i-MOS Document

Architecture, file locations and functions

(1st draft)

Prepared by Ben, Saikin and Lining

June 2013

General Idea

The IMOS URI can be interpreted as /controller/function/param1/param2/etc..
.htaccess defines some rewrite rules, that will rewrite the URI, for example, http://i-mos.org/imos/simulation will be rewritten as http://i-mos.org/imos/index.php?/simulation .
Actually, it will centralize all the linking and request, and the framework will do URI routing, construct the corresponding controller, and call the specific function.

The IMOS site is constructed with the PHP framework - CodeIgniter. It separates with three main parts which are controller, model and view (MVC). The Model represents the data structures. Typically your model classes will contain functions that help you retrieve, insert, and update information in your database. The View is the information that is being presented to a user. A View will normally be a web page, but in CodeIgniter, a view can also be a page fragment like a header or footer. It can also be an RSS page, or any other type of "page". The Controller serves as an intermediary between the Model, the View, and any other resources needed to process the HTTP request and generate a web page.

General Architecture

Controller

The constructor initializes required library, helper class and model.

Some pages need not to do any SQL query, the function then directly pass the data to the view. The framework performs *extract($data)* function so that the view page can get the variable.

Some pages need access control, thus, the constructor load the account_model. And the authentication has been done before output the view. The function call *$this->Account_model->IsAuth()*, that checks the session first and compare the id to the user id which is stored by SQL. If the comparison is false, it will redirect to *authErr* page. Otherwise, it will check the user's class whether have enough permission to access the page.

Some pages need to do form validation. It requires form_validation library provided by framework. The validation process loads the config file that function specified and follow the user-defined rules to check whether the input data are valid. Moreover, the verification code implements google reCaptcha API. The reCaptcha helper is loaded to verify the verification code. And then call corresponding model to query data to SQL.

Model

The main ideas of model perform SQL query, compose email, and manage session data.

View

The view use template inheritance helper class. Most pages use 'layout.php' as the base layout. In 'layout.php', some block markers are defined. For those pages extend the layout.php page, they can either override or extend block. By calling *get_extended_block(),* the page will inherit layout.php and further extend the content of the block. Otherwise, the content will be overridden.

Authentication System

Registration

When the request create_user is made, if the validation is success, the model then creates a user. The password of users are encrypted by using Drupal API, which using SHA-512 with salt, running the hash function numerous times. Then the function generates a unique id for activation use.

Activation

The recipient will receive an email with the URI baseuri/account/activate/uuid. The request will be routed to account controller, activate function. And the account model will check the uuid corresponds to the logined user. If so, the uuid record

will be removed and the user becomes activated.

Login Part

For loading each page, the page will make an ajax request to baseuri/account/login_block, to retrieve login or logined form. When loading successes, it will create some click event, login and logout action. When the login action is invoked, it will also make an ajax request to baseuri/account/login, and pass the json data containing username and password to the server. And the server will return three messages, “ok”, “noactive” and “noaccpass”.

Model and Simulation

General

For each model page, it implements some Javascript APIs, including JQuery UI, Knockout JS, JqPlot. There are numerous user-defined bindings.

Tabs

Let’s talk about tabs, it includes numerous steps behind. When knockout starts binding, the callback function *init* in *ko.bindingHandlers.tabs* will be called. The tabIndex would be zero first as the value of selectedTab is zero first. Since the tabIndex is observable and the callback function *activate* will be initialized, and tabs with multiple panel will also be set up. For function *activate*, when the user selects another tab, the tabIndex will be changed to newTab index. After the panel changes success, the callback function *update* will be called. The *update* function is used to update the index of active panel.

Moreover, the extender is created for selectedTab, when the value get updated, the callback function *localPersist* will be called and the corresponding parameter get passed. Insides *localPersist* function, if clients do not support local storage, it is a dummy function. Otherwise, the function is *persistExtender*. The updated value will be stored to local storage with corresponding key.

Data Storage

The parameters, checkboxes data and also tab index are stored in the local storage provided by HTML5. The self-invoke function in *knockout.localPersist.js* will check whether the client support local storage.

Buttons

Knockout also binds corresponding function to the buttons. Take Run simulation as example, the function *simulate* has bounded to the click event of the button. When the click event is invoked, it will do validation and pass the data to *baseuri/modelsim/simulate*. The server will *getNetlist* and perform *simulate*. After server response, the json data will pass to client side. The function *loadPlotData* get called and data is retrieved by a series of ajax request.

File locations and functions

Home

\imos2\css\home.css	
\imos2\js\home.js	define the showcase jQuery plugin for picture showing
\imos2\application\controllers\home.php	controller for home page
\imos2\application\models\Account_model.php	access user information from database
\imos2\application\models\Resource_model.php	Access imos activities and users' experiences
\imos2\application\view\home	view for the home page
\imos2\application\view\layout.php	basic view for all the pages
\imos2\application\config\home.php	user's experience

Models

\imos2\css\model.css	define the font color, size, etc in model page
\imos2\css\disscusion.css	define those in discussion page
\imos2\css\font\font-awesome.min.css	Font Awesome
\imos2\css\jquery-ui\themes\base\jquery-ui.css	Jquery-UI library
\imos2\css\jquery.jqplot.css	define those in graphic output
\imos2\js\fivestar.js	Model rating
\imos2\js\disscusion.js	Client side discussion page checking
\imos2\js\star-rating\	Rating module
\imos2\js\library	Library files
\imos2\js\modelsim\controller.js	User library operations, ko is knockout
\imos2\application\controllers\modelsim.php	controller for model page
\imos2\application\controllers\starrating.php	controller for model rating
\imos2\application\models\modelsim_model.php	Access model information from database
\imos2\application\models\Account_model.php	access user information from database
\imos2\application\models\Discussion_model.php	integrating the comments page
\imos2\application\models\Ngspice_model.php	Configure the ngspice running environment, like the directory, the netlist template
\imos2\application\view\simulation Mode_list.php Model.php \descriptions	 display the model list page manage different models display different models
\imos2\application\config\simulations.php	Location of the ngspice bin file
\imos2\system\helpers\download_helper.php	for example, the download function
\imos2\system\libraries\Upload.php $this->load->library('upload');	for example, the upload function. JSON is used in the data uploading process.

Text Simulation

\imos2\css\node_sim.css	Define some icon, etc
\imos2\css\jquery.jqplot.css	define those in graphic output
\imos2\css\txtsim.css	Define display style for circuit sim
\imos2\css\font\font-awesome.min.css	Font Awesome
\imos2\css\font\font-awesome-ie7.min.css	For IE7
\imos2\js\library	Library files
\imos2\js\simulation\utilities.js	Form submit rules, etc
\imos2\js\simulation\simulation.js	Communication with server
\imos2\js\simulation\plot.js	Chart plotting module
\imos2\js\simulation\param_set.js	Handling user library
\imos2\js\txtsim	For the plotting function , and etc
\imos2\js/txtsim\doc_ready.js	Plotting, x-lable, y-label, etc
\imos2\application\controllers\txtsim.php	controller for this page
\imos2\application\models\account_model	Check the user information
\imos2\application\models\txtsim_model	Get the model cards from server, generating the netlist, run spice, etc
\imos2\application\config\simulation	Location of the ngspice bin file
\imos2\application\models\simulation_model.php	Get user library, etc
\imos2\application\views\txtsim \imos2\application\views\txtsim_template	display the txtsim page template for raw input

Content Management System

\imos2\css\resources.css	Display style for the CMS
\imos2\js\resources.js	Form validate
\imos2\application\controllers\resources.php	Controller for CMS
\imos2\application\models\resources_model.php	Database communication for CMS
\imos2\application\view\resources	View of CMS

Discussion

\imos2\css\disscusion.css	Display style
\imos2\js\disscusion.js	
\imos2\application\controllers\disscusion.php	Controller for discussion
\imos2\application\models\disscusion_model.php	Get comments from database, etc
\imos2\application\view\disscusion	View for discussion page

Resources

\imos2\application\controllers\resources.php	Controller for resource page
\imos2\application\models\resources_model.ph	Read information from database, arrange it according to, say, time
\imos2\application\view\resources	Include all the separate pages
\imos2\application\config\resources.php	Configure the category

Account

\imos2\css\account_create.css	Display style for creating account page
\imos2\js\account_create.js	Client side checking
\imos2\application\controllers\account.php	Controller for account info
\imos2\application\models\account_model.php	Check with database
\imos2\application\view\account	View settings for account
\imos2\application\view\account_create.php	View for account creating
\imos2\application\config\account_create_form.php	Configure form page
\imos2\application\config\account_info_update.php	Configure update page

Contact

\imos2\css\contacts.css	Display style for contact page
\imos2\application\controllers\contact.php	controller for the cotact page
\imos2\application\models\contact_model.php	for sending a message to some email addresses
\imos2\application\view\contact	view for the contact page, for example, the i-mos team member list
\imos2\application\config\contacts_form.php	configuration file for this page