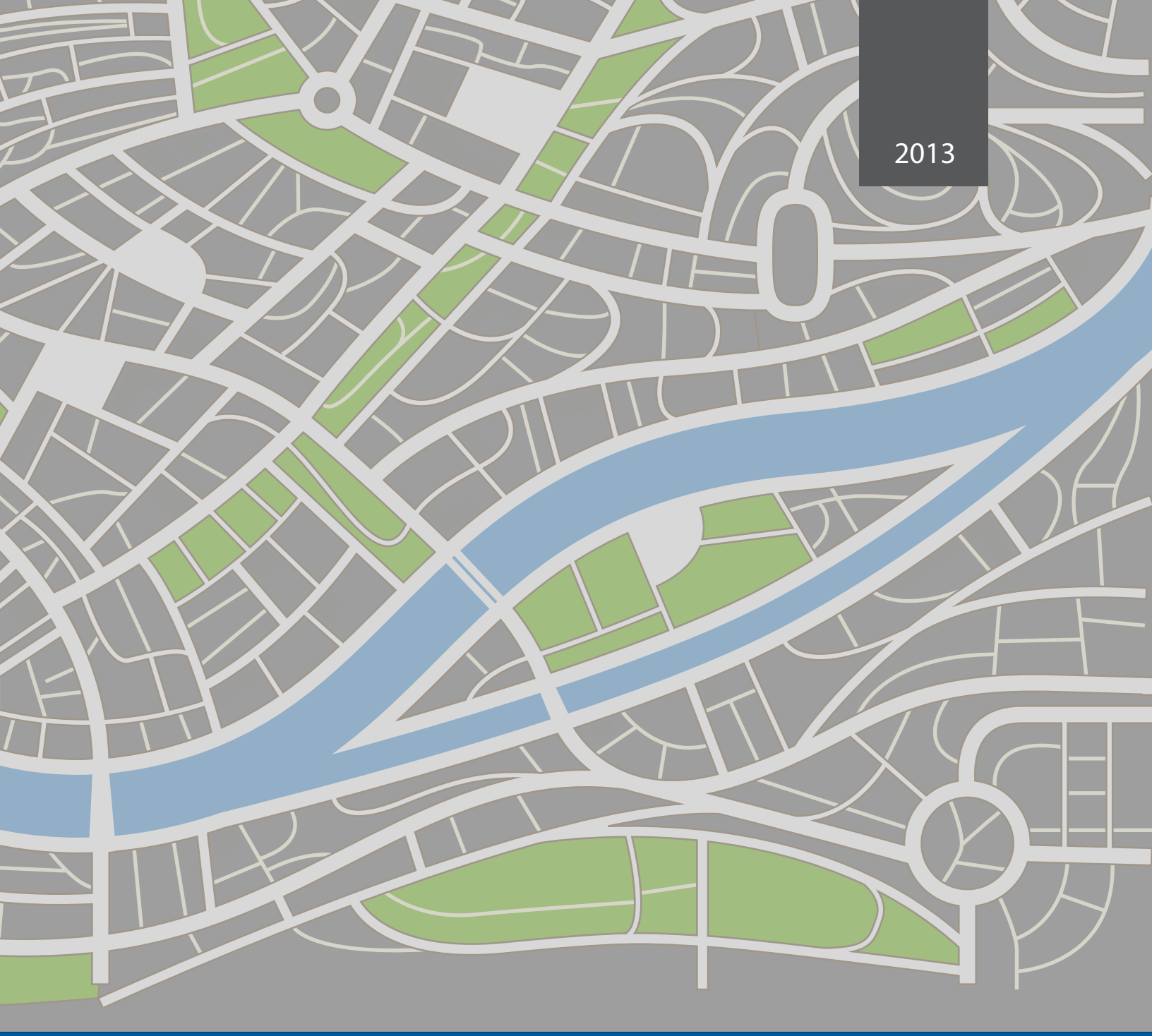

2013

CARISKA

Training Manual for Software Developers

OpenDRI
OPEN DATA FOR RESILIENCE INITIATIVE

GFDRR
Global Facility for Disaster Reduction and Recovery

THE UNIVERSITY OF THE WEST INDIES
ST. AUGUSTINE CAMPUS

THE WORLD BANK
Working for a World Free of Poverty

This volume is a product composed of myriad contributions from global citizens, consultants, development partners, open-source resources, and World Bank staff. Notable contributions were made by Jean-Mark Wright and Bishwa Pandey.

Published in Washington DC - February 2013
World Bank
www.worldbank.org/lcrdrm
Contact: Bradley Lyon (blyon@worldbank.org)
Bishwa Pandey (bpandey@worldbank.org)

University of West Indies - St. Augustine
www.sta.uwi.edu/eng/surveying
Contact: Earl Edwards (Earl.Edwards@sta.uwi.edu)

CARISKA

CARIBBEAN RISK ATLAS PROJECT

Training Manual for Software Developers

Jean-Mark Wright, Bishwa Pandey

OpenDRI OPEN DATA FOR RESILIENCE INITIATIVE
GFDRR Global Facility for Disaster Reduction and Recovery
THE UNIVERSITY OF THE WEST INDIES ST. AUGUSTINE CAMPUS
THE WORLD BANK Working for a World Free of Poverty

Table of Contents

1 Introduction

This manual aims to introduce participants to key concepts and foundational topics necessary for the implementation of the Caribbean Risk Atlas (CARISKA) project. The CARISKA online platform was created as a deliverable out of a grant agreement signed by the World Bank and the UWI Disaster Risk Reduction Centre. CARISKA is meant to be an online platform that houses information on flood, hurricanes and earthquakes. The main goal of CARISKA is to make this data online and available so that the capacity to analyze disaster risk is strengthened in the Caribbean Region. It is intended that this data would be used primarily by physical planners, disaster managers and persons involved in the modelling of risk.

This document presumes that readers are familiar with the Linux scripting environment and the use of common scripting commands. CARISKA was built upon the Geonode 1.2 framework and requires knowledge of HTML, CSS, Python and Django. This document builds upon this knowledge. The reader is taught how to create a custom project to achieve select fundamental functionality from the development and deployment of the CARISKA project. CARISKA was built on a Ubuntu 11.04 system. In particular, the following functionality from the CARISKA platform will be covered:

- Creating a custom Geonode installation
- Installing applications in a custom Geonode installation
- Integrating Disqus to replace Geonode's default comment system
- Integrating the display of flags based on layers uploaded
- Packaging the custom project

The instructions given here are expected to work on Ubuntu versions 11.04, 11.10 and 12.04. The Geonode version is also version 1.2. The instructions contained here may need to be adjusted for other platforms, operating systems or versions of Geonode.

2 Getting Started

In order to get started, the user needs to begin by having a working directory. The working directory represents the directory that the user will place all files for modification for the purposes of this project. The working directory used in this document is (/home/user/web). The user needs to be thoroughly acquainted with the following terminology, conventions and commands.

- ~ - refers to the HOME directory of the user, typically /home/user. (/home/user is analogous to ~).
- cd newdir – changes the user's current directory to "newdir"
- cp source dest – copies a file from source to destination.
- Git – a distributed version control system that is used to track changes to files in a project.
- clone – the act of copying a repository to the user's local drive.
- uwinode – this was the codename of the CARISKA project. The project folder containing the code and customizations to be done for CARISKA will be called "uwinode".
- pip – this binary is used to manage Python packages. The following are the most common commands that are used by pip in the form "pip command".
    - install – installs a package.
    - install==version – installs a specific version of a package.
    - uninstall – uninstalls a package.
    - search – searches for a package.
- Python Virtual Environment – Python virtual environments are used to isolate python installations. Typically, different projects may require different packages (and possibly specific versions). A virtual environment (often called "virtualenv") allows packages to be installed within that environment without affecting any other package versions installed on that system.
- Virtual environments can be created by running the following command:

```
$ virtualenv envname
```

- Virtual environments can be activated and deactivated using the following commands. Note that when the environment is activated the prompt changes and the name of the environment is in brackets before the "$" prompt. The "deactivate" command is used to exit the environment.

```
$ virtualenv envname # create a virtualenv called "envname"
$ cd envname # go into the folder
$ source bin/activate # activate the environment
```

```
(envname) $
$ deactivate
```

- Github - GitHub is a web-based hosting service for software development projects that use the Git revision control system. GitHub offers both paid plans for private repositories, and free accounts for open source projects. [Wikipedia]

3 Preliminaries

3.1.1 Exercise: Checking out GeoNode

3.1.1.1 Introduction

The purpose of this task is to re-acquaint the user with the process of checking out Geonode in developer mode. Geonode needs to be configured in development mode for a custom project to be created based on Geonode.

3.1.1.2 Objectives

At the end of this lesson, the user will be able to:

- Clone Geonode from Github and set it up in developer mode.

3.1.1.3 Outline

The CARISKA project relies on a development setup for Geonode. Therefore, the first step is to setup Geonode in development mode. The following changes are made to the typical way that Geonode is setup in developer mode.

- A custom Github repository is used (jaywhy13) instead of the official GeoNode repository.
- The openjdk-7-jdk package is used to replace openjdk-6 in the packages to be installed.
- In order to accommodate a java issue in Ubuntu 12.04, a change was made in pavement.py in line 236. sh(“mvn clean compile”) is replaced by sh(“ant zip”).

In order to setup Geonode in development mode, start by executing the following commands. The following commands will execute the following procedures:

- Install prerequisite packages
- Clone Geonode repository

- Build Geonode

Execute the following commands below:

```
sudo apt-get install -y --force-yes openjdk-7-jdk
sudo apt-get install -y vim zip unzip subversion git-core binutils
build-essential python-dev python-setuptools python-imaging gdal-bin
libproj-dev libgeos-dev python-urlgrabber python-nose pep8 python-
virtualenv python-gdal python-pastescript postgresql-contrib libpq-dev
gettext python-psycopg2 libxml2-dev libxslt1-dev
sudo apt-get install -y --force-yes ant maven2 --no-install-recommends
sudo apt-get install git -y # Install GIT for version control


mkdir ~/web # make a directory called "web" in your home directory
cd ~/web # go into the directory

git clone git://github.com/jaywhy13/geonode.git geonode
cd geonode
git submodule update --init
python bootstrap.py --no-site-packages
source bin/activate # activate the geonode virtual env
paver build
```

4 Creating a Custom Installation

4.1 Exercise: Creating a custom Geonode project (uwinode)

4.1.1 Introduction

The goal of this segment is to familiarize the user with the process of creating a custom project based on Geonode and making customizations to this projects. The customizations will consist of custom Django application installations, template and settings modifications. This segment also shows the user how to run the multi-threaded server and start up Geoserver while in developer mode, in preparation for further customizations to the custom application.

4.1.2 Objectives

At the end of this lesson, the user will be able to:

- Create a custom Geonode project
- Install additional Django applications into the custom project
- Verify the installation
- Test the application using the browser

4.1.3 Outline

4.1.3.1 Step 1: Making changes to your custom project and to Geonode

In order to create a custom project, a new Django project can be created based on a template project created by Ariel Nunez.

The following commands can be used to create the custom project.

```
# in the ~/web dir, make sure the Geonode env is still active
git clone git://github.com/ingenieroariel/geonode-project.git geonode-
template
django-admin.py startproject --template=geonode-template -epy,rst
uwinode
pip install -e uwinode # makes a link to your project in the Geonode
virtual env
```

First, the geonode-project is cloned from Ariel Nunez's repository online, then a custom project is created by passing using the --template argument to pass in the geonode-template directory as the directory that should be used as the template for the creation of the uwinode project.

The pip command makes a link to the project in the Geonode virtual environment. This is necessary to create a link between the code that will be adjusted and the Geonode source. The custom project will rely on Geonode's code base, however, the uwinode project needs to be in the Geonode's virtual environment. Using pip in this fashion creates this linkage.

4.1.3.1.1 Step 1a: Changing uwinode settings.py

After the uwinode project has been created, the next step is to edit settings to prepare our project for extension. Recall that each Django project has a settings.py file. The settings.py file is the main file that holds all the settings for a Django project. In this scenario, since the uwinode project has been created, we will edit and use the uwinode's settings.py file (as opposed to Geonode's) to make the changes we need. The following changes and additions will be made to the settings file.

- Add imports for the uwinode project – the template project does not include an import statement for the custom project, this needs to be added.
- Add applications to the INSTALLED_APPS variable. The INSTALLED_APPS variable holds a list of Django applications that are used by the Django project. New applications are added simply by adding applications to that list. The additionally functionality we need to add will be provided by custom applications, hence, this requires additions to the INSTALLED_APPS variable. The following applications are added:
    - Taggit – an application that is referenced by Geonode but not included in the template project.
    - Django-Forms-Bootstrap – Bootstrap version of forms for Django. Twitter created an open source toolkit called Bootstrap for the easy creation of form related interface tools.
    - uwinode – the custom project.
    - disqus – The Django Disqus project that provides the functionality for Disqus integration.
- Template context processors are functions that modify the context that is available for in views. Geonode has a template context processor that needs to be added. An additional Django request context processor is also added.

Important settings that are required to be in the file are also pointed out, these settings may or may not exist in the file. The user needs to confirm that these settings exist.

```
# This import can be added near the top of the file
import uwinode

# The following addition are required...
INSTALLED_APPS = (
...
'taggit',
'django_forms_bootstrap',
'uwinode',
'disqus',
)

# Ensure these settings exist
DB_DATASTORE = False
# Database datastore connection settings
DB_DATASTORE_NAME = ''
DB_DATASTORE_USER = ''
DB_DATASTORE_PASSWORD = ''
DB_DATASTORE_HOST = ''
DB_DATASTORE_PORT = ''
DB_DATASTORE_TYPE = ''
AUTH_PROFILE_MODULE = 'maps.Contact'

SITEURL = http://localhost:8000/


GEOSERVER_BASE_URL = "http://localhost:8001/geoserver/"

STATICFILES_DIRS = [
os.path.join(PROJECT_ROOT, "static"),
os.path.join(GEONODE_ROOT, "static"),

]

TEMPLATE_DIRS = (
os.path.join(PROJECT_ROOT, "templates"),
```

```
os.path.join(GEONODE_ROOT, "templates"),
)

ROOT_URLCONF = 'uwinode.urls' # uwinode urls are to be used instead of
Geonode's

# Also add this to the TEMPLATE_CONTEXT_PROCESSORS
TEMPLATE_CONTEXT_PROCESSORS = [
...
"django.core.context_processors.request", # Insert this below media
...
"geonode.maps.context_processors.resource_urls" # Insert this at the
end
]


# Add MIDDLEWARE_CLASSES
MIDDLEWARE_CLASSES = (
 'django.middleware.common.CommonMiddleware',
 'django.contrib.sessions.middleware.SessionMiddleware',
 'django.contrib.messages.middleware.MessageMiddleware',
 # The setting below makes it possible to serve different languages
per
 # user depending on things like headers in HTTP requests.
 'django.middleware.locale.LocaleMiddleware',
 'django.middleware.csrf.CsrfViewMiddleware',
 'django.contrib.auth.middleware.AuthenticationMiddleware',
)
```

4.1.3.1.2 Step 1b: Install Additional Packages

Requirements will also be added to the Geonode's requirements.txt file in the geonode/shared directory. When Geonode is installing, it installs all the packages in requirements.txt. An easy way to get Geonode to install custom Python packages is to amend this file. Also, pip has a "-r" option that is used to install all requirements (not already installed) from a file passed to the argument. Open the

shared/requirements.txt, add the following packages, then run pip install to install the additional packages.

```
django-disqus
googlemaps
django-forms-bootstrap
```

Once the file has been edited, the following command can be run to install the additional packages. Ensure that the Geonode environment is still active when this command is run.

```
$ pip install -r shared/requirements.txt
```

Run ./manage.py validate from inside the uwinode directory to ensure that there are no errors. If there are no errors, a database can now be created for the project.

```
./manage.py validate # This should say "0 error(s) found"
./manage.py syncdb # A super user may be created here
./manage.py migrate
./manage.py createsuperuser
```

4.1.3.1.3 Step 1c: Testing the server

Next the shared folder from the geonode directory must be copied to make some changes. The shared folder also contains a file called “dev-paste.ini”. This file is a configuration file that is passed to paster. Paster is a multi-threaded (better alternative to manage.py runserver) that is used to run the web server. It also proxies for geoserver, i.e. it provides a port whereby the application can access GeoServer. GeoServer will also be started. In a production environment, the Tomcat server is used to host GeoServer and Apache is typically used to serve Geonode. However, for development purposes, a startup.sh script is provided to start GeoServer and Paster is used as a substitute for Apache. Finally, the urls.py in the uwinode project needs to be replaced by the one found in the geonode project since the version in the geonode-template project is old.

```
# from the ~/web dir
cp -r geonode/shared uwinode/

# Open uwinode/shared/dev-paste.ini and change geonode.settings to
```

```
uwinode.settings

cp geonode/src/GeoNodePy/geonode/urls.py uwinode/uwinode



# Start Paster
cd uwinode
paster serve shared/dev-paste.ini

# Start up geoserver
cd geonode/src/geoserver-geonode-ext/
./startup.sh # run the startup geoserver script
```

If this is done correctly, http://localhost:8000/ should show the Geonode application. At this point the user may upload a layer. This layer will be needed for testing Disqus.

4.2 Exercise: Integrating Disqus

4.2.1 Introduction

This segment seeks to teach the reader how to integrate Django Disqus into a custom installation for GeoNode. This procedure relies on an account at Disqus.com and a custom Django project called "Django-Disqus".

4.2.2 Objectives

At the end of this lesson the user is expected to be able to:

- Setup a Disqus key in the Django settings.py
- Replace Geonode comments system with Disqus

4.2.3 Outline

4.2.3.1 Step 1: Setup Django Key

Disqus requires that developers use an API key. API keys are tied to developer accounts. To setup Disqus, the user will need a Disqus account. An account can be created at http://disqus.com. Once the user is logged in; the url: http://disqus.com/api/get_my_key/ can be used to get the key that is needed for the

settings.py file. Once the key is obtained, the following can be added to the settings.py file in the uwinode project.

```
DISQUS_API_KEY = '...'

DISQUS_WEBSITE_SHORTNAME = '...'
```

4.2.3.2 Step 2: Editing Templates

At this point, the user can now edit templates to replace Geonode's default comment system with a Disqus powered comment system. Recall that Django uses template overriding. In section 4.1.3.1.1 earlier we saw how uwinode's template were placed before Geonode's template in the TEMPLATE_DIRS variable, implying that any template in the uwinode directory will override a template found in the Geonode directory. Therefore, the Geonode maps/layer.html template will be copied to the uwinode project and edited.

```
# Go into the main web directory
cd ~/web
mkdir -p uwinode/uwinode/templates/maps
# copy over the layer.html template that shows a particular layer...
cp geonode/src/GeoNodePy/geonode/templates/maps/layer.html
uwinode/uwinode/templates/maps/
```

4.2.3.3 Step 3: Swap out Geonode's comment system

Open layer.html (uwinode/uwinode/templates/maps/layer.html) and search for the div with the id "comments_container", and delete the contents of the div. Replace the contents with:

```
{% load disqus_tags %}
{% disqus_dev %}
{% disqus_show_comments %}
```

The load disqus tags loads in Disqus tags, the disqus_dev activates developer mode and the last directive tells Disqus to load the comments. That setup is adequate for testing, however for production Django-Disqus requires more. The following are required:

- Each page needs to be assigned an identifier so that Disqus can differentiate pages.
- A site name needs to be created at Disqus.com. The Django-Disqus plugin tries to retrieve embed code based on the site name, therefore this requires that the Django contrib site has a name and that a proper name is specified for the DISQUS_WEBSITE_SHORTNAME property in settings.

The following is typical of a production setup:

```
{% load disqus_tags %}

{% set_disqus_identifier "layer_" layer.pk %}

{% set_disqus_url request.build_absolute_uri %}

{% disqus_show_comments %}
```

4.3 Exercise: Integrating Flags

4.3.1 Introduction

For the next exercise, a flag will be shown next to the title of the layer to indicate the country that the layer is for. In order to complete this exercise, the following activities will be completed:

- Install Django countries application
- Download flags for countries
- Create template tags to show the flag image based on the country ISO
- Edit the layer template to show the flag

4.3.2 Objectives

At the end of this section, the user will be able to:

- Install the Django countries application
- Write code to extract the center of a layer
- Create template tags to show country flag
- Edit layer template to show a flag
- Suggest better implementations of the flag exercise

4.3.3 Outline

4.3.3.1 Step 1: Installing Django Countries Application

A Python package called "Countries", provides the functionality to show a flag based on the ISO code for the country. The Countries application also comes with database fixtures that has a list of countries and their corresponding ISO codes. The countries application will need to be added to the INSTALLED_APPS variable and syncdb and migrate re-run to update the database with the countries application data. The actual flag image files are not packaged with the application and will be downloaded separately.

```
pip install countries

# Open up settings and add countries to INSTALLED_APPS

# Also add COUNTRIES_FLAG_PATH = STATIC_URL + 'flags/%s.png' near the
bottom

./manage.py validate # make sure all is well

./manage.py syncdb

./manage.py migrate
```

4.3.3.2 Step 2: Downloading Flags for countries

The zip file containing all the images will be downloaded and extracted into the static folder of the uwinode application. The flag zip file is downloadable from http://www.famfamfam.com/lab/icons/flags/famfamfam_flag_icons.zip.

```
mkdir ~/tmp

cd ~/tmp

wget http://www.famfamfam.com/lab/icons/flags/famfamfam_flag_icons.zip

unzip famfamfam_flag_icons.zip

mkdir -p ~/web/uwinode/uwinode/static # ensure the destination
directory exists

mv png flags # rename the png dir to flags

mv flags ~/web/uwinode/uwinode/static/ # copy over the flags dir
```

Once this is done http://localhost:8000/static/flags/jm.png should show a picture of the Jamaican flag. Ensure that paster is running on port 8000 (paster serve --reload shared/dev-paste.ini).

4.3.3.3 Step 3: Create template tag and filter to show flag

A template tag will be created to allow the flag to be shown easily given a layer. The template tag can be loaded then the filter used within any template where the context has a layer. The source code below shows the code that is necessary to setup the template tag. First a templatetags directory needs to be created and an __init__.py file placed within so Python can recognize the directory as a package. Next, any python file that is within that directory will be recognized as a template tag. For example, if the user placed a foo.py within the templatetags directory, the user could then type {% load foo %} to load in the filters and tags used in the foo namespace.

```
# Go into the Geonode folder
cd ~/web/geonode
source bin/activate # activate the Geonode env

cd ~/web/uwinode/uwinode # go into the application folder

mkdir templatetags # create a template tag dir

cp __init__.py templatetags # copy the __init__.py so python will
recognize this folder as package

# go into the templatetags folder


cd templatetags
```

Next, use a text editor and create a file called "layer_extras.py" in the templatetags folder. Use the following starter code to populate your new layer_extras.py file.

```
def get_layer_bbox(layer):
 """ Returns the bounding box for a layer in the format: (lon1 lon2
lat1 lat2)
 """
 resource = layer.resource
 bbox = {}

 for i in range(0,4):
 bbox[i] = float(resource.latlon_bbox[i])

 dx = float(bbox[1]) - float(bbox[0])
 dy = float(bbox[3]) - float(bbox[2])
```

```
return bbox

def get_layer_center(layer):
 """ Returns the center
 """
 bbox = get_layer_bbox(layer)
 if bbox:
 lat = (bbox[2] + bbox[3]) / 2.0
 lng = (bbox[0] + bbox[1]) / 2.0
 return {
 "lat" : lat,
 "lng" : lng,
 "x" : lng,
 "y" : lat
 }
 return None
```

The code above contains two functions. The get_layer_bbox gets the bounding box from the layer and returns it in a way that it can be passed to the get_layer_center function. The get_layer_center function returns the lat, long center of the layer.

4.3.3.4 Step 4: Writing image generation code

Given that the basic structure for the template tag is already setup, the following activities are left to be done:

- Use Google geocode to get the name of the country based on the lat, long coordinate.
- Complete the template filter that will automatically show the flag given a layer.

Consider the code below:

```
from googlemaps import GoogleMaps
from django import template
from countries.models import *
from django.template.defaultfilters import stringfilter
```

```
from django.conf import settings
from django.utils.safestring import mark_safe
register = template.Library()

@register.filter()
def layer_flag(layer):
 if layer:
       center = get_layer_center(layer)
       gmaps = GoogleMaps(settings.GOOGLE_API_KEY)
       lat = center.get('lat',None)
       lng = center.get('lng', None)
       if lat and lng:
              addr = gmaps.latlng_to_address(lat, lng)
              pieces = addr.split(",")
              country = pieces[-1].strip()
              flag_location = country_flag(country)
              if flag_location:
                     flag_location = flag_location.lower()
                     return mark_safe("<img src='%s'/>" % flag_location)
 return ""


def country_flag(country):
 """
 Returns a full path to the ISO 3166-1 alpha-2 country code flag image
based on the country name
 """
 if not country:
       return u''
 result = Country.objects.filter(name__icontains=country)
 if result:
       c = result[0]
       iso = c.iso
       flag_location = settings.COUNTRIES_FLAG_PATH % iso
       return flag_location
```

```
    return u''
```

The code snippets start by making imports for required libraries within the code snippets. The following imports are worth mentioning:

- GoogleMaps is imported from googlemaps – this provides the geocoding functionality to map a coordinate back to an address and ultimately, a country.
- The Country model is imported from the Django countries application that was installed earlier in 4.3.3.1.
- The mark_safe function is imported from django.utils.safestring in order to mark our HTML output as being safe. This will prevent the HTML from being escaped.
- The settings dictionary is imported from django.conf so we can access the COUNTRIES_FLAGS_PATH.

4.3.3.4.1 The layer_flag function

The layer_flag function expects to be passed a layer. The function first gets the center of the layer. Next, the GoogleMaps suite is initialized into the “gmaps” variable. The API_KEY from the settings is passed in to the constructor of GoogleMaps. Next, the latitude and longitude coordinates are retrieved from the dictionary. If both are not blank, then a call is made to the latlng_to_address function to get back an address. When this function runs it will return a string of the form "Road Name, Community, Parish, Country". In this scenario, the country needs to be obtained. Therefore, the output of the latlng_to_address function is split the by comma and only the final piece is saved. When the country is obtained, the country_flag function is then used to get the URL for the flag. This is then used to construct the image HTML and is return from the layer_flag function. Also note that a decorator (@register.filter) is used to register the function as a filter that may be used in templates.

4.3.3.4.2 The country_flag function

The country_flag function is also quite simple in its implementation. It first checks to ensure that a non-null input was passed in, then it checks the database to see if the country exists with the name supplied. If such a country exists, the ISO code is used to construct the URL for the flag.

4.3.3.5 Step 5: Editing template to use template filter

The layer.html template (uwinode/uwinode/templates/maps/layer.html) can now be edited to include the template tags and filter just created. The flag will be inserted beside the title of the layer on the layer page.

Once the file has been opened, the user will need to search for the div with the id “description” and edit the HTML so that it resembles the code below.

```
{% load layer_extras %}

<div id="description">
 {{ layer|layer_flag }}<h3>Admin Boundaries</h3>
...
```

The load layer_extras loads in layer_extras template tag that was created earlier. It also makes all the register layer_flag filter available for use in the template. In this scenario, the layer_flag filter was registered. The line {{ layer|layer_flag }} invokes the layer_flag function on the layer that is passed in and returns the HTML output. This is all that is required to print out the image of the flag.

4.3.3.6 Better way of implementing flags

This current way of implementing the flags is actually quite inefficient since the call to latlng_to_address has to contact Google’s server. This delays the response and hence the page may take a while to load. A better way to code the flags would be to use load the page then use an asynchronous request to load the flag image so that the response is not delayed.

5 Packaging your custom application

5.1 Introduction

In this section the steps necessary to make a Debian package out of the user's installation will be presented. A method for setting a simple Ubuntu apt repository will also be shown.

5.2 Objectives

After this section, the user will be able to:

- Make a distributable Debian executable of their custom project
- Adjust Geonode versioning information
- Provide download for custom application via Ubuntu apt repository

5.3 Outline

When Geonode makes a release it pulls on certain configuration files in the shared/package folder to make a release tar, which is then given to a special geonode debian project that uses Debian commands to generate the .deb file. This document presents two methods for Geonode to build a custom project.

- Edit the Geonode config files in places and change references to the user's project name. This method is presented in Step 1a.
- Edit the function that packages the files so that it is able to read configurations from an external source. This is presented in step 1b.

5.3.1 Step 1a: Edit Geonode configuration

Go into the shared/package folder and run the following commands to replace all instances of geonode.settings with uwinode.settings.

```
cd ~/web/geonode/shared

grep -r -l "geonode.settings" package | xargs sed -i
's/geonode.settings/uwinode.settings/g'
```

5.3.2 Step 1b: Edit Uwinode configuration

Go into the shared/package folder for uwinode and run the following commands to replace all instances of geonode.settings with uwinode.settings.

```
cd ~/web/uwinode/shared

grep -r -l "geonode.settings" package | xargs sed -i
's/geonode.settings/uwinode.settings/g'
```

Changes also need to be made to the source of pavement.py. Open the pavement.py file in the Geonode directory and replace the function for make_release with the version provided below:

```
@task

@cmdopts([

    ('name=', 'n', 'Release number or name'),

    ('no_svn', 'D', 'Do not append svn version number as part of name
'),

    ('append_to=', 'a', 'append to release name'),

    ('skip_packaging', 'y', 'Do not call package_all when creating a
release'),

    ('extra_pkg_tree=', 't', 'Also walk this pkg tree please')

])

def make_release(options):

    """

    Creates a tarball to use for building the system elsewhere
```

```
    (production, distribution, etc)
    """


    if not hasattr(options, 'skip_packaging'):
        call_task("package_all")
    if hasattr(options, 'name'):
        pkgname = options.name
    else:
        pkgname = create_version_name()
        if hasattr(options, 'append_to'):
            pkgname += options.append_to


    if not hasattr(options, 'extra_pkg_tree'):
        extra_pkg_tree = None
    else:
        extra_pkg_tree = options.extra_pkg_tree


    with pushd('shared'):
        out_pkg = path(pkgname)
        out_pkg.rmtree()
```

```
        path("./package").copytree(out_pkg)



        tar = tarfile.open("%s.tar.gz" % out_pkg, "w:gz")

        for file in out_pkg.walkfiles():

            #print "Adding file %s to tar " % file

            tar.add(file)



        if extra_pkg_tree:

            with pushd(extra_pkg_tree):

                for file in path(".").walkfiles():

                    filename = str(file).replace("./","") # I don't
like how "./filename" looks

                    fullname = "%s/%s" % (pkgname, filename)

                    tar.add(file, fullname) # hmmm..  any side
effects? Adding the same file twice



        tar.add('./README.release.rst', arcname=('%s/README.rst' %
out_pkg))

        tar.close()



        out_pkg.rmtree()
```

```
        info("%s.tar.gz created" % out_pkg.abspath())
```

The modification given above allows make_release to take an extra argument that specifies an extra directory tree that should be walked to add files to the tar. In the next step, the shared/package path will be provided as an argument to the function.

5.3.3 Step 2: Make the release

The next step is to make the release. An extra argument is needed if the user previous following Step 1b.

```
cd ~/geonode/

paver make_release # if Step 1A was run

paver make_release extra_pkg_tree=../uwinode/shared # if Step 1B was
run
```

5.3.4 Step 3: Edit versioning information (optional)

If the user wishes to edit versioning information, the file in ~/web/geonode/src/GeoNodePy/geonode/__init__.py can be edited. The version number can be changed in this file. The print out below gives a sample of the file at version 1.2 for CARISKA.

```
__version__= (1, 2, 0, 'alpha', '+uwi')



def get_version():

    import geonode.utils

    return geonode.utils.get_version(__version__)
```

5.3.5 Step 4: Build the .deb file

Once the release is built, a Geonode*.tar.gz file will be placed in the shared/package directory. The next step is to clone the geonode-deb repository and create the .deb file.

```
$ cd ~/web

$ git clone git://github.com/jaywhy13/geonode-deb.git

$ cp geonode/shared/Geonode*.gz geonode-deb

$ cd geonode-deb
# Build the deb file
$ debuild –uc -us
```

Once this command completes successfully, a deb file will be placed in the folder outside the geonode-deb folder (~/web folder).

5.3.6 Step 5: Setup an Ubuntu apt repository

Given Apache is installed, run the following steps to copy the package to /var/www then generate package information for the install.

```
$ cd /var/www

/var/www$ sudo mkdir repo

/var/www$ cd repo

/var/www/repo$ sudo mkdir binary

/var/www/repo$ cp ~/web/*.deb binary


/var/www/repo$ sudo su
```

```
/var/www/repo$ dpkg-scanpackages binary /dev/null | gzip -9c >
binary/Packages.gz
```

Once this is completed, another can try adding the line “deb http://<<host>>/repo /binary” to the /etc/sources.list file, running an apt-get update then installing geonode.

6 Automating Packaging

This section will discuss techniques for deploying the custom application using Fabric. Using Fabric, python code can be written that will get executed on a remote server. Highlights are made to key areas of approach in the code as it relates to automating the packaging of a custom application. For this exercise the user will be shown how to utilize Fabric functions to automate the process described in section 5. This section is designed to motivate how the packaging process may be automated. The full code base is available for the user in Appendix (Section 8.3).

6.1 Utility Functions

The following functions are used as library functions to provide core functionality that is used by the main functions that are directly related to packaging.

- with – Python's with keyword is used to setup a context within all code is executed within that context. For this scenario, it is used to enter and exit virtual environments and to enter and exit directories on the remote system.
- run – Fabric function used to run a command remotely with normal user permissions.
- sudo – Fabric function used to run a command remotely with elevated user permissions.
- virtualenv – a function used to activated a virtual environment. This function is used along with the Python "with" function. This enables the developer to execute commands within a virtual environment. The example below runs paver build within the "geonode" virtual environment.

```
with virtualenv("geonode"):
      run("paver build")
```

- git_clone – this function clones a Github project.
- check_sudo – this function tries to execute a sudo command. It will fail if no sudo is available. This is used to escape from a function in the event that no sudo is available.
- setup_geonode_packages – this function is used to install all the Geonode packages that are required.
- setup_geonode_build_packages – this function is used to install all the Geonode packages that are required for setting up geonode in Developer mode.

6.2 Cloning Functions

The following functions are directly related with cloning both projects (Geonode and Uwinode) and putting Uwinode in developer mode. Note that, this step requires that the uwinode project is a project in Github.

- checkout_geonode – this function is used to clones Geonode and set it up in developer mode. This function relies on the with keyword to enter the geonode, then on the run function to execute the commands necessary for putting Geonode in developer mode. These commands were covered in 3.1.1.3.
- checkout_uwinode - this function clones uwinode and ensures the requirements are all installed (pip install –r requirements.txt). See 2 for notes on pip.

6.3 Packaging Functions

The package uwinode function performs automation based on the steps described in 5.3. It executes the following steps remotely:

- Ensures all geonode build pacakges are installed on the server.
- Clones the geonode-deb project.
- Makes a directory tree that resembles the one used for Geonode.
- Replaces all instances of geonode.settings with uwinode.settings.
- Runs paver make_release --extra_pkg_tree=uwinode
- Runs debuild to build the Debian package

7 Editing HTML and CSS

7.1 Introduction

The following exercise will acquaint the user with several HTML and CSS concepts. In this exercise, the user will be asked to write style information so that the basic interface provided GeoNode can be extended to resemble the work done with CARISKA. In this exercise, the user will need to write CSS styles and will also need to add, remove or modify components as is necessary for the design of the website.

7.2 Objectives

At the end of this exercise, the user will be able to:

- Apply the following CSS properties easily:
    - background
    - width
    - height
    - clear
    - padding
    - margin
    - float
    - color
- Use Twitter Bootstrap conventions to add, remove or modify HTML on the interface.
- Suggest what template may need to be edited to change a certain area of the website.

7.3 Outline

7.3.1 Step 1: Preliminaries

Templates are widely used in Django to separate a page in different logical areas. Template inheritance is widely used to allow a dynamic page to only supply information for certain sections, while allowing other static pages such as a header or footer to construct the outline of the page. To begin this exercise, the base.html template will be copied from the Geonode folder. This folder outlines the structure of the HTML page. This page has links to several Javascript files and CSS files. Many pages will use inherit from this page and fill in the areas of functionality as is needed. This file will now be edited to include a CSS file that will be edited in later exercises.

```
$ cp geonode/src/GeoNodePy/geonode/templates/base.html
uwinode/uwinode/templates/
```

Now open the file and add the following line at approximately line 36.

```
<link href="{{ STATIC_URL }}styles.css" rel="stylesheet"/>
```

After this code is added, whenever a page is loaded, the user's browser will try to fetch a CSS file from the link /static/styles.css. This file now needs to be created.

```
$ touch uwinode/uwinode/static/styles.css
```

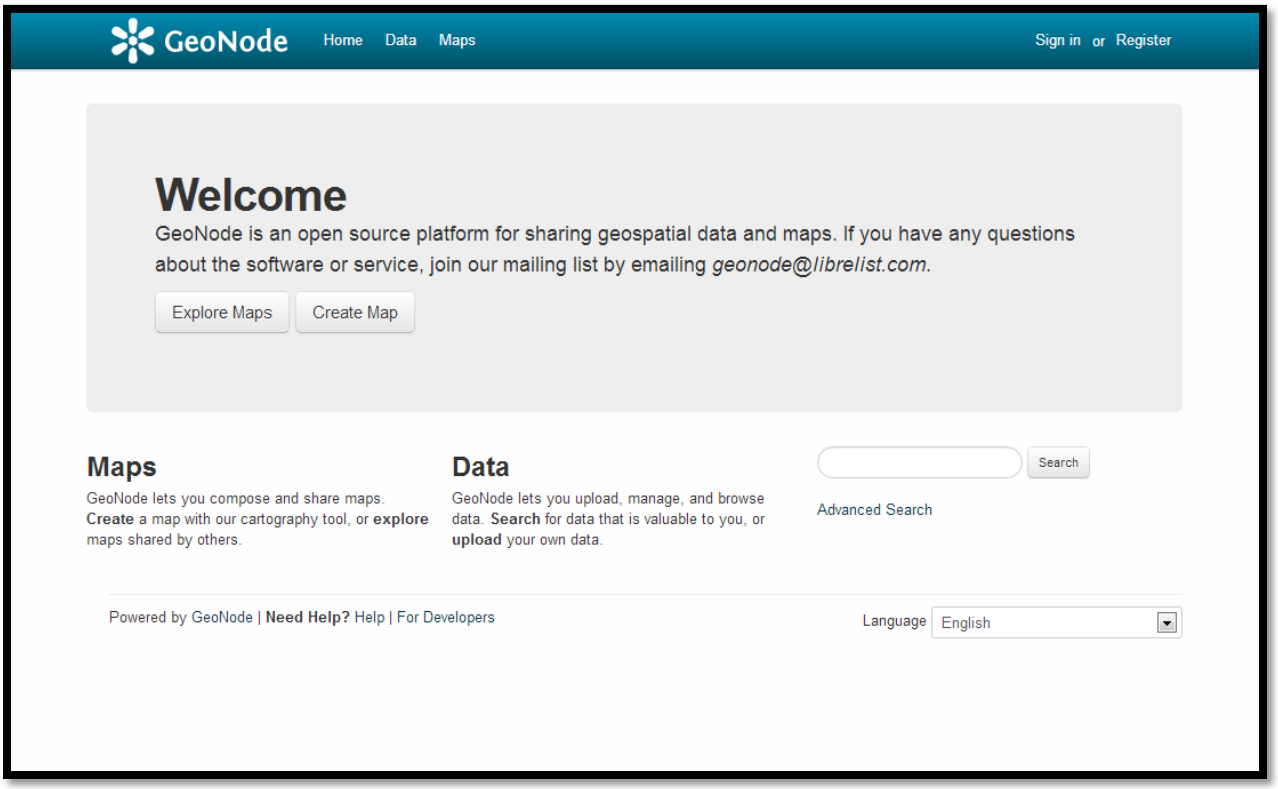

Figure 1- Geonode with Bootstrap

The user is now encouraged to attempt to style the header and footer using different CSS techniques. Figure 1 shows how Geonode currently looks with Bootstrap, the user is to add styles so that the interface resembles that of Figure 2. For this question, the user will style the header and footer, in particular the following components need to be styled:

- Banner area
    - CARISKA logo
    - Search area
- Ribbon area
    - Ribbon
    - Icons (Home, Data, Maps)
- Footer
    - Powered by message
    - Drop down at right

The following information should prove useful:

- Banner
    - CARISKA logo
        - The distance between the CARISKA logo and the left hand side of the page is 70px.
        - The CARISKA logo is white.
        - The orange color is #D95723
        - The name of the CARISKA logo is cariska_logo.png
        - The logo is floated left
    - Background
        - The background color is #4E4E4E
    - CARISKA ribbon
        - The ribbon is a PNG (apart from the colours, the other sections of the image are transparent)
        - If a layer with a black background is placed before the ribbon, a negative margin may be used to shift the ribbon upwards to create the illusion of the ribbon being on the black background.
        - HTML changes are needed to implement the ribbon properly.
    - Strip
        - The multi-coloured strip that runs at the bottom may need to be shifted 36px pixels to the left because the first color is grey and it blends in with the background.
        - The strip is called cariska_strip.png
    - Main menu icons
        - The icon container needs to be floated to the right and it needs to clear both.
        - Home icon

- 82x61 – home_icon.png
- Data icon
  - 70x61 – data_icon.png
- Maps icon
  - 82x61 – maps_icon.png

- Sign In
  - The sign in button can use #EB8651 or #C6602A as the background color.

- Footer
  - The background color is black.
  - The height is 35px, a padding of 15px is sufficient.

7.3.3 Step 3: Edit main section

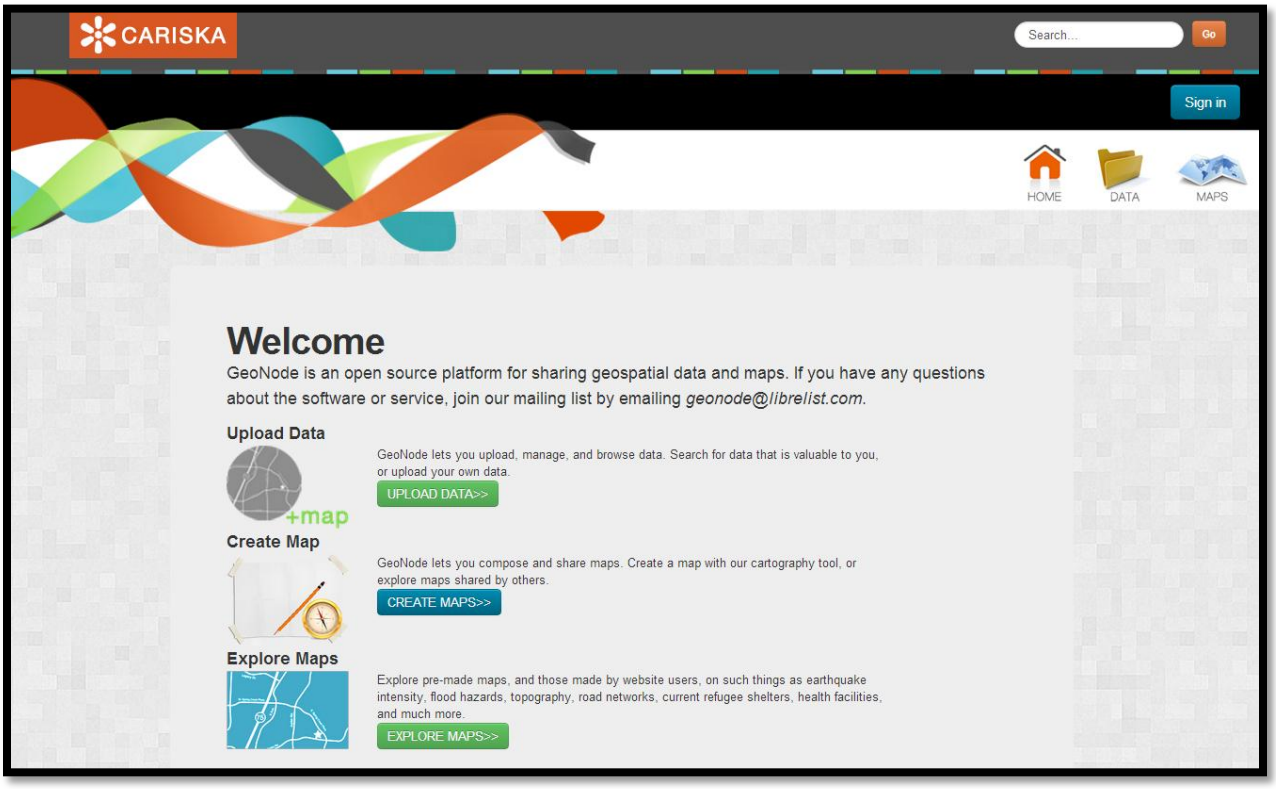

Figure 2 - Geonode with CARISKA customizations

For this section the user needs to copy another template that is responsible for displaying information on the home page. The index.html file overrides the base.html template and overrides many content blocks,

providing more functionality. Of particular interest is the “body_outer” that it overrides. This section contains the main content area of the website. This section will be edited next to reflect some of the changes done for CARISKA. Again, this section is presented as a user assignment, the user is challenged to attempt to style the main section of the page and the background. Only the main welcome section of the page and the background need to be styled. The following information should prove useful:

- When setting the background, the user may need to completely set the entire background variable instead of just setting a part of it to ensure that the entire value is overridden.
- Each row in the welcome area (Upload Data, Create Maps and Explore Maps) can be treated as a Bootstrap row, using a span2 for the icons and span8 for the writeup.
- The BODY uses a background called noise.png.

7.3.4 Step 3: Implementation Review

In implementing the different areas of the interface, the following approaches were taken to implement the different areas of functionality. Note that the full solution for the CSS can be found in 8.1.

- Banner area
    - A 70px left padding was assigned to the navbar container. This was used to adequately push the CARISKA logo inward.
    - The background was first set to none, then the background color was set to #4E4E4E. This was done to clear all settings for the background. the background-repeat was also set to repeat-x.
    - A div called “navbar-inner” exists within the “navbar” div and has the same height. Therefore, this div was used to hold the strip background. A left margin of -36px was used to pull the navbar-inner 36px to the left to hide the grey part of the image but then a more specific selector (.navbar .navbar-inner) assign a left padding for the navbar-inner. Recall that padding is applied inside the component while margin is applied outside the component.
    - The CARISKA logo was assigned a right padding but the background position was used to center the image in the center. The image was also floated left.
    - The base.html file was adjusted so that the main menu had the following hierarchical structure:

```
<div class=”main-menu-bg”></div>
<div class=”main-menu”>
```

- The main-menu-bg has a black background that is used as a backdrop for the ribbon. The main-menu-bg is given a height of 100px. The main-menu is only used as a container to which a padding is applied and given a negative margin of 100px to move the inner container upwards so that the ribbon is shown above the black backdrop. main-menu-inner has uses CSS3’s multiple backgrounds to show the background of the ribbon and the white background that is behind the ribbon.

- The main-menu-signin and main-menu-icons control needed to be floated right.
- The main menu icons need a negative top margin of 62px.
- The search HTML was moved into the “container” div. It was floated right and given a padding of 30px.

- Main Page
  - The footer had to be removed from the container div and placed at the same level with the body. See the full solution for the index.html page in 8.2.
  - An “intro-icon” class was created for all icons in the main section of the website and a width of 130px assigned to it.
  - Each item in the main area of the page (Upload Data, Create Map, Explore Maps) was declared as a bootstrap row (each containing div has the class “row”).
    - The first div inside each row (the div containing the icon) was given a class of “span2” to ensure that the div spans over 2 columns.
    - The second div inside each row (the div containing the text) was given a class of “span8”.
  - Each button was created using an A tag and assigning the classes “btn”. Both the Upload Data and Explore Maps buttons use the “btn-success” class to make it appear green. The “Create Maps” buttons use the “btn-primary” class to make it appear blue.

8 Appendices

8.1 Appendix A – styles.css

```
.navbar .container {

    width: auto;

    padding:0 0 0 70px;

}



/** background at the top **/

.navbar-fixed-top {

    background:none;

    background-color:#4E4E4E;

}



/** this has the stripe below the header **/

.navbar-inner {

    background:none;

    background-image:url("/static/cariska_assets/cariska_strip.png");

    background-position:bottom left;

    background-repeat:repeat-x;
```

```
    /** need to shift background left **/
    margin-left:-36px;
}


/** shift it back 36px, needed more specificity **/
.navbar .navbar-inner {
    padding-left:36px;
}


/** change the logo **/
.nav-logo {
    background:url("/static/cariska_assets/cariska_logo.png") no-
repeat #D95723;
    /** Add 10px padding on one side **/
    padding:0 10px;
    /** Put the icon in the center **/
    background-position:center center;
    float:left;
}


/** background styles **/
```

```
BODY {

    background:url("/static/cariska_assets/noise.png");

}



/** the main menu bg that has a black bg **/

.main-menu-bg {

    background-color:black;

    height:100px;

    /** move the main menu upwards to meet the banner **/

    margin-top:-30px;

}



/** style the main menu **/

.main-menu {

    width:100%;

    height:200px;

    /** add this padding to push main-menu-inner down a little **/

    padding-top:20px;

    /** move this up 100px into the black bg provided by the main-
menu-bg **/

    margin-top:-100px;
```

```
}



/** a container inside the main menu that holds the ribbon and the
ribbon bg **/

.main-menu-inner {

    height:200px;

    background:url("/static/cariska_assets/cariska_ribbon.png") no-
repeat,

    url("/static/cariska_assets/cariska_ribbon_bg.png") repeat-x;

}



/** this holds the signin button **/

.main-menu-signin {

    /** move it back up 20px to undo the top-padding on the parent **/

    margin-top:-20px;

    height:100px;

    padding:28px 20px 10px 0px;

    float:right;

}
```

```
/** main menu icons **/

.main-menu-icons {

    clear:both;

    float:right;

    margin-top:-62px;

}



#home_icon {

    height: 61px;

    width: 50px;

    background-image: url("/static/cariska_assets/home_icon.png");

}



#data_icon {

    height: 61px;

    width: 70px;

    background-image: url("/static/cariska_assets/data_icon.png");

}



#maps_icon {
```

```
    height: 61px;

    width: 82px;

    background-image: url("/static/cariska_assets/maps_icon.png");

}




/** style the footer **/

.footer {

    height:35px;

    padding:15px 15px 5px 15px;

    background:url("/static/cariska_assets/cariska_strip.png") repeat-
x left top #000;

    margin-bottom:0px;

}



/** style the main welcome region **/

.intro-icon {

    width:130px;

}



/** style sign in **/
```

```
.login {

    float:right;

    padding-right:30px;

}



/** add some style for the login button **/

.login button {

    background: -webkit-linear-gradient(top, #eb8651 0%,#c6602a 100%);

    color:white;

}
```

8.2 Appendix B – index.html

```
{% extends "base.html" %}

{% load i18n %}



{% block title %} {% trans "Welcome!" %} - {{ block.super }} {%
endblock %}



{% block body_outer %}

  <div class="hero-unit">

    <h1>{% trans "Welcome" %}</h1>

    <p>

      {% blocktrans %}

        GeoNode is an open source platform for sharing geospatial data
and maps.  If you have any questions about the software or service,
join our mailing list by emailing <em>geonode@librelist.com</em>.

      {% endblocktrans %}

    </p>



    <div class="into_maps">

      <!-- Upload Data -->

      <h3>Upload Data</h3>
```

```
      <div class="row welcomerow">

        <div class="span2">

          <img class="intro-icon"
src="/static/cariska_assets/upload_map_icon.png"/>

        </div>

        <div class="span8">

          GeoNode lets you upload, manage, and browse data. Search for
data that is valuable to you, or upload your own data.

          <p>

            <a class="btn btn-success" href="/data/upload">UPLOAD
DATA&amp;gt;&amp;gt;</a>

          </p>

        </div>

      </div>



      <!-- Create maps -->

      <h3>Create Map</h3>

      <div class="row welcomerow">

        <div class="span2">

          <img class="intro-icon"
src="/static/cariska_assets/create_map_icon.png"/>

        </div>
```

```
        <div class="span8">

          GeoNode lets you compose and share maps. Create a map with
our cartography tool, or explore maps shared by others.

          <p>

            <a class="btn btn-primary" href="/maps/new">

              CREATE MAPS&amp;gt;&amp;gt;

            </a>

          </p>

        </div>

      </div>



      <!-- Explore maps -->

      <h3>Explore Maps</h3>

      <div class="row welcomerow">

        <div class="span2">

          <img class="intro-icon"
src="/static/cariska_assets/explore_map_icon.png">

        </div>

        <div class="span8">

          Explore pre-made maps, and those made by website users,

          on such things as earthquake intensity, flood hazards,
```

```
          topography, road networks, current refugee shelters, health
          facilities, and much more.
          <p>
            <a class="btn btn-success" href="/maps/search">
              EXPLORE MAPS&amp;gt;&amp;gt;
            </a>
          </p>
        </div>
      </div>
    </div>


  </div>
  <div class="row">
    <div class="span4">
        <form class="form-search" action="{% url data_search %}"
method="POST">
              {% csrf_token %}
          <input type="text" class="input-medium search-query"
name="q">
          <button type="submit" class="btn btn-small">{% trans
"Search" %}</button>
```

```
        </form>

        <a href="{% url data_search %}">{% trans "Advanced Search"
%}</a>

    </div>

  </div>

{% endblock %}
```

8.3 Appendix C: Fabric Scripts for deployment

```
import os

from fabric.api import *

from fabric.operations import *

from fabric.contrib import files

from contextlib import contextmanager as _ctxmgr



GEONODE_PACKAGES = (

    'openjdk-7-jdk',

    'vim',

    'zip',

    'unzip',

    'subversion',

    'git-core',

    'binutils',

    'build-essential',

    'python-dev',

    'python-setuptools',

    'python-imaging',

    'gdal-bin',
```

```
    'libproj-dev',
    'libgeos-dev',
    'python-urlgrabber',
    'python-nose',
    'pep8',
    'python-virtualenv',
    'python-gdal',
    'python-pastescript',
    'postgresql-contrib',
    'libpq-dev',
    'gettext',
    'python-psycopg2',
    'libxml2-dev',
    'libxslt1-dev',
    'ant',
    'maven2',
    'dpkg-dev'
)


GEONODE_BUILD_PACKAGES = (
```

```
    'debhelper',

    'devscripts',

)



### Preresuite Code

@_ctxmgr

def virtualenv(folder=None):

    if not folder:

        folder = 'bin'

    with prefix('source %s/activate' % folder):

        yield




def git_clone(project, account='jaywhy13'):

    with cd('/home/mgi'):

        run('mkdir -p web')

        with cd('web'):

            if project.endswith('.git'):

                project = project.replace(".git","")
```

```
            run('git clone http://github.com/%s/%s.git' % (account, project))



def check_sudo():

    """ Checks that we can sudo

    """

    sudo('ls')



### GEONODE stuff

def setup_geonode_packages():

       """ Install all the Geonode packages

       """

    install_package(" ".join(GEONODE_PACKAGES),['--force-yes', '--no-install-
recommends'])



def setup_geonode_build_packages():

       """ Install all the Geonode packages necesssary to build a release

       """

    install_package(" ".join(GEONODE_BUILD_PACKAGES))



def checkout_geonode(git_protocol="http"):

    """ Checks out geonode in development mode
```

```
    """

    print " ===== Checking out geonode"

    if files.exists("geonode"):

        print "Geonode dir exists"

        run("rm -rf geonode")



    git_clone("geonode", protocol=git_protocol) # check out our Geonode
forked repos

    with cd("geonode"):

        if git_protocol is "http":

            run("git submodule init")

            run("sed -i \"s/git:\/\//http:\/\//g\" .gitmodules")

            run("git submodule sync")

            run("git submodule update")

        else:

            run("git submodule update --init")

        run("python bootstrap.py --no-site-packages")

        with virtualenv():

            run("paver build")
```

```
def checkout_uwinode():

    """ Checks out uwinode from our Github repos

    """

    print " ===== Checking out uwinode"

    if files.exists("uwinode"):

       run("rm -rf uwinode")

    git_clone("uwinode")

    with cd("geonode"):

        with virtualenv(): # enter the geonode virtual env

            run("pip install -r ../uwinode/requirements/production.txt") #
install our pcakages

            run("pip install -e ../uwinode")

            run("../uwinode/manage.py syncdb")

            run("../uwinode/manage.py migrate")



def deploy_uwinode(purge=False):

      """ Sets up and deploys our custom installation of uwinode.

             If there is already a uwinode directory, this is only

             overwritten if purge is True.

      """

    print " ===== Deploying uwinode"
```

```
    print " ===== Setting up APT"



    check_sudo() # make sure we can sudo

    setup_geonode_packages()

    with cd("/tmp"):

        # checkout geonode

        if not files.exists("geonode") or purge:

            checkout_geonode()



        # checkout uwinode

        if not purge or not files.exists("uwinode"):

            checkout_uwinode()



        # package uwi node and create a deb

        package_uwinode()



def package_uwinode(base_dir=None):

    print " ===== Packaging uwinode"

    if not base_dir:

        base_dir = "/tmp"
```

```
    with cd(base_dir):

        print " ===== Setting up geonode dev packages"

        setup_geonode_build_packages() # make sure this is done

        if not files.exists("geonode-deb"):

            git_clone("geonode-deb") # check out the geonode-deb project for
building Geonode

        with cd("geonode"):

            with virtualenv():

                # copy the package folder in shared and make our changes
there...

                if files.exists("shared/uwinode"):

                   run("rm -rf shared/uwinode")

                run("mkdir -p shared/uwinode/support") # make the directory
tree



                # copy over the files we need to change

                print " ===== Copying package files to make custom changes "

                # we need to replace geonode.settings with uwinode.settings
in these files...

                for filename in ["install.sh", "support/geonode.binary",
"support/geonode.local_settings", "support/geonode.wsgi"]:
```

```
                    target_dir = "shared/uwinode/"

                    if "/" in filename:

                        filepath = "/".join(filename.split("/")[:-1])

                        target_dir = target_dir + filepath + "/"



                    run("cp shared/package/%s %s" % (filename, target_dir))



                pkg_tree = "uwinode"

                print " ===== Making our custom changes "

                # make some changes to the geonode settings

                          # Copy over settings from geonode/shared/package
and replace geonode.settings with uwinode.settings

                # we need the install settings to point to our installation

                run("sed -i \"s/geonode\.settings/uwinode\.settings/1\"
shared/%s/install.sh" % pkg_tree)

                run("sed -i
\"s/\$GEONODE_LIB\/src\/GeoNodePy\/geo/\$GEONODE_LIB\/src\/GeoNodePy\/uwi/1\"
shared/%s/install.sh" % pkg_tree)

                run("sed -i \"s/geonode\.settings/uwinode\.settings/1\"
shared/%s/support/geonode.binary" % pkg_tree)

                run("sed -i \"3aimport uwinode\"
shared/%s/support/geonode.local_settings" % pkg_tree)
```

```
                run("sed -i \"4aUWINODE_ROOT =
os.path.dirname(uwinode.__file__)\" shared/%s/support/geonode.local_settings"
% pkg_tree)

                files.append("shared/%s/support/geonode.local_settings" %
pkg_tree,"TEMPLATE_DIRS = (os.path.join(UWINODE_ROOT,
'templates'),'/etc/geonode/templates',os.path.join(GEONODE_ROOT,
'templates'),)")

                run("sed -i \"s/geonode\.settings/uwinode\.settings/1\"
shared/%s/support/geonode.wsgi" % pkg_tree)

                files.append("shared/requirements.txt", "-e
git+https://github.com/jaywhy13/uwinode.git#egg=uwinode") # TODO need to
figure out a way to isolate this file

                print " ==== Building our custom release with paver"

                # Make the release

                          run("paver make_release --extra_pkg_tree=uwinode")

                run("mv shared/GeoNode*.tar.gz ../geonode-deb/")

        with cd("geonode-deb"):

            print " ===== Building geonode deb"

            run("debuild -uc -us -A")
```

CARISKA

Training Manual for Software Developers

OpenDRI OPEN DATA FOR RESILIENCE INITIATIVE
GFDRR Global Facility for Disaster Reduction and Recovery
THE UNIVERSITY OF THE WEST INDIES ST. AUGUSTINE CAMPUS
THE WORLD BANK Working for a World Free of Poverty