\\Web\www\phramewrk\views\images\control\phramewrk.png  
Documentation

[About Phramewrk 5](#_Toc272357463)

[Purpose and Goals 6](#_Toc272357464)

[Purpose 6](#_Toc272357465)

[Goals 6](#_Toc272357466)

[Architecture 7](#_Toc272357467)

[License Agreement 8](#_Toc272357468)

[Options 8](#_Toc272357469)

[Change Log 9](#_Toc272357470)

[Credits 10](#_Toc272357471)

[Kirk Ouimet 10](#_Toc272357472)

[Kam Sheffield 10](#_Toc272357473)

[Seth Jensen 10](#_Toc272357474)

[Matt Smith 10](#_Toc272357475)

[David Bailey 10](#_Toc272357476)

[Grant Collard 10](#_Toc272357477)

[Installation 11](#_Toc272357478)

[Server Requirements 12](#_Toc272357479)

[Downloading Phramewrk 13](#_Toc272357480)

[Installation Instructions 14](#_Toc272357481)

[Provisioning a New Instance 14](#_Toc272357482)

[Upgrading from Previous Versions 15](#_Toc272357483)

[Troubleshooting 16](#_Toc272357484)

[Getting Started 17](#_Toc272357485)

[Introduction 18](#_Toc272357486)

[Setup an Instance 19](#_Toc272357487)

[Control Panel 20](#_Toc272357488)

[Core 21](#_Toc272357489)

[Phramewrk 22](#_Toc272357490)

[Router 23](#_Toc272357491)

[Routes 24](#_Toc272357492)

[Controllers 26](#_Toc272357493)

[Security 27](#_Toc272357494)

[Debugging 28](#_Toc272357495)

[Exceptions 29](#_Toc272357496)

[HTTP Errors 30](#_Toc272357497)

[Logging 31](#_Toc272357498)

[Error Handling 32](#_Toc272357499)

[Continuous Integration 34](#_Toc272357500)

[Modules 35](#_Toc272357501)

[Overview 36](#_Toc272357502)

[Creating Your Own Module 38](#_Toc272357503)

[Accounting 39](#_Toc272357504)

[Advertising 40](#_Toc272357505)

[Affiliates 41](#_Toc272357506)

[Analytics 42](#_Toc272357507)

[API 43](#_Toc272357508)

[Blogs 44](#_Toc272357509)

[Cache 45](#_Toc272357510)

[Cart 46](#_Toc272357511)

[Charting 47](#_Toc272357512)

[Chat 48](#_Toc272357513)

[Communication 49](#_Toc272357514)

[Content 50](#_Toc272357515)

[Cookies 51](#_Toc272357516)

[Customers 52](#_Toc272357517)

[Databases 53](#_Toc272357518)

[Documentation 54](#_Toc272357519)

[Encryption 55](#_Toc272357520)

[Forms 56](#_Toc272357521)

[Messaging 57](#_Toc272357522)

[Models 58](#_Toc272357523)

[Optimization 59](#_Toc272357524)

[Profiling 60](#_Toc272357525)

[Questions and Answers 61](#_Toc272357526)

[Routes 62](#_Toc272357527)

[Sales 65](#_Toc272357528)

[Scheduled Tasks 66](#_Toc272357529)

[Sessions 67](#_Toc272357530)

[Social Objects 68](#_Toc272357531)

[Source Code 69](#_Toc272357532)

[Support 70](#_Toc272357533)

[Tasks 71](#_Toc272357534)

[Transactions 72](#_Toc272357535)

[Users and Accounts 73](#_Toc272357536)

[Utilities 74](#_Toc272357537)

[Arr 75](#_Toc272357538)

[Dir 76](#_Toc272357539)

[File 77](#_Toc272357540)

[HtmlDocument 78](#_Toc272357541)

[HtmlElement 79](#_Toc272357542)

[Image 80](#_Toc272357543)

[Json 81](#_Toc272357544)

[Logger 82](#_Toc272357545)

[Mail 83](#_Toc272357546)

[Network 84](#_Toc272357547)

[Number 85](#_Toc272357548)

[Object 86](#_Toc272357549)

[String 87](#_Toc272357550)

[Time 88](#_Toc272357551)

[Url 89](#_Toc272357552)

[Scripts 90](#_Toc272357553)

[Styles 91](#_Toc272357554)

About Phramewrk

Purpose and Goals

Purpose

Designed specifically with real-world business problems in mind, Phramewrk is a web business framework. Phramewrk is intended to be as user friendly as WordPress and as developer friendly as CodeIgniter.

Goals

Performance by Lightweight Design

Rapid Business Development Using Modules

Ease of Management through Central Control Panel

Architecture

Like many popular web frameworks, Phramewrk uses the Model-View-Controller design pattern.

License Agreement

Options

Open for the public to download for free for non-profits, licensing fee per domain for businesses and others. Only gives basic modules

We sell additional modules in the Phramewrk store at a premium per month or one off

We have code in Phramewrk that gets called and reports home randomly when used in all versions

Developers can submit modules to the Phramewrk store

Developers can review documentation and comment

We also offer design services and hosting of Phramewrk for you

We will host your Phramewrk project for free (under 5000 visits)

Change Log

Version 1

Credits

Kirk Ouimet

Creator

Kam Sheffield

System Design and Programmer  
Modules: Databases, Models

Seth Jensen

Module Design and Programmer  
Modules: Forms

Matt Smith

Tester

David Bailey

Business Consultant

Grant Collard

Business Consultant

Installation

Server Requirements

Minimum Requirements

Apache 2.x with mod\_rewrite enabled

PHP 5.2+

Optional Requirements

MySql, SQLlite, MSSQL, Oracle

Memcache

Downloading Phramewrk

Phramewrk is available for download at:

<http://www.phramewrk.com/download/>

You may use the Linux command *wget* to retrieve a zip file from the server:

wget http://www.phramewrk.com/download/latest.zip

Installation Instructions

File Structure

Provisioning a New Instance

Download phramewrk and extract it to a folder  
Run /create.php and pass in the directory you want the site to live  
The directory will be created along with a bunch of files  
By default phramewrk will use only one main phramewrk core dir  
and support multiple sites at the same time.  
Make sure you have a virtual host setup for the new site  
Visit the main site directory and you will be given a setup prompt  
You can set  
Site name  
Databases  
Other configuration options  
Routes  
Internationalization  
Put site in development mode (debugging and perf testing)  
Put site offline with a message (Down for maintentance)  
All of these can be edited any other time using the /phramewrk/ route  
All site content can be managed from the phramewrk interface  
You can set a dev server and a production server, and the dev server  
can talk to the production server and update it

Upgrading from Previous Versions

Write after version 1 release.

Troubleshooting

Community support is available online at:

<http://www.phramewrk.com/support/>

Getting Started

Introduction

Phramewrk…

Setup an Instance

Control Panel

Core

Phramewrk

Phramewrk

Sets the config

Router

Identifies the current controller and passes the data to it via Security. Returns the reponse of the security which has the response of the controller

Or, retrieves cached data

Security

Identifies what the data is and encodes it appropriately (JSON, arrays, files, sessions, cookies, etc)

Verifies the data being sent to the controller is clean

Passes data to the controller

loads security settings from security.php in site phramewrk

Has a controller which has a view

Cache

On disk and memory caching for string and database responses

Controller

Models have functions that get Html based on the model

Model lists manage grousp of models

Collects views from Models and actual views

Functions in controllers call views and pass variables in them and and return views

Interacts with the database models and loads the appropriate view and passes the view raw data

By default, home is the function that runs without a specific function call

Each function in the controller represents a different page

Will always return a View

/bio/socwall.com/search-engine/google/google-pagerank/

a route declares that bio is to activate the Bio controller which will be in controllers/bio/BioController

it receives the variables socwall.com,search-engine,google,and google-pagerank in a data array where the keys are defined by the route

domain = socwall, category = search engine, sub cat = google, metric = pagerank

the bio controller loads the SearchEngine controller which loads the google controller

The bio controller has a default function called BioController that

Database (model)

Handles all calls to the database

Record (model)

Does crud for

Records (model)

Manages groups of records

View

A view is anything that has a to string method

The final output to the user

type

data

Document is a type of View

Extends response and element

Element

Independenct component

Default

/(HomeController)about(view)/(parameter)

name represents root folder

functions inside represent sub folder

function parameters reprent further sub folders

Controller accesses models to get information from the database

Provides raw data (not elements, html, etc) to the view

View iterates and builds html

template -> aggregate of views

Router

Routes

Controllers

The controller uses models to build the headers and views (documents, elements, JSON, etc)

Controller may use caching to retrieve data from the models

Views can access the user class to check permissions for conditionally displaying items

and returns them to the route, which will cache them and display them

Security

Examines all of the $\_REQUEST data (post and get) for anything malicous. Automatically identifies files, images, json objects, arrays, strings, and integers and puts them in the appropriate type

Security receives all of the routes in order from parent to child

By default, all requests are granted access, unless there are policies to evaluate, in which case they are denied access by default. A logical OR is applied to all policies (if one fails, it goes to the next until it finds true or fails)

Debugging

Exceptions

HTTP Errors

Logging

Error Handling

Exception, fatal error, warning, and notice handling:

**Development and Staging Mode**

Shows all notices, warning, and fatal errors on the view they are generated from

Errors are displayed in raw format

basically just error\_reporting(E\_ALL)

**Production Mode**

Hide all notices and warnings, no matter what

Redirect fatal errors to a page (something like the fail whale)

Tell the user that the error has been logged

Store information about the error in the database or some other type of log (developer's choice)

E-mail the error to an array of e-mail addresses (developer's choice)

Store the error in the bug tracking module

Error class forms the appropriate message for the Logger class

Logger class is used to log the error

Logger class can also be used to e-mail the error

bug system?

Control Panel for Errors

You can configure how you want errors handled in production mode in settings - this is configured per instance

What error levels you want to send a message about and the user or e-mail addresses you want to receive it (sms, e-mail)

What error levels you want to store and where (log file or database)

What controller function you want to route to on for each of the HTTP status codes (custom message for each one)

What controller&/function you want to route to on fatal errors

This way the user can specify specifically how they want errors to display

By default there will be a phramewrk controller that will handle the errors

The function will receive an array of values -> error title, error code, error description

You should get a ton of information - everything from \_SERVER and \_REQUEST (post/get) the file, stack trace, everything, exception information, etc

You shouild be able to use the control panel to review all of the recent errors, see which ones had e-mails sent out, when they happened, and get all of the information you need

Error and warning pages

There are default error views for every error code and a catch-all customizable one - distributed in the template

These need to be descriptive by default, but also be easily changed and pulled into an existing site through a controller function

Exceptions

An exception is thrown and not caught

Dev mode: details concerning the exception are output on the screen

Prod mode: user is redirected to an error page with a custom message

Continuous Integration

Modules

Overview

Modules section in settings

dependendies

run setup on them if you want them enabled

Module is a class that is extended

Module class has a setup function that will copy files, generate schema models, etc

modules have datamodels

modules need a registry to get and store options in

unit testing

continous integration

commits to a project must pass certain requirements before being rolled out

Module::install

Module::checkDependencies

Module::activate

Module::deactive

Module::delete

Registry::get(

problem:

having 15 sites run off of one phramewrk directory that is shifting beneath them

response:

the phramewrk core doesn't shift

refactor your 15 sites

have a migration wizard to show you why you can upgrade yet

unit testing for each module

Modules add additional function to a site

\* modify the control panel navigation

\* have custom images, scripts, and styles, and views that can be pulled into an existing instance

On init:

Check which mode you are in and set the error reporting for the level you are on

if you are in production - make sure you set an error handler that catches crazy errors and redirects them appropriately

Creating Your Own Module

Accounting

Advertising

Affiliates

Analytics

API

Blogs

Blog controller can be extended in control panel - autogen models

Cache

Cart

Charting

Chat

Communication

Content

Content::get('key') uses memcache then db

Content::set('key') uses memcache then db

Cookies

Customers

Databases

Documentation

Documentation controller can be extended

Encryption

Forms

Messaging

Models

Optimization

Profiling

Configure on the instance level

Set a probability on running

Stores information about the server side processing of each request

Questions and Answers

Routes

Router::error(code, title = '', message = ''); ?

Route policies

routes have policies which follow the amazon access policy structure

routes can have subroutes, where when the parent route has been matched against it will then perform checks on the subroutes

there is a root route which can have policies on it, which is the instance access path

Route rule

Conditions

the user must be logged in

the user must belong to an account with account type Phramewrk

the user must belong to an account named Phramewrk

the user must have a username kirkouimet

the user must have the account role \_\_\_\_\_

the user must have the account privilege \_\_\_\_\_\_\_\_

this rule is in affect at these times

time of day

Sample rules

/bio/

the user must belong to account type pagemass and have the account feature bio or they are redirected to the buy bio page

rules:[

]

/phramewrk/

the user must belong to the account type phramewrk and belong to the phramewrk account

If conditions are met

point to a file, or point to a controller

If conditions aren't met

Redirect to a particular page and set certain variables (like why they can't get in)

Could be a login page, could be an error page

A user can access the control panel if they are logged in and belong to an account named "Owners".

If the user has the account role "Employee" then they can only login during Monday-Friday, not weekends.

If the user has the name "root" they can login regardless of any other rule. If none of the rules succeed,

then redirect the user to a page telling them specifically why they are unable to get to the control panel.

E.g., "It's the weekend and you are only allowed to login Monday-Friday."

Evaluation logic:

Browser sends a request

htaccess

If the request is for a anything in instanceAccessPath/images or instanceAccessPath/scripts or instanceAccessPath/styles, the request is immediately served by htaccess

Note: The user can specify a file or folder in any of those directories which will force Phramewrk evaluation of the resource

Note: if the request ends in .php, it needs to be processed (watch out for security problems here)

Everything else is passed to Phramewrk via the index.php file

Sales

Scheduled Tasks

Sessions

What if session has timed out and they submit a form?

This only matters if they need to be logged in in order to access the form

Social Objects

SocialObject controller can be extended in control panel - autogen models

SocialObject, SocialObjectRating, SocialObjectComment

Source Code

Support

Tasks

Transactions

Users and Accounts

Permissions

Route level policies and statements determine general access to different resources

Stored in settings under the route

User level permissions determine access to specific resources or actions

Dynamic cascading ACLs determine static access to resources

Stored in the account\_type\_permission table

Account level policies can be issued and expire

User level policies can be issued and expire

Login -> if users-and-accounts module is not set, then just use the settings login, hide the remember me button

Login - lost your password for database and for settings logins

What about checking permissions if the user has root access?

What about checking permissions at the routing level?

What about checking permissions at the routing level on a jformer request?

What if the user doesn't have root access and what about checking user permissions?

if(User::can('View Phramewrk')) {

Remember me cookies need to expire in 60 days - this should be customizable from settings

Login throttling on settings or database logins

Login throttling on cookie logins

Store only essential information in the session, not the entire user object

Garbage collection on user\_cookie\_login

Garbage collection on user\_session

Updating user\_session and tying the appropriate user to it

reset password

Utilities

Arr

Dir

File

HtmlDocument

HtmlElement

Image

Json

Logger

Mail

Network

Number

Object

String

Time

Url

Scripts

Styles