

*COMP3760 Project S1 2008*

# Simple Rostering System

## Project Report

*Ben Griffiths (4302927)*

### **Abstract**

Simple rosters can be found in all manner of communities throughout society, including homes, workplaces, and community groups. They are a necessary tool whenever there is a group of individuals taking joint responsibility for a routine task.

Despite the prevalence of this tool throughout society, its most popular implementation is still a paper-based one. There is presently a shortage of tools available that provide a viable alternative to a paper roster in terms of ease of use, accessibility and reliability. The purpose of this project was therefore to demonstrate that a web-based rosters system could be developed to fill this niche.

The Simple Rostering System (SRS) is the system developed to meet that objective. It provides a simple, flexible, and accessible web-based interface through which users can manage one or more rosters. This includes the ability to create and modify rosters, as well as view them and configure email notifications.

The system was developed for the LAMP architecture (Linux, Apache, MySQL, PHP) to make it cost effective to implement and easy to customise. The web interface is delivered to a web browser in the form of XHTML 1.0 (for content) and CSS 2.0 (for presentation).

The outcome of the project was a functional framework for the system that demonstrates the feasibility of a web-based rosters system that provides a more convenient and accessible alternative to the traditional paper-based roster. Development of an efficient user-friendly web interface for the system is left as future work, to bring the system to its full potential.

***Table of Contents***

1	Introduction.....	6
1.1	Aims.....	6
1.2	Methods.....	7
1.2.1	Design.....	7
1.2.2	Implementation.....	7
1.3	Contributions.....	7
1.4	Limitations.....	7
1.5	Structure of This Report.....	8
2	Background.....	9
2.1	Concepts.....	9
2.1.1	Rostering System.....	9
2.1.2	Web-based System.....	9
2.2	Significance.....	9
2.2.1	Significance for development.....	9
2.2.2	Significance for use in society.....	9
2.3	Illustration of Need (User Stories).....	9
2.3.1	In the Office.....	9
2.3.2	At a School.....	10
2.4	Technologies.....	11
2.4.1	HTML.....	11
2.4.2	CSS.....	11
2.4.3	PHP.....	11
2.4.4	MySQL.....	11
2.4.5	Roles in the Simple Rostering System.....	11
2.5	Related Systems.....	12
2.5.1	Paper roster.....	12
2.5.2	Manual email roster.....	12
2.5.3	Commercial rostersing software.....	12
2.5.4	Web-based system (PHP/SQL).....	12

2.5.5 Influences on the Simple Rostering System.....	12
3 Requirements (Scope).....	13
3.1 Core Features.....	13
3.1.1 Actions Available At Each Access Level.....	13
3.2 Desirable Features.....	14
4 Schedule.....	15
4.1 Initial Plan.....	15
4.2 Due Dates.....	15
4.3 Accuracy of the Initial Plan.....	15
5 Design & Modelling.....	16
5.1 System Design.....	16
5.2 Modelling.....	16
5.3 Interface design.....	16
6 Implementation.....	17
6.1 Database implementation.....	17
6.1.1 Series, Roster, Task Structure – Issues.....	17
6.1.2 UML Model.....	17
6.2 Code implementation.....	17
6.2.1 Overall Code Structure.....	18
6.2.2 Interface Between The Code and The Database.....	18
6.2.3 Email Module.....	19
6.2.4 PHP Files.....	19
6.3 Reminder Implementation.....	19
6.3.1 Scheduling.....	19
6.3.2 Format.....	20
6.4 User Accounts Implementation.....	20
6.4.1 User Class.....	20
6.4.2 Session Management.....	21
6.4.3 Password authentication.....	21
6.5 Interface implementation.....	21

7	Testing & Evaluation.....	22
7.1	Integration testing.....	22
7.2	Interface Testing.....	22
7.3	Subjective comparison to existing systems.....	23
7.3.1	Paper roster.....	23
7.3.2	Commercial Rostering Software.....	23
7.4	Acceptance testing.....	23
8	Discussion.....	24
8.1	Conclusions.....	24
8.2	Issues.....	24
8.2.1	Project Management.....	24
8.2.2	Database Implementation Issues.....	24
8.2.3	Code Implementation Issues.....	24
8.3	Future Work.....	25
8.3.1	User Interface Development.....	25
8.3.2	Open Source Future.....	25
8.3.3	Adaptation for Content Management Systems.....	25
8.3.4	Other Potential Enhancements.....	25
9	References.....	26

## Glossary

### SMTP

An acronym for Simple Mail Transfer Protocol (SMTP). It refers to the protocol most often used to send email messages between mail servers. An SMTP server is a server that sends emails using this protocol.

### Protocol

The language used by digital devices to communicate over digital networks. It involves details such as handshakes (or greetings), timings, format of the data to be transmitted, and error detection.

### Content Management System

A Content Management System (CMS) is a web application platform used to facilitate the development of large scale websites. A CMS uses server-side programming (usually in combination with a database server) to generate pages on the fly rather than having to store every page in a website statically.

### Database

A database is a program that stores tables of information. There will be a number of tables (often representing objects) in a typical database, each with a number of columns (representing attributes), and rows (representing instances of the object).

Here is an example of a (simplified) database table for a Roster:

ID	Name	Start Date	End Date	Series ID
5	Kitchen Duties	2008-06-09 00:00:00	2008-06-13 00:00:00	2
6	Household Chores	2008-06-08 00:00:00	2008-06-14 00:00:00	3

Here there are two rosters in the database, each with their own name and start/end dates. They also each have a Series ID, which is a foreign key (see below).

### Cron job

A scheduled task for the server to perform at a preset time and/or date. The term “cron job” commonly refers to this functionality in Unix-based operating systems, but the same functionality is available (as “Scheduled Tasks”) in Windows operating systems.

### Foreign key

A foreign key is a field within a database table that serves as an identifying reference to another table in the database. For example, the Series table may have an ID field which uniquely identifies each Series. The Roster table may then have a field called 'Series ID' which is a foreign key that identifies which Series it is a part of.

### Foreign key constraints

Foreign key constraints prevent there being entries in the database with a foreign key that refers to a non-existent entity (for example, a Roster with a 'Series ID' that refers to a Series that does not exist).

### MySQL Storage Engine

The storage engine is essentially the system MySQL uses to store the tables in the database. Different database storage engines have different capabilities and limitations.

# 1 Introduction

Simple rosters are used in some form by people all over the world, in any situation in which a group of individuals seek to perform some routine task on a rotational basis. They provide a way for people to schedule which individual in a group will be responsible for a given task at a particular time.

Due to the nature of these kinds of rosters in representing a diverse range of different activities in varying time frames, most rosters today are simply printed or drawn by hand on paper. There is currently a shortage of digital rostering tools to facilitate the creation and sharing of these kinds of rosters.

There is an opportunity for a system to be developed that will fill this void and bring simple rosters into the digital age. Most participants in any roster today will have regular access to email and a web browser, particularly in an office environment. The near-ubiquitous nature of these facilities in today's society presents an opportunity for a web-based rostering system that is far superior to existing systems in terms of ease of use, convenience, and accessibility.

The Simple Rostering System (SRS) is a web-based rostering system that was created over the course of this project to take advantage of the opportunity described above. It is targeted toward a wide variety of simple rostering scenarios, such as the following:

- Routine tasks in an office or place of business,
- Shared roles in community groups, and
- Odd jobs around the home.

The SRS will provide a simple, flexible, and accessible web-based interface through which users can create and modify rosters (if authorised to do so), as well as view them and configure notifications. The system will feature the following:

- Flexible rostering options to suit a variety of tasks,
- Email confirmations and notifications, and
- Authentication and authorisation of users.

## 1.1 Aims

The core aim of this project was the development of a functional framework for the Simple Rostering System that would demonstrate how a web-based system can provide efficient and effective rostering capabilities.

The ultimate aim for the Simple Rostering System itself is for it to provide a viable web-based alternative to traditional paper-based rosters, and actually achieve practical real-world use.

## 1.2 Methods

### 1.2.1 Design

The system was designed using a combination of aspect-oriented thinking (separation of concerns), top-down approaches (functional break-down), and bottom-up approaches (data modelling). Diagramming techniques used include entity-relationship diagrams and unified modelling language, as well as some non-standard notations to represent specific concepts.

The design of certain aspects of the system was an iterative process, as the structure of the PHP code and database was refined in order to achieve correct functionality, and an appropriate compromise between flexibility and complexity.

### ***1.2.2 Implementation***

The system was developed using the LAMP architecture (Linux operating system, Apache web server, MySQL database server, and PHP scripting language). The development process itself was conducted by manually writing the bulk of the PHP code of the system in an advanced text editor, and uploading the code to a (shared hosting) server via FTP for testing with a web browser. Database administration was performed via the phpMyAdmin web interface.

### **1.3 Contributions**

The key contributions of this system are the project code itself and the associated documentation (including this report). The project code provides a framework or a prototype for anyone wanting to develop this type of system. The documentation provides supporting information so that anyone interested in the system is able to learn how it was developed and how it can be modified.

### **1.4 Limitations**

Limitations on the system as it was submitted include a basic user interface and the lack of some desirable time-saving features (such as editing whole roster series at once). The system provides all of the necessary functionality for creating, modifying, and sharing rosters, but there are many improvements yet to be made that will improve efficiency and ease of use. This includes more thorough validating of user input. The PHP code will protect the database from malicious input, but simple JavaScript input validation is not implemented in the system, meaning errors can be easily generated by unexpected input.

The basic user interface included with the system is tested with Firefox 2, Internet Explorer 7, and Opera 9. Older and less well known browser may therefore not display the interface as intended.

Other more general limitations on the system include scalability (as determined by the performance of the web and database servers used, and what the user considers an acceptable loading time for pages) and the need for the server on which the system is installed to support the technologies used by the system (see Section 2.4).

### **1.5 Structure of This Report**

The first part of this report introduces the project in Section 1 and provides some background to the project in Section 2. This includes what the system is, what the goals of the project are, as well as describing some similar systems, and the technologies used.

Section 3 outlines the requirements for the project by specifying what features were needed for the system, as well as features that were desirable but not essential. This section also specifies which of these features will be accessible to users at each access level.

Section 4 provides the planned schedule for the project in the form of gantt chart, and lists some of the key dates (such as that of the final presentation, which was another component of the project).

The next part of the report covers how the project itself was carried out, including how it was designed in Section 5, how it was implemented in Section 6, and how the system was tested in Section 7.

Section 8 then discusses the conclusions arising from the project, as well as the challenges faced during the project and some ideas for future work on the system.

## 2 Background

This section will cover some of the concepts and technologies that were involved in the project, as well as significance of the project and the role of the system in society. Related systems are also discussed, including other types of rostering systems and how they relate to the Simple Rostering System.

### 2.1 Concepts

#### 2.1.1 Rostering System

A rostering system is a tool for generating a schedule of activities which a group of individuals participate in on a regular basis. Participants in a roster will usually rotate or 'take turns' completing an activity, and the roster assists them in determining who is responsible for completing the activity whenever it is called for.

*Roster: "a list or plan showing turns of duty or leave in an organization."* - AskOxford [8]

#### 2.1.2 Web-based System

A web-based system is a software system with which the end user interacts via web services (such as through a web browser) over the Internet. Web-based systems that are delivered via a web browser are accessible from any device that has a web browser and Internet connectivity. Web-based systems suffer from the limitations of common web browsers, which include very limited interactivity and constraints on layout & presentation. This results from the necessary use of standard mark-up such as HTML, albeit with some enhancements offered by technologies such as CSS and JavaScript.

### 2.2 Significance

#### 2.2.1 Significance for development

The Simple Rostering System provides a working example of a system for storing and manipulating simple rosters in a web server environment. The system may be used in the future to assist with the production of similar systems. It serves as a starting point which developers can study to determine what aspects of their own system could be similar, and what should be different.

#### 2.2.2 Significance for use in society

The Simple Rostering System provides the groundwork for a system that can be used in an enormous variety of different scenarios throughout society – in homes, businesses, and community groups.

Rosters of this simple kind are used in one form or another by people all over the world for many different tasks. The dominant method used to create and share these rosters is printing or manually drawing them up on paper. The SRS has the potential to make the processes of roster creation and sharing much more efficient and effective by providing a central web-based system through which they can be organised. Other features such as automated email notifications also offer users significant enhancements to the usefulness of their rosters.

### 2.3 Illustration of Need (User Stories)

#### 2.3.1 In the Office

The most typical use for this system would be in an office environment, where tasks such as cleaning the staff kitchen are often shared between employees. A typical roster for such duties will be coordinated by single person via email, produced using a spreadsheet application, and printed for display somewhere in the office (such as the staff kitchen).

Not only can coordination be lengthy and difficult, but alterations are not easily incorporated into the existing roster, particularly one-off changes which staff members will typically be required to remember.

SRS will centralise email coordination by allowing a single administrator to invite all participants by email, who can then register on the system. Once the administrator then generates the roster according to their specific needs, a link to it will be emailed to all participants, who can view it and request alterations. The administrator can set the roster to repeat until a certain date, and individual roster events can be modified without affecting the rest of the series. Participants will be notified of any changes that are made to the roster, and they can request an email reminder be sent to them prior to their turn coming around. The roster can also be printed off by any participant, so that they can display it like a regular paper roster if desired.

### ***2.3.2 At a School***

As another example, consider the environment of a school, in which teaching staff take turns to supervise the children during lunch breaks. A roster will typically be drawn up specifying which staff member will be on duty each day over a period of one or two weeks. The roster will be created by a staff member whose responsibility it is to manage the lunch duty roster. The staff member may request the availability of all teaching staff and then collate that information to create a roster, either by hand or using a spreadsheet application. The roster will then be printed and posted somewhere, such as in the staff room.

This process would have a similar structure using the SRS, but would be significantly streamlined. The staff member responsible for the roster will send out requests by email to the staff members whose participation is required. The staff member will follow a link in the email and go through a simple and brief sign-up process. They can then notify the administrator of their availability for the roster they have been invited to. The administrator of the roster will then create a roster in the system according to the availability submitted by participants. The roster can be configured to be the desired length (eg. one or two weeks long), with one task to be completed daily by one participant, and the roster can be repeated for the rest of the school term. The SRS would also allow the rostering of multiple tasks on the roster (eg. Different playground areas).

Once the roster is finalised, an email will be sent to all participants with a link to the roster and a summary of their participation in it. Participants can then view the roster online at any time through the email link or by logging into the service. All participants will be emailed whenever changes to the roster occur (at the discretion of the administrator). Each participant will also (if desired) be sent an email reminder notifying them when they will soon be required to go on duty. Participants are also able to notify the roster administrator when they are unable to fulfil their obligations on the roster, and the administrator can alter the participant responsible for lunch duty on that day without altering the rest of the weeks in the term. Any staff member can also print off the roster at any time, to keep at their desk, or in the staff room.

## 2.4 Technologies

### 2.4.1 HTML

Hypertext Markup Language (HTML) is a language for marking up the content of a text document so that a web browser can display it. HTML is specified by the World Wide Web Consortium (W3C). The specification that was used for the SRS web interface is XHTML1.0. [4]

HTML is limited in that it is inherently static (hyperlinks are the only interactive elements), and it is designed primarily for on-screen display. In order to create printable output, the system will need to use CSS (see below) or PDF output.

### 2.4.2 CSS

Cascading Style Sheet (CSS) is a specification for formatting documents that use mark-up languages such as HTML. It allows web developers to separate the presentation of their web documents from the content, and alter the presentation of a whole set of documents from a single file. Both of these features were highly desirable for the SRS web interface. CSS is specified by the World Wide Web Consortium (W3C). The specification that will be used for the SRS web interface is CSS 2.1. [5]

The role of CSS in this project is peripheral, as it specifies presentation rather than providing functionality. The major goals of this project are functional, therefore CSS is not a core technology.

### 2.4.3 PHP

Hypertext pre-processor (PHP) is a scripting language which is interpreted by web servers to generate web pages. It is a popular language for generating dynamic content in web pages. As well as basic dynamic content, PHP also facilitates interaction with a database server. Other common applications using PHP include scripts for sending out emails and user authentication. As with any server-side scripting, there are security concerns that need to be addressed to protect the web and database servers[6].

PHP is a flexible language, and has very little structure built into it. This meant that a structure had to be devised for the system that was modular, flexible and efficient. The “*Joomla!*” content management system was studied as a basis for such a structure, as well as the applicable security measures it uses. For information on *Joomla!* and why it has been selected for study, see Section 2.5.4 entitled *Related Systems – Web-based System (PHP/SQL)*.

### 2.4.4 MySQL

MySQL is an open source database software package that can be used in conjunction with PHP. PHP scripts can interface with a MySQL server to run queries and store/retrieve data. Queries to the MySQL database(s) use the Structured Query Language (SQL) and can be used to request, modify, add or remove data. SQL queries can also be used to create and erase database tables. [7]

### 2.4.5 Roles in the Simple Rostering System

These technologies were combined in the Simple Rostering System to provide an efficient, reliable, and accessible rostering system delivered through a web interface and via email.

To illustrate how these technologies interact, take the simple example of a user viewing a roster. The PHP code receives the request and loads the template for a roster web page. The PHP code then retrieves the details of the requested roster from the MySQL database using SQL queries. This allows for the fast and efficient retrieval of the data, which is used to fill out the template. The completed template then forms a HTML document which is passed to the user’s browser. The HTML document contains a reference to the applicable CSS file(s), which the browser downloads and uses to display the HTML document to the user as intended.

## 2.5 Related Systems

### 2.5.1 Paper roster

A paper roster consists of a basic roster drawn up or printed out on a sheet of paper, and posted in an appropriate place for participants to view.

This rostering method is quite simple, and is used for similar purposes to the planned SRS system. Paper rosters can be generated quite efficiently with some spreadsheet applications, but are still limited in terms of accessibility. This is due to the necessity for participants in the roster to travel to and physically view the roster (wherever it may be posted). It also adds printing costs to roster production.

Rosters generated using these systems are usually either authored by hand or using spreadsheet software. Neither of these methods is ideal.

### 2.5.2 Manual email roster

Participants are emailed by the person responsible for managing the roster when something is required of them. Email rosters are sometimes not formally recorded (if at all), so this method relies on the coordinator of the rostering remembering (or recording) who is scheduled to do what when, and for reminding participants when it is their turn on the roster. Communication is limited, as participants can only view the roster if the coordinator provides it to them. This also means that if participants wish to swap with someone, lengthy email correspondence is required between participants and the coordinator.

### 2.5.3 Commercial rostering software

Most of these systems are very complex as they are intended for staff rostering, and have payroll and skill-level features. These systems are advanced and far more efficient than any other method for their purposes; however the complex interface means the system has a steep learning curve and makes the system inefficient when only simple rosters are required.

One such system is Oriador Rota [1] which features much of the complexity described above. Another is VersaERS [2], which features a web-based interface that mimics an application. This means that it doesn't follow standard web page conventions and is similar in complexity to Oriador.

### 2.5.4 Web-based system (PHP/SQL)

These systems feature dynamic web pages generated by PHP scripts (with access to one or more databases using SQL queries). Desirable capabilities offered by PHP/SQL systems include user accounts and the ability to store & retrieve rosters. Typical applications include forums, news websites, and e-commerce websites.

#### 2.5.4.1 Joomla!

*Joomla!* (spelled with an '!') is one example of such a web-based system [3]. Joomla! is a flexible open source Content Management System which uses a highly extensible platform built with PHP and SQL. As a community developed system that has been through numerous revisions, Joomla! is a good example of a flexible, modular, secure and efficient PHP/SQL system.

### 2.5.5 Influences on the Simple Rostering System

The SRS can be used for similar applications to the paper roster system, but it combines some of the functionality of commercial rostering systems such as Oriador with the technology of PHP/SQL web-based platforms such as Joomla! The result is a simple, portable, flexible, and accessible rostering system.

## 3 Requirements (Scope)

### 3.1 Core Features

The following features constitute the core functionality of the SRS:

- User accounts based on email addresses
- Different access levels for user accounts
- Ability for users to (depending on access level):
  - Invite new users to the system
  - Authorise users to create & manage their own rosters
  - Modify rosters within the system
  - Create and store custom rosters
  - Add participants to rosters
  - Remove participants from rosters that they have created or which they administer
  - Modify rosters that they have created or that they administer
  - Set repeat interval of rosters and alter individual instances in a series
  - Optionally specify a time of day for tasks within a roster
  - View rosters they participate in and/or administer

#### 3.1.1 Actions Available At Each Access Level

Users are given access to certain features of the system depending on what access level they have been granted by the administrator. The table below outlines which of the core system features are available at each level of access.

Action	System Administrator	Roster Administrator	Participant User
View rosters (if participating in or administering)	✓	✓	✓
Create and store custom rosters	✓	✓	
Add participants to rosters	✓	✓	
Remove participants from rosters that they have created or which they administer	✓	✓	
Modify rosters that they have created or that they administer	✓	✓	
Set repeat interval of rosters and alter individual instances in a series	✓	✓	
Optionally specify a time of day for tasks within a roster	✓	✓	
Invite new users to the system	✓		
Authorise users to create & manage their own rosters	✓		
Modify rosters within the system	✓		

## 3.2 Desirable Features

The following features were to be added to the system as the time frame permitted:

- Advanced roster scheduling (first Thursday of every second month, etc)
- Advanced print formatting (using CSS) and PDF output
- Automatic roster recommendation based on availability
- Multiple tasks per roster
- Ability for participants to request swaps with other participants
- Ability to generate roster calendars over any time period (eg. weekly roster for the year)
- Ability to create rosters with no date associated with the activities
- Multiple participants assigned to a single instance of an activity

## 4 Schedule

### 4.1 Initial Plan

Tasks	Week	2	3	4	5	6	7	8	9	10	11	12	13	14
Project/Supervisor Selection		Red												
Contract Finalised			Red											
Requirements Determination			Red	Red	Red	Red								
Initial Report & Presentation				Red										
Project Report				Red	Red	Red	Red	Red	Red	Red	Red	Red	Red	Red
Prototyping					Green	Green	Green	Green						
Database Design/Modelling					Blue	Blue								
PHP Code Design/Modelling						Blue	Blue							
Interface Design							Blue	Blue						
Prototype & Report Outline									Red					
Database Implementation							Green	Green	Green					
PHP Code Implementation								Green	Green	Green				
Interface Implementation									Green	Green	Green			
Testing & Evaluation							Blue	Blue	Blue	Blue	Blue	Blue	Blue	
Finalize DB/Code/Interface												Green	Green	
Final Presentation													Red	
Final Submission														Red
Tasks	Week	2	3	4	5	6	7	8	9	10	11	12	13	14

### 4.2 Due Dates

**Week 4:** Initial Presentations - includes overview of project topic and project plan timetable

**Week 9:** Mid project result (including prototype, outline of final report) due to supervisor(s).

**Week 13:** Final Presentations - includes demonstration where applicable

Projects due on **4pm Friday 13 June** (2 hard copies of final report plus soft copy of report / final presentation slides / project artifacts to projects over-coordinator)

### 4.3 Accuracy of the Initial Plan

The initial plan was followed quite closely in the undertaking of the project, until approximately week 10. At that point the PHP code development began to draw out significantly longer than was anticipated. For further discussion on why this occurred, see Section 8.2. As a result of this the *Interface Implementation* and *Testing & Evaluation* stages were cut shorter than was initially planned.

# 5 Design & Modelling

## 5.1 System Design

The design of the system was top-down, dividing it into functional areas which would be reflected in the interface and the back end implementation. The three functional areas are the main area for performing the basic functions of the system (such as the homepage), the user area for managing user accounts, and the roster area for managing the rosters themselves. This structural breakup provided an even and natural breakdown of the system that could be accurately reflected both in the front end interface and the back end implementation.

## 5.2 Modelling

The rostering component of the system was modelled using an entity-relationship diagram during the design phase.

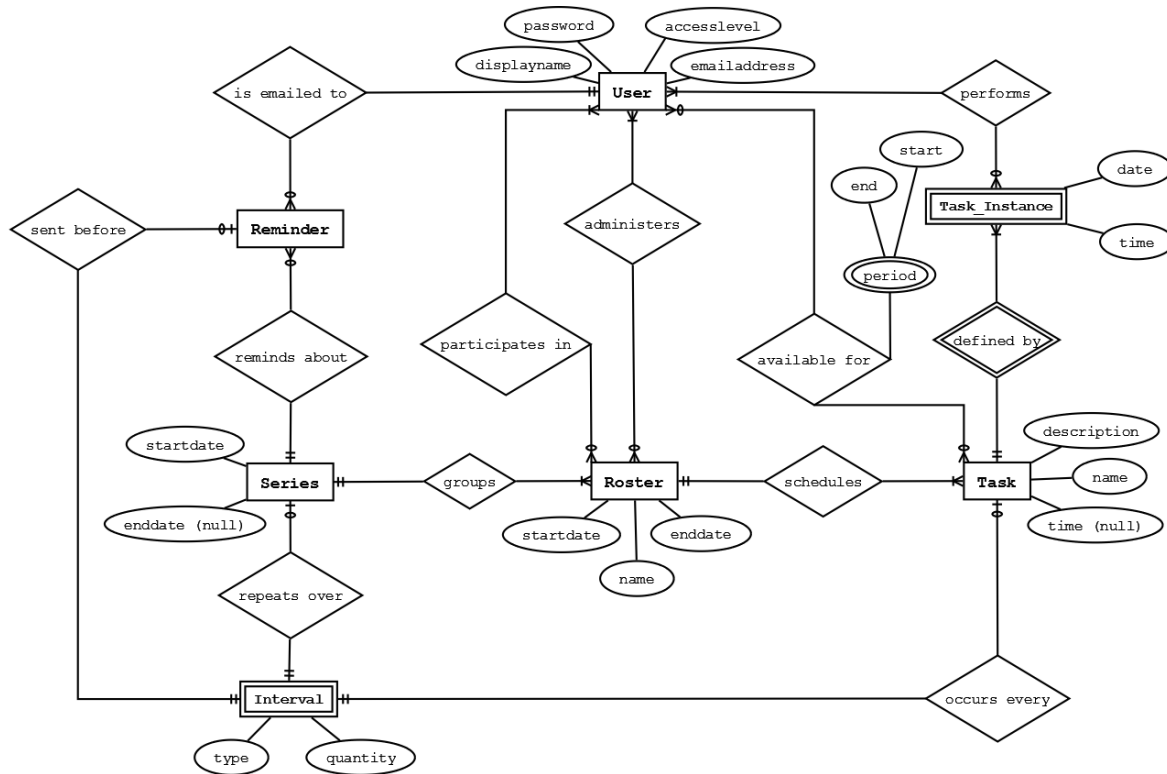


Figure 1: ER diagram for the rostering system (See enlarged version in Appendix A)

## 5.3 Interface design

The interface was designed to be simple and task-oriented, using only terminology that would already be known to the user (ie. No system-specific jargon).

The interface follows the same overall structure as the system itself (main, user, and roster areas), but the navigation presented to the user was designed to be task-oriented, to streamline use as much as possible.

# 6 Implementation

This section covers the main components of the system implementation, and where applicable, some key points within each. The implementation in it's entirety is not fully explained in this section due to the size and complexity of the system, but a general overview of each component is given, as well as the key considerations.

## 6.1 Database implementation

The database was implemented using MySQL, with the assistance of phpMyAdmin, a web-based database administration interface made available by the hosting provider of the development server.

### 6.1.1 Series, Roster, Task Structure – Issues

One of the key decisions in developing the database structure was determine whether a series (of rosters) should be able to be extended without requiring reference to the existing roster instances (as they may have been altered). This introduced additional complexity to the ER model, including circular referential fields between series and roster. It was concluded that it would be sufficient for the user to set the period that the roster will cover at the time that the roster is created.

### 6.1.2 UML Model

The database implementation was documented using a UML model of the table structure.

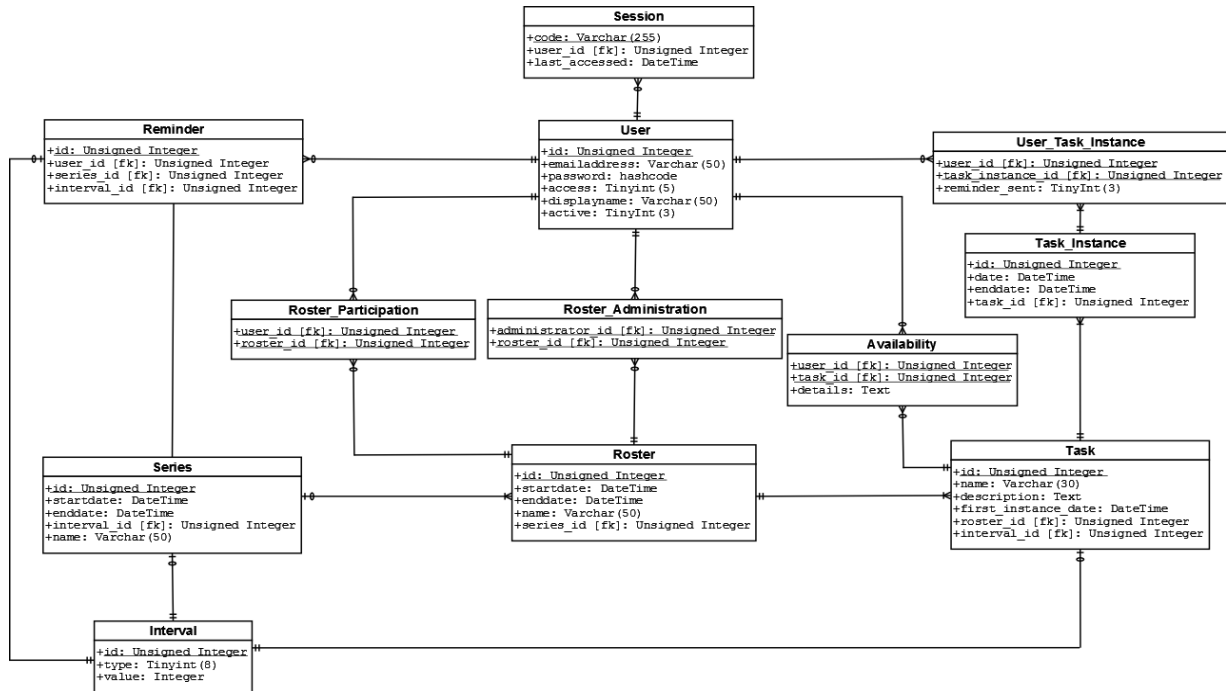


Figure 2: UML Model of the SRS database (See enlarged version in Appendix A)

## 6.2 Code implementation

The system code was implemented in a series of PHP files. The root PHP file (index.php) calls the database and user modules to initialise the database and check the session status of the user (respectively). It then calls the main.php file to build the page, capturing all output (to prevent HTTP headers being sent before the session cookie has been processed). Once the page building has been completed, the index.php file releases the captured output, and it is sent to users' browser.

The final version of the system consists of approximately 3000 lines of PHP code across 20 files.

### 6.2.1 Overall Code Structure

The code is divided into 3 functional areas, and within these areas different tasks trigger calls to different functions. The below diagram offers a cut-down view of the way the files are organised. The lines indicate the flow of function calls, originating at the index.php file.

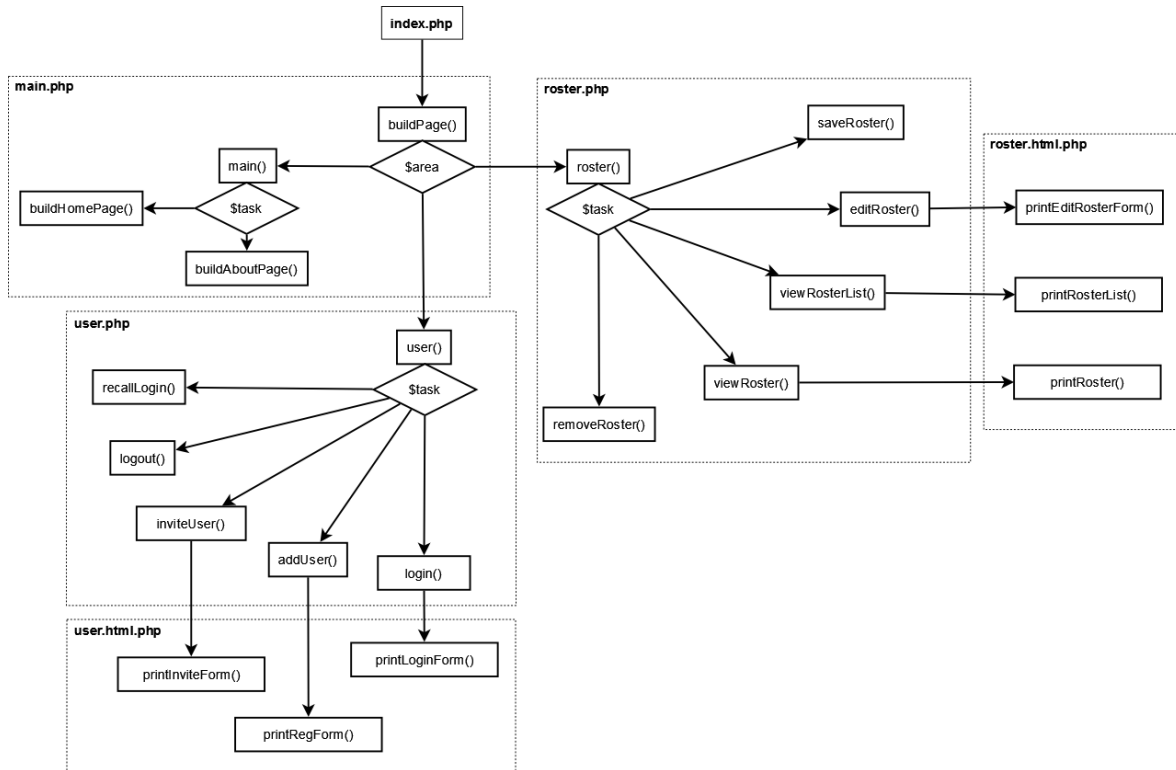


Figure 3: Basic code structure of the SRS system (See enlarged version in Appendix A)

### 6.2.2 Interface Between The Code and The Database

Two possible methods for structuring the code to access the database were considered.

1. One method was to initiate the interface to the database before each query, and close it afterwards. This method was rejected as it involved the database username and password being available to all areas of the code (as this information is required to initiate the interface). One advantage of this method is that the code is not engaged with the database for any more time than is necessary to execute it's queries (as opposed to leaving the database connection open). This reduces the load on the database and improves security.
2. The other method that was considered was to extract the database interactions and centralise them in a single module. This method requires all of the queries within the code to be submitted to the database via functions in the database module, and the same is true for retrieving the results of a query. This allows for information hiding with the database interface itself, and minimises the repetition of code. The other significant advantage of this method is that it improves security and convenience by only including the username and password for the database in one location. Note that this method is based on that used in *Joomla!* (See Section 2.5.4 for information on *Joomla!*).

The second method was ultimately selected, as it represents better programming practice with regard to security, lack of repetition, and information hiding.

### 6.2.3 Email Module

Email is a key functionality of this system and it was therefore necessary to have emailing functions within the code that were efficient and secure (in particular, they must be difficult to exploit by spammers). Through research into existing implementations, an appropriate structure was devised which involves a single email module that can be called from any other function within the code. Within this module all inputs to be applied to the email messages are thoroughly checked for signs of exploitation, and suspicious requests will be rejected with a polite message to the user. These security checks are quite lengthy, hence by separating email functions into a discrete module a significant amount of code duplication is avoided. It also makes it easier to change details such as the From address.

### 6.2.4 PHP Files

The following PHP files form the code of the simple rostering system:

- *configuration.php* – Sets global variables such as site name and current template
- *cron.php* – Run scheduled activities such as email reminders (run daily by the server)
- *includes/phpmail.php* – Cleans, validates, and sends emails for the system
- *includes/srs\_db.php* – Manages the database connection
- *index.php* – The root file, which is requested by users' browser for most system tasks
- *main.php* – The main content file, which controls page building
- *page/bodyfooter.html.php* – Generates the HTML for the bottom portion of the output
- *page/bodyheader.html.php* – Generates the HTML for the top portion of the output
- *page/bodynavigation.html.php* – Generates the HTML for the site navigation
- *page/footer.html.php* – Generates the HTML footer (ie. “</body>\n</html>”)
- *page/header.html.php* – Generates the HTML header (including title and link tags)
- *rosters.php* – Manages database queries and data manipulation for rostering tasks
- *rosters.html.php* – Generates HTML output for the rostering component
- *setup.php* – Install script (initialises the database, and can uninstall it)
- *users.php* – Manages database queries and data manipulation for users component
- *users.html.php* – Generates HTML output for the users component

## 6.3 Reminder Implementation

### 6.3.1 Scheduling

Reminders are implemented through the use of a cron job. At scheduled intervals (eg. hourly or daily), the server will execute the *cron.php* file, which checks the database for any outstanding reminders and sends them. This is one of the few aspects of the system that requires some manual configuration to set up. The user must manually set up the cron job on their server in order to have reminders function. If cron job functionality is unavailable for any reason, outstanding reminders will be sent out whenever someone accesses the *cron.php* file using their browser.

### **6.3.2 Format**

Below is an example reminder email sent by the system:

Dear Sir/Madame,

This is a message from The Simple Rostering System to remind you of an upcoming task. You requested to be notified on the day of this task.

You are scheduled for the following activity:

Name: Wipe down benches

Date: 09-06-2008

Description:

Use the cloth hanging over the sink to wipe the kitchen benches clean. When done, rinse out the cloth and hang it back up to dry.

This task is part of the 'Kitchen Duties' roster. To view this roster, click the link below (you will need to login):

<http://<Site URL>/index.php?area=roster&task=view&id=1>

Regards,

The Simple Rostering System Administrator

## **6.4 User Accounts Implementation**

User authentication was implemented as a pair of PHP files, one of which is included (or called) by the index.php file of the website. The users.php file contains functions to check whether a user is logged in, validate a login, and log a user out.

### **6.4.1 User Class**

The users.php file specifies a User object class. This class is used to represent the logged in user (if applicable). The processLogin (for logging in a user) and recallLogin (for checking if a user is logged in) functions both return a User object if successful.

```
class SRS_User:
  private $access;
  private $active;
  private $id;
  private $name;
  private $email;

  function SRS_User($uid,$uaccess,$uname,$uemail,$uactive);
  function isActive();
  function getAccess();
  function getUserID();
  function getUsername();
  function getEmail();
  function changePassword($oldPass, $newPass);
```

### 6.4.2 Session Management

Session management essentially involves logging users in and preserving their logged in status in between page requests. This is achieved through the use of cookies. When a user successfully logs in, a cookie is sent to their browser with a unique session code. The browser will then include the cookie value (the session code) in every subsequent page request for the duration of that browser session (until the user closes their browser). If the user logs out manually using the 'Logout' link, the cookie is overwritten, ending the session.

### 6.4.3 Password authentication

The system reduces the risk of password theft by not storing users passwords in plain text (or in any form where it may be feasibly translated back to plain text). The system uses a hash function and some additional data to generate a unique hash code for the password, which is then stored in the database.

This means that the actual password is never stored in the system, which adds an extra barrier of protection against disclosure. The disadvantage of this is that users who have forgotten their password cannot retrieve it – the password must be reset. If a user requests to reset their password, a new random password is generated and emailed to the user. The user can then log in using this password, and change it to one they will (ideally) remember

## 6.5 Interface implementation

The interface was implemented using HTML and CSS. The content of each page was delivered as an XHTML document, the presentation data was specified using CSS (linked from the XHTML file). Some interactivity (as well as form validation) is to be programmed using JavaScript (which was also linked from the XHTML file).

All interactions with the system follow standard web conventions in order to minimise learning curve. Adding or editing roster, tasks, and configuration data is all done using standard HTML forms.

Unfortunately the scope of this project did not allow for the level of user interface refinement that would have brought the system to it's full potential with regard to ease of use. This includes the JavaScript features described above, as well as the visible interface.

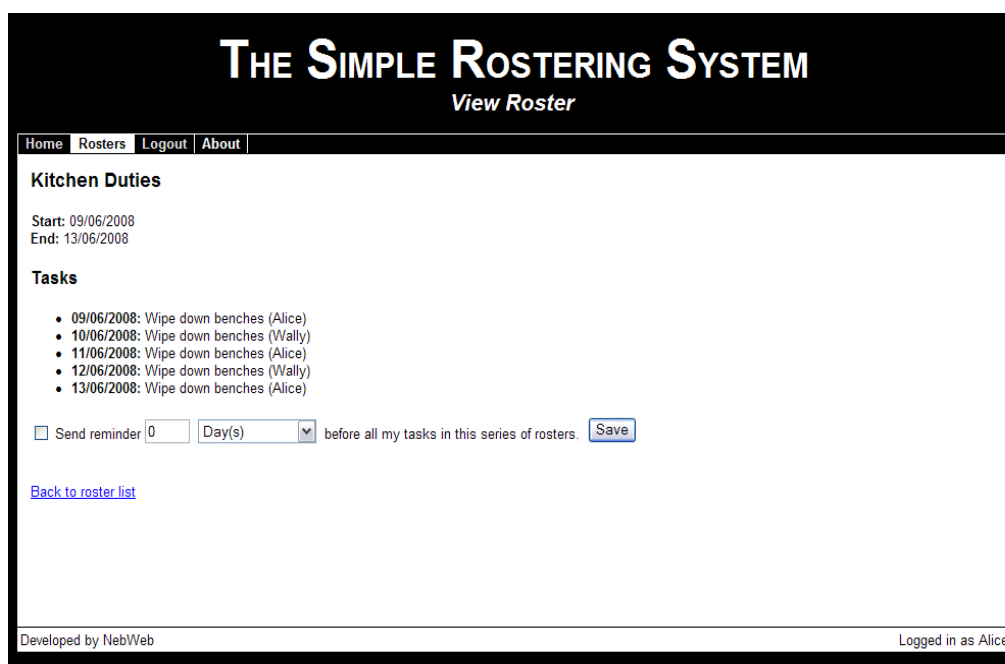


Figure 4: Screenshot of the SRS interface  
(Note that NebWeb is the business name of Ben Griffiths)

## 7 Testing & Evaluation

### 7.1 Integration testing

Integration testing involves testing the whole system once it has been completed. This includes testing specific functions of the system, such as creating a new roster.

Due to the nature of the system and its development environment (the system is tested on a remote server), integration testing formed the backbone of the test suite for the system.

The following functional test were used to verify the system, and successful outcomes can be reproduced using real data:

Task Attempted	Bugs Found	Date of test
Invite a new user	9	26/05/08
New user registration	5	26/05/08
Log in (correct details)	3	24/05/08
Attempt log in (incorrect details or code injection)	2	24/05/08
Log out	1	24/05/08
Reset password	1	26/05/08
Create new roster	1	01/06/08
Invite user to roster	3	03/06/08
Remove user from roster	3	03/06/08
Edit and save roster	11	01/06/08
Create a new task for roster	6	02/06/08
Edit and save task for roster	3	02/06/08
Remove task from roster	2	02/06/08
Assign user to task instance	5	01/06/08
Re-assign task instance to another user	1	01/06/08
Configure reminder	2	08/06/08
Receive reminder	4	09/06/08
Notify participants of changes to a roster	1	09/06/08
Notify participants of changes to a task	2	09/06/08

*Table 1: A representative sample of test outcomes*

As the system is intended to be intuitive and simple to use, the method of execution and desired outcome of the above tests should be self-evident to a user attempting to reproduce the results. If the method of execution is not readily apparent, or the results cause uncertainty, then the system is flawed.

During testing, the system needed to meet all of the requirements specified under *Core Features* (Section 5.1), with no errors or undesired behaviour.

### 7.2 Interface Testing

The user interface was tested on Mozilla Firefox 2.0, Internet Explorer 7.0, and Opera 9.0. These tests involved viewing each individual page in the system and verifying that all elements of the interface displayed correctly and consistently across all browsers.

The XHTML+CSS template adapted for the user interface was previously cross-browser verified, and because very little further CSS development was done in the system, there were no significant issues with the display of the web interface in any of the browsers tested.

### **7.3 Subjective comparison to existing systems**

One of the major goals of this project is to develop a rostering system that is more suited to common shared tasks such as household chores and shared responsibilities in an office environment. The system should achieve all of the objectives of a rostering system effectively (creation, modification, reminders, etc).

#### ***7.3.1 Paper roster***

A paper roster was created in line with the first user story (Section 2.3.1). The same roster was then generated using the Simple Rostering System. The experiences were then compared and contrasted to determine to what extent the goals of the project have been met.

The outcome of the test was that the system was able to fulfill the functional requirements for all tasks related to the rostering scenario, but that it didn't surpass the paper roster in terms of convenience due to the limitations of the user interface. This was a satisfactory outcome in line with the project goals.

#### ***7.3.2 Commercial Rostering Software***

Another measure of the effectiveness of the system (but one which was not carried out as part of this project) would be to perform the same test from Section 7.3.1 above, but using a commercial rostering software package rather than a paper roster. Again the experience would be compared to that of using the Simple Rostering System. This test was not performed as a part of this project, as acquiring a commercial rostering software package was not within the scope of the project.

### ***7.4 Acceptance testing***

Acceptance testing for the system will be carried out by the client, Eric McCreath. It will include use of the system on a trial basis to determine whether it meets his requirements, and to evaluate the feasibility of using it in the long term.

This test will be conducted outside the time frame of this project, and will provide feedback for potential future work.

## 8 Discussion

### 8.1 Conclusions

It can be concluded from the outcome of this project that a simple rostering system such as this one is indeed a feasible concept, and it can streamline roster generation, management, and sharing as compared with paper rosters and commercial rostering software.

One significant conclusion I am able to draw after completing this project, is that having a tidy code structure can make a system appear much simpler, but doesn't necessarily reduce the volume of code required. The opposite can occur, whereby a lot of the complexity of the system is hidden.

### 8.2 Issues

#### 8.2.1 Project Management

The two major project management issues that came up in this project were judging the scope at the outset, and pacing work in accordance with the schedule.

The additional work involved in creating the PHP code for the rostering component (on account of the database structure used) meant that the scope of the work was underestimated. This led to the scope needing to be altered as the project progressed, which is an undesirable step to have to take, and involves difficult decisions. As the user interface wasn't a core concern of the project, the work done on that part of the system was scaled back to allow the project be completed within the time frame.

#### 8.2.2 Database Implementation Issues

One of the main issues to arise with the implementation of the database was the use of foreign keys\*.

It was necessary to specify a particular storage engine in MySQL in order to create foreign key constraints (as the default engine used by MySQL does not support them). It was important to include foreign key constraints in the database to help keep the different tables in synchronisation and prevent old records cluttering up the database.

There was then the issue that the initial database design had circular foreign key references (ie. A Roster had a Series\_ID and a Series had a Template\_Roster\_ID). This caused problems as it isn't possible to create a foreign key constraint to a table that doesn't exist yet, and the constraints were initially being added at the same time as the tables were being created. This meant that the tables had to be created in a particular order, and for the circular foreign keys no order would work (as one of the tables had to be created first, so the other one would not yet exist and errors would result).

This issue was solved by removing the circular foreign key references (which was done for other reasons as well), and the difficulty with creation ordering was resolved by executing a separate set of queries to add the constraints *after* the tables had all already been created.

#### 8.2.3 Code Implementation Issues

It took some iterative refinement to arrive at the final file structure used in the system. The system initially had the users.php and users.html.php files combined (and the same with rosters), and most of the functionality of the main.php file (which did not exist) was located in index.php. This proved to be an unwieldy implementation, and was ultimately abandoned in favour of the 3-area division used in the design of the system, as well as data/output split used by *Joomla!* components.

\*If you are not familiar with databases and their implementations, it is recommended that you read the explanations of 'Database', 'Foreign Key', 'Foreign key constraints', and 'MySQL storage engine' in the Glossary before reading this section.

## **8.3 Future Work**

### ***8.3.1 User Interface Development***

The next step for the development of the system is to create a more efficient and user-friendly web interface. This should include proper calendar-style roster display and more intuitive data input. The system navigation should also be redesigned to a better standard of user-centric task-oriented simplicity, always providing the user with their context in the system and not requiring them to guess how to get where they want to go next.

### ***8.3.2 Open Source Future***

The system can be adapted/modified to a number of purposes and needs by developers worldwide. Any further work carried out on the system by myself (beyond the user interface as described above) will be primarily concerned with improving documentation and code style to facilitate open source development.

### ***8.3.3 Adaptation for Content Management Systems***

There are a number of freely available content management systems that use the LAMP architecture, and the SRS could quite readily be adapted to serve as a plugin/component for such systems.

### ***8.3.4 Other Potential Enhancements***

Any features on the 'Desirable Features' list that weren't included in the finished project are potential future enhancements.

## 9 References

1. V4 Solutions Limited, 2007, *STAFF SCHEDULING SOFTWARE, ORIADOR STAFF ROTA*, Victoria Place, Carlisle, UK, viewed 20 March 2008, <<http://oriador-staff-scheduling.com/>>.
2. VersaDev, 2008, *VERSADEV.COM : HELP DESK, CRM, E-GOVERNMENT AND INTRANET SOFTWARE SOLUTIONS*, Gilbert Street, Adelaide, AUSTRALIA, viewed 20 March 2008, <<http://www.versadev.com/versaershome.aspx/>>.
3. OpenSourceMatters Inc., 2008, *JOOMLA! – WHAT IS JOOMLA! ?*, New York, NY, USA, viewed 20 March 2008, <<http://www.joomla.org/>>.
4. W3C, 2002, *XHTML 1.0: THE EXTENSIBLE HYPERTEXT MARKUP LANGUAGE (SECOND EDITION)*, Cambridge, MA, USA, viewed 20 March 2008, <<http://www.w3.org/TR/xhtml1/>>.
5. W3C, 2007, *CASCADING STYLE SHEETS LEVEL 2 REVISION 1 (CSS 2.1) SPECIFICATION*, Cambridge, MA, USA, viewed 20 March 2008, <<http://www.w3.org/TR/CSS21/>>.
6. The PHP Group, 2008, *PHP: HYPERTEXT PREPROCESSOR*, viewed 20 March 2008, <<http://www.php.net/>>.
7. MySQL AB, 2008, *MYSQL : THE WORLD'S MOST POPULAR OPEN SOURCE DATABASE*, Cupertino, CA, USA, viewed 20 March 2008, <<http://www.mysql.com/>>.
8. Oxford University Press, 2008, *ASK OXFORD: ROSTER*, Great Clarendon Street, Oxford, UK, viewed 20 March 2008, <[http://www.askoxford.com/concise\\_oed/roster?view=uk](http://www.askoxford.com/concise_oed/roster?view=uk)>.

*Appendix A – Full-sized Figures*

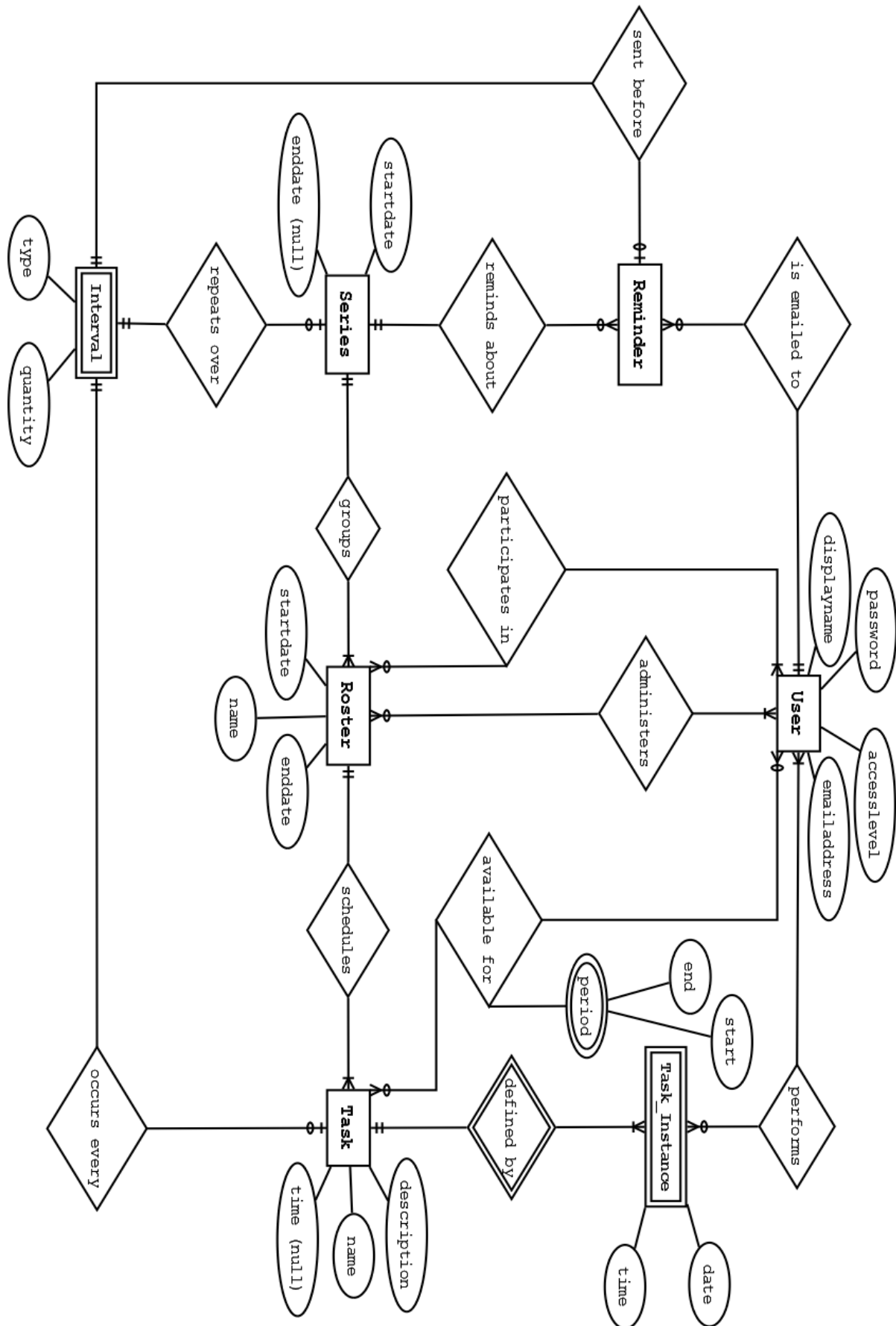


Figure 1: ER diagram for the rostering system

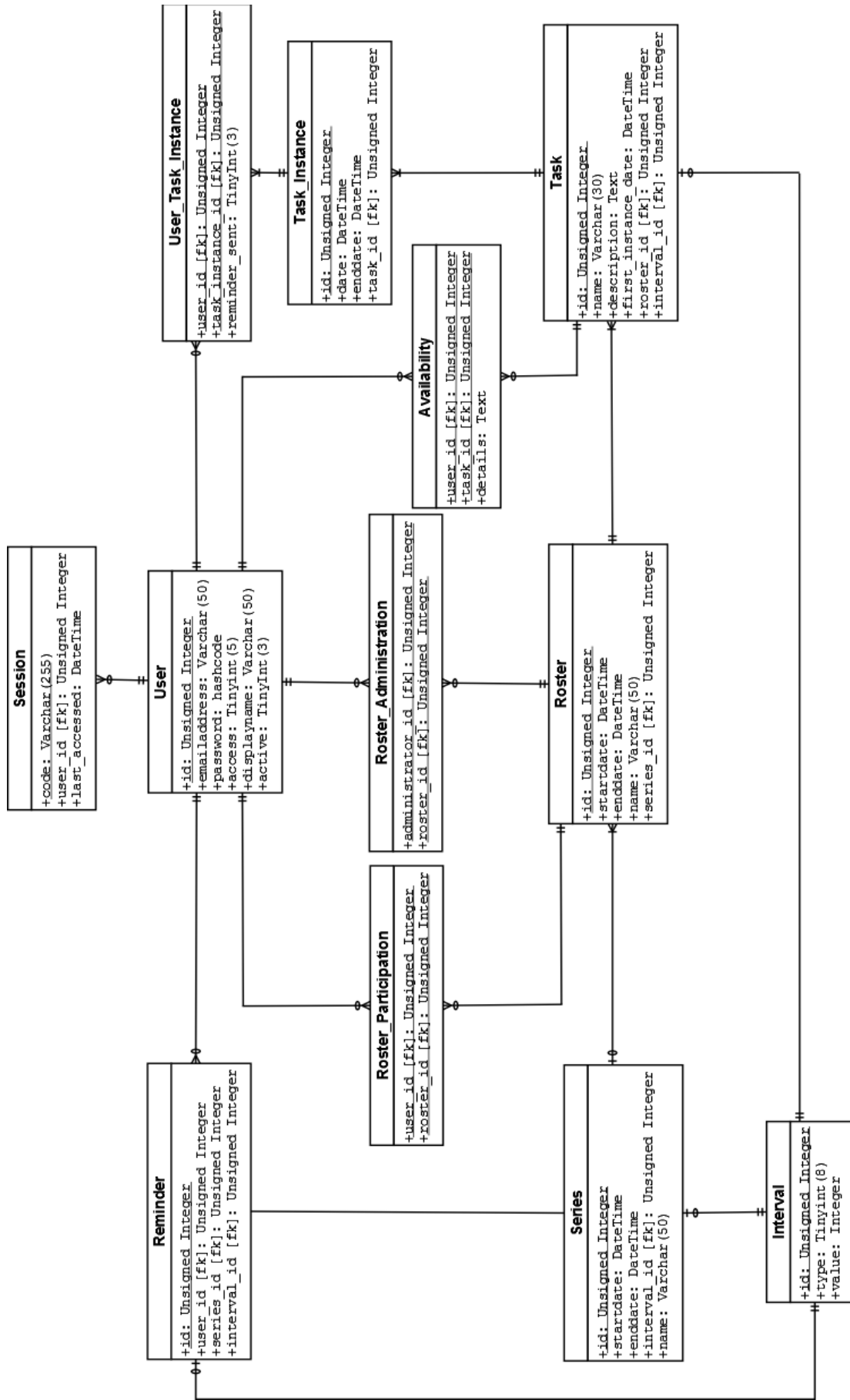


Figure 2: UML model for the rostering system

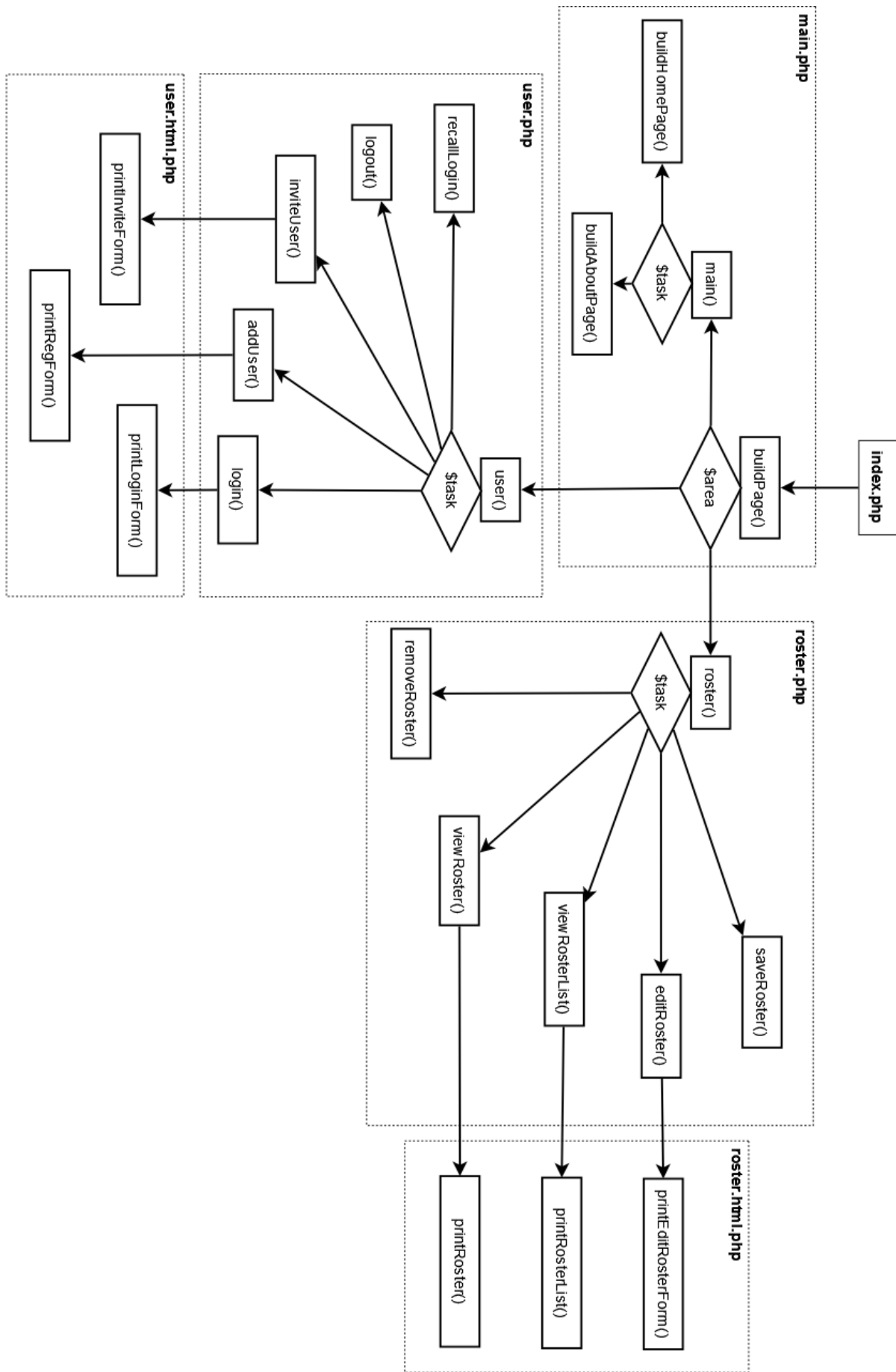


Figure 2: Basic code structure of the SRS system

## ***Appendix B – Installation Guide***

To set up the simple rostering system, you will need to have the following:

- A web server (or hosting account) with:
  - A Linux operating system
  - PHP (version 5.0 or later)
  - MySQL (version 4.0 or later)
  - Configurable cron jobs (for reminders to be sent automatically)
- Sufficient privileges to:
  - Upload/write files to the location at which the system is to be installed
  - Edit files on the server (either remotely or by downloading and uploading)
  - Set up cron jobs (or request that they be set up)
  - Create databases and database users in MySQL

Once you have the above, follow these steps to get the system up and running:

1. Create a database for the system, and a user with full privileges on the database (note the database name, MySQL hostname, username and password, as they will be required later)
2. Upload/write the install package to the server at the location it is to be installed and unpack/unzip it (if you haven't done this already)
3. Edit the configuration.php file by entering your site name, site location, and any other settings you wish to customise where indicated
4. Edit the includes/srs\_db.php file by entering your database name, username, password, and hostname where indicated
5. Use your browser to navigate to the setup.php file within the installation, and follow the prompts (entering your desired administrative account details)
6. Once setup.php has completed the installation, you will be ready to log in to the system for the first time.
7. Once setup is complete and the system is running properly, ***delete the setup.php file from its publicly accessible location as it can be used to uninstall the system.***
8. In order for reminders to work as intended, it is necessary for the cron.php file to be accessed at least daily, either by an automated cron job or manually by your web browser.
9. Congratulations, you're ready to start using the Simple Rostering System. If you have any issues or enquiries, please don't hesitate to contact [srs@nebweb.com.au](mailto:srs@nebweb.com.au)

If you wish to uninstall the system at any time, you can do so by restoring setup.php from the install package to its location on the server, and accessing setup.php?option=uninstall from your browser (***Note that this will result in the loss of all data in the SRS system***). You will then need to manually remove the database you created for the system, and delete the SRS files from their location.