**Students Database**

Technical Design

**Document Revision History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ver. | Date | Author | Reviewed by | Approved by | Changes |
| 0.1 | 01/01/2011 | Someone | Someone else | Someone else | Initial draft |
|  |  |  |  |  |  |

Table of Contents

[1 Title 1 3](#_Toc307307689)

[1.1 Title 2 3](#_Toc307307690)

[1.1.1 Title 3 3](#_Toc307307691)

# General Design

## Objectives

The two main objectives are:

1. Good separation of responsibilities into small components, to ease the development by several people, and making easier to build the application step by step.
2. Keep the design simple, so anyone can easily be part of the development, without long training

To reach the first objective, each component will:

* be located into a separate directory
* have the responsibility of his own tables and data in the database, and provide functionalities to access for other components
* defines its own web pages, if applicable

To reach the second objective:

* we will not use a complete framework which may be long to learn, but may use small part of it or small libraries at some points to avoid doing something that already exist (like importing Excel files…)
* but still we will implement few principles usually provided by frameworks, as good design practices

## Structure

In order to ensure security, and enforce every component follow the same good practices, the components will not be directly accessible. Instead, only one entry point will be provided (index.php), and every request will be processed by this entry point. This entry point will then analyze the request, and then, if it is correct, it will dispatch it to the right component. However this feature will be hidden from the user point of view.

Each component, located in its own directory (component/<component\_name>) will follow the structure below:

* page/<page\_name>.php
* static/<filename>
* service/<service\_name>.php
* <component\_name>.inc

The *page* directory contains one file for each screen provided by this component. This file should have the extension *.php*, all other files in this directory must not have this extension.

The *static* directory will contain static resources (images, scripts…) which is not dependent of the user using the application, and thus can be easily cached by web browsers or web proxies, even among different user sessions.

The *service* directory will contains one file for each service provided by this component. A service is different from a page, in the sense that a page is in HTML format to be display in a browser, while a service may be return data in any format, is not intended to be displayed, and may be called either by a screen, or by a third-party system.

The file <component\_name>.inc will:

* Ensure every module implements security:
  + Declare the list of pages, together with the needed access rights to access the page
  + Declare the list of services, together with the needed access rights to access the service
* Provide any other functionalities that may be accessed by other modules
* Define the database elements it manages

From outside, the files will be accessed using the URLs as follow:

|  |  |  |
| --- | --- | --- |
| Type | URL | Will access in the structure to |
| Page | /dynamic/<component>/page/<page> | /component/<component>/page/<page>.php |
| Service | /dynamic/<component>/service/<service> | /component/<component>/service/<service>.php |
| Resource | /static/<component>/<filename> | /component/<component>/static/<filename> |

The reason to separate dynamic elements from static elements into two different sub-directories from outside point of view is technical: to allow caching (no cookies outside of /dynamic/)

# Components

## Authentication

### Responsibilities

The Authentication component is responsible to validate a username together with a password.

As students and staff already have username and password in different systems, we will not define new usernames and new passwords again. Instead, we will use an external system, where people already have their username and password.

Depending on the project, the external system may be different (Active Directory in Cambodia, Linux in Philippines…). That’s why during authentication we will also provide with a *domain*, specifying on which project the user belongs to, and so which external system should we use to do the authentication.

### Functionalities



The authentication component will not manage any data, but only provide one function.

## User Management

### Responsibilities

The UserManagement component is responsible to attach rights to users. The rights define what a user can do in the application.

A right can be attached directly to a specific user.

Roles can be defined. A role define a set of rights. Roles can then be attached to users. A user can have several roles.

The total rights a user have is the union of (1) the rights directly attached to this user (2) the union of the rights of all the roles of this user. When the same rights is present several times, the less restricted is kept.

### Functionalities



UserManagement depends on 2 components:

* People: as a user is usually a person. UserManagement is responsible to attach a *People profile* to a user.
* Authentication: to check the password when a user login

The component will keep information about the user currently authenticated:

* Its username
* Its roles
* The result of the computation of all its rights
* The identifier of people, to access to personal information about the user

The component will provide functionalities to other components:

* Login and logout
* Check if the user has a specific right

### Data

