Working Waterfronts Documentation

Release 1.0

OSU Open Source Lab

February 11, 2015

Contents

API and Usage Documentation:

CHAPTER 1

Usage

CHAPTER 2

API Endpoints

CHAPTER 3

Install

CHAPTER 4

Planning

4.1 Draft API

4.1.1 Format

Responses will be returned in standard JSON format. An attempt will be made to keep the structure simple. Https will be used for all endpoints.

Null values (optional fields that do not have data), will be empty strings: “”.

4.1.2 Versions

The API will be versioned with simple version integers, 1, 2, 3, ...

ex: https://working-waterfronts.org/1/pois

4.1.3 Errors

Error records will be returned in every message, and will consist of a dictionary containing the error status, error name, error text and error level. The status field will indicate the presence of an error condition, and should be checked before attempting to process the rest of the response.

example:

```
error: {error_status: true, error_name: 'not_found_error', error_text: 'poi with id=232 could not be
```

4.1.4 Extended Fields

To allow for future expandability, a dictionary call ‘ext’ will be included with every response. This dictionary will either contain no records, or will contain additional first-class records that were not included in the original specification. For instance, if a new attribute “color” is later added to the product response, it can be included in the extended attributes array. Applications can choose to discover/use these new fields or ignore them without effecting backwards compatibility. Response validation should include the presence of ext, but not its contents.

4.1.5 Endpoints

/pois

Return a dictionary containing a record for every poi in the database. This data is unlikely to change frequently, it should be in long-term storage on the device and refreshed periodically.

```
{
        error: {error_status: bool, error_name: text, error_text: text, error_level},
        pois:  [
    {id: int,
                name: text,
                alt_name: text or null,
                summary: text,
    lat: float,
    long: float,
    street: varchar,
    city: varchar,
    state: varchar,
    zip: varchar,
    description: text
    location_description: text,
    history: text,
    facts: text,
    contact_name: varchar,
    phone: varchar,
    website: url,
    hazards: [
          {
                name: varchar,
                description: varchar,
                id: int
          },
          ...
    ],
    categories: [
          {
                category: varchar,
                id: int
          },
          ...
    ],
    images: [
          {
                link: varchar,
                name: varchar,
                caption: varchar
          },
          ...
    ],
    videos: [
          {
                link: varchar,
                name: varchar,
                caption: varchar
          },
          ...
    ],
    email: email,
```

```
                created: datetime,
                modified: datetime,
                ext: {attribute: value, attribute: value...} or {}},
            {...},
            {...}
    ]
}
```

/pois/<id>

Returns a single poi record identified by <id>. This will return all available details about a poi.

```
{
        error: {error_status: bool, error_name: text, error_text: text, error_level},
    id: int,
                name: text,
                alt_name: text or null,
                summary: text,
    lat: float,
    long: float,
    street: varchar,
    city: varchar,
    state: varchar,
    zip: varchar,
        description: text,
    history: text,
    facts: text,
    location_description: text (optional),
    contact_name: varchar,
    phone: varchar (optional),
    website: url (optional),
    email: email (optional),
    hazards: [
          {
                name: varchar,
                description: varchar,
                id: int
          },
          ...
    ],
    categories: [
          {
                category: varchar,
                id: int
          },
          ...
    ],
    images: [
          {
                link: varchar,
                name: varchar,
                caption: varchar
          },
          ...
    ],
    videos: [
          {
                link: varchar,
                name: varchar,
```

```
                caption: varchar
            },
            ...
    ],
                created: datetime,
                modified: datetime,
                ext: {attribute: value, attribute: value...} or {},
}
```

/pois/categories/<id>

Returns a list of pois in the category identified by <id>.

```
{
        error: {error_status: bool, error_name: text, error_text: text, error_level},
        pois:  [
    {id: int,
                name: text,
                alt_name: text or null,
                summary: text,
    lat: float,
    long: float,
    street: varchar,
    city: varchar,
    state: varchar,
    zip: varchar,
                created: datetime,
                modified: datetime,
                ext: {attribute: value, attribute: value...} or {}},
            {...},
            {...}
    ]
    ...
}
```

4.1.6 Additional parameters

These parameters can be added to any endpoint request

?location=<lat>,<long>

or

?lat=<float>&long=<float>

These parameters represent the latitude and longitude of either the mobile device's current location, or a pre-defined location such as "Newport, OR". These will cause the results to be sorted by proximity, closest items first. This parameter will be ignored with the /stories endpoint. Depending on how the device handles the coordinates, it may be more convenient to send a single parameter, 'location=<lat>,<long>' and use the latitude and longitude as positional arguments.

examples:

?limit=<int>

This parameter will limit the number of records returned to <int>. In combination with the location parameter, it can be used to return the 5 nearest pois selling tuna:

?proximity=<int>

This parameter will restrict the returned results to those within <int> miles (or configurable distance unit) of the given location. Ignored if no location is given.

4.2 Draft Data model

4.2.1 pois

```
    id                      int (pk)
    name                    varchar
    alt_name                varchar (optional)
    location                point
    street                  varchar
    city                    varchar
    state                   varchar
    zip                     varchar
    description             text
    location_description    text (optional)
history                 text
facts                   text
    contact_name            varchar
    phone                   varchar (optional)
    website                 url (optional)
    email                   email (optional)
    created                 datetime
    modified                datetime (auto-update on modification)
```

4.2.2 categories

```
id          int (pk)
category    varchar
```

4.2.3 images

```
    id          int (pk)
name        varchar
poi_id      int (foreign key to poi)
    image       image (file)
    caption     text (optional)
    created     datetime
    modified    datetime (auto-update on modification)
```

4.2.4 videos

```
    id              int (pk)
name            varchar
poi_id          int (foreign key to poi)
    video           link
    description     text (optional)
    created         datetime
    modified         datetime (auto-update on modification)
```

4.2.5 hazards

```
id              int (pk)
name            varchar
description     text
    created         datetime
    updated         datetime (auto-update on modification)
```

4.2.6 pois_hazards

```
poi_id      int (foreign key to poi)
hazard_id   int (foreign key to hazard)
```

4.2.7 pois_categories

```
poi_id          int (foreign key to poi)
category_id     int (foreign key to category)
```

Developer Setup:

CHAPTER 5

Developer Guide

5.1 Project Structure

5.2 Issue Tracking

The bug tracker for the Working Waterfronts API is at code.osuosl.org, and all bugs and feature requests for the Working Waterfronts API should be tracked there. Please create an issue for any code, documentation or translation you wish to contribute.

5.3 Repository Layout

We loosely follow Git-flow for managing repository. Read about the branching model and why you may wish to use it too.

master Releases only, this is the main public branch.

release/<version> A release branch, the current release branch is tagged and merged into master.

develop Mostly stable development branch. Small changes only. It is acceptable that this branch have bugs, but should remain mostly stable.

feature/<issue number> New features, these will be merged into develop when complete.

When working on new code, be sure to create a new branch from the appropriate place:

- **develop** - if this is a new feature
- **release/<version>** - if this is a bug fix on an existing release

5.4 Code Standards

We follow PEP 8, "the guide for python style".

5.4.1 Developing with Docker

5.5 Platform dependent specifics

If you are using Linux you will need to prefix all of the following commands with sudo. If you are using OS X you will need to use the boot2docker tool.

5.6 Postgis image

The Working Waterfronts Docker workflow relies on the kartoza/postgis image available on the docker hub. To pull this image run:

```
$ docker pull kartoza/postgis
```

The image can take two optional environment variables to specify a user and password to the database. These will be specified with the -e option. A port should be provided with the -p followed by the port to communicate with the host machine, a colon, and the port to communicate with the container. Make sure the environment variables passed to this container match those which are passed to the Working Waterfronts Docker image. Reasonable defaults can be found in the Dockerfile. Postgres typically runs on port 5432. To run the image:

```
$ docker run -d --name postgis -p $HOSTPORT:$CONTAINERPORT -e  USERNAME=$USERNAME -e PASS=$PASSWORD
```

Make sure that the What's Fresh project container connects to the database over the host port.

5.7 Building the What's Fresh docker image

```
$ docker build -t="osuosl/working_waterfronts:dev" .
```

5.8 Running the What's Fresh docker image

The Dockerfile included in the root of the repository will load the code from the current directory. This means that any changes you made to your copy of the repository will be run. Environment variables can be passed with the -e option. The Dockerfile specifies a reasonable default set of environment variables, which can be overridden with the -e option.

Before the app is ready, create the database and run migrations.

```
$ docker exec -it postgis bash
# createdb -U $USERNAME -h localhost $DBNAME
# psql -U $USERNAME -h localhost
DBNAME=# create extension postgis;
CREATE EXTENSION
DBNAME=# ^D
# ^D
$ docker run --link postgis:postgis osuosl/working_waterfronts:dev python manage.py migrate
```

Next, connect to the database with psql and create the relevant user.

```
$ psql -h localhost -U docker -p $HOSTPORT
```

Running the server is similar:

```
$ docker run --link postgis:postgis -p 8000:8000 osuosl/working_waterfronts:dev
```

If you are running linux, connect to http://localhost:8000 in your browser. If you are running OS X, get the IP address of your boot2docker vm

```
$ boot2docker ip
192.168.59.103
```

Next connect to http://192.168.59.103:8000 in your browser.

On occasion it may be necessary to obtain a shell in the container:

```
$ docker run -it osuosl/working_waterfronts:dev bash
```

Some developers may prefer to mount their copy of the application as a volume when they run the app:

```
$ docker run -v /path/to/code/:/opt/whats_fresh --link postgis:postgis osuosl/whats_fresh:dev
```

5.8.1 Developing

5.9 Requirements

This project uses a Vagrant virtual machine to create a homogeneous development environment and allow developers to destroy and recreate their environment in the case that something goes horribly, horribly wrong.

To set up this environment on your own machine, you'll need a few things:

Vagrant

To install Vagrant, just use your package manager:

```
sudo yum install vagrant # Debian or Ubuntu
sudo apt-get install vagrant # Centos
```

vagrant-berkshelf and vagrant-omnibus

These plugins are used to configure the Vagrant machine. To install these plugins, you'll need to use Vagrant's plugin manager:

```
vagrant plugin install vagrant-berkshelf
vagrant plugin install vagrant-omnibus
```

5.10 Running the Django project

5.11 Testing

Model and View documentation:

CHAPTER 6

Indices and tables

- *genindex*
- *modindex*
- *search*