NASA EVA
Gamification

HANDOVER DOCUMENT - MediaWiki Basics

PHASE II

Updated By (Phase 2)
Laura Addiego
Samia Alam
Kelli Corey
Charles Milk
Adeola Odusola
Prepared By (Phase 1)
Kevin Fortier
Michael Salgo
Victoria Guadagno
Okechukwu Ogudebe
Jacqueline Macfadyen
Montrell Nuble

Table of Contents

Introduction 3

  Document Overview 3

Need For MediaWiki 3

How Extensions Work 3

  File hierarchy 4

    extension.json 4

    body.php 4

    i18n/*.json (Localization) 4

  Special Pages 4

  Hooks 5

Database Interaction 5

Debugging Tips 5

Helpful Resources From Stakeholder 6

Support and Contact Information 6

Inspiration Credit 6

Versioning 6

Version Number	**Description of Change**	**Author**	**Date**
1.0	Initial Creation of Document	Kevin Fortier	4/6/2018
1.1	Added Inspirations and Contact Information	Victoria Guadagno	4/8/2018
1.2	Revised Document for Formatting and content	Michael Salgo	4/9/2018
1.3	Updated Cover page	Samia Alam	8/2/2018
1.4	Added Need For MediaWiki section	Samia Alam	8/3/2018
1.5	Added MediaWiki Resources from Stakeholder	Samia Alam	8/3/2018

Introduction

Document Overview

This document is designed to supplement the information that can be found at mediawiki.org, as it specifically relates to the continued development of the NASA EVA Gamification extension. The goal of this document is to provide an easier learning curve for future teams selected to work on this capstone project for UMUC's SWEN 670 course. It is assumed that most students in the course may not have prior MediaWiki knowledge.

This document assumes a MedaWiki version of 1.27.

Need For MediaWiki

Students will not be able to get access to actual NASA wikis. They are firewalls-restricted and access-controlled based on the type of data that gets posted. For this development work, you'll need to set up your own wiki. The stakeholder Daren Walsh has suggested to use meza [0], which is a set of scripts that takes a bare bones CentOS machine and installs and configures everything you need for your own wiki farm. You can then either create some pages with fake data to get started, or you could download a few pages from Wikipedia. The content is really not important for this project since the project focuses more on user activity.

How Extensions Work

As the name implies, extensions add functionality to a core product; in this case, to MediaWiki. There needs to be a working MediaWiki installation already in place before extensions can be added. Because MediaWiki is Open Source software, anyone can write an extension and add it to the community. The following link is a complete list of the community extensions https://www.mediawiki.org/wiki/Special:AllPages/Extension:. These extensions can be used for reference to see how others have implemented ideas.

Every extension is different. Some are designed to change the look-and-feel of a wiki, add API functionality, handle additional media types, or something else. As such, each extension could have files from different programming or scripting languages (i.e. JavaScript, CSS, PHP, and DDL statements from a DBMS).

File hierarchy

When creating or improving an extension there are a minimum of three files that need to be included, as detailed at https://www.mediawiki.org/wiki/Manual:Developing_extensions. Additional details of these files are provided below.

extension.json

JSON is an open-standard data exchange format. Every extension in MediaWiki contains an extension.json file, that MediaWiki is hard-coded to look for. Most values in this file are optional to include; only manifest_version and name are required. https://www.mediawiki.org/wiki/Manual:Extension.json/Schema contains the full list of values understood in the schema.

body.php

By convention, the filename should take the form of ExtensionName_body.php, but it is not enforced. This file contains the main part of the extension code. If the extension is complex enough, convention says to put additional source code in an /includes directory, to minimize what is at the root level of the extension file structure. MediaWiki leaves the rules relaxed, to allow for more creative freedom for extension creativity. Only the programming constructs of the PHP language are enforced.

i18n/*.json (Localization)

The i18n localization concept allows for internationalization of the content. The files are kept in a /i18n directory and consist of a master qqq.json file and as many additional language files as there have been translations.

This concept works better with an example. Take the familiar “Hello World” notion. If an extension wants to print “Hello World”, the correct MediaWiki way is to define a message tag in the qqq.json file; something like ‘hello-world-message.’ In the qqq.json file, the ‘hello-world-message’ is also given a description of what it is and what it is used for, so future programmers and translators understand where the message is used. Then, the message is translated into each language file the extension has. In the en.json file for English, ‘hello-world-message’ would simply be “Hello World.” In the fr.json file, ‘hello-world-message’ would be “Bonjour le monde.” The jp.json file would have “こんにちは世界.” When a user changes what language the wiki will display in, the new i18n json files are used for display purposes.

Special Pages

While this topic may fall under a higher-level “how MediaWiki works,” each extension has the freedom to create Special Pages for their own purposes. Wiki sites are designed to serve pages to users based on categories and groups of the content they contain. If any action falls outside of this norm, it may be handled by a concept called a Special Page. Take, for example, a way for an administrator to see a list of users or IP addresses which have been blocked from accessing the site. This functionality is handled by a Special Page, as it does not neatly fall into categories or groups and is outside of what users would normally do on a wiki.

Most extensions limit their Special Pages to administrator-like functionality, if they have them. The underlaying PHP can be written to check for correct permissions to accomplish this. However, some Special Pages, such as the Gamification extension, were written to aggregate additional non-core MediaWiki data in a presentation that may differ than the rest of the wiki it was installed to. This allows for greater freedom for the extension to display data in a manner that aligns with the Gamification process.

Hooks

MediaWiki is an event-driven platform. If a user edits a page, one or more edit actions are executed to satisfy the request. As of this writing, MediaWiki has over two hundred (200) events, detailed at https://www.mediawiki.org/wiki/Manual:Hooks.

An extension can append additional code to each of these actions, by hooking onto them. The additional code should aim to be minimalistic in nature, as the user and even administrators of the wiki will notice the effects of an extension that changes the action response times of a wiki.

Database Interaction

Interacting with the database layer must go through the MediaWiki API, as MediaWiki is designed to work with different DBMS software and the API creates this layer of abstraction. As such, extension code should never call SQL DDL statements directly. The API layer, through wrapper functions, offer access to the database; https://www.mediawiki.org/wiki/Manual:Database_access#Wrapper_functions offers the full list and examples.

Because wikis could be complex, with database replication and multi-server load balancing, the extension's PHP should be coded in a way that isolates read-only SELECT queries from any code that could alter or write to the database. The read-only statements should be done against DB_REPLICA or DB_SLAVE constant values, to limit the load on the DB_MASTER database which should handle all UPDATE, INSERT, or DELETE actions.

Debugging Tips

During extension development work, adding and saving new code will frequently cause the wiki to respond with Error 500-Internal Server Error, without any additional information. The following tips and tricks were found to help investigate where to look for what is causing the problem.

- To check if the problem exists in the PHP syntax, open a command prompt or terminal window to the file location and run: `php <filename.php>`
  Any syntax errors will display to the screen.
- The Spring 2018 group found that adding the following 4 settings to the MediaWiki's LocalSettings.php file aided in debugging MediaWiki processes on the website:
  ```
  $wgShowDebug = true;
  $wgDebugComments = true;
  $wgEnableParserCache = false;
  $wgCachePages = false;
  ```
  These settings will cause MediaWiki to produce debug messages in addition to the normal page display and tell it to bypass any cache settings, so code changes are reflected in the page output quicker.
  For Meza environments, these settings should be placed in the /opt/conf-meza/public/postLocalSettings.d directory as a stand-alone php file. This way, the settings are retained after a deployment of the environment.

- The above ShowDebug setting will also display the concatenated SQL command which the database wrapper functions will produce. This can aid in determining why function call is failing.
- Calls to wfDebug() can also be used to display message on the debug section of pages.
- If all the above fail, MediaWiki is configured to redirect PHP or MediaWiki errors to log files. In a Meza environment, these files are located at /opt/data-meza/logs/php_errors.php and /opt/data-meza/logs/php_errors.php However, those logs typically do not capture all issues and should be a last resort.

Helpful Resources From Stakeholder

- Link provided by Daren Walsh regarding Gamification design framework for MediaWiki: https://github.com/darenwelsh/Profiler/blob/master/FRAMEWORK.md
- Link regarding profiler extension to measure and profile user activity: https://github.com/darenwelsh/profiler
- MediaWiki extension to track each users' progression from new contributor to mastery: https://github.com/darenwelsh/UserJourney
- Detailed analysis of Gamification: https://yukaichou.com/gamification-examples/octalysis-complete-gamification-framework/

Support and Contact Information

Developer Support Portal: https://discourse-mediawiki.wmflabs.org/

Technical Resource:

Cindy Cicalese

ccicalese@wikimedia.org

Ms. Cicalese is a neighbor of Dr. Michael Brown and is helping out as a favor to him. She recommends posting your question to the Developer Support Portal and emailing her to look for it there.

Inspiration Credit

StackOverflow's Profile Page (www.stackoverflow.com) inspired the Spring 2018 project team. We used the look and feel from this page as the basis of our design.

Versioning

The extension.json file informally documents the version of the extension. This version information can be handled separately or in conjunction in a repository such as GitHub and does not need to be a part of the extension.json file, but helps with pairing together versions installed with technical support.