Spiff Documentation

Release 0.1.0

Torrie Fischer

July 19, 2014

Contents

Author Torrie Fischer <tdfischer@hackerbots.net>

Contact irc://chat.freenode.net/#synhak

Date July 19, 2014

Version 0.1.0

Spaceman Spiff (or just Spiff) is a hackerspace management tool that helps hackers manage a hackerspace. It comes with a builtin web interface for end users, but is also exceptionally machine friendly.

Management of a hackerspace includes several topics:

- Membership
- Documentation
- Communication
- Infrastructure
- Governance

It is made available to the public under the AGPL.

CHAPTER 1

Dependencies

```
Django==1.6
Jinja2==2.6
South==0.8.3
Sphinx==1.1.3
git+https://github.com/LeadSift/django-gravatar.git#egg=django-gravatar
django-notification
git+https://github.com/tdfischer/django-openid-auth.git#egg=django-openid-auth
django-openid-provider==0.4
django-webfinger==0.2
django-bootstrap-toolkit==2.15.0
django-tastypie==0.10.0
docutils==0.10
isodate==0.4.9
python-openid==2.2.5
qrcode==2.4.2
requests==1.0.4
stripe==1.7.7
wsgiref==0.1.2
xrd==0.1
mimeparse==0.1.3
django-nose==1.2
django-cors-headers
PyJWT
```

1.1 Features

- Track member dues to see who is paid for the month
- Multiple ranks with independent monthly dues
- Keep track of space resources and metadata associated with each resource
- Create arbitrary membership fields with visibility settings such as “Door Keycode” (editing/viewing limited to officers), “Enjoys Smooth Jazz” (viewing limited to members only), or “Nobel Prizes Earned” (public to the internet).
- A thorough REST api to access everything
- A skill tracking system

- Simple interface to sensors
- Perform actions when sensors are updated
- An implementation of the SpaceAPI
- Accept member dues through Stripe
- Use of Django's builtin admin interface to provide low-level database editing.
- Keep track of who is certified to use equipment
- Merit based equipment skill level ranking system

Contents:

1.1.1 Installation

Since Spiff is written with Django, the Django installation docs may be a helpful primer.

In Brief

1. Create a local_settings.py that contains any values you want to override from settings.py.
2. Install your dependencies:

   $ pip install -r pip-requirements
3. $ ./manage.py syncdb –migrate
4. Go nuts.

The default settings use sqlite3 as the database, with /path/to/spiff/spiff.sqlite3 as the file.

Common Environments

MySQL

Here is an example configuration to put in local_settings.py:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'spiff',
        'USER': 'spiff',
        'PASSWORD': 'hunter2',
        'HOST': '',
        'PORT': ''
    }
}
```

Leaving HOST and PORT empty uses 'localhost' at the default mysql port.

Please refer to the Django MySQL docs for more details.

Apache

This section is included as an example to get Spiff and Apache to work together in harmony. It is more or less exactly how we run things at synhak.org

First, decide where you're going to serve up spiff. Keep in mind: this URL should probably never ever ever change in your space's lifetime. QR codes, hardware sensors, door swipes, and whatever else you have talking to Spiff will need reconfigured if things ever move. We run our instance at https://synhak.org/auth/

Our git clone of Spiff is located in /usr/share/spiff/.

$ git clone git://github.com/SYNHAK/spiff.git /usr/share/spiff/ $ cd /usr/share/spiff/ *Configure your local_settings.py here* $ ./manage.py syncdb –migrate

In /etc/httpd/conf.d/synhak.org.conf:

```
<VirtualHost *:80>
  LoadModule wsgi_module modules/mod_wsgi.so
  WSGIScriptAliasMatch ^/auth(/([^~].*)?)$ /usr/share/spiff/spiff/wsgi.py$1
  Alias /auth/static /usr/share/spiff/spiff/static
  WSGIPassAuthorization On
  WSGIDaemonProcess spiff-1 user=apache group=apache threads=25
  WSGIProcessGroup spiff-1
</VirtualHost>
```

That is all you need. You may then access spiff at http://your-space.org/auth/

Other information for using Django with mod_wsgi can be found at the Django mod_wsgi howto guide.

1.1.2 Usage

Configuring your Space

To rename your hackerspace, configure the only available Site object in the Django admin interface. If you have more than one Site object, configure one to use the correct domain name, delete the others, and optionally let the developers know how it got there since Spiff should only ever have one.

Other properties are configurable through the SpaceConfig object in the Django admin UI:

site Refers to the Django object. This shouldn't be changed unless you've got a good reason to.

logo URL that points to your space's logo. It may be absolute, or relative to your Spiff URL.

openIcon Used as part of the *SpaceAPI* to indicate a graphic to display if the space is open.

closedIcon Same as the `openIcon`.

url Your space's website.

open Determines if your space is currently open or closed. Also see Open Sensor, which provides programatic access to this.

lat and lon Latitude and longitude

address Your space's physical location.

status A free-form field that is shown in the *SpaceAPI*. For example, "Open to members only", or "Closed due to inclement weather".

openSensor See Open Sensor.

Open Sensor

It is possible to configure Spiff to automatically handle opening/closing your space with the *Sensors* system, for whatever definition of "open" or "closed" you have. The associated sensor must be a boolean type. When it has a true value, the space is reported as open through the *SpaceAPI*. It is reported as closed for false values. Refer to the *Sensors* chapter for details.

Membership

Spiff provides a basic system for staff to manage a member database, self-service membership management, and accepting dues via Stripe.

Each member has a few basic fields that should be filled out:

- Email
- First Name
- Last Name
- Birthday
- Profession

These fields are public to the general internet, except for hidden users.

Administrators can add extra fields such as phone number, mailing address, emergency contact information, etc. These extra fields have three flags available:

- Required - The profile will not save and a user can't register without this field being filled out. Examples: emergency contact information, digital signature proving they read the rules, preferred objective description of the color #33ff62.
- Public - Other members can read the field, but not the entire internet. Members can edit their own public fields. Fields that are not public can still be read by those with the `can_view_private_fields` permission. Examples: IRC nickname, membership sponsors, or exact reasons for disliking ABBA.
- Protected - Only those with the `can_edit_protected_fields` permission can view and edit the field. Members can read the value of their own protected fields, but can't edit them. This is useful for things that members should know about themselves, but others shouldn't know about others, and members shouldn't be able to change. Examples: a key/RFID token ID number, a note proving that they signed a liability waiver, a third meta-item that points out this is the third item in the third list of lists of threes and thus three times as funny.

Ranks

Many spaces have a set of ranks, such as "Basic Membership", "Board Member", etc. Spiff allows you to model this via Spiff's Ranks and Django's builtin groups.

To create a new rank, such as "Basic Membership", create a new Group object via the Django admin interface. This automatically creates a Rank object, which has several properties:

monthlyDues How much it costs per month for this rank, in USD.

group The Django group object this rank refers to. There shouldn't be a need to ever change this.

isActiveMembership If a member is in this rank, they are considered an active member. This property is used to determine if a user pays dues, and to show the list of active members.

isKeyholder If a member is in this rank, they are considered a keyholder. This property is used by the *SpaceAPI* to list keymasters.

Each underlying Django group object can have a set of permissions attached to it, which grants all members of the group those permissions.

Those with the `membership.can_change_member_rank` permission may edit a user's ranks by visiting the user's profile page.

See Also:

Permissions

Membership Dues

Managing membership dues is fairly straightforward, and involves very little usage of the confusing Django administration interface: Simply configure the `isActiveMembership` and `monthlyDues` properties of your roles and forget about the admin interface.

A member's profile page will list their recent due payments, along with an option to record a payment that was not handled by Spiff, such as cash or some other payment method.

Recording partial payments are supported. This is useful for instances such as a member paying $10 in cash and the last $40 via Stripe, or forgetting that dues are $35 and not $30.

To enable stripe, set your API key in local_settings.py:

:: STRIPE_KEY = "sk_test_foofoofoo"

Resources

In every hackerspace, theres a bunch of equipment sitting around that not everyone might know how to use or even what it is called. Spiff solves that problem.

You can create a Resource object in Spiff for each real-world resource. After it is created, metadata can be attached to it and edited by users with the correct permissions. Members can also keep track of their training on the site, along with their relative skill ranks.

Users require the inventory.certify permission to be able to add and remove certifications from members.

Skill ranking works on an honor system that requires users undergo a vetting process by other users:

- Your hackerspace acquires a nice new lathe.
- A member adds the lathe to the database, prints out the QR code and sticks it on the machine.
- Another member who happens to be a master metalworker sees that there is a Lathe, scans the code (or visits the resource page) and clicks "I have used this!" to indicate that they have used a Lathe at some point in their life.
- A second member (who is a total newbie to metalworking) also clicks "I have used this!". Spiff says that both the newbie and the master are ranked at the same skill level, so they click "They are better than me".
- Spiff now indicates that the master is better trained at the lathe than the newbie and sorts them accordingly.

At no point can the newbie say that they are better than the master without the master explicitly promoting the newbie to their level. Additionally, the newbie can't demote the master. Members are ranked relative to each other based on this feedback system.

Not all resources in a hackerspace are trainable! For instance, it makes no sense to say that someone is more skilled at using the classroom or meeting area. When creating a resource, you can specify if a resource can be trainable or not.

Events

Spiff also allows for tracking of events. Anyone with a proper permission can create an event (and later edit it). Members can easily RSVP for an event with a link on the event page. There is no special permission required to state that you are attending an event.

If an event requires the use of some resource (which could be a classroom, or maybe its a class on using the lathe), it is possible to reserve the use of a resource by adding it to the event. This reservation system is purely an advisory one at the moment. Nothing will stop someone from reserving an already reserved item, or physically blocking you from using it.

Events can have multiple organizers, who are able to edit an event's description and reserve resources. Organizers may only be added or removed by the event creator.

Sensors

See *Sensors* for complete documentation.

Management Commands

There are a number of management commands available through manage.py.

list_members Lists the email addresses of active members. At SYNHAK, we pipe the output of this through to mailman's sync_members script to subscribe active members to the members only list.

stripe_sync Currently useless. Will soon be used to support automatic billing and advanced invoicing through Stripe. It creates a Customer in stripe for each Member in Spiff.

permission_list Lists all permissions in Spiff and Django.

1.1.3 Permissions

Spiff has a set of permissions that say who can do what on the site.

It is recommended to create a general purpose "Active Member" rank to keep track of who is and isn't a member and to provide a base set of permissions that apply to all members. Afterwards, you can create new ranks for each membership level in your hackerspace.

Note: Run ./manage.py permission_list to retrieve a list of permissions and brief descriptions. Only permissions used in Spiff codebase are documented. See the Django auth reference for information about how permissions work inside Django.

---

auth.change_user

The user can edit the profiles of other users.

auth.delete_user

The user can delete other users.

events.add_event

The user can create events and edit their own events.

events.can_reserve_resource

The user can attach resources to their own events.

events.change_event

The user can edit other user's events. This along with can_reserve_resource is required for being able to attach resources to events that they don't own.

inventory.certify

The user may grant and remove certifications for resources from members.

inventory.change_resource

The user can add and modify resource metadata.add_metadata, change_metadata, etc are not used at all in Spiff.

inventory.can_train

The user can promote other users' trainings and add themselves to a resource at the lowest level.

membership.add_duepayment

The user may add previous due payments to Spiff.

membership.can_change_member_rank

The user may view modify the ranks a member belongs to.

membership.can_edit_protected_fields

The user can edit and view profile fields that are protected.

membership.can_view_hidden_members

The user is able to view members that have the hidden flag set.

membership.can_view_member_rank

The user is able to view another user's ranks.

membership.can_view_private_fields

The user can view any field that does not have the Public flag set.

1.1.4 Sensors

Spiff is intended to be the central brain of a hackerspace. As such, it includes functionality for tracking various sensors in the space.

There are five basic types of sensors:

- number
- string
- binary
- json
- temp
- boolean

The type of sensor is just a hint to tell API users how to display the data if the exact purpose of the sensor is unknown. For example, the spiff web UI will show a history graph for number sensors. The sensor types adhere to the SpaceAPI standard: http://hackerspaces.nl/spaceapi/

To update a sensor, send a POST request to the sensor's page (i.e. /sensors/1) with a single 'data' parameter containing the new sensor data:

$ curl –data "data={'test': true}" http://example.com/sensors/1

The data can be anything: an image, a number, a basic string that says "Hello!", more strutured JSON data, or whatever else you want to put in there. Spiff doesn't care (except for Boolean Sensors, it just stores the data until someone else wants it.

Boolean Sensors

Spiff's REST API translates certain values into native values for requested serialization formats, and for the *Open Sensor*.

Accepted values that mean false:

- The string "false" (case-insensitive)
- The string "0"
- An empty string

Anything else is interpreted as true.

Pamela

Pamela is described as a "very cool way to visualize any kind of data". You can find it at http://www.hackerspace.be/Pamela

Spiff is totally 100% compatible with Pamela's basic API.

To use pamela's ARP scanner with Spiff:

```
$ ./pamela/scanner/pamela-scanner.sh -i "eth0" -o \
    "http://example.com/sensors/1" -t mac.csv -d \
    "/var/lib/dhcpd/dhcpd.leases"
```

Please see Pamela's documentation for more details.

Sensor Actions

Spiff can do stuff when sensors are updated. There are 4 kinds of actions that can be added via the Django admin interface:

- http: Sends a HTTP GET request to the given url
- exec: Runs a command via **subprocess.call_**
- python: Executes some python code. A spiff.sensors.models.Sensor object is available in the 'sensor' variable.
- script: Writes a blob of text to a temporary file, performs chmod +x, and runs it.

Warning: Be extremely careful with the exec, python, and script actions! The commands are ran by the python process, which also means don't run spiff as root ever! Scary things can happen! Don't forget that your members trust you with the information you keep in spiff!

1.1.5 API Documentation

Spiff has had two separate APIs over its lifetime, a versioned cherrypy powered one, and a hacked together homebrew one.

All active development continues on the versioned API, while the deprecated one is scheduled for removal before the first full release. It is documented here for posterity, as there already exist a number of experimental deployments.

REST API

Documentation is forthcoming. Check out /v1/?format=json in your browser for some hints.

General purpose information about the space is available by fetching /status.json, as per the *SpaceAPI*.

SpaceAPI

Spiff currently implements version 0.12 of the SpaceAPI. You can read more about it at http://hackerspaces.nl/spaceapi/

You may access the SpaceAPI through /status.json, as per the standard. If your spiff installation is accessed via a different url (eg: https://synhak.org/auth/), status.json will be at /auth/status.json.

See Also:

Configuring your Space

Spiff Extensions Spiff adds a few minor extensions to the SpaceAPI:

x-spiff-url The URL used to access spiff. Can be used to build full REST urls, such as http://example.com/path/to/spiff/resources/1.json

x-spiff-version The version of the Spiff REST API. This is not the version of spiff installed!

x-spiff-open-sensor The ID of the sensor used to determine if the space is open or not. See *Open Sensor* for details.

Deprecated REST API

Spiff's versioned API addresses a number of shortfalls in the previous API:

- It was a bear to maintain
- Didn't implement a number of important features, such as result pagination
- POSTing updates had limited authentication control

Just about everything in Spiff is accessible through REST. It is as easy as adding .json to the end of your URLs:

$ http://example.com/sensors/1.json curl http://localhost:8000/sensors/1.json { "description": "A list of devices in the space", "name": "pamela", "value": { "stamp": "2012-10-31T15:28:53.901053+00:00", "sensor": "#Sensor#1", "id": 6, "value": "{'test': true}" }, "id": 1

Due to the cyclic nature of the database, some values are references to other objects. This is indicated by the syntax "#Type#ID", such as "#Sensor#1".

Other serialization formats may be added later if there is enough demand.

REST documentation pages

Resources

- /resources/.json

  Returns a list of all resources

- /resources/{id}.json

  Returns data for the specific resource.

- /resources/{id}/meta

  Parameters:

  - name - The name of the metadata value
  - value - The value
  - type - The type of metadata.

- /resources/{id}/train
- /resources/{id}/promote

Sensors

- /sensors/.json

  Returns a list of all sensors.

- /sensors/{id}.json

  Returns data for the given sensor.

  Parameters:

  - data - The data to be updated with.

1.1.6 Developing Spiff

Spiff is hosted on GitHub: http://github.com/synhak/spiff

The preferred coding style is Pep 8. If you submit a patch or merge request that is not formatted with Pep 8, thats ok, it'll get cleaned up.

Spiff is a side effect of SYNHAK. You can find some of the developers and users in #synhak on chat.freenode.net.

1.1.7 Spaces using Spiff

Spiff is kinda new, but if you're looking to know more about how others use it, here's a list of spaces using Spiff:

- SYNHAK, Akron, Ohio, USA

If your space uses Spiff, here is a bitcoin address you may use to contribute to: 15fvBnRowyDshudNRKAiBYhTyKEUwnnywQ

CHAPTER 2

Indices and tables

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