### DEVELOPER GUIDE UNIWEB

## Inroduction

\*\*UniWeb\*\* is web application which helps researchers in maintaining their data in for the purpose of yearly updates as well as is integrated to Canadian Common CV which is website for the researcher to apply for the fundings online. The Researcher need to submit their yearly report to the institute as well as use their CV and data for applying for different awards and findings. The application eliminates the need to redo the work to submit their reports for their funding and to the institution separately. The web application is also a great way for researchers to maintain their public profile where people looking for a particular field of research can know the people actively working in the field . It can also acts as a networking platform for the institutes resarchers and students.

Uniweb has been modified over the time with the advancment in technology the design has been modified to follow a more modular approach.As Uniweb is a single page application which means only a single page is dowloaded from the server and rest all the pages are created virtually (\*\*Virtual Web Pages\*\*). The traditionol model follows the pattern by getting the client request from the client side and then creating each manager which is responsible for submitting AJAX request and then the server handles the request given and gives the desired page. As working over the time we realised there is alot of re writing of the code so as to over come the challange we started to reuse our code and assemble the web pages by implemeting the code in form of widgets.

In the new version of \*\*Uniweb 2.0\*\* the idea is to avoid the rewriting the code and use the previously availaible widgets and cards by just giving a simple set of instructions in the form of YAMl files for assembling the page. The The new format creates the Virtual pages. For the explaining the new approach we can think as Uniweb restaurant which has different pages (virtual web pages) as dishes. There will always be a home page or main page on top of that the new virtual web pages will be created. If user wants to create a new page the user needs to provide the recipie or list of ingredients in yaml and then that recipe will be given to the kitchen/cook (php) for assembling and cooking the desired page. The page once assembled is the delivered as a response to the client.

The more technical aspects of the Uniweb Software are explained in the Later Part of the Code.

Uniweb application is a \*\*single page\*\* application which creates a single request and downloads only a single page from the server \*\*i.e. index.php\*\*. The basic structure of the page is loaded under the views/index.html. The change of the URL triggers an event which is handled to stop the reloading of the page and the pages are dynamically handled through an AJAX request from client to server. Such changes are taken from the browsers and managed by the UNIWeb client JS code instead. When the one and only index.php page is loaded, the server sends all the necessary JS code and configuration parameters for the website. The parameters include specifications about how to render certain pages, and some of the data schemas needed by \*widgets\* in some pages. The main configuration file for all \*\*virtual webpages\*\* is located in `configs/apps/uniweb/ui/menus.yml`. That configuration file defines the menus shown in the standard navbar of UNIWeb, and the optional \*\*tabs\*\* or \*\*views\*\* offered in all pages referenced from the navbar. In other words, `menus.yml` defines \*standard\* virtual pages and their optional \*views\*.

When \*index.phtml\* is loaded it first sets the Head for the HTML page, containing setting the language, Choose the appropriate "base" href to make requests from \*\*index.phtml\*\*. Set the meta data like title and description for the page. The \*metacards\* for the user is defined if user is not login as the data is derived for the user from fb, twitter and other sources. So it sets the default values for a guest user.

The next step after defining the head is to define the body for the HTML page which is done using the \*getBody\* class which returns the object of a client UI. Features like warning a user for oldbrowser notification is included. The main navigation bar for the web page is loaded by rendering the yml file index.yml, which contains the configuration for main navbar. The function calls get frame that return the YML object notation to HTML markup and returns schema frame

in form of html string. The client footer is decided by finding absolute file path of deepest config data file with given name that also helps in dealing with multiple languages. Then the main page CSS, JS is loaded for the entire app at once by the function \*makePageIncludes\*, this function sets the configuration of the page from ‘pages.yml’ and adds the link for CSS, json, JS to HTML as requested. The JavaScript for the page handles the client action. At first the active user info such as member\_id, first name, middle name, and the last name of a user in form of an associated array. If the user is login for the first time or the application is not able to find the last login date a splash window displaying some generic info(\*html/splash\_window\_\*) appears. Script also saves the language selected by the user for the current page and save changes for entire session.

When webpage is ready, the \*UniwebCore\* class is instantiated using client static data. It triggers the \*\*\*init\*\*\* function in the UniwebCore class which is responsible for handeling events like Blur and beforeunload, window resize, changes in URL. Whenever there is a change in URL init function validates the href if the href is correct and not = “#” it prevents the default behaviour and calls the draw content function. In other words, a change in the URL path (ie, not a domain change), will call the `uniweb.content.draw()` method of the singleton `uniweb` object (class `UniwebCore`). The result is that different URL act as \*\*virtual webpages\*\* in the uniweb app.

Once the data and settings are set by the \*\*\*init\*\*\* function the controll is then passed to the draw function of the \*\*CardsWidget\*\*.

Each page has a dedicated manager to render itself in the client. For instance, the home page will have a \*\*\*.yml file\*\*\* and a \*\*HomeManager\*\* class (home\_manager.js) and a \*\*MembersCards\*\* Class in \*\*\*home\_cards.php\*\*\*. This class is responsible for the connection with database. When the URL is changed to redirect to members, The draw() method of the generic ManagerClass will begin by getting known draw specifications of the page via the method getTabDrawParams('draw', params). : {action: getCardList", contentType: "generic", viewType: "home.cards"}. This method takes the given URL params and looks for the draw property in the configs/apps/uniweb/ui/menus.yml specifications received when the real index.php page was loaded.

## How to create a new "virtual webpage"

Let's first test what happens when you try reaching a page that doesn't exist. For example,

https://uniweb-domain.com/test/something-else

will show a page like

![404 example](../assets/404.png)

This page is automatically generated by both the client JS code and the server PHP code. In the client, the request of a page is managed by the singleton `uniweb.content` object (`ContentCore` class). The method `draw()` analyses the URL and determines which `manager` class is in charge of rendering the page. The fist level in the URL path is taken as the name of the manager, with the first letter replaced to its corresponding uppercase character, and the word \*\*Manager\*\* as suffix. That logic is defined in the implementation of `uniweb.createManagerForContent(contentName, params)`, which in turn calls `ManagerClass.makeManager(className, params)` to create the \*manager\* object.

In the example above, this means that `content.draw()` will find \*\*test\*\* as the first level in the URL path and will look for the JS class name `TestManager`, will create an object from it and call its `draw()` method to render the page.

What happens if the is no matching manager? In other words, what happens in the example above if there is no class named `TestManager`?

If there requested manager class doesn't exist, then aon object of the generic `ManagerClass` is created. Since all manager classes inherit from `ManagerClass`, this means that a `ManagerClass` object is always created by any URL. The only difference is whether it is a specialized manager of class `XYZManager` or `ManagerClass`.

\*\*UniWeb 1.0\*\*

The `draw()` method of the generic `ManagerClass` will begin by getting known \*draw\* specifications of the page via the method `getTabDrawParams('draw', params)`. This methods takes the given URL params and looks for the \*\*draw\*\* property in the `configs/apps/uniweb/ui/menus.yml` specifications received when the real `index.php` page was loaded. If the target content type is not defined by those specs, they `draw()` calls `uniweb.ui.drawGenericPage(params)` which in turn will contact the server to request the page to draw. To that end, the method shares the URL parameters with the server-side endpoint `pages.php`. This means that any kind of "webpage" can be made by the server's `PagesController` for URLs that are not handled by specialized client-side manager objects. In particular, the `404 page` is made by the `PagesController` as a last resort.

\*\*UniWeb 2.0\*\*

Overtime, we realized that that is not necessary. The generic method can now cover a wide range of complex pages that require no dedicated manager to render them in the client. Because of this reasoning, we will start by considering how to render webpages using the server-side method. To that end, let's assume that we want the URL

https://uniweb-domain.com/test/something-else

to render something as defined in the server code.

If a virtual webpage is meant to have a navbar link to it (from the main uniweb page), the place to go is `configs/apps/uniweb/ui/menus.yml` because in it we define the menus shown in the standard navbar of UNIWeb, and the optional \*views\* or \*tabs\* offered by such pages.

For our first Exercise, we will not be building a standard page. This means that the page will not be accessible from the main navbar of the UNIWeb application. Start by creating the following YAML file under `configs/apps/uniweb/pages/test.yml`

```yml

contentData:

tag: h1

value: Hello world

```