Juturna Developer Manual

Last Updated: August 17, 2016

Table of Contents

1. Introduction....1
2. System Overview....1
   - 2.1 General Client-Server Interaction....2
   - 2.2 Overview of Key Software....2
     - 2.2.1 Database....2
     - 2.2.2 Database Administration....2
     - 2.2.3 Web Server....2
     - 2.2.4 Website Framework....3
     - 2.2.5 Website Display....3
3. Projects that Use the Juturna Framework....3
   - 3.1 CVC Well-Being and Your Watershed....3
     - 3.1.1 Introduction....3
     - 3.1.2 Setting Up a Developer Instance....3
       - 3.1.2.1 Source Code....3
       - 3.1.2.2 Setting Up the Project Database....3
       - 3.1.2.3 How to Create an Entity or Modify Existing Entity....4
     - 3.1.3 Database....5
       - 3.1.3.1 List of Database Tables....5
       - 3.1.3.2 Relationship Map....8
     - 3.1.4 Debugging the Code....9
     - 3.1.5 Commenting the Code....9
     - 3.1.6 AODA Compliance....9

Illustration Index

Illustration 1: General client-server interaction....2
Illustration 2: Adding a new extension to the database....4
Illustration 3: Database relationship map....8

1. Introduction

The Juturna web-based GIS platform was created by the Department of Environmental Studies at York University for use with various projects. It provides a user interface, database storage, and web-mapping display for user interaction with various spatial data. It can be customized to various applications where user input and interaction with spatial data is desired.

2. System Overview

The framework is constructed from a PHP (open source server-side scripting language) framework

called Symfony hosted on an Apache HTTP Server (web server). PHP language is used to both query a PostgreSQL database (with a PostGIS extension for locational data) and construct the HTML for the web server to display. The PostgreSQL database is administered by a web-based application called phpPgAdmin. The final display of the HTML is handled by Bootstrap, Leaflet, GeoJSON, TopoJSON, and jQuery.

2.1 General Client-Server Interaction

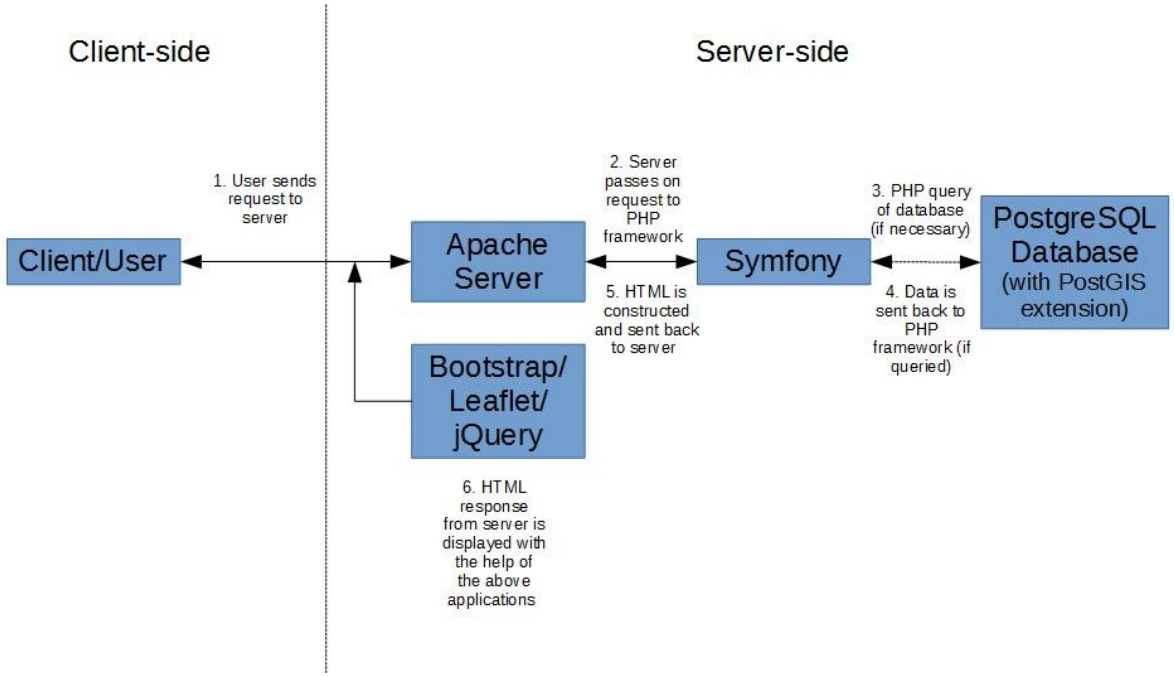

Illustration 1: General client-server interaction

2.2 Overview of Key Software

All of the software used is open source. "Open source software is software that can be freely used, changed, and shared (in modified or unmodified form) by anyone. Open source software is made by many people, and distributed under licenses that comply with the Open Source Definition" ("Welcome to The Open Source Initiative," n.d.).

2.2.1 Database

PostgreSQL is a database application which has been extended with PostGIS to enable the handling of geographic objects. All of the site data is stored in this database, including spatial data for mapping, user information, user input, and website structure.

2.2.2 Database Administration

phpPgAdmin is a web-based administration tool that is used to interface with the PostgreSQL database. This tool is used by the database administer to create tables and add data to the database.

2.2.3 Web Server

Apache HTTP Server is used to host the website and process user requests/responses.

2.2.4 Website Framework

Symfony is the PHP (open source server-side scripting language) framework from which the website is constructed. It resides on the web server. Its PHP framework allows for querying of the data in the database and the construction of HTML documents for display.

2.2.5 Website Display

Bootstrap is an open source HTML, CSS, and Javascript framework. It is used for display of the web content (i.e., how the HTML content provided by Symfony should be displayed).

Leaflet is an open-source JavaScript library for interactive maps. It powers the interactive map that provides the final graphical display of the geographic data queried by Symfony from the PostgreSQL database.

GeoJSON and TopoJSON spatial data formats are used to improve map display performance.

jQuery is a javascript library.

3. Projects that Use the Juturna Framework

3.1 CVC Well-Being and Your Watershed

3.1.1 Introduction

This web application “Well-being and Your Watershed” was created for the Credit Valley Conservation Authority. It's main goal is to get watershed residents to familiarize themselves with their watershed, recognize the important benefits that natural ecosystems provide, encourage interaction with those natural ecosystems, and share personal stories of those interactions. It uses the Juturna framework to accomplish these goals through the use of watershed spatial data and an interactive map.

3.1.2 Functional Specifications

The below list outlines what the web application is designed (or is being designed) to accomplish:

- Spatially represent various layers of GIS data
    - Link this data to specific benefits for watershed residents
        - Provide referenced information on each benefit and the links to ecosystem services
    - Link these benefits to both ecosystem services and human health domains as identified in the Millennium Ecosystem Report
    - Allow users to turn off and on GIS data via a legend
        - Legend has capability for multiple categories

- User registration for user input and customized user interface
    - Allow for user-generated spatial features: points, lines, polygons (along with associated text and files)
    - Allow registered user to set map centre and zoom quickly using three preset zoom levels
    - Allow registered users to measure distances and areas with the ability to save these measurements
- Scenario planning tool to show results of specific landscape interventions (in development)

3.1.3 Setting Up a Developer Instance

3.1.3.1 Source Code

The application source code is located on GitHub at https://github.com/josephzhao/cvcbrowser.

To clone the repository:

```
% git clone git://github.com/josephzhao/cvcbrowser.git cvcbrowser
```

3.1.3.2 Setting Up the Project Database

Step 1. Install the database software PostgreSQL (version 9.3 or more recent), database management software phpPgAdmin, and the PostGIS extension.

Step 2. Within phpPgAdmin create a database called “cvcbrowser” and add “postgis” as a new extension to that database.

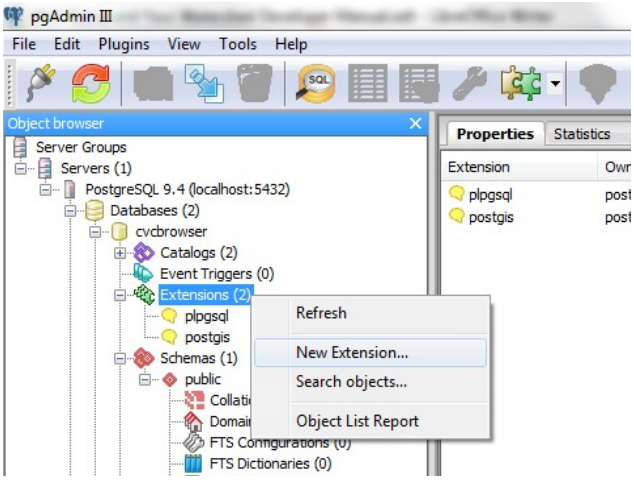

Illustration 2: Adding a new extension to the database

Step 3. Within the source code copy the file at /app/config/paramaters.yml.dist to /app/config/paramaters.yml and set the following:

- database_driver: pdo_pgsql

- database_host: 127.0.0.1
- database_port: ~
- database_name: cvcbrowser
- database_user: <username>
- database_password: <password>

Step 4. To process uploaded spatial data, the Geospatial Data Abstraction Library (GDAL) 1.11 or more recent version should be installed in order to use the ogr2ogr command to input shapefiles into the PostgreSQL database. See http://www.gdal.org/drv_pg.html.

Step 5. Update the database schema with the cvcbrowser project defined entity. The composer.phar file from the source code must be downloaded to the cvcbrowser database folder or be installed. In the command prompt execute the following code to create the needed tables within the database:

- goto <cvcbrowser folder>
- php -r "readfile('https://getcomposer.org/installer');" | php
- php app/console doctrine:schema:update –force

3.1.3.3 How to Create an Entity or Modify Existing Entity

See: http://symfony.com/doc/current/bundles/SensioGeneratorBundle/commands/generate_doctrine_entity.html

Use Symfony 2.3 LTS. All logic layer code is inside the Controller folder. The display layer is a Twig template file.

Before starting coding, make sure that your local system is properly configured for Symfony.

Execute the “check.php” script from the command line:

- php app/check.php

The script returns a status code of 0 if all mandatory requirements are met, 1 otherwise.

Access the “config.php” script from a browser:

- http://localhost/path/to/symfony/app/web/config.php

If you get any warnings or recommendations, fix them before moving on.

3.1.4 Database

3.1.4.1 List of Database Tables

The following table lists all of the tables within the database. A description is included for important tables and it is also indicated if the data is spatially represented on the interactive map.

Table 1: List of Database Tables

Database Table	Description	Spatial?
acl_classes		
acl_entries		
acl_object_identities		
acl_object_identity_ancestors		
acl_security_identities		
additionalreference_indicatorbenefits	Stores the many-to-many relationship between the indicators and additional references that were not used in the text	
categories	Stores all the ecosystem services and health and well-being domains	
category_contentdetails	Stores referenced information about the subcategories listed in the category_contents table	
category_contents	Stores subcategories of the ecosystem services and health and well-being domains	
categorycontent_dlayers	Stores the many-to-many relationship between the subcategories and the display layers	
classification__category		
classification__collection		
classification__context		
classification__tag		
cluster_layers	Stores cluster layer information	Yes
clusterlayers_indicatorbenefits	Stores the many-to-many relationship between the indicators and the cluster layers[1]	
contacts		
databasebackup	Stores information on database back-ups	
eco_system_service	Stores the ecosystem services	
ecosystemservice_indicatorbenefits	Stores the many-to-many relationship between the ecosystem services and the indicator benefits	
ext_log_entries		
ext_translations		
feature_contents		
fos_user_group	Stores grougs of users	
fos_user_user	Stores user information	
fos_user_user_group	Stores the many-to-many relationship between users and groups	
geoserver_layers	Stores information for connecting to remote	Yes

[1] “Cluster layers” refers to those layers that have been set up to agglomerate on the map depending on the display scale (e.g., the Public Recreation Areas has been set up to cluster)

	geoservers	
geoserverlayers_indicatorbenefits	Stores the many-to-many relationship between the geoserver layers and the indicator benefits	
groups		
heatmap_bookmark		
heatmap_gradient		
heatmap_layers		
homepage_description	Stores the home page text	
homepage_flashs	Stores images for the flash animation on the home page	
homepage_image	Stores images for the home page	
homepageheader	Stores images for the home page header	
human_well_being_domain	Stores the health and well-being domains	
humanwellbeingdomain_indicatorbenefits	Stores the many-to-many relationship between the health and well-being domains and the indicator benefits	
indicator	Stores indicators of ecosystem services (e.g., canopy cover)	
indicator_benefit	Stores benefits derived from indicators	
indicator_benefit_image	Stores images associated with benefits	
indicator_reference	Stores references associated with indicator information	
indicatorbenefitimage_indicatorbenefits	Stores the many-to-many relationship between indicator benefits and the indicator benefit images	
indicatorreference_indicatorbenefits	Stores the many-to-many relationship between indicator benefits and the indicator references	
logos	Stores logos of project sponsors	
map2u__forum_board		
map2u__forum_category		
map2u__forum_faq		
map2u__forum_faq_answers		
map2u__forum_forum		
map2u__forum_post		
map2u__forum_registry		
map2u__forum_subscription		
map2u__forum_topic		
map_bookmark		
media__gallery		
media__gallery_media		

media__media		
news__comment		
news__post		
news__post_tag		
notification__message		
otherlinks_indicatorbenefits	Stores the many-to-many relationship between external references and indicator benefits	
privileges		
schema_migrations		
sessions		
spatial_ref_sys		
stories	Stores user uploaded stories	
system_settings		
systembackup		
systemparams		
temp_polygon		
thematic_map		
uploadfile_layers	Stores the spatial data that should be displayed to user	Yes
uploadfilelayers_indicatorbenefits	Stores the many-to-many relationship between the display layers and the indicator benefits	
user_map_share		
user_membership		
userdrawgeometries	Stores user drawn shapes	Yes
userdrawgeometries_geom		
userdrawlayers	Stores the layer that displays user added data	Yes
userdrawlayers_translation		
useruploadfile	Stores user uploaded spatial data	Yes
useruploadfile_geoms		

3.1.4.2 Relationship Map

Below is the relationship map of the database. If you have trouble viewing it, please right click on the image and save as an SVG file. Then open this file in your browser.

A solid line indicates an identifying relationship, whereas a dashed line indicates a non-identifying relationship.

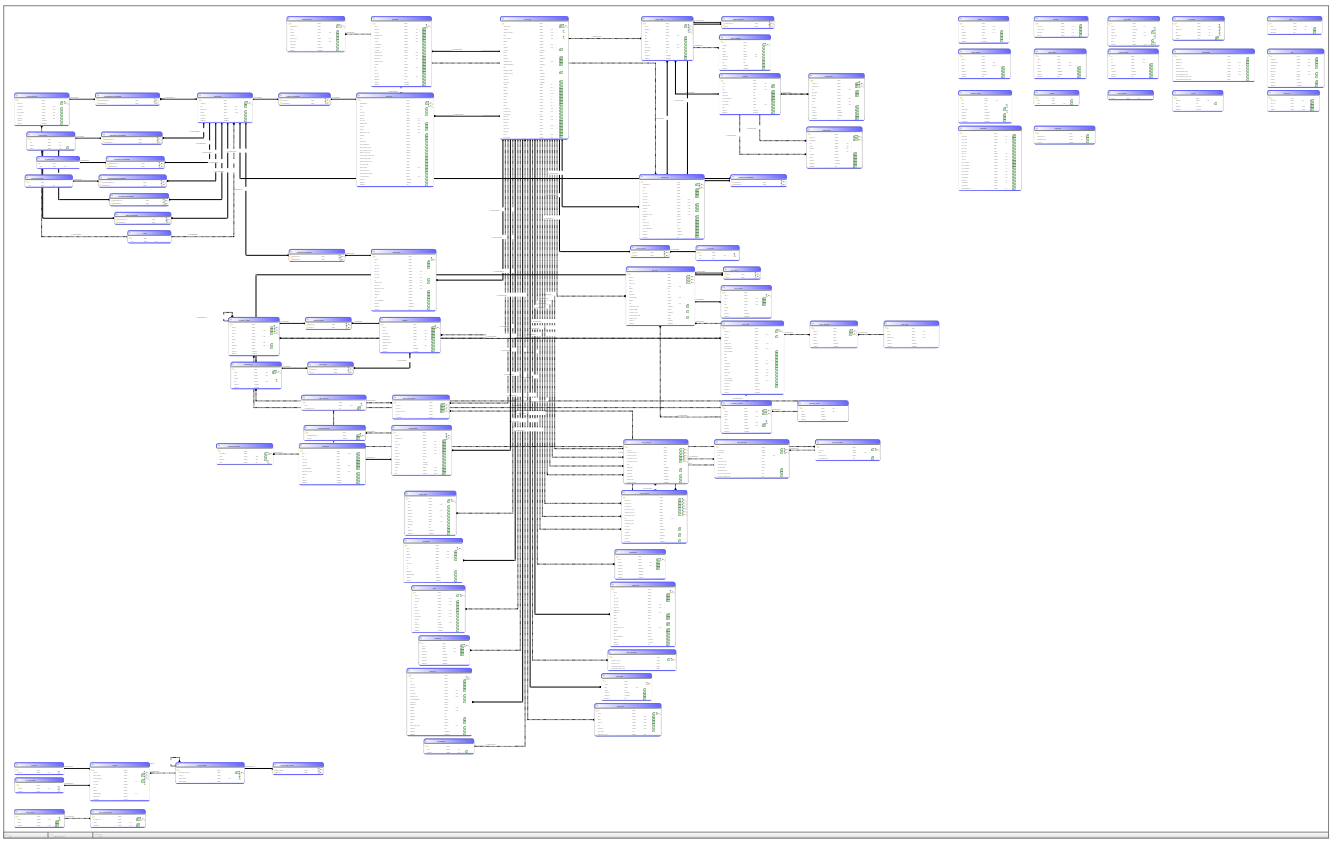

Illustration 3: Database relationship map

3.1.5 Debugging the Code

The source code was debugged using the xdebug extension for PHP and SensioLabsInsight

3.1.6 Commenting the Code

The in-code documentation has been improved using phpDocumentor.

3.1.7 AODA Compliance

The website has been updated to comply with the Accessibility for Ontarians with Disabilities Act (AODA) through the following website: http://achecker.ca/checker/index.php.