RapidSMS Rwanda Documentation

Release 0.9.6

RapidSMS Rwanda

Jul 06, 2017

Contents

RapidSMS is a free and open-source framework for dynamic data collection, logistics coordination and communication, leveraging basic short message service (SMS) mobile phone technology. It is written in Python and uses Django.

The documentation, and other resources for the RapidSMS framework can be found below.

- **Website**: http://www.rapidsms.org/
- **Documentation**: http://rapidsms.readthedocs.org/
- **Code**: http://github.com/rapidsms/rapidsms
- **Google Group**: http://groups.google.com/group/rapidsms
- **Youtube Channel**: http://www.youtube.com/user/rapidsmsdev
- **RapidSMS IRC Channel archives**: http://irc.rapidsms.org (#RapidSMS also accessible via browser at http://webchat.freenode.net/)

CHAPTER 1

Rwanda Custom Release

RapidSMS Rwanda was built to track mHealth-related data from Community Health Workers (CHWs), originally in the Musanze district, and eventually counry-wide.

It's initial features include:

- Registration of pregnant mothers
- Reminders sent out for pre-natal and ante-natal check-ups
- Tracking of birth, death, and other vital statistics of the fetus and newborn

Features currently in development are:

- Enhanced charting and mapping
- Enhanced alerts and feedback
- Additions for the "1000 days" project, tracking infant weight and height through 2 years of age

Installing and running RapidSMS Rwanda

Basic Environment Setup

If you're already familiar with working on python/Django projects, you'll likely be able to skip this section. The following steps walk you through the first steps of setting up python, i.e. basic package management and development environments. This allows you to more easily install the dependencies for RapidSMS-Rwanda, while keeping your base installation of python clean. This assumes that you already have python installed.

Install easy_install (if necessary):

```
$ wget peak.telecommunity.com/dist/ez_setup.py
$ python ez_setup.py
```

Install pip (if necessary):

```
$ easy_install pip
```

Set up virtualenv, a tool for creating isolated python environments (to keep your dependencies seperate and in local user space, rather than in /usr/):

```
$ pip install virtualenv
$ pip install virtualenvwrapper
```

Configure virtualenv. First, create a folder to organize all your virtual environments within:

```
$ mkdir ~/virtualenvs
```

Append to ~/.bashrc:

```
export WORKON_HOME=$HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
```

Make the virtual environment:

```
$ mkvirtualenv rwanda
```

Installing RapidSMS-rwanda

First, clone the project directly:

```
$ git clone git://github.com/pivotaccess2007/RapidSMS-Rwanda
```

You can then install the dependencies:

```
$ pip install -r pip-requires.txt
```

Configuring RapidSMS-Rwanda

The following parameters need to be set in order to get RapidSMS-Rwanda up-and-running from a base install.

First, edit rapidsms.ini and add the basic configuration parameters. Yours may vary, depending on what sort of database you have, what you want to name it, your time zone, etc.:

```
[database]
engine=django.db.backends.postgresql_psycopg2
name=rwanda
user=postgres

...
# Revence thinks this is necessary.
TIME_ZONE=Africa/Kampala
BASE_TEMPLATE=layout.html  # This needed to be added
```

To apps, add *patterns* (this seemed to be missing upon syncdb):

```
apps=webapp,ajax,admin,reporters,locations,messaging,httptester,logger,ubuzima,echo,
↪ambulances,patterns
```

Create the database:

```
$ sudo -u postgres createdb rwanda
```

Syncdb:

```
$ python manage.py syncdb
```

Load in essential fixtures:

```
$ python manage.py loaddata fosa_location_types groups reminder_types reporting
```

View-Level Documentation

The next few sections provide a tab-by-tab (in Django-speak, view by view) breakdown of the functionality of RapidSMS Rwanda. Each page will explain the high-level functional specifications of the page, followed by a more detailed technical explanation: example code of critical logic, models involved, etc.

The main tabs are:

- *Reporters and Groups*
- *Child & Maternity Health*
- *Ambulances*
- *Messaging*
- *Message Log*

Contents

Reporters and Groups

Index

The main reporters view is at webserver/reporters, which accesses the view at *apps.reporters.views.index*

This view retrieves all reporters for which the user has permission, based on the UserLocation associated with this user account.

Further, it uses an optimization, LocationShorthand, to provide easy recursive lookups within a Location tree mirroring Rwanda's administrative boundaries.

See *Models* for a more detailed discussion of UserLocation and LocationShorthand, also see *apps.reporters.views.location_fresher* and *apps.reporters.views.reporter_fresher* for implementations using these lookups.

Export to CSV

One additional feature of the reporters app is that it has mirror views for export to CSV, containing all the same filtering logic, but replacing a standard *render_to_response* call with logic for emitting a csv file.

See *apps.reporters.views.*_csv* for examples.

match_inactive

One important thing to note to avoid confusion is that *match_inactive* is actually a function used by all the following views (error log, inactive and active reporters), it simply filters users by location, using LocationShorthand to ease the process.

Active Reporters

The active reporters view is at reporters/active, which accesses the view at *apps.reporters.views.view_active_reporters*.

Its critical section is in *apps.reporters.views.active_reporters*:

```
def active_reporters(req,rez):
    active_reps=[]
    reps=Reporter.objects.filter(groups__title='CHW',**rez)
    pst=reporter_fresher(req)
    for rep in reps.filter(**pst):
        if not rep.is_expired():
            active_reps.append(rep)
    return active_reps
```

The important method here is *is_expired* in the Reporter model, which iterates over the *last_seen* attributes of all related PersistantConnections, checking to see if this reporter has last been seen within the desired date range.

Finally, this view chains to location_fresher, only returning those reporters that the user has authorization to view.

Inactive Reporters

The inactive reporters view is at reporters/inactive, which accesses the view at *apps.reporters.views.view_inactive_reporters*.

Its critical section is in *apps.reporters.views.active_reporters*:

```
def inactive_reporters(req,rez):
    active_reps=[]
    reps=Reporter.objects.filter(groups__title='CHW',**rez)
    pst=reporter_fresher(req)
    for rep in reps.filter(**pst):
        if rep.is_expired():
            active_reps.append(rep)
    return active_reps
```

The important method here is *is_expired* in the Reporter model, which iterates over the *last_seen* attributes of all related PersistantConnections, checking to see if this reporter has last been seen within the desired date range.

Note that this method is a mirror of *active_reporters* with only a *not* missing in the *if rep.is_expired()* branch.

Finally, this view chains to location_fresher, only returning those reporters that the user has authorization to view.

Error Log

The inactive reporters view is at reporters/errors, which accesses the view at *apps.reporters.views.error_list*.

This view looks for appropriate location, filtered by UserLocation, and then looks for *apps.ubuzima.models.ErrorNote* objects associated with each reporter location and within the appropriate time ranges:

```
if 'location__id' in l.keys(): rez['errby__location__id']=l['location__id']
elif 'location__in' in l.keys(): rez['errby__location__in']=l['location__in']
elif 'location__id' in pst.keys():  ps['errby__location__id']=pst['location__id']
elif 'location__in' in pst.keys():  ps['errby__location__in']=pst['location__in']
try:
    rez['created__gte'] = filters['period']['start']
    rez['created__lte'] = filters['period']['end']+timedelta(1)
except KeyError:
    pass
errs=ErrorNote.objects.filter(**rez).order_by('-created')
```

Reporters and Groups

I'm getting the following error when I try to access this view. Help?:

```
Traceback (most recent call last):

  File "/home/david/Projects/PythonEnv/rwanda/lib/python2.6/site-packages/django/core/
↪servers/basehttp.py", line 280, in run
    self.result = application(self.environ, self.start_response)

  File "/home/david/Projects/PythonEnv/rwanda/lib/python2.6/site-packages/django/core/
↪servers/basehttp.py", line 674, in __call__
    return self.application(environ, start_response)

  File "/home/david/Projects/PythonEnv/rwanda/lib/python2.6/site-packages/django/core/
↪handlers/wsgi.py", line 241, in __call__
    response = self.get_response(request)

  File "/home/david/Projects/PythonEnv/rwanda/lib/python2.6/site-packages/django/core/
↪handlers/base.py", line 141, in get_response
    return self.handle_uncaught_exception(request, resolver, sys.exc_info())

  File "/home/david/Projects/PythonEnv/rwanda/lib/python2.6/site-packages/django/core/
↪handlers/base.py", line 180, in handle_uncaught_exception
    return callback(request, **param_dict)

  File "/home/david/Projects/PythonEnv/rwanda/lib/python2.6/site-packages/django/
↪views/defaults.py", line 24, in server_error
    return http.HttpResponseServerError(t.render(Context({})))

  File "/home/david/Projects/PythonEnv/rwanda/lib/python2.6/site-packages/django/
↪template/__init__.py", line 173, in render
    return self._render(context)

  File "/home/david/Projects/PythonEnv/rwanda/lib/python2.6/site-packages/django/
↪template/__init__.py", line 167, in _render
    return self.nodelist.render(context)

  File "/home/david/Projects/PythonEnv/rwanda/lib/python2.6/site-packages/django/
↪template/__init__.py", line 796, in render
    bits.append(self.render_node(node, context))

  File "/home/david/Projects/PythonEnv/rwanda/lib/python2.6/site-packages/django/
↪template/__init__.py", line 809, in render_node
    return node.render(context)
  File "/home/david/Projects/PythonEnv/rwanda/lib/python2.6/site-packages/django/
↪template/loader_tags.py", line 103, in render
    compiled_parent = self.get_parent(context)

  File "/home/david/Projects/PythonEnv/rwanda/lib/python2.6/site-packages/django/
↪template/loader_tags.py", line 97, in get_parent
    raise TemplateSyntaxError(error_msg)

TemplateSyntaxError: Invalid template name in 'extends' tag: ''. Got this from the
↪'base_template' variable.
```

Reminders

Lorem Ibsum

Triggers

Lorem Ibsum

Statistics

Lorem Ibsum

Ambulances

Lorem ibsum

Messaging

The messaging view is similar to standard messaging view in rapidsms.contrib, however only Reporter objects are allowed to be messaged, and the logic has been adapted accordingly.

Message Log

Message Log is accessed via the url */logger/*, and is the standard rapidsms.contrib.logger view.

See the RapidSMS documentation for further details (this is a joke, RapidSMS doesn't actually have detailed documentation).

SMS Messaging Logic

The SMS messaging logic for the RapidSMS project is all housed within a monolithic app within *apps.ubuzima.App*. It uses the keyworder parser and several handler methods to handle customized parsing for each message type, with validation and parsing logic interspersed throughout each message type's method, as well as creation of any supplemental models or reports based on the type of message.

Additionally, this is where *ErrorNote* objects are created when an erroneous message is encountered.

One critical portion of the logic is the *create_report* method, invoked by the majority of the handler methods to create the *Report* object associated with the message. See *Models* for a more detailed description of *Report* and *Field* objects. The *create_report* method is as follows:

```
def create_report(self, report_type_name, patient, reporter):
    """Convenience for creating a new Report object from a reporter,
    patient and type """

    report_type = ReportType.objects.get(name=report_type_name)
    report = Report(patient=patient, reporter=reporter, type=report_type,
                    location=reporter.location, village=reporter.village)
    return report
```

The individual message types are described below.

Registration messages

Registration messages are of the form:

```
SUP YOUR_ID CLINIC_ID LANG VILLAGE
or
REG YOUR_ID CLINIC_ID LANG VILLAGE
```

Depending on if the user's role is 'Supervisor' or 'CHW', respectively. This message updates or creates an associated *Reporter* object, and no reports.

Pregnancy Messages

Pregnancy registrations are of the form:

```
PRE MOTHER_ID LAST_MENSES ACTION_CODE LOCATION_CODE MOTHER_WEIGHT
```

Only registered reporters are allowed to send this message, and a correct message sent from a registered reporter creates a Report object of type 'Pregnancy', with any associated Field objects.

Models and Extensions

This page covers an overview of the pertinent models added by the Rwanda customization of RapidSMS, as well as any extensions made via the ExtensibleModels classes within the RapidSMS Framework (Locations, Contacts, Connections, etc).

Locations

LocationShorthand

RapidSMS Locations in version 1.2 suffer from a lack of efficient capabilities for recursive expansion in the Location tree, having only a simple foreign key to self called *parent*.

In order to solve this problem, *apps.ubuzima.models* contains a model, *LocationShorthand*, which enables this expansion to be looked for any level of the tree, following a hard-coded structure that mirrors Rwanda's administrative boundaries that are pertinent to RapidSMS Rwanda, namely district and province.

This allows for simple, fast lookups such as:

```
LocationShorthand.objects.filter(district__name='Musanze')
```

Which would return all child Locations of the Musanze district.

The pertinent model implementation is below:

```
class LocationShorthand(models.Model):
    'Memoization of locations, so that they are more-efficient in look-up.'

    original = models.ForeignKey(Location, related_name = 'locationslocation')
    district = models.ForeignKey(Location, related_name = 'district')
    province = models.ForeignKey(Location, related_name = 'province')
```

UserLocation

Ubuzima also provides a way of mapping web users to the locations they are authorized to administer, via the User-Location model:

```
class UserLocation(models.Model):
    """This model is used to help the system to know where the user who is trying to␣
↪access this information is from"""
    user=models.ForeignKey(User)
    location=models.ForeignKey(Location)
```

Reporters

See *apps.reporters.models*.

Rwanda uses a separate model for known reporters that are validated for submitting reports within the system. This model doesn't extend Contact, and is created as a result of the registration process.

Further, it uses *PersistantConnection* and *PersistantBackend* models, mirroring RapidSMS' *Backend* and *Connection* models, but tying in the idea that a single reporter, using a single phone number (SIM card), could end up communicating over different backends, depending on the current state of the provider networks and active SMPP connections.:

```
class Reporter(models.Model):
    """This model represents a KNOWN person, that can be identified via
       their alias and/or connection(s). Unlike the RapidSMS Person class,
       it should not be used to represent unknown reporters
       ...
    """
    alias      = models.CharField(max_length=20, unique=True)
    first_name = models.CharField(max_length=30, blank=True)
    last_name  = models.CharField(max_length=30, blank=True)
    groups     = models.ManyToManyField(ReporterGroup, related_name="reporters",␣
↪blank=True)
    ...
    location   = models.ForeignKey(Location, related_name="reporters", null=True,␣
↪blank=True)
    role       = models.ForeignKey(Role, related_name="reporters", null=True,␣
↪blank=True)
    ...
    village    = models.CharField(max_length=255, null=True)
```

Reports

apps.ubuzima.models contains a model for taking note of parsing errors when reporters submit reports:

```
class ErrorNote(models.Model):
    '''This model is used to record errors made by people sending messages into the␣
↪system, to facilitate things like studying which format structures are error-prone,␣
↪and which reporters make the most errors, and other things like that.'''
    errmsg  = models.TextField()
    errby   = models.ForeignKey(Reporter, related_name = 'erring_reporter')
    created = models.DateTimeField(auto_now_add = True)
```

This model is also used for the “Error Log” view in the *reporters* app.

Finally, all reports consist of the main Report object, and (potentially) any fields associated with it:

```
class Report(models.Model):
    reporter = models.ForeignKey(Reporter)
    location = models.ForeignKey(Location)
    village = models.CharField(max_length=255, null=True)
    fields = models.ManyToManyField(Field)
    patient = models.ForeignKey(Patient)
    type = models.ForeignKey(ReportType)
    # meaning of this depends on report type..
    # arr, should really do this as a field, perhaps as a munged int?
    date_string = models.CharField(max_length=10, null=True)

    # our real date if we have one complete with a date and time
    date = models.DateField(null=True)

    created = models.DateTimeField(auto_now_add=True)
```

Because all fields are of numeric type, the Field model is fairly simple:

```
class Field(models.Model):
    type = models.ForeignKey(FieldType)
    value = models.DecimalField(max_digits=10, decimal_places=5, null=True)
```

Upgrading to the new RapidSMS Core

To upgrade to the new RapidSMS core, simply check out the new branch (until it is merged into main):

```
$ git checkout new-rapidsms
```

A number of database modifications need to occur:

```
alter table locations_locationtype add column "slug" varchar(50);
update locations_locationtype set slug=lower(name);
alter table locations_locationtype add unique(slug);

alter table "locations_location" add column "parent_type_id" integer;
update locations_location set parent_type_id = (select id from django_content_type␣
↪where model = 'location');
alter table locations_location add check (parent_id >= 0);
alter table locations_location add constraint "locations_location_parent_type_id_fkey
↪" foreign key (parent_type_id) references django_content_type(id) deferrable␣
↪initially deferred;
```

```
alter table locations_location drop constraint parent_id_refs_id_47ca058b;

CREATE TABLE "locations_point" (
    "id" serial NOT NULL PRIMARY KEY,
    "latitude" numeric(13, 10) NOT NULL,
    "longitude" numeric(13, 10) NOT NULL
);
alter table "locations_location" add column "point_id" integer REFERENCES "locations_
↪point" ("id") DEFERRABLE INITIALLY DEFERRED;

alter table locations_location add column "type_slug" varchar(50) REFERENCES
↪"locations_locationtype" ("slug") DEFERRABLE INITIALLY DEFERRED;

update locations_location set type_slug = (select slug from locations_locationtype t␣
↪where t.id = type_id);

alter table locations_location drop column type_id;

alter table locations_locationtype drop constraint "locations_locationtype_pkey";

alter table locations_locationtype drop column id;

alter table locations_locationtype alter column slug set not null;

alter table locations_locationtype add constraint locations_locationtype_pkey primary␣
↪key(slug);

alter table locations_location rename column type_slug to type_id;
```

Then, migrate points over to their own class, in shell_plus:

```
for l in Location.objects.extra(select={'longitude':'longitude','latitude':'latitude'}
↪).all():
    if l.longitude and l.latitude:
        l.point = Point.objects.create(longitude=l.longitude, latitude=l.latitude)
        l.save()
```

Finally, drop longitude and latitude from the locations table:

```
alter table locations_location drop column latitude;
alter table locations_location drop column longitude;
```

- genindex
- modindex
- search