to be visible for anonymous users visiting the web site.
Documents are hiearchicaly classified into categories. To enter the web administration module follow the link on the top of the interface of the system. Web admin module has 5 submodules.

- Documents - interface for editors
- Approval - interface for approvers
- Header - Editing the header of each page
- Footer - Editing the footer
- CSS - Cascade Stylesheet definitions

If you do not see some of the submodules it means that you do not have enough permissions, i. e. you do not have appropriate user roles. Role called "edit" is for editing while role "approve" is for approving. The rest of the submodules should be visible to all users who have access to the web admin module.

6.2 Editor's interface

Editor's interface is located in "Documents" submodule. There is a list of all documents with several links. "E" means to edit document, "D" means to delete document, and "A" means to send document for approval. Documents with no links are already approved, therefore they cannot be edited or deleted. There is one more link "*" by means of which the subordinate document is created.



Documents | Approval | Header | Footer | CSS

E - Edit document; D - Delete document; A - Send to approval process

. . . 0|Home Page (Approved) [*]
. . . 10|Open Community (Approved) [*]
. . . 10|Open Community$$20|Core of the System (Approved) [*]
[E] [D] [A] 10|Open Community$$20|Core of the System$$10|DB Access Component (Concept) [*]
. . . 10|Open Community$$60|Downloads (To Approve) [*]
[E] [D] [A] 15|CZFree.net AP (Concept) [*]
. . . 55|To do (To Approve) [*]
. . . 999|User Login (Approved) [*]
. . . HH|Second to approve (Approved) [*]

Figure 6.1: Screenshot - Editor's interface

Creating a document

Below on the page is a form with several fields.

- Doc name - Short name of the document. Identifier also used in the menu
- Title - Longer name of the document. Text is displayed at the top of each document
- Abstract - Summary of the document
- Sortstr - String by means of which the list of documents is sorted
- Type - type of the document (HTML / Plaintext)
- Reference - parent of current document
- Content - Text / HTML / ... the body of the document

If these fields are filled in and the form submitted, the document is created.

Inserting an attachment

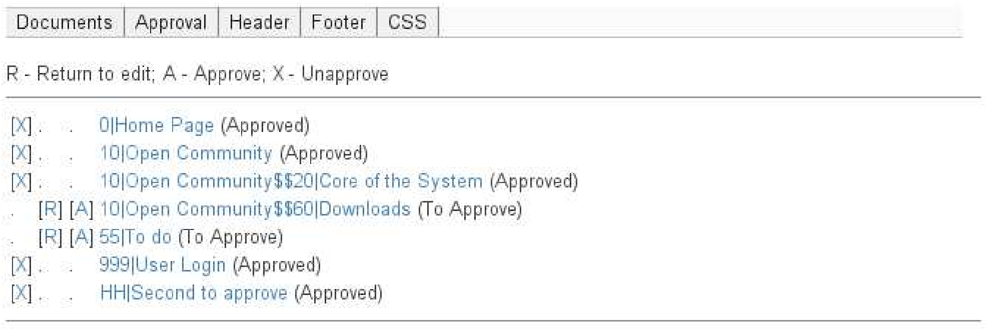
If any document is viewed or edited, there is a form for insertion of an attachment below. Simply browse for a file and press ">>" button. The file should be uploaded. The attachment is then shown at the bottom of the document.

Sending document to approval

There are two ways to send a particular document to approval. It can be done by clicking on "A" in the list or by clicking the "Send to Approve" button while displaying the document. After the document is sent for approval, there is no way of deleting it or changing its data.

6.3 Approver's interface

If current user has appropriate role, he enters this interface by clicking on "Approval" sub-module. The list with all documents contain 3 links. "A" - approve document, "R" - return to editors, "X" - unpublish.



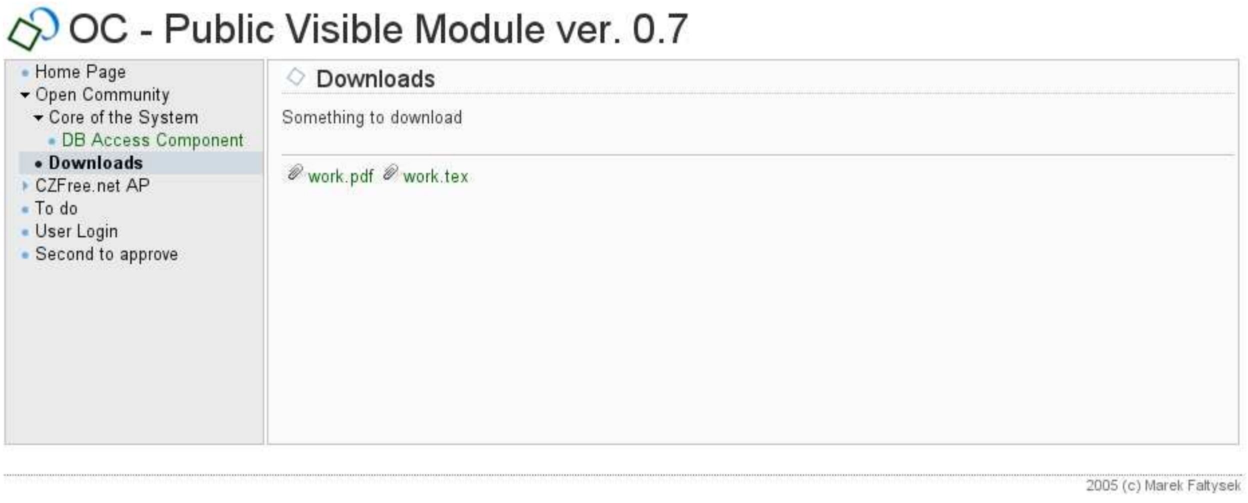
R - Return to edit; A - Approve; X - Unapprove

[X] . . 0|Home Page (Approved)
[X] . . 10|Open Community (Approved)
[X] . . 10|Open Community$$20|Core of the System (Approved)
. [R] [A] 10|Open Community$$60|Downloads (To Approve)
. [R] [A] 55|To do (To Approve)
[X] . . 999|User Login (Approved)
[X] . . HH|Second to approve (Approved)

Figure 6.2: Screenshot - Approver's interface

6.4 Publication

Every page is divided into 4 elements. Header, hiearchical menu, document body, and footer. How to control header, footer, and document body was described before. Menu is generated from the list of all approved documents. Documents are shown hiearchicaly and the menu can expand and collapse. The source of it is written in Javascript and it works in most of today's browsers (IE 4+, Mozilla, Firefox). To see the menu and all possibilities of a presentation, please visit http://kmlinux.fjfi.cvut.cz/~faltysekm1/oc/ or see the chapter with extra screenshots which is at the end of this document.


OC - Public Visible Module ver. 0.7

Downloads

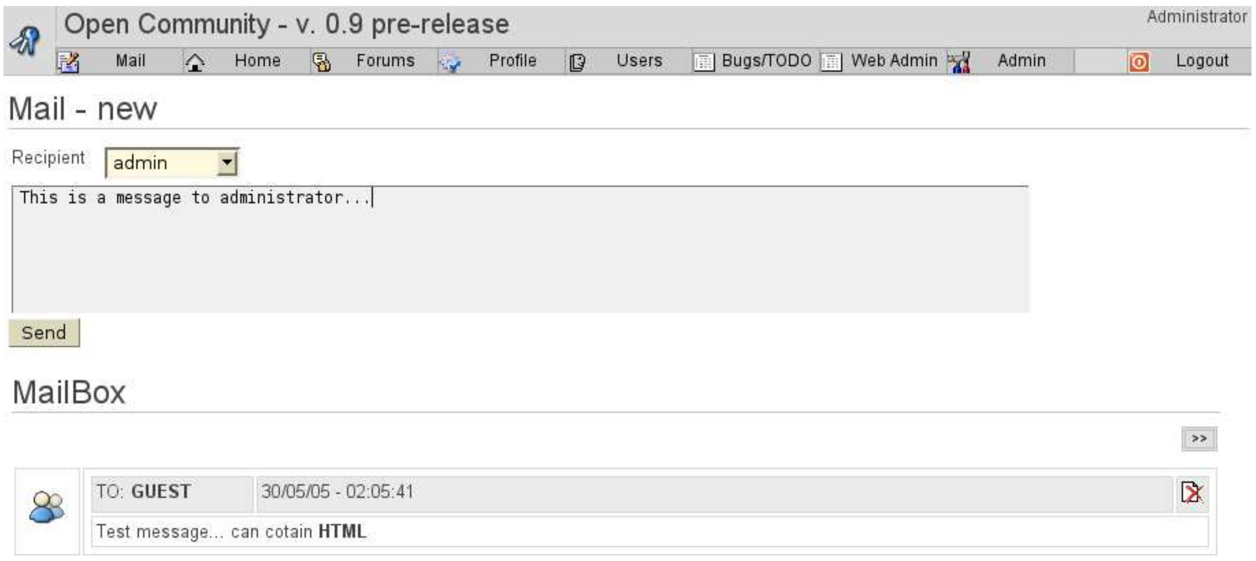
Something to download

2005 (c) Marek Faltysek

Figure 6.3: Screenshot - Presentation

Chapter 7

Further screenshots

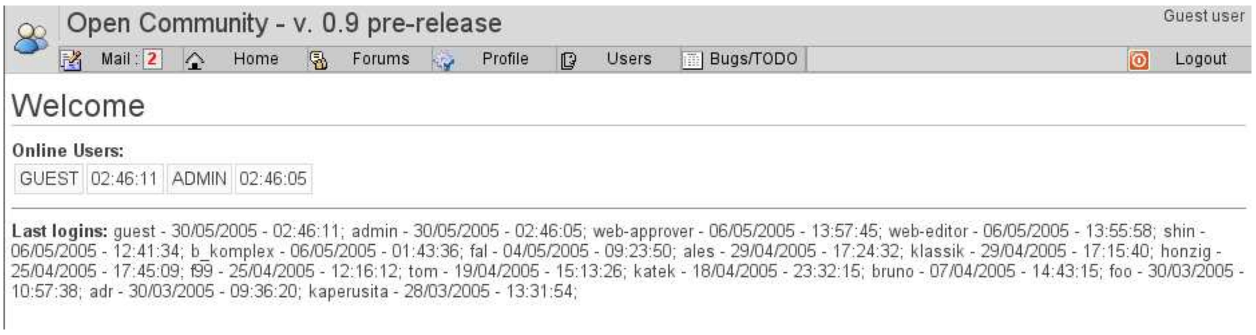


This is a message to administrator...

TO: GUEST 30/05/05 - 02:05:41

Test message... can cotain **HTML**

Figure 7.1: Screenshot - Mail module

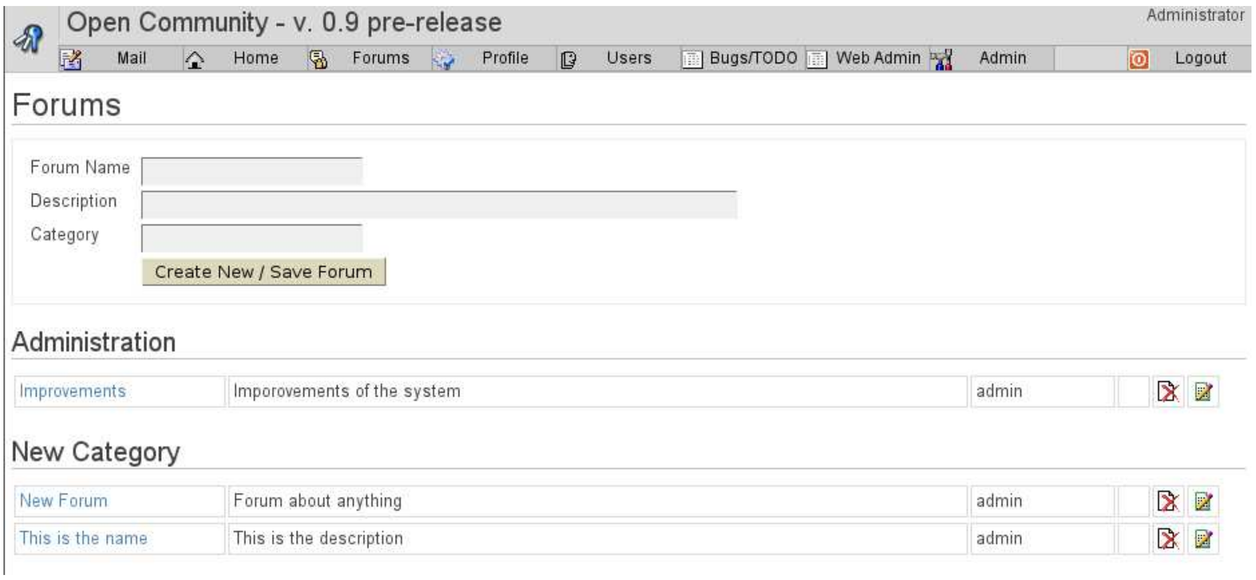


Online Users:

GUEST	02:46:11	ADMIN	02:46:05

Last logins: guest - 30/05/2005 - 02:46:11; admin - 30/05/2005 - 02:46:05; web-approver - 06/05/2005 - 13:57:45; web-editor - 06/05/2005 - 13:55:58; shin - 06/05/2005 - 12:41:34; b_komplex - 06/05/2005 - 01:43:36; fal - 04/05/2005 - 09:23:50; ales - 29/04/2005 - 17:24:32; klassik - 29/04/2005 - 17:15:40; honzig - 25/04/2005 - 17:45:09; f99 - 25/04/2005 - 12:16:12; tom - 19/04/2005 - 15:13:26; katek - 18/04/2005 - 23:32:15; bruno - 07/04/2005 - 14:43:15; foo - 30/03/2005 - 10:57:38; adr - 30/03/2005 - 09:36:20; kaperusita - 28/03/2005 - 13:31:54;

Figure 7.2: Screenshot - Module's additional code example - new mail



Forums

Forum Name

Description

Category

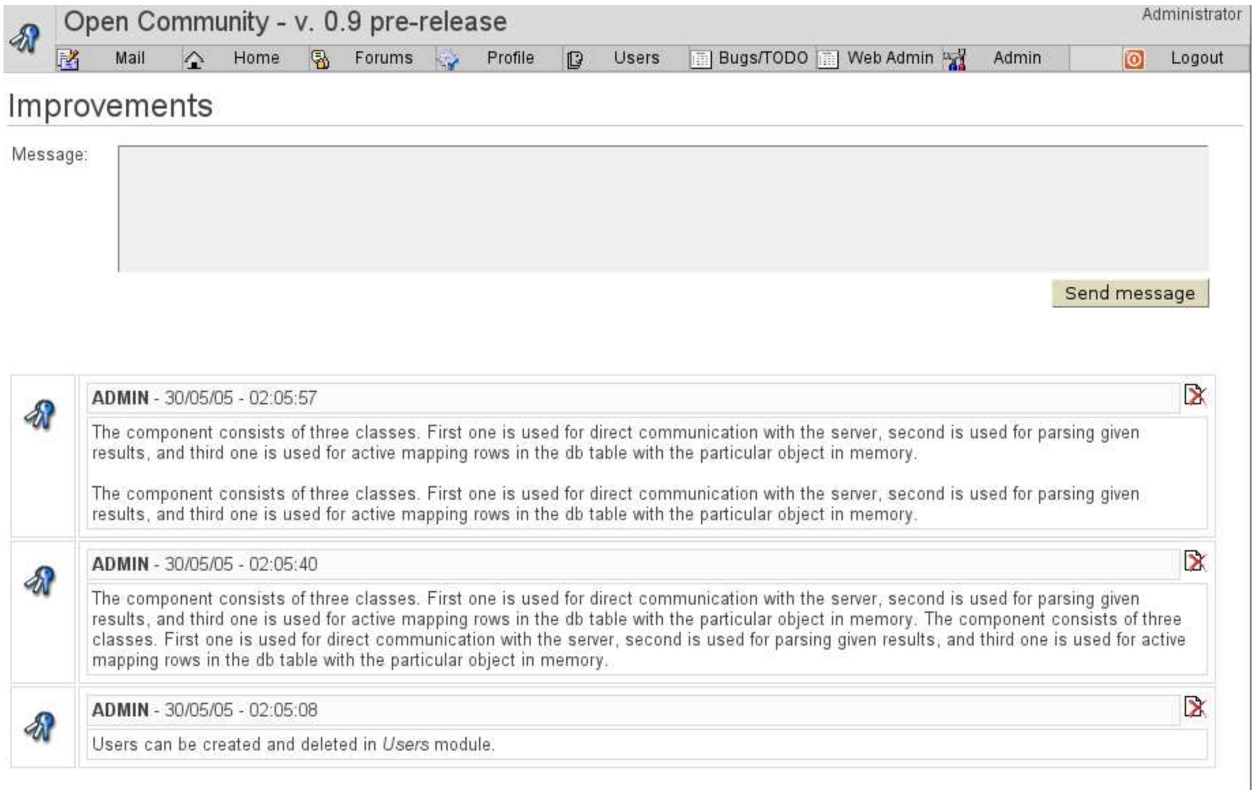
Administration

Improvements	Imporovements of the system	admin

New Category

New Forum	Forum about anything	admin
This is the name	This is the description	admin

Figure 7.3: Screenshot - Forums module



Improvements

Message:

ADMIN - 30/05/05 - 02:05:57

The component consists of three classes. First one is used for direct communication with the server, second is used for parsing given results, and third one is used for active mapping rows in the db table with the particular object in memory.

The component consists of three classes. First one is used for direct communication with the server, second is used for parsing given results, and third one is used for active mapping rows in the db table with the particular object in memory.

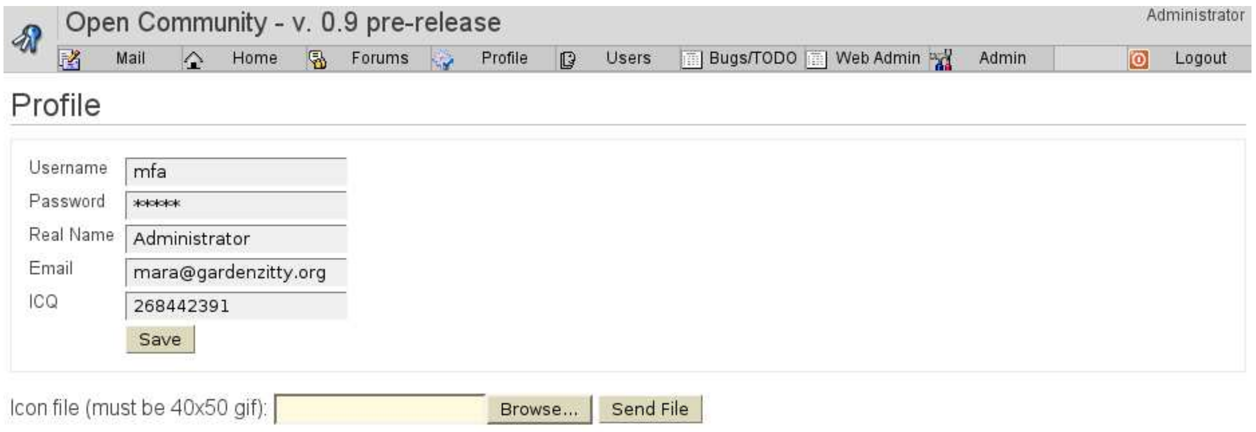
ADMIN - 30/05/05 - 02:05:40

The component consists of three classes. First one is used for direct communication with the server, second is used for parsing given results, and third one is used for active mapping rows in the db table with the particular object in memory. The component consists of three classes. First one is used for direct communication with the server, second is used for parsing given results, and third one is used for active mapping rows in the db table with the particular object in memory.

ADMIN - 30/05/05 - 02:05:08

Users can be created and deleted in *Users* module.

Figure 7.4: Screenshot - Displaying particular forum

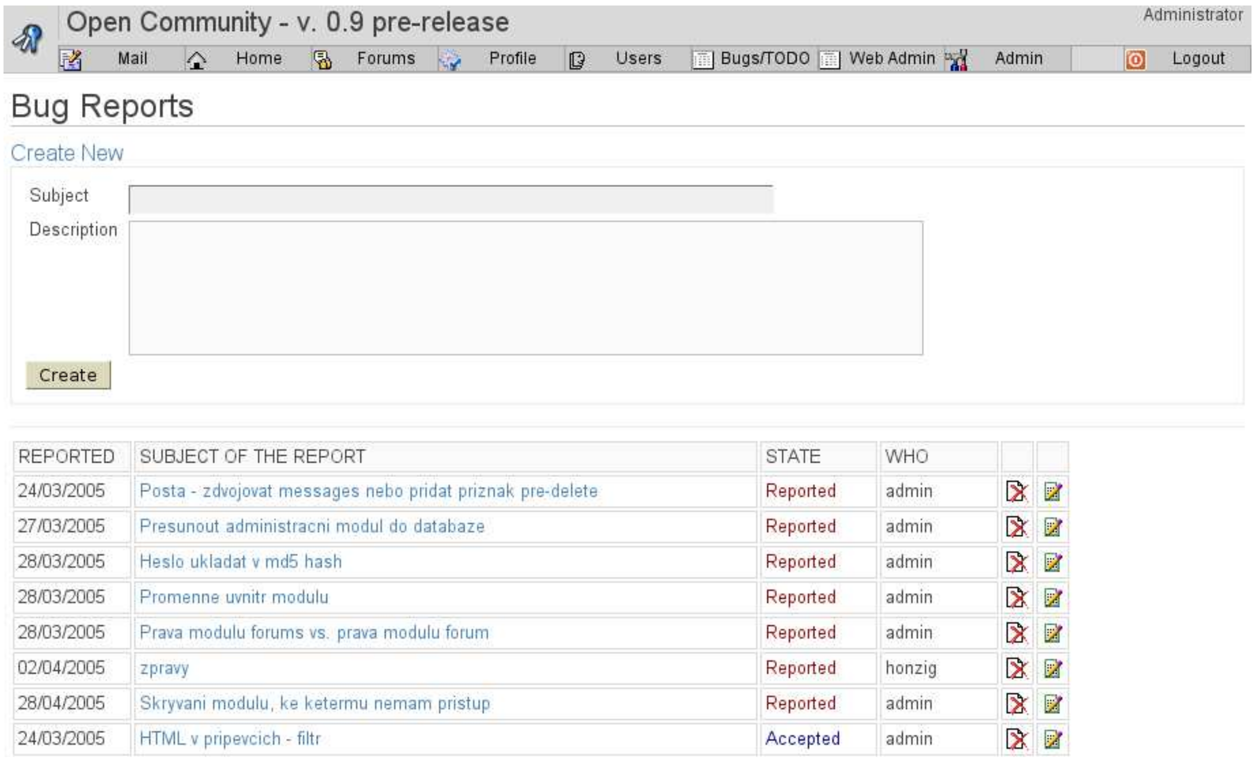


Profile

Username	mfa
Password	*****
Real Name	Administrator
Email	mara@gardenzitty.org
ICQ	268442391

Icon file (must be 40x50 gif):

Figure 7.5: Screenshot - User's profile editing



Bug Reports

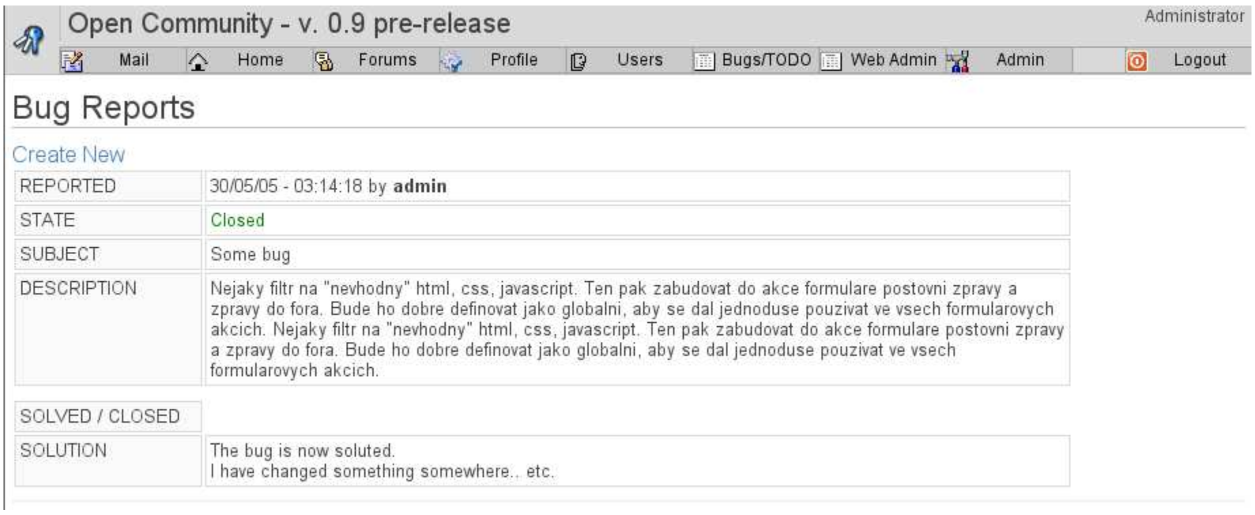
Create New

Subject

Description

REPORTED	SUBJECT OF THE REPORT	STATE	WHO		
24/03/2005	Posta - zdvojovat messages nebo pridat priznak pre-delete	Reported	admin		
27/03/2005	Presunout administracni modul do databaze	Reported	admin		
28/03/2005	Heslo ukladat v md5 hash	Reported	admin		
28/03/2005	Promenne uvnitr modulu	Reported	admin		
28/03/2005	Prava modulu forums vs. prava modulu forum	Reported	admin		
02/04/2005	zpravy	Reported	honzig		
28/04/2005	Skryvani modulu, ke ketermu nemam pristup	Reported	admin		
24/03/2005	HTML v pripevcich - filtr	Accepted	admin		

Figure 7.6: Screenshot - Bug reporting module

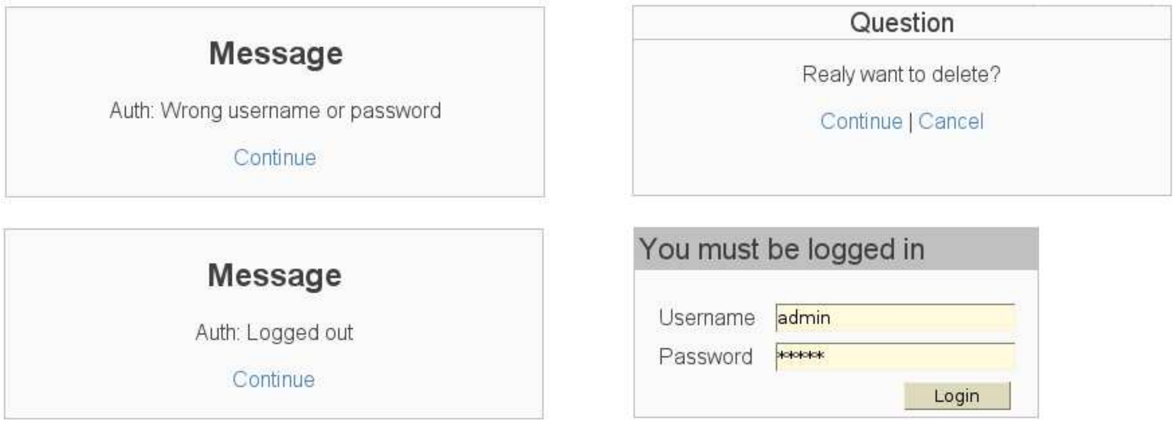


Bug Reports

Create New

REPORTED	30/05/05 - 03:14:18 by **admin**
STATE	Closed
SUBJECT	Some bug
DESCRIPTION	Nejaky filtr na "nevhodny" html, css, javascript. Ten pak zabudovat do akce formulare postovni zpravy a zpravy do fora. Bude ho dobre definovat jako globalni, aby se dal jednoduse pouzivat ve vsech formularovych akcich. Nejaky filtr na "nevhodny" html, css, javascript. Ten pak zabudovat do akce formulare postovni zpravy a zpravy do fora. Bude ho dobre definovat jako globalni, aby se dal jednoduse pouzivat ve vsech formularovych akcich.
SOLVED / CLOSED	
SOLUTION	The bug is now soluted. I have changed something somewhere.. etc.

Figure 7.7: Screenshot - Details of bug and the solution


Message

Auth: Wrong username or password

Continue

Question

Realy want to delete?

Continue | Cancel

Message

Auth: Logged out

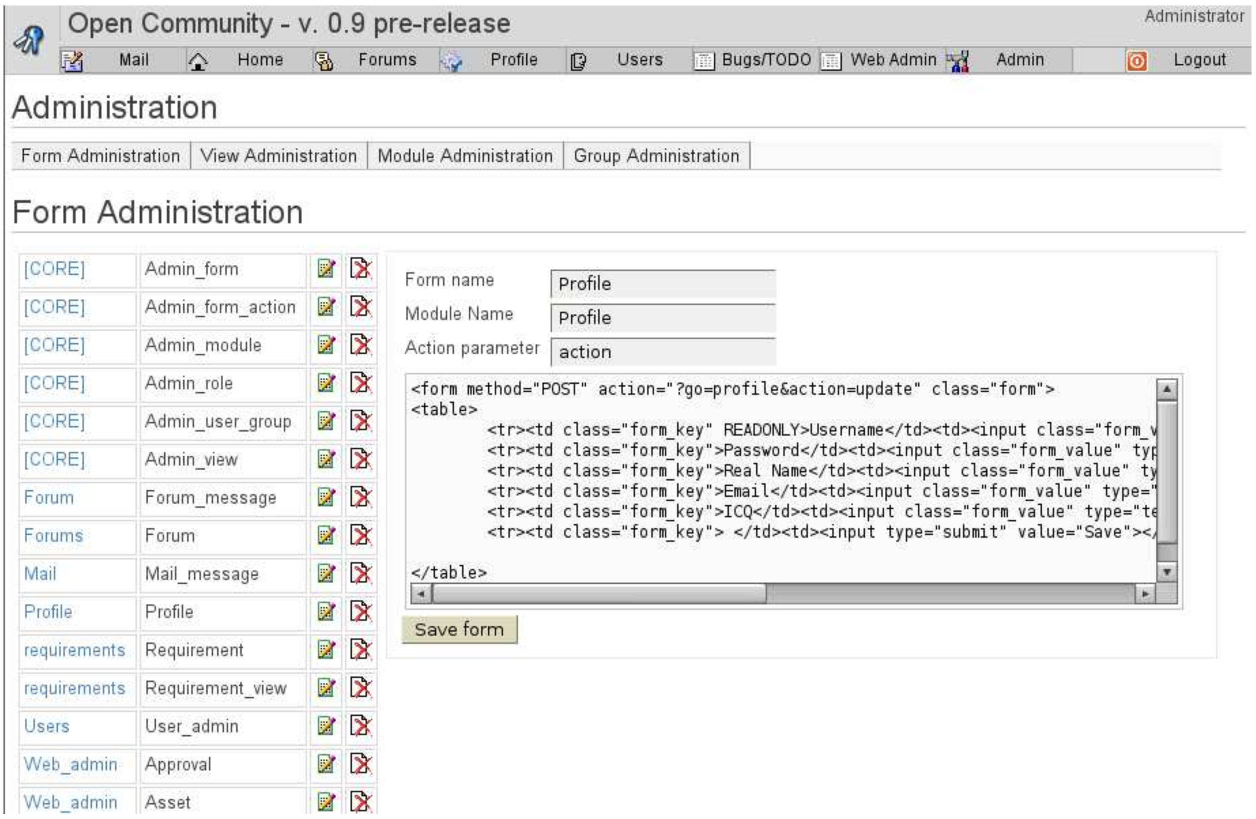
Continue

You must be logged in

Username	admin
Password	*****

Login

Figure 7.8: Screenshot - Examples of messages

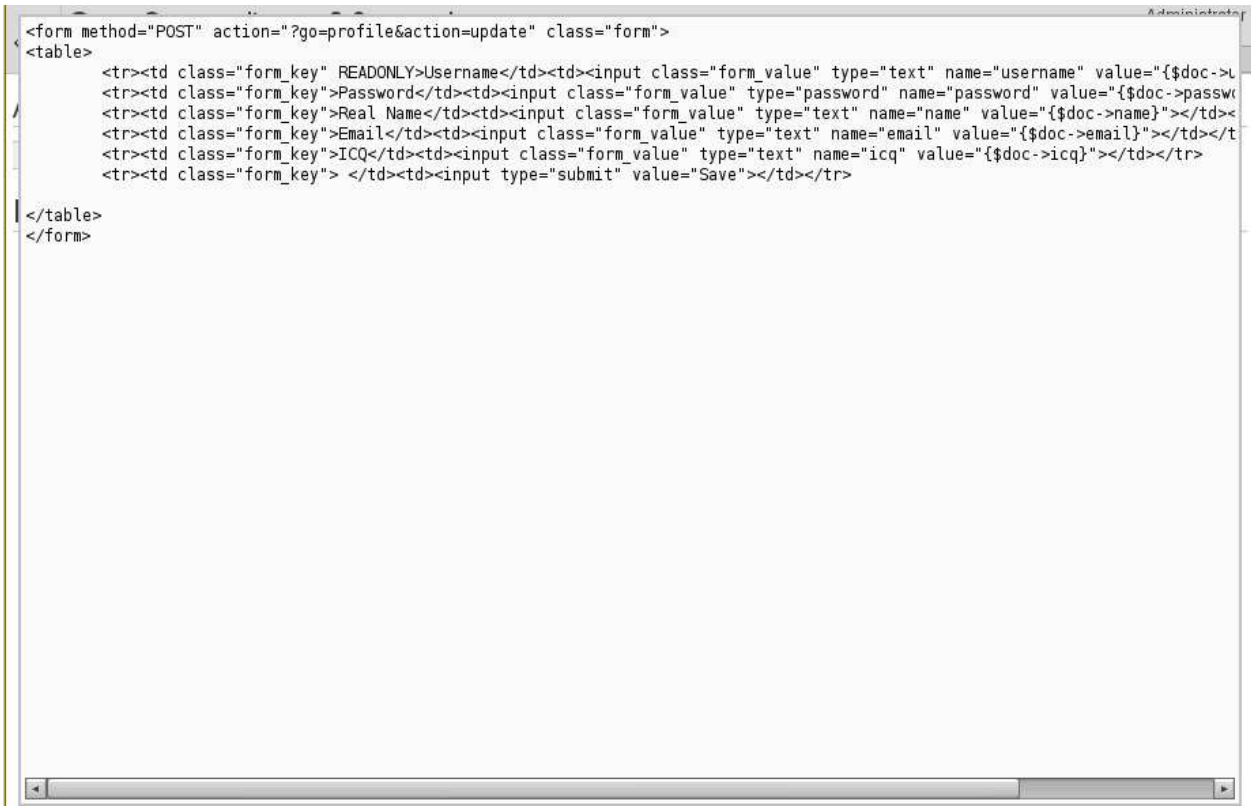


[CORE]	Admin_form
[CORE]	Admin_form_action
[CORE]	Admin_module
[CORE]	Admin_role
[CORE]	Admin_user_group
[CORE]	Admin_view
Forum	Forum_message
Forums	Forum
Mail	Mail_message
Profile	Profile
requirements	Requirement
requirements	Requirement_view
Users	User_admin
Web_admin	Approval
Web_admin	Asset

Form name	Profile
Module Name	Profile
Action parameter	action

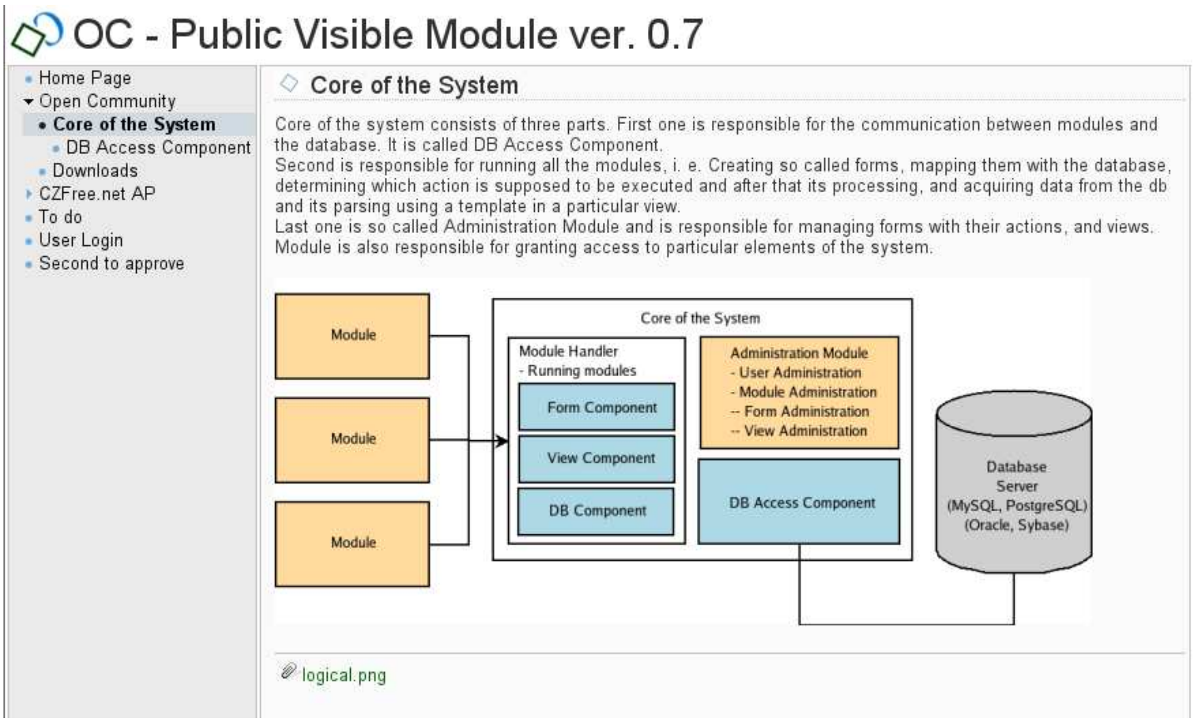
```
<form method="POST" action="?go=profile&action=update" class="form">
<table>
        <tr><td class="form_key" READONLY>Username</td><td><input class="form_v
        <tr><td class="form_key">Password</td><td><input class="form_value" typ
        <tr><td class="form_key">Real Name</td><td><input class="form_value" ty
        <tr><td class="form_key">Email</td><td><input class="form_value" type="
        <tr><td class="form_key">ICQ</td><td><input class="form_value" type="te
        <tr><td class="form_key"> </td><td><input type="submit" value="Save"></

</table>
```

Figure 7.9: Screenshot - Editing form properties



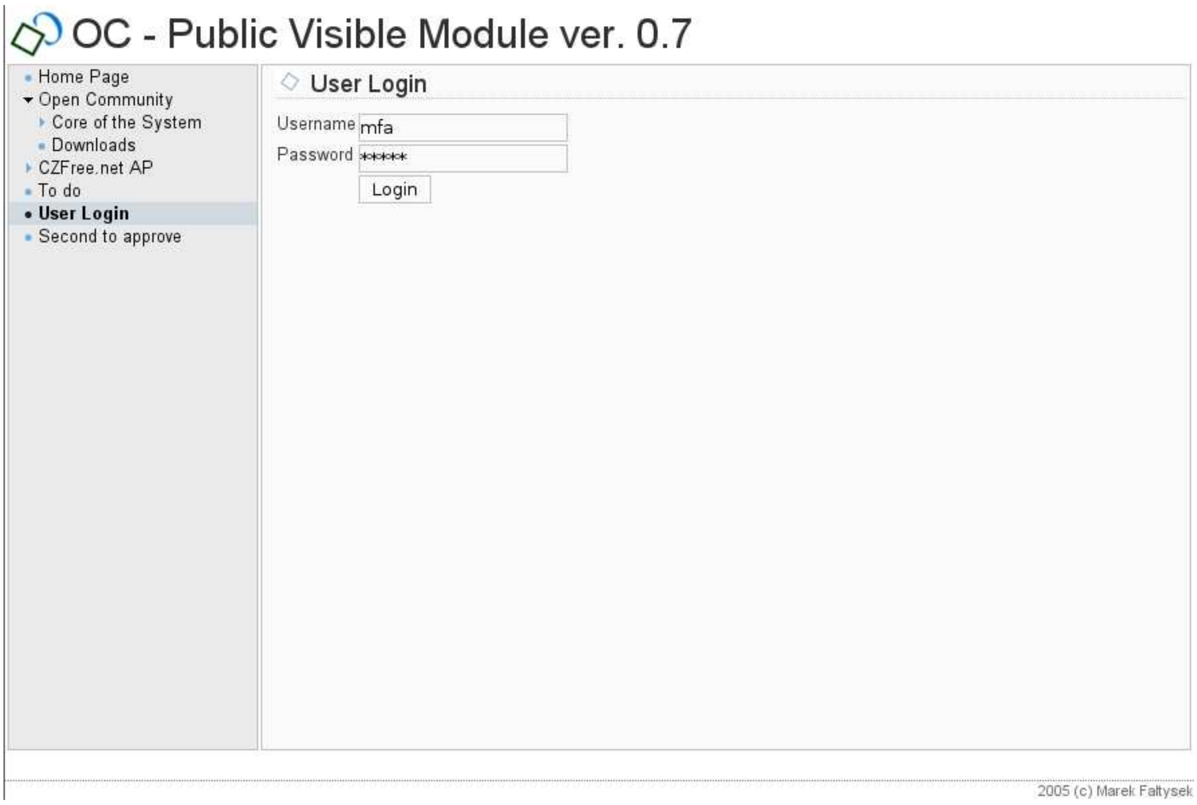
```
<form method="POST" action="?go=profile&action=update" class="form">
<table>
        <tr><td class="form_key" READONLY>Username</td><td><input class="form_value" type="text" name="username" value="{$doc->u
        <tr><td class="form_key">Password</td><td><input class="form_value" type="password" name="password" value="{$doc->passw
        <tr><td class="form_key">Real Name</td><td><input class="form_value" type="text" name="name" value="{$doc->name}"></td><
        <tr><td class="form_key">Email</td><td><input class="form_value" type="text" name="email" value="{$doc->email}"></td></t
        <tr><td class="form_key">ICQ</td><td><input class="form_value" type="text" name="icq" value="{$doc->icq}"></td></tr>
        <tr><td class="form_key"> </td><td><input type="submit" value="Save"></td></tr>

</table>
</form>
```

Figure 7.10: Screenshot - Editing the same form properties with maximalised field



Core of the system consists of three parts. First one is responsible for the communication between modules and the database. It is called DB Access Component.
Second is responsible for running all the modules, i. e. Creating so called forms, mapping them with the database, determining which action is supposed to be executed and after that its processing, and acquiring data from the db and its parsing using a template in a particular view.
Last one is so called Administration Module and is responsible for managing forms with their actions, and views. Module is also responsible for granting access to particular elements of the system.

Figure 7.11: Screenshot - Presenting a document to the public



Username mfa
Password *****

Figure 7.12: Screenshot - Presenting a document to the public

Chapter 8

Conclusion

Despite a number of minor mistakes which have not yet been removed due to the limited time available for the final ”tune-up”, there was a success in creating a simple groupware-like system which enables user groups to communicate with one another in various ways, and to publish the results of their communication. Plenty of furtherpossible modules come to mind - depending on the specific purposes for which the system may be used. Modules can be easily added and removed. In the future, the created modules and the entire documentation should be available for downloading on the main pages of the project - i.r. http://kmlinux.fjfi.cvut.cz/˜faltysekm1/oc/

Chapter 9

Bibliography

- Kosek J.: *PHP - tvorba interaktivních internetových aplikací*, Grada Publishing, 1999
- Kosek J.: *HTML - tvorba dokonalých www stránek*, Grada Publishing, 1998
- Schlossnagle, George: *Pokročilé programování v PHP 5*, Zoner Press, 2004
- doc. Ing. Richta K., CSc., Ing. Sochor J., CSc.: *Softwarové inženýrství I*, ČVUT, 1996
- Prof. RNDr. J. Pokorný, CSc., Ing. I. Halaška: *Databázové Systémy*, ČVUT, 2003
- http://www.php.net - PHP project website
- http://www.mysql.com - MySQL project website
- http://www.jakpsatweb.cz - HTML, CSS, and Javascript reference