# 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

## Overview

The application is separated into *components*, each component being responsible of specific functionalities and data.

A component can provides different resources:

* Dynamic pages: web pages which can be displayed to the user, and being dynamically generated (according to the user, its permissions, content from database…)
* Static resources: resources which are not dependent of a user, or database (typically an image, css file, a javascript…)
* Services: a service is also dynamic (depends on the user, its permissions, do actions on database…), but the difference with a dynamic page is a service is not supposed to provide something to display to the user. A service is typically used to manipulate data: add a new information in database, retrieve data, modify data…
* Translations: in order to make the application multi-languages, each word or sentence can be provided in different languages.

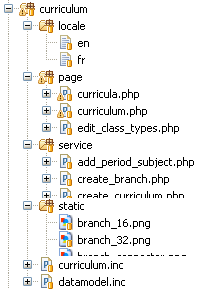
A component can also provide functionalities to other components, to be used directly in PHP and not exposed to the client.

The application also contains a *common* part: some resources (PHP, JavaScript, images…) which are not specific to a functional area, but are aimed to be shared and used by any part of the application. For example, it contains common images, so every page can use the same images for common purposes, or some UI widgets, which will make the application more homogeneous. Also some functionalities like access to the database, or management of translations.

## Structure of a component and URL to access a component

Each component must follow the same structure: each kind of resources must be located in a specific directory, and will be accessed through a common URL structure.

Here is an example if a component *curriculum*:



Component implementation and data model specification

Static resources

URL to access:

/static/curriculum/branch\_16.png

URL to access:

/dynamic/curriculum/service/create\_branch

Services

URL to access:

/dynamic/curriculum/page/curricula

URL to access:

/locale/curriculum/

Dynamic pages

Localized strings: translations in English (file *en*) and French (file *fr*)

### Dynamic pages

Every dynamic page must be located in directory *page* of the component, and have the extension *php*. The URL to access it has the format /dynamic/<component\_name>/page/<page\_name>. For example, the URL /dynamic/curriculum/page/curricula will access to the file curriculum/page/curricula.php

### Services

In the same way, services must be located in directory *service* of the component, and have the extension *php*. It is accessed through URL format /dynamic/<component\_name>/service/<service\_name>

### Static resources

Static resources must be located in directory *static* of the component, and are accessed through the URL format /static/<component\_name>/<filename>

### Localized strings

Translations of words or sentences used by the component must be located in the directory *locale* of the component. For each language, a file named with the language code will contain the translations (example, for English, the language code is *en*). More details about translations will be provided in next sections of this document.

### Component implementation

A component must provide a class implementing the abstract class *Component*.

D:\_PN\student_soft\repositories\lecousin\doc\system\diagrams\Common_PNApplication_.wmf

This class will mainly allow to access to pages and services, implementing security checks, and provide the list of permissions (rights) the component supports. This class will also allow to provide functionalities to other components.

More details will be provided in the next sections of this document about PNApplication and Component classes.

### Data model specification

Each component must declare its data model (tables, format of columns, links between tables, permissions needed to access to tables or columns…). This will be used, first to ensure security checks so we can be sure that if a user should not access to specific data, even a component *forget* to check the permission, the access will be automatically denied. It will also be used to implement generic functionalities to manipulate data. More details about data model will be provided in the next sections of this document.

## URL explanation

### Static and dynamic

The reason to separate clearly dynamic and static resources into URLs starting either with /dynamic/ or /static/ is for performance purpose for static resources.

Indeed, static resources should be cached by the browser of the user: the first time, the browser will download the resource, and then will store it so it does not need to download it again the next time it needs it.

For a resource to be cached by the browser, there are several conditions: the resource must be marked as cacheable, and the request to access it must be exactly the same.

As we are using PHP, with sessions, by default PHP will add a cookie containing the session ID, that the browser will include in every request to the server. That means that every time the same user connect to the application, it will get a different session ID, so the request to a static resource will be different, avoiding the browser to reuse the resource previously downloaded.

Cookies may be restricted to a given path: in our application, we will specify that the cookie containing the PHP session ID is restricted to the path /dynamic/, it means every request to an URL starting with /dynamic/ will contain the session ID, but any other request won’t. So the URLs starting with /static/ will not contain any cookie, thus may be cached.

### Single entry point

In order to ensure security checks, and that we will not allow to access to a file which is not supposed to be sent to the browser, all requests are filtered: the web server is configured to redirect all the requests to a single entry point: index.php

This PHP file will analyze the requested URL, process it, and reject any restricted resource or unexpected request.

## Security for pages and services

Every component must implement security checks. In order to make this sure, any page or service is by default restricted. Thus, a component must explicitly allow its access, after needed security checks.

When the URL /dynamic/selection/page/dashboard is received by the server, here are the steps:



1. The application receive the URL “/dynamic/selection/page/dashboard”. It is analyzed by the single entry point “index.php”: this is a request for page *dashboard* of component *selection*.
2. The component *selection* is retrieved through the *PNApplication* object.
3. The page *dashboard* is requested to the component.
4. The component checks the access to this page is allowed. If not the request is rejected.
5. An object *Page* is created, and requested to generate the page.
6. The file *dashboard.php* located in directory *page* of component *selection* is executed.

Details about the class Page is provided in the next sections of this document.

A similar process is used for services.

Any static resource is allowed: if a resource is restricted, it means it depends on the user, so this is not static anymore. Same for translations, there is no need to secure access to translations.

## Security of data

In order to ensure security of data, direct access to database from a component must be avoided. To access to the database, a PHP script must use one of the classes provided in the *common* part (SQLQuery or DataModel), which will ensure security checks.

Each component declares its part of the data model in a file *datamodel.inc*. The data model is declared using the class DataModel, allowing to:

* Declare tables
* For each table
  + Declare columns, with name and type
  + Specify access restrictions to the table: who can access the table, can add or remove entries in the table…
  + Specify restrictions on each column: who can access to data on this column, who can change a data…

Then, typical way to access to the database is using SQLQuery: this class provides functionalities to easily create SQL queries, and will do all necessary security checks before to send it to the database.

More details on those classes are provided in next sections of this document.