
GeoNode Developers Workshop Documentation

Release 2.0

GeoNode

February 08, 2013

CONTENTS

1	Introduction to GeoNode development	3
1.1	GeoNode Components	3
1.2	Standards	4
1.3	GeoNode Architecture	6
1.4	Development References	7
2	Development Prerequisites and Core Modules	9
2.1	GeoNode's Development Prerequisites	9
2.2	GeoNode's Core Modules	12
3	Customized GeoNode Projects	17
3.1	Introduction to GeoNode Projects	17
3.2	Setting up your GeoNode Project	18
3.3	Theming your GeoNode Project	22
3.4	Adding Additional Django apps to your GeoNode Project	30
3.5	Integrating your Project with other Systems	39
4	Setting up a GeoNode development environment	41
4.1	GeoNode Development Tools	41
4.2	Git Repository Setup	41
4.3	Installing GeoNode's Python Package	41
4.4	Pavement.py and Paver	41
4.5	Manually Deploying your Development Environment	41
5	Loading Data into a GeoNode	43
5.1	GeoServer Data Configuration	43
5.2	Using ogr2ogr to load data into GeoNode	43
5.3	Loading OSM Data into GeoNode	43
6	GeoNode APIs	45
6.1	OGC Services	45
6.2	GeoServer REST API	45
6.3	GeoServers Import and Print APIs	45
6.4	GeoNode's Ad-Hoc API	45
7	GeoNode debugging techniques	47
7.1	Debugging GeoNode's Python Components	47
7.2	Debugging GeoNode in the Browser	47
7.3	Debugging GeoServer	47

8	GeoNode's development process	49
8.1	GeoNode's Issue Tracking System	49
8.2	Testing in GeoNode	49
8.3	GeoNode's Patch Review Process	49
8.4	GeoNode Improvement Proposals	49
8.5	GeoNode's Roadmap Process	49
8.6	Development Resources	49

Welcome to the GeoNode Developers Workshop! This workshop will teach how to develop with and for the [GeoNode](#) software application.

Introduction to GeoNode development Learn about GeoNode's core components, its Architecture, the tools it is developed with and the standards it supports.

Development Prerequisites and Core Modules Learn about the pre-requisites you will need in order to develop with GeoNode. Take a look at its core modules and how they work together to provide a complete web mapping tool.

Customized GeoNode Projects Learn how existing projects leverage GeoNode and create your own GeoNode based project.

Setting up a GeoNode development environment Learn how to set up a GeoNode development environment so you can contribute to GeoNode's core.

Loading Data into a GeoNode Learn how to load data into a GeoNode with GeoServer, on the command line or programatically with scripts.

GeoNode APIs Learn about the APIs GeoNode leverages and provides.

GeoNode debugging techniques Learn how to debug GeoNode instances and projects.

GeoNode's development process Learn about GeoNode's development process and how to work with the GeoNode community.

INTRODUCTION TO GEONODE DEVELOPMENT

This module will introduce you to the components that GeoNode is built with, the standards that it supports and the services it provides based on those standards, and an overview its architecture.

GeoNode is a web based GIS tool, and as such, in order to do development on GeoNode itself or to integrate it into your own application, you should be familiar with basic web development concepts as well as with general GIS concepts.

A set of reference links on these topics is included at the end of this module.

1.1 GeoNode Components

GeoNode's architecture is based on a set of core tools and libraries that provide the building blocks on which the application is built. Having a basic understanding of each of these components is critical to your success as developer working with GeoNode.

Lets look at each of these components and discuss how they are used within the GeoNode application.

1.1.1 Django

GeoNode is based on [Django](#) which is a high level Python web development framework that encourages rapid development and clean pragmatic design. Django is based on the Model View Controller ([MVC](#)) architecture pattern, and as such, GeoNode models layers, maps and other modules with Django's [Model](#) module and and these models are used via Django's [ORM](#) in views which contain the business logic of the GeoNode application and are used to drive HTML templates to display the web pages within the application.

1.1.2 GeoServer

[GeoServer](#) is a an open source software server written in Java that provides OGC compliant services which publish data from many spatial data sources. GeoServer is used as the core GIS component inside GeoNode and is used to render the layers in a GeoNode instance, create map tiles from the layers, provide for downloading those layers in various formats and to allow for transactional editing of those layers.

1.1.3 GeoExplorer

[GeoExplorer](#) is a web application, based on the [GeoExt](#) framework, for composing and publishing web maps with OGC and other web based GIS Services. GeoExplorer is used inside GeoNode to provide many of the GIS and

cartography functions that are a core part of the application.

1.1.4 PostgreSQL and PostGIS

PostgreSQL and PostGIS are the database components that store and manage spatial data and information for GeoNode and the django modules that it is composed of, pycsw and GeoServer. All of these tables and data are stored within a geonode database in PostgreSQL. GeoServer uses PostGIS to store and manage spatial vector data for each layer which are stored as a separate table in the database.

1.1.5 pycsw

pycsw is an OGC CSW server implementation written in Python. GeoNode uses pycsw to provide an OGC compliant standards-based CSW metadata and catalogue component of spatial data infrastructures, supporting popular geospatial metadata standards such as Dublin Core, ISO 19115, FGDC and DIF.

1.1.6 Geospatial Python Libraries

GeoNode leverages several geospatial python libraries including gsconfig and OWSLib. gsconfig is used to communicate with GeoServer's REST Configuration API to configure GeoNode layers in GeoServer. OWSLib is used to communicate with GeoServer's OGC services and can be used to communicate with other OGC services.

1.1.7 Django Pluggables

GeoNode uses a set of Django plugins which are usually referred to as pluggables. Each of these pluggables provides a particular set of functionality inside the application from things like Registration and Profiles to interactivity with external sites. Being based on Django enables GeoNode to take advantage of the large ecosystem of these pluggables out there, and while a specific set is included in GeoNode itself, many more are available for use in applications based on GeoNode.

1.1.8 jQuery

jQuery is a feature-rich javascript library that is used within GeoNode to provide an interactive and responsive user interface as part of the application. GeoNode uses several jQuery plugins to provide specific pieces of functionality, and the GeoNode development team often adds new features to the interface by adding additional plugins.

1.1.9 Bootstrap

Bootstrap is a front-end framework for laying out and styling the pages that make up the GeoNode application. It is designed to ensure that the pages render and look and behave the same across all browsers. GeoNode customizes bootstraps default style and its relatively easy for developers to customize their own GeoNode based site using existing Bootstrap themes or by customizing the styles directly.

1.2 Standards

GeoNode is based on a set of Open Geospatial Consortium (OGC) standards. These standards enable GeoNode installations to be interoperable with a wide variety of tools that support these OGC standards and enable federation

with other OGC compliant services and infrastructure. Reference links about these standards are also included at the end of this module.

GeoNode is also based on Web Standards ...

1.2.1 Open Geospatial Consortium (OGC) Standards

Web Map Service (WMS)

The Web Map Service (WMS) specification defines an interface for requesting rendered map images across the web. It is used within GeoNode to display maps in the pages of the site and in the GeoExplorer application to display rendered layers based on default or custom styles.

Web Feature Service (WFS)

The Web Feature Service (WFS) specification defines an interface for reading and writing geographic features across the web. It is used within GeoNode to enable downloading of vector layers in various formats and within GeoExplorer to enable editing of Vector Layers that are stored in a GeoNode.

Web Coverage Service (WCS)

The Web Coverage Service (WCS) specification defines an interface for reading and writing geospatial raster data as “coverages” across the web. It is used within GeoNode to enable downloading of raster layers in various formats.

Catalogue Service for Web (CSW)

The Catalogue Service for Web (CSW) specification defines an interface for exposing a catalogue of geospatial meta-data across the web. It is used within GeoNode to enable any application to search GeoNode’s catalogue or to provide federated search that includes a set of GeoNode layers within another application.

Tile Mapping Service (TMS/WMTS)

The Tile Mapping Service (TMS) specification defines an interface for retrieving rendered map tiles over the web. It is used within GeoNode to enable serving of a cache of rendered layers to be included in GeoNode’s web pages or within the GeoExplorer mapping application. Its purpose is to improve performance on the client vs asking the WMS for rendered images directly.

1.2.2 Web Standards

HTML

CSS

REST

1.3 GeoNode Architecture

1.3.1 Django Architecture

- MVC
- WSGI

1.3.2 GeoNode and GeoServer

- Configuration via the REST API
- Authentication and Authorization

1.3.3 GeoNode and PostgreSQL/PostGIS

- Configuration and Application Information
- Vector Data Layer Storage

1.3.4 GeoNode and pycsw

GeoNode is built with pycsw embedded as the default CSW server component.

Publishing

Since pycsw is embedded in GeoNode, layers published within GeoNode are automatically published to pycsw and discoverable via CSW. No additional configuration or actions are required to publish layers, maps or documents to pycsw.

Discovery

GeoNode's CSW endpoint is deployed available at `http://localhost:8000/catalogue/csw` and is available for clients to use for standards-based discovery. See <http://pycsw.org/docs/tools.html> for a list of CSW clients and tools.

1.3.5 Javascript in GeoNode

- GeoExplorer
- jQuery Functionality

1.4 Development References

1.4.1 Basic Web based GIS Concepts and Background

- OGC Services
 - <http://www.opengeospatial.org/>
 - http://en.wikipedia.org/wiki/Open_Geospatial_Consortium
- Web Application Architecture
 - http://en.wikipedia.org/wiki/Web_application
 - <http://www.w3.org/2001/tag/2010/05/WebApps.html>
 - <http://www.amazon.com/Web-Application-Architecture-Principles-Protocols/dp/047051860X>
- AJAX and REST
 - [http://en.wikipedia.org/wiki/Ajax_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming))
 - http://en.wikipedia.org/wiki/Representational_state_transfer
- OpenGeo Suite
 - <http://workshops.opengeo.org/suiteintro/>
 - <http://suite.opengeo.org/opengeo-docs/>
- GeoServer Administration
 - <http://suite.opengeo.org/opengeo-docs/geoserver/>
 - <https://docs.google.com/a/opengeo.org/presentation/d/15fvUDYg0TO6WGFQIMLM2J1qiTVBYpfjCp0aQBDT0GrM/edit#>
 - <http://suite.opengeo.org/docs/sysadmin/index.html#sysadmin>
- PostgreSQL and PostGIS Administration - <http://workshops.opengeo.org/postgis-intro/> - <http://workshops.opengeo.org/postgis-spatialdbtips/>

1.4.2 Core development tools and libraries

- python
 - <http://docs.python.org/2/tutorial/>
 - <http://www.learnpython.org/>
 - <http://learnpythonthehardway.org/book/>
- django
 - <https://docs.djangoproject.com/en/dev/intro/tutorial01/>
 - <https://code.djangoproject.com/wiki/Tutorials>
- javascript
 - <http://www.crockford.com/javascript/inheritance.html>
 - <http://geoext.org/tutorials/quickstart.html>
- jquery
 - <http://www.w3schools.com/jquery/default.asp>

- http://docs.jquery.com/Tutorials:Getting_Started_with_jQuery
 - <http://www.jquery-tutorial.net/>
- bootstrap
 - <http://twitter.github.com/bootstrap/>
 - <http://www.w3resource.com/twitter-bootstrap/tutorial.php>
- geotools/geoscript/geoserver
 - <http://docs.geotools.org/stable/tutorials/feature/csv2shp.html>
 - <http://geoscript.org/tutorials/index.html>
 - <http://docs.geotools.org/stable/tutorials/>
 - <https://github.com/dwins/gsconfig.py/blob/master/README.rst>
- geopython
 - <http://pycsw.org/docs/documentation.html>
 - <http://geopython.github.com/OWSLib/>
 - <https://github.com/toblerity/shapely>
 - <https://github.com/sgillies/Fiona>
 - <http://pypi.python.org/pypi/pyproj>
- gdal/ogr
 - http://www.gdal.org/gdal_utilities.html
 - http://www.gdal.org/ogr_utilities.html

DEVELOPMENT PREREQUISITES AND CORE MODULES

This module will introduce you to the

2.1 GeoNode's Development Prerequisites

2.1.1 Basic Shell Tools

ssh and sudo

ssh and sudo are very basic terminal skills which you will need to deploy, maintain and develop with GeoNode. If you are not already familiar with their usage, you should review the basic descriptions below and follow the external links to learn more about how to use them effectively as part of your development workflow.

ssh is the network protocol used to connect to a remote server where you run your GeoNode instance whether on your own network or on the cloud. You will need to know how to use the ssh command from the terminal on your unix machine or how to use a ssh client like putty or winscp on windows. You may need to use pki certificates to connect to your remote server, and should be familiar with the steps and options necessary to connect this way. More information about ssh can be found in the links below.

- <http://winscp.net/eng/docs/ssh>

sudo is the command used to execute a terminal command as the superuser when you are logged in with a normal user. You will use sudo in order to start, stop and restart key services on your GeoNode instance. If you are not able to grant yourself these privileges on the machine you are using for your GeoNode instance, you may need to consult with your network administrator to arrange for your user to be granted sudo permissions. More information about sudo can be found in the links below.

- <http://en.wikipedia.org/wiki/Sudo>

bash

Bash is the most common unix shell which will usually be the default on servers where you will be deploying your GeoNode instance. You should be familiar with the most common bash commands in order to be able to deploy, maintain and modify a geonode instance. More information about Bash and common bash commands can be found in the links below.

- [http://en.wikipedia.org/wiki/Bash_\(Unix_shell\)](http://en.wikipedia.org/wiki/Bash_(Unix_shell))

apt

apt is the packaging tool that is used to install GeoNode on ubuntu and other debian based systems. You will need to be familiar with adding Personal Package Archives to your list of install sources, and will need to be familiar with basic apt commands. More information about apt can be found in the links below.

- http://en.wikipedia.org/wiki/Advanced_Packaging_Tool

2.1.2 Python Development Tools

The GeoNode development process relies on several widely used python development tools in order to make things easier for developers and other users of the systems that GeoNode developers work on or where GeoNodes are deployed. They are considered best practices for modern python development, and you should become familiar with these basic tools and be comfortable using them on your own projects and systems.

virtualenv

virtualenv is a tool used to create isolated python development environments such that the the versions of project dependencies are sandboxed from the system-wide python packages. This eliminates the commonly encountered problem of different projects on the same system using different versions of the same library. You should be familiar with how to create and activate virtual environments for the projects you work on. More information about virtualenv can be found in the links below.

- <http://pypi.python.org/pypi/virtualenv>
- <http://www.virtualenv.org/en/latest/>

virtualenvwrapper is a wrapper around the virtualenv package that makes it easier to create and switch between virtual environments as you do development. Using it will make your life much easier, so its recommended that you install and configure it and use its commands as part of your virtualenv workflow. More info about virtualenvwrapper can be found in the links below.

- <http://www.doughellmann.com/projects/virtualenvwrapper/>

pip

pip is a tool for installing and managing python packages. Specifically it is used to install and upgrade packages found in the Python Package Index. GeoNode uses pip to install itself, and to manage all of the python dependencies that are needed as part of a GeoNode instance. As you learn to add new modules to your geonode, you will need to become familiar with the use of pip and about basic python packaging usage. More information about pip can be found in the links below.

- <http://www.pip-installer.org/en/latest/>
- <http://pypi.python.org/pypi/pip>
- [http://en.wikipedia.org/wiki/Pip_\(Python\)](http://en.wikipedia.org/wiki/Pip_(Python))

miscellaneous

ipython is a set of tools to make your python development and debugging experience easier. The primary tool you want to use is an interactive shell that adds introspection, integrated help and command completion and more. While not strictly required to do GeoNode development, learning how to use ipython will make your development more productive and pleasant. More information about ipython can be found in the links below.

- <http://ipython.org/>
- <http://pypi.python.org/pypi/ipython>
- <https://github.com/ipython/ipython>
- <http://en.wikipedia.org/wiki/IPython>

pdb is a standard python module that is used to interactively debug your python code. It supports setting conditional breakpoints so you can step through the code line by line and inspect your variables and perform arbitrary execution of statements. Learning how to effectively use *pdb* will make the process of debugging your application code significantly easier. More information about *pdb* can be found in the links below.

- <http://docs.python.org/2/library/pdb.html>

2.1.3 Django

GeoNode is built on top of the *Django web framework*, and as such, you will need to become generally familiar with Django itself in order to become a productive GeoNode developer. Django has excellent documentation, and you should familiarize yourself with Django by following the Django workshop and reading through its documentation as required.

Model Template View

Django is based on the Model Template View paradigm (more commonly called Model View Controller). Models are used to define objects that you use in your application and Django's ORM is used to map these models to a database. Views are used to implement the business logic of your application and provide objects and other context for the templates. Templates are used to render the context from views into a page for display to the user. You should become familiar with this common paradigm used in most modern web frameworks, and how it is specifically implemented and used in Django. The Django tutorial itself is a great place to start. More information about MTV in Django can be found in the links below.

- <http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
- <http://www.codinghorror.com/blog/2008/05/understanding-model-view-controller.html>
- <https://docs.djangoproject.com/en/1.4/>

HTTP Request Response

Django and all other web frameworks are based on the HTTP Request Response cycle. Requests come in to the server from remote clients which are primarily web browsers, but also can be api clients, and the server returns with a Response. You should be familiar with these very basic HTTP principles and become familiar with the way that Django implements them. More information about HTTP, Requests and Responses and Django's implementation in the links below.

- <http://devhub.fm/http-requestresponse-basics/>
- http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- <https://docs.djangoproject.com/en/dev/ref/request-response/>

Management Commands

Django projects have access to a set of management commands that are used to manage your project. Django itself provides a set of these commands, and django apps (including GeoNode) can provide their own. Management commands are used to do things like synchronize your models with your database, load data from fixtures or back up your

database with fixtures, start the development server, initiate the debugger and many other things. GeoNode provides management commands for synchronizing with a GeoServer or updating the layers already in your GeoNode. You should become familiar with the basic management commands that come with Django, and specifically with the commands that are part of GeoNode. The GeoNode specific commands are covered in section. More information about management commands can be found in the links below.

- <https://docs.djangoproject.com/en/dev/ref/django-admin/>

Django Admin Interface

Django provides a build-in management console that administrators and developers can use to look at the data in the database that is part of the installed applications. Administrators can use this console to perform many common administration tasks that are a necessary part of running a GeoNode instance, and as a developer, you will use this interface during your development process to inspect the database and the data stored in your models. More information about the django admin interface can be found in the links below.

- <https://docs.djangoproject.com/en/dev/ref/contrib/admin/>

Template Tags

Django templates make use of a set of tags to inject, filter and format content into a rendered HTML page. Django itself includes a set of built-in template tags and filters that you will use in your own templates, and GeoNode provides a geonode specific set of tags that are used in the GeoNode templates. You should become familiar with the built-in tag set and with GeoNode's specific tags as you work on developing your own templates or extending from GeoNode's. More information about Django template tags can be found in the links below.

- <https://docs.djangoproject.com/en/dev/ref/templates/builtins/>

2.2 GeoNode's Core Modules

GeoNode is made up of a set of core Django pluggable modules (known as apps in Django) that provide the functionality of the application. Together they make up the key components of a GeoNode site. While your own use case and implementation may not require that you work directly on these modules, it is important that you become familiar with their layout, structure and the functionality that they provide. You may need to import these apps into your own apps, and as such, becoming familiar with them is an important step in becoming a proficient GeoNode developer.

2.2.1 geonode.layers

geonode.layers is the most key GeoNode module. It is used to represent layers of data stored in a GeoNode's paired GeoServer. The layer model class inherits fields from the ResourceBase class which provides all of the fields necessary for the metadata catalogue, and adds fields that map the object to its corresponding layer in GeoServer. When your users upload a layer via the user interface, the layer is imported to GeoServer and a record is added to GeoNode's database to represent that GeoServer layer within GeoNode itself.

The Layer model class provides a set of helper methods that are used to perform operations on a Layer object, and also to return things such as the list of Download or Metadata links for that layer. Additional classes are used to model the layers Attributes, Styles, Contacts and Links. The Django signals framework is used to invoke specific functions to synchronize with GeoServer before and after the layer is saved.

The views in the layers app are used to perform functions such as uploading, replacing, removing or changing the points of contact for a layer, and views are also used to update layer styles, download layers in bulk or change a layers permissions.

The forms module in the layer app is used to drive the user interface forms necessary for performing the business logic that the views provide.

The Layers app also includes a set of templates that are paired with views and used to drive the user interface. A small set of layer template tags is also used to help drive the layer explore and search pages.

Some helper modules such as `geonode.layers.metadata` and `geonode.layers.ows` are used by the layer views to perform specific functions and help keep the main views module more concise and legible.

Additionally, the GeoNode specific management commands are a part of the `geonode.layers` app.

You should spend some time to review the layers app through GitHub's code browsing interface.

<https://github.com/GeoNode/geonode/tree/master/geonode/layers>

2.2.2 geonode.maps

The `geonode.maps` app is used to group together GeoNodes multi layer map functionality. The `Map` and `MapLayer` objects are used to model and implement maps created with the `GeoExplorer` application. The `Map` object also extends from the `ResourceBase` class which provides the ability to manage a full set of metadata fields for a `Map`.

The views in the maps app perform many of the same functions as the views in the layers app such as adding, changing, replacing or removing a map and also provide the endpoints for returning the map configuration from the database that is used to initialize the `GeoExplorer` app.

The maps app also includes a set of forms, customization of the Django admin, some utility functions and a set of templates and template tags.

You can familiarize yourself with the maps app on GitHub.

<https://github.com/GeoNode/geonode/tree/master/geonode/layers>

2.2.3 geonode.security

The `geonode.security` app is used to provide object level permissions within the GeoNode Django application. It is a custom Django authentication backend and is used to assign Generic, User and Group Permissions to Layers, Maps and other objects in the GeoNode system. Generic permissions are used to enable public anonymous or authenticated viewing and/or editing of your data layers and maps, and User and Group specific permissions are used to allow specific users or groups to access and edit your layers.

2.2.4 geonode.search

The `geonode.search` module provides the search API that is used to drive the GeoNode search pages. It is configured to index layers, maps, documents and profiles, but is extensible to allow you to use it to index your own model classes. This module is currently based on the Django ORM and as such has a limited set of search features, but the GeoNode development team is actively working on making it possible to use this module with more feature-rich search engines.

2.2.5 geonode.catalogue

The `geonode.catalogue` app provides a key set of metadata catalogue functions within GeoNode itself. GeoNode is configured to use an integrated version of the `pycsw` library to perform these functions, but can also be configured to use any OGC compliant CS-W implementation such as `GeoNetwork` or `Deegree`. The metadata app allows users to import and/or edit metadata for their layers, maps and documents, and it provides an OGC compliant search interface for use in federating with other systems.

2.2.6 geonode.geoserver

The `geonode.geoserver` module is used to interact with GeoServer from within GeoNode's python code. It relies heavily on the `gsconfig` library which addresses GeoServer's REST configuration API. Additionally, the `geonode.geoserver.uploader` module is used to interact with GeoServers Importer API for uploading and configuring layers.

2.2.7 geonode.people

The `geonode.people` module is used to model and store information about both GeoNode users and people outside of the system who are listed as Points of Contact for particular layers. It is the foundational module for GeoNode's social features. It provides a set of forms for users to edit and manage their own profiles as well as to view and interact with the profiles of other users.

2.2.8 geoexplorer

GeoNode's core GIS client functions are performed by GeoExplorer. The GeoExplorer app is in turn based on GeoExt, OpenLayers and ExtJS. It provides functionality for constructing maps, styling layers and connecting to remote services. GeoExplorer is the reference implementation of the OpenGeo Suite SDK which is based on GXP. GeoNode treats GeoExplorer as an external module that is used out of the box in GeoNode, but it is possible for you to create your own Suite SDK app and integrate it with GeoNode.

2.2.9 Static Site

The front end of GeoNode is composed of a set of core templates, specific templates for each module, cascading style sheets to style those pages and a set of javascript modules that provide the interactive functionality in the site.

Templates

GeoNode includes a basic set of core templates that use Django's template inheritance system to provide a modular system for constructing the web pages in GeoNode's interface. These core templates drive the overall page layout and things like the home page. You will start the process of customizing your GeoNode instance by overriding these templates, so you should familiarize yourself with their tructure and how they inherit from each other to drive the pages.

Additionally, most of the apps described above have their own set of templates that are used to drive the pages for each module. You may also want to override these templates for your own purposes and as such should familiarize yourself with a few of the key ones.

CSS

GeoNode's css is based on Twitter's Bootstrap Library which uses the lessc dynamic stylesheet language. GeoNode extends from the basic Bootstrap style and you are able to create your own bootstrap based style to customize the look and feel of your own GeoNode instance. Sites like bootswatch.com also provide ready made styles that you can simply drop in to your project to change the style.

Javascript

The interactive functionality in GeoNode pages is provided by the jQuery javascript framework and a set of jQuery plugins. The core set of GeoNode javascript modules closely aligns with the apps described above, and there are also

a few pieces of functionality provided as javascript modules that are used through out all of the apps. You are able to add your own jQuery code and/or plugins to perform interactive functionality in your own application.

CUSTOMIZED GEONODE PROJECTS

This module will teach you about how to setup your own GeoNode based project and how to customize your project by changing the theme, adding additional modules and how to integrate your project with other systems. When complete, you should understand how Downstream GeoNode projects work, and how to setup your own for your own use cases and needs.

3.1 Introduction to GeoNode Projects

GeoNode enables you to setup a complete site by simply by installing the packages and adding your data. If instead you want to create your own project based on GeoNode, there are a several options available to you that enable you to customize the look and feel of your GeoNode site, to add additional modules that are necessary for your own use case and to integrate your GeoNode project with other external sites and services.

This module assumes that you have installed a GeoNode site with the Ubuntu Packages and that you have a working GeoNode based on that setup. If you want to follow this same methodology on a different platform or setup, you should follow this module and adapt as necessary for your desired deployment environment.

3.1.1 Overview

GeoNode by itself is an out of the box, full featured Spatial Data Infrastructure Solution, but many GeoNode implementations require either customization of the default site, or the use of additional modules whether they be third-party Django Pluggables or modules developed by a GeoNode implementor himself. There are quite a few existing Downstream GeoNode projects some of which follow the methodology described in this module. You should familiarize yourself with these projects and how and why they extend GeoNode. You should also carefully think about what customization and additional modules you need for your own GeoNode based project and research the options that are available to you. The [Django Packages](#) site is a great place to start looking for existing modules that may meet your needs.

3.1.2 Existing Downstream GeoNode Projects

- Harvard Worldmap
- MapStory
- Risiko/SAFE

3.1.3 Django Template Projects

GeoNode follows the django template projects paradigm introduced in Django 1.4. At a minimum, a django project consists of a settings.py file and a urls.py file and django apps are used to add specific pieces of functionality. The GeoNode development team has created a template project which contains these required files with all the GeoNode configuration you need to get up and running with your own GeoNode project. If you would like learn more about Django projects and Apps, you should consult the [Django Documentation](#)

3.2 Setting up your GeoNode Project

This section will walk you through the steps necessary to setup your own GeoNode project. It assumes that you have installed GeoNode from the ubuntu packages and that you have a working GeoNode site.

3.2.1 Setup Steps

If you are working remotely, You should first connect to the machine that has your GeoNode installed on it. You will need to perform the following steps in a directory where you intend to keep your newly created project.

1. Activate GeoNode's Virtual Environment:

```
$ source /var/lib/geonode/bin/activate
```

2. Create your GeoNode project from the Template:

```
$ django-admin.py startproject --template=https://github.com/GeoNode/geonode-project/zipball/master my_geonode
$ cd my_geonode
```

3. Update your local_settings.py:

You will need to check the local_settings.py that is included with the template project and be sure that it reflects your own local environment. You should pay particular attention to the Database settings especially if you intend to reuse the database that was setup with your base GeoNode installation.

4. Synchronize your Database:

```
$ python manage.py syncdb --all
```

5. Run the test server:

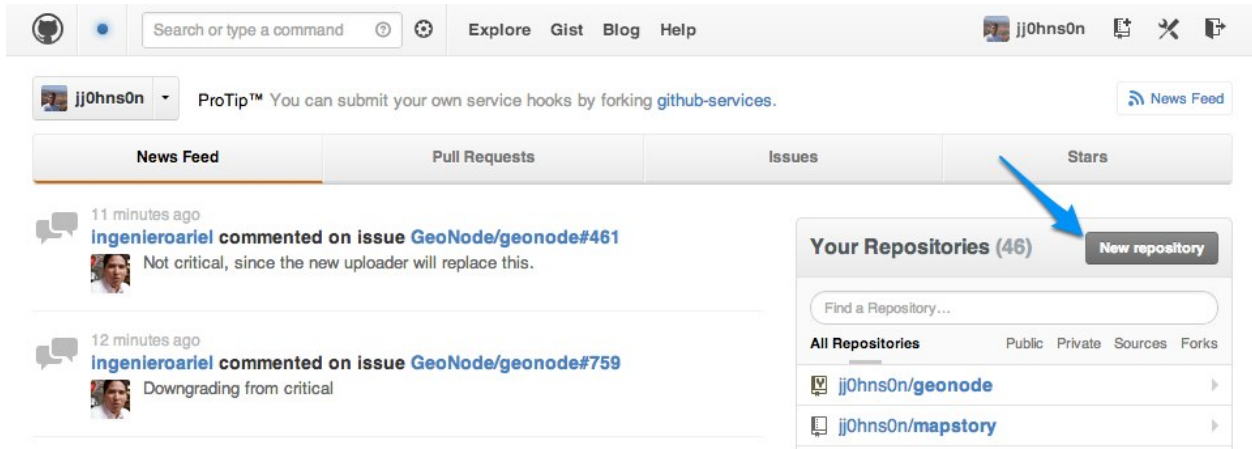
```
$ python manage.py runserver
```

6. Visit your new GeoNode Site.

<http://localhost:8000>

3.2.2 Source Code Revision Control

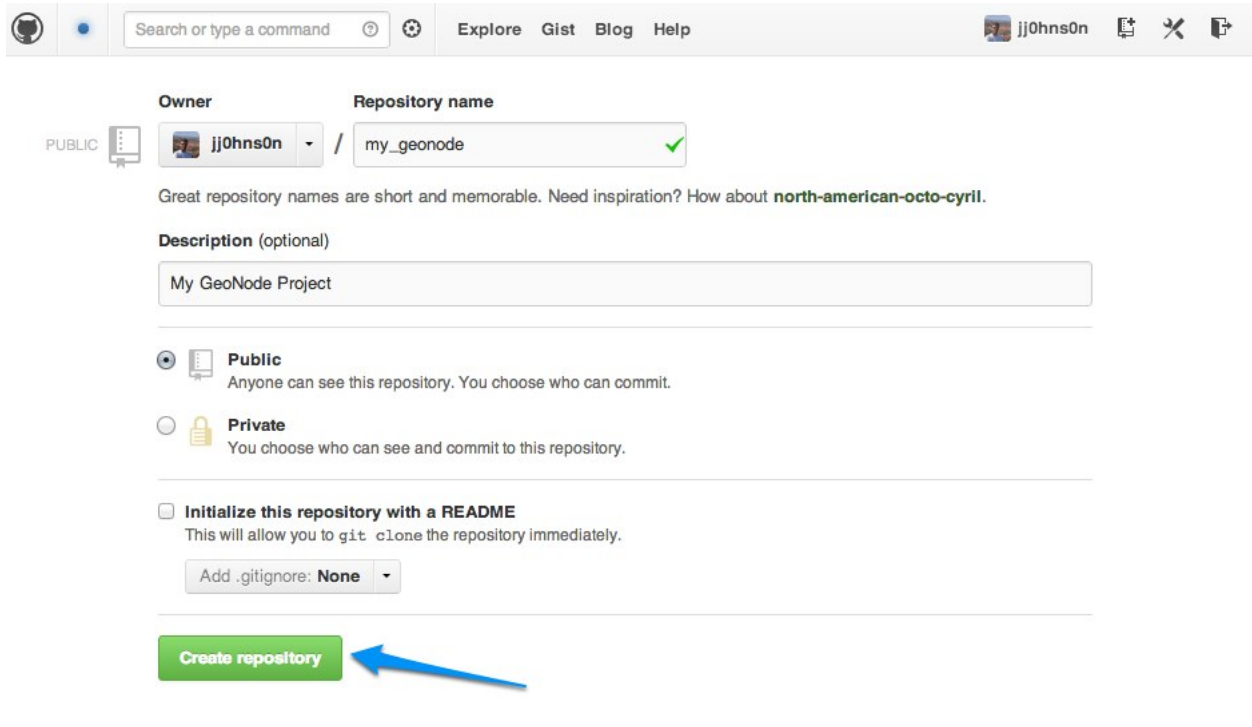
It is recommended that you immediately put your new project under source code revision control. The GeoNode development team uses Git and GitHub purpose and recommends that you do the same. If you do not already have a GitHub account, it is recommended that you set one up. A full review of Git and distributed Source Code Revision Control systems is beyond the scope of this tutorial, but you may find the [Git Book](#) useful if you are not already familiar with these concepts.



1. Create a new Repository in GitHub:

You should use the GitHub user interface to create a new repository for your new project.

Creating a new GitHub Repository From GitHub's Homepage



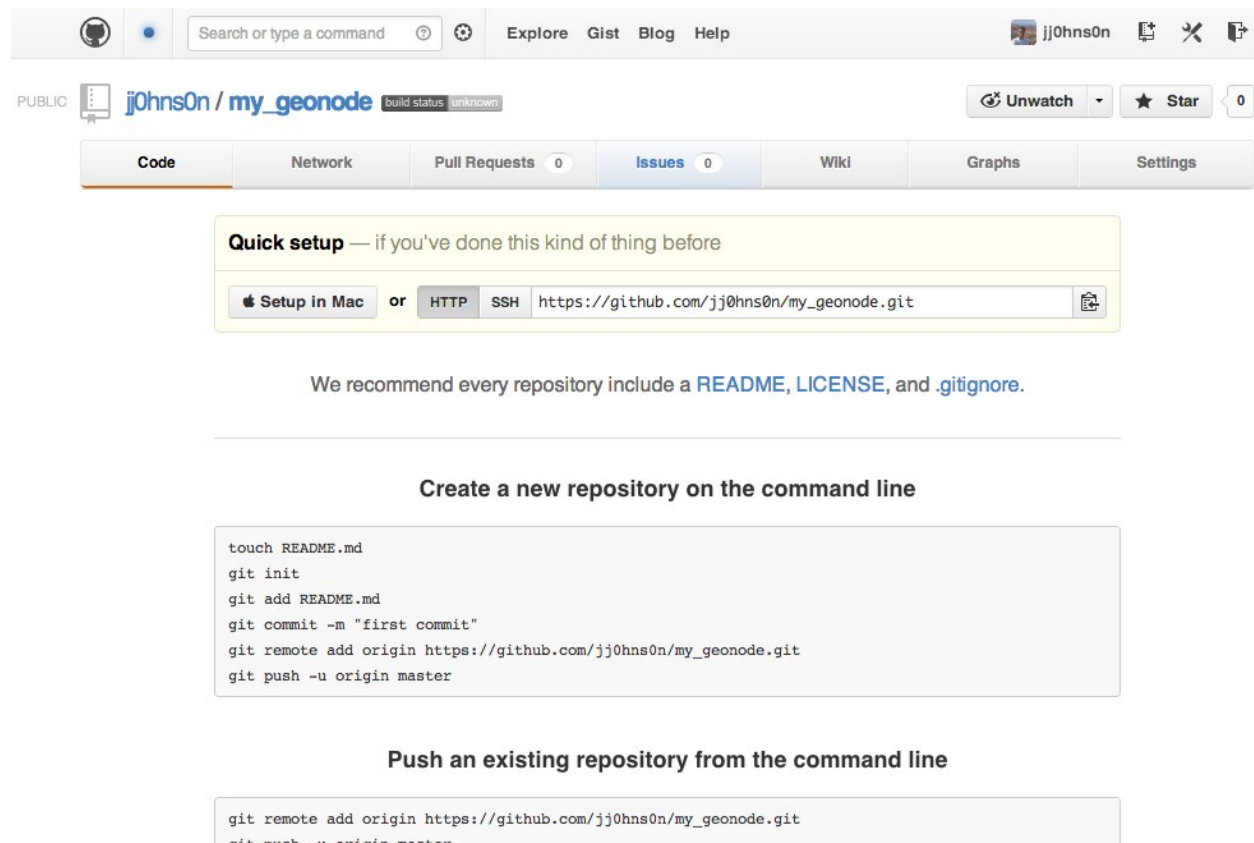
Specifying new GitHub Repository Parameters

Your new Empty GitHub Repository

2. Initialize your own repository:

```
$ git init
```

3. Add the remote repository reference to your local git configuration:



```
$ git remote add
```

4. Add your project files to the repository:

```
$ git add .
```

5. Commit your changes:

```
$ git commit -am "Initial commit"
```

6. Push to the remote repository:

```
$ git push origin master
```

3.2.3 Your GeoNode Project's Struture

Your GeoNode project will now be structured as depicted below:

```
|-- README.rst
|-- manage.py
|-- my_geonode
|   |-- __init__.py
|   |-- settings.py
|   |-- static
|   |   |-- README
|   |   |-- css
|   |   |-- site_base.css
```



```
| | | -- img
| | | -- README
| | | -- js
| | | -- README
| | -- templates
| | | -- site_base.html
| | | -- site_index.html
| | -- urls.py
| | -- wsgi.py
|-- setup.py
```

You can also view your project on github.

The screenshot shows the GitHub interface for the repository 'my_geonode' by user 'jj0hns0n'. The repository has 0 pull requests, 0 issues, 0 stars, and 0 forks. The 'Code' tab is selected, showing the repository's clone options (Mac, ZIP, HTTP, SSH, Git Read-Only) and the repository URL: `git@github.com:jj0hns0n/my_geonode.git`. The 'Files' tab is also visible, showing the initial commit structure with files: `my_geonode`, `README.rst`, `manage.py`, and `setup.py`, all added 'just now' by 'jj0hns0n'.

Each of the key files in your project are described below.

`manage.py`

`settings.py`

`urls.py`

`wsgi.py`

`setup.py`

`static`

`templates`

3.2.4 Deploying your GeoNode Project

Now that your own project is setup, you will need to replace the existing default configuration with configuration for your own project in order to visit your new project site.

1. Update Apache Configuration
2. Check GeoServer Configuration
3. Check Database Configuration

3.2.5 Staying in Sync with Mainline GeoNode

One of the primary reasons that we setup your own GeoNode project using this method is so that you can stay in sync with mainline geonode as the core GeoNode development team makes new releases. Your own project should not be adversely affected by these upstream changes, but you will receive bug fixes and other improvements by staying in sync.

1. Upgrade GeoNode:

```
$ apt-get update
$ apt-get install geonode
```

2. Verify that your new project works with the upgraded GeoNode:

```
$ python manage.py runserver
```

Visit <http://localhost:8000/>

3.3 Theming your GeoNode Project

There are a range of options available to you if you want to change the default look and feel or theme of your GeoNode project. Since GeoNode's style is based on bootstrap you will be able to make use of all that bootstrap has to offer as you customize the theme of your GeoNode. You should consult bootstrap's documentation as your primary guide once you are familiar with how Geonode implements bootstrap and how you can override GeoNode's theme and templates in your own project.

3.3.1 Logos and Graphics

GeoNode intentionally does not include a large number of graphics files to drive its interface. This keeps browser loading times to a minimum and makes for a more responsive interface. That said, you are free to customize your GeoNode's interface by simply changing the default logo, or by adding your own images and graphics to deliver a GeoNode experience the way you envision it.

Your GeoNode project has a directory already setup for you to store your own images at `my_geonode/static/img`. You should place any image files that you intend to use for your project in this directory.

Let's walk through an example of the steps necessary to change the default logo.

1. Change directories into the `img` directory:

```
$ cd my_geonode/static/img
```

2. Grab your logo image from the web using `wget`. This is just an example, you will need to change this URL to match the location of your file or copy it to this location from somewhere else on disk or your network:

```
$ wget http://www2.sta.uwi.edu/~anikov/UWI-logo.JPG
$ cd ../../..
```

3. Override the CSS that displays the logo by editing `my_geonode/static/css/site_base.css` with your favorite editor and adding the following lines:

```
.nav-logo {
    width: 373px;
    height: 79px;
    background: url(../img/UWI-logo.JPG) no-repeat;
}
```

4. Restart your GeoNode project and look at the page in your browser:

```
$ python manage.py runserver
```

Visit your site at <http://localhost:8000/> or the remote URL for your site.

You can see that the header has been expanded to fit your graphic. In the following sections you will learn how to customize this header to make it look and work the way you want.

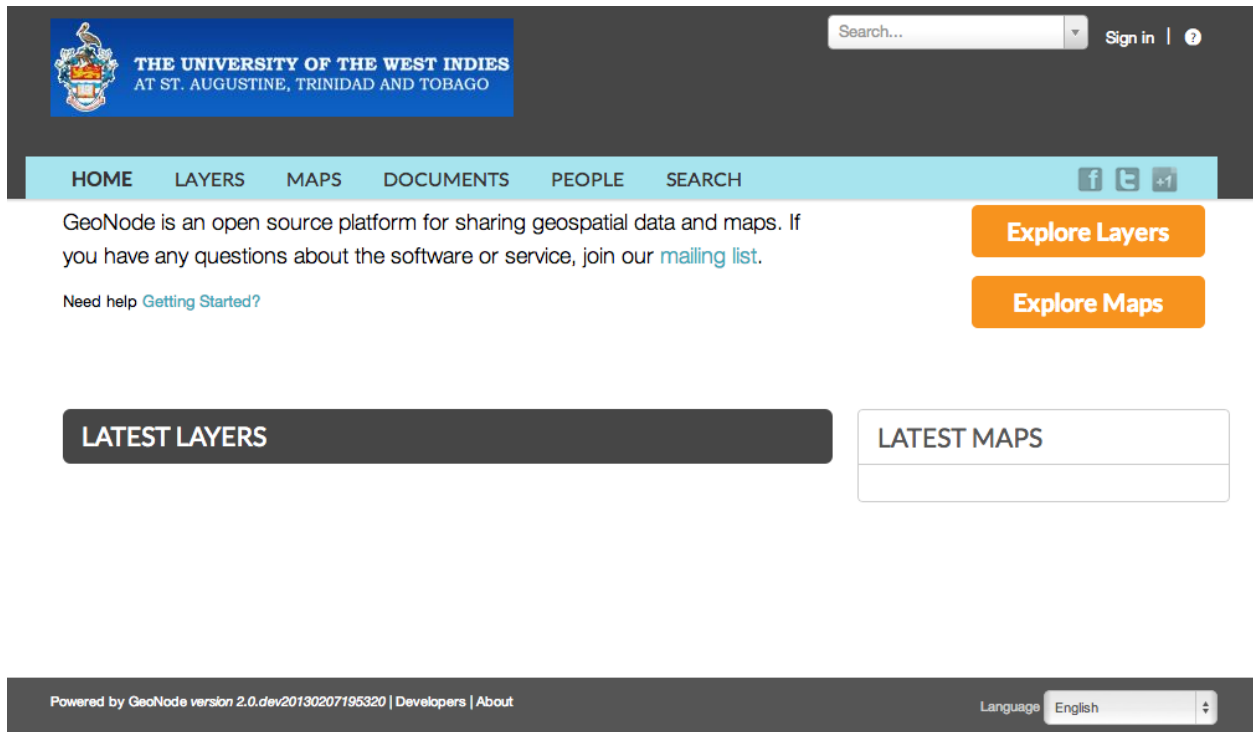
Note: You should commit these changes to your git repository as you progress through this section, and get in the habit of committing early and often so that you and others can track your project on github. Making many atomic commits and staying in sync with a remote repository (i.e. on GitHub) makes it easier to collaborate with others on your project.

3.3.2 Cascading Style Sheets

In the last section you already learned how to override GeoNode's default CSS rules to include your own logo. You are able to customize any aspect of GeoNode's appearance this way. In the last screenshot, you saw that the main Hero area in the homepage is now covered up by the expanded header.

First, let's walk through the steps necessary to displace it downward so it is no longer hidden, then let's change the background color of the header to match the color in our logo graphic.

1. Reopen `my_geonode/static/css/site_base.css` in your editor and add the following rule after the one we added in the previous step:



```
.content-wrap {  
    margin: 75px 75px;  
}
```

2. Add a rule to change the background color of the header to match the logo graphic we used:

```
.navbar .navbar-inner {  
    background: #0e60c3;  
}
```

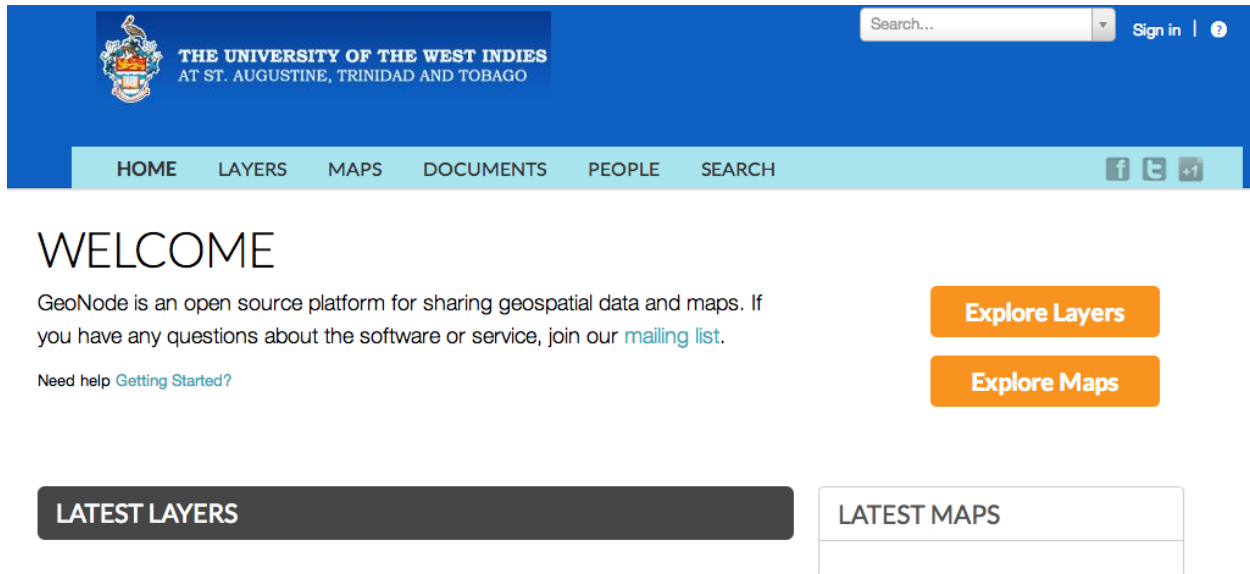
3. Your projects css file should now look like this:

```
.nav-logo {  
    width: 373px;  
    height: 79px;  
    background: url(../img/UWI-logo.JPG) no-repeat;  
}  
  
.content-wrap {  
    margin: 75px 75px;  
}  
  
.navbar .navbar-inner {  
    background: #0e60c3;  
}
```

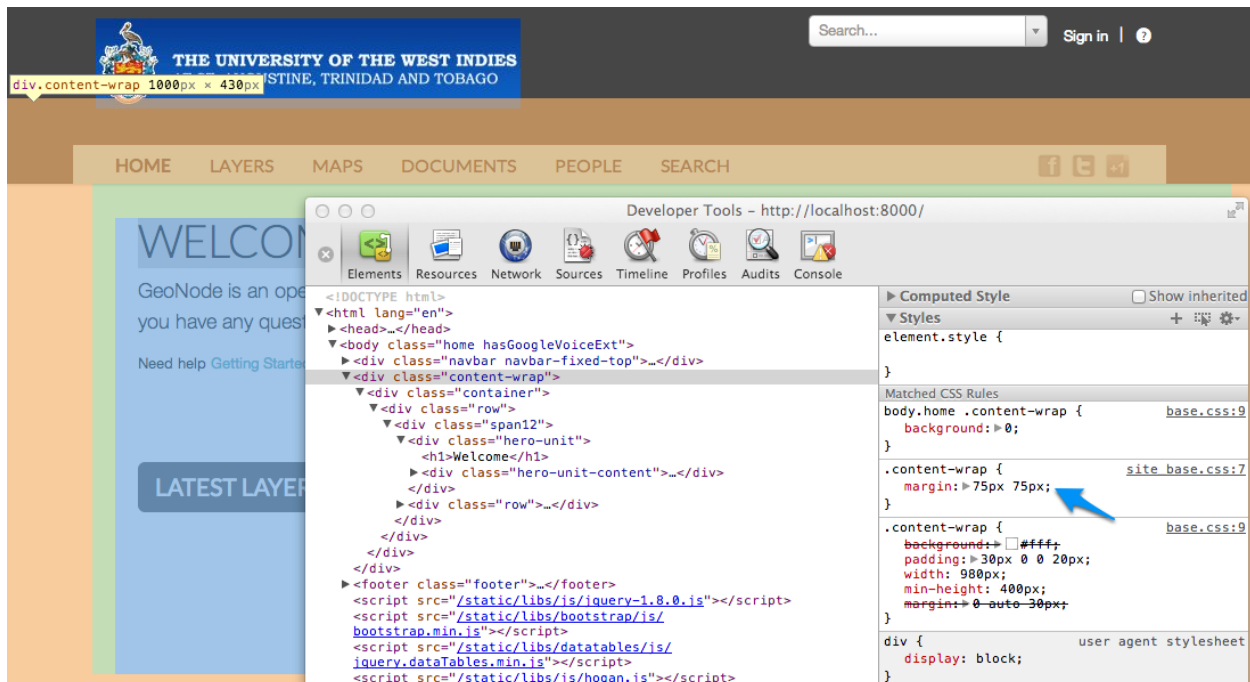
4. Restart the development server and reload the page:

```
$ python manage.py runserver
```

Your GeoNode projects homepage should now look like this



Note: You can continue adding rules to this file to override the styles that are in geonodes base css file which is built from base.less. <https://github.com/GeoNode/geonode/blob/master/geonode/static/geonode/less/base.less> You may find it helpful to use your browsers development tools to inspect elements of your site that you want to override to determine which rules are already applied. You can see an example of that in the screenshot below. Another section of this workshop covers this topic in much more detail.



Screenshot of using Chrome's debugger to inspect the css overrides added in the previous steps.

3.3.3 Templates and Static Pages

Now that we have changed the default logo and adjusted our main content area to fit the expanded header, its time to update the content of the homepage itself. Your GeoNode project includes 2 basic templates that you will use to change the content of your pages.

site_base.html is the basic template that all other templates inherit from and you will use it to update things like the header and navbar, site wide announcement, the footer and also to include your own javascript or other static things that are included in every page in your site. Its worth taking a look at GeoNode's base file for this on the GitHub site here. <https://github.com/GeoNode/geonode/blob/master/geonode/templates/base.html> You have several blocks available to you to for overriding, but since we will be revisiting this file in future sections of this workshop lets just look at it for now and leave it unmodified.

Open my_geonode/templates/site_base.html in your editor:

```
{% extends "base.html" %}
{% block extra_head %}
    <link href="{{ STATIC_URL }}css/site_base.css" rel="stylesheet"/>
{% endblock %}
```

You will see that it extends from base.html which is the GeoNode template referenced above and it currently only overrides the extra_head block to include our projects site_base.css which we modified in the previous section. You can see on line 14 of GeoNodes base.html (<https://github.com/GeoNode/geonode/blob/master/geonode/templates/base.html#L14>) template that this block is included in an empty state and is setup specifically for you to include extra css files as your project is already setup to do.

Now that we have looked at site_base.html, lets actually override a different template.

site_index.html is the template used to define for your GeoNode projects homepage. It extends GeoNode's default index.html template and gives you the option to override specific areas of the homepage like the hero area, but also allows you leave things like the Latest Layers and Maps and the Contribute section as they are if you want to keep them. You are of course free to override these sections if you choose and this section shows you the steps necessary to do that below.

1. Open my_geonode/templates/site_index.html in your editor
2. Edit the h1 element on line 13 to say something other than "Welcome":

```
<h1>{% trans "UWI GeoNode" %}</h1>
```

3. Edit the intro paragraph to include something specific about your GeoNode project:

```
<p>
    {% blocktrans %}
        UWI's GeoNode is setup for students and faculty to collaboratively
        create and share maps for their class projects. It is maintained by the
        UWI Geographical Society.
    {% endblocktrans %}
</p>
```

4. Change the Getting Started link to point to another website::

```
<span>
    For more information about the UWI Geographical society,
    <a href="http://uwigsmona.weebly.com/">visit our website</a>
</span>
```

5. Add a Graphic to the hero area above the paragraph we replaced in step 3:

```
<img src = 'http://uwigsmona.weebly.com/uploads/1/3/2/4/13241997/1345164334.png'>
```

6. Your edited `site_index.html` file should now look like this:

```
{% extends 'index.html' %}
{% load i18n %}
{% load maps_tags %}
{% load layers_tags %}
{% load pagination_tags %}
{% load staticfiles %}
{% load url from future %}
{% comment %}
This is where you can override the hero area block. You can simply modify the content below or r
{% endcomment %}
{% block hero %}
    <div class="hero-unit">
        <h1>{% trans "UWI GeoNode" %}</h1>
        <div class="hero-unit-content">
            <div class="intro">
                <img src = 'http://uwigsmona.weebly.com/uploads/1/3/2/4/13241997/1345164334.png'>
            <p>
                {% blocktrans %}
                UWI's GeoNode is setup for students and faculty to collaboratively
                create and share maps for their class projects. It is maintained by the
                UWI Geographical Society.
                {% endblocktrans %}
            </p>
            <span>
                For more information about the UWI Geographical society,
                <a href="http://uwigsmona.weebly.com/">visit our website</a>
            </span>
        </div>
        <div class="btns">
            <a class="btn btn-large" href="{% url "layer_browse" %}">
                {% trans "Explore Layers" %}
            </a>
            <a class="btn btn-large" href="{% url "maps_browse" %}">
                {% trans "Explore Maps" %}
            </a>
        </div>
    </div>
{% endblock %}
```

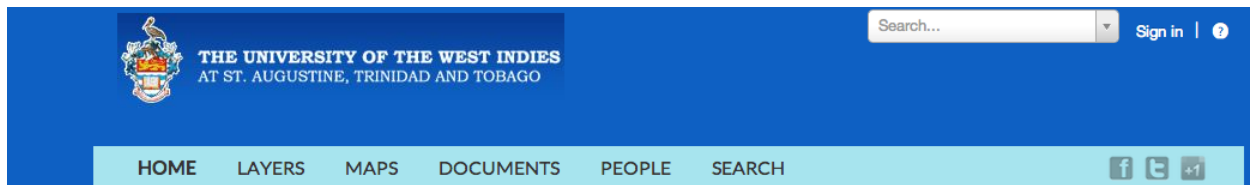
7. Restart your GeoNode project and look at the page in your browser:

```
$ python manage.py runserver
```

Visit your site at <http://localhost:8000/> or the remote URL for your site.

You can see that the homepage now includes our changes.

From here you can continue to customize your `site_index.html` template to suit your needs. This workshop will also cover how you can add new pages to your GeoNode project site.



UWI GEONODE

[Explore Layers](#)[Explore Maps](#)

UWI's GeoNode is setup for students and faculty to collaboratively create and share maps for their class projects. It is maintained by the UWI Geographical Society.

For more information about the UWI Geographical society, [visit our website](#)

3.3.4 Other Theming Options

You are able to change any specific piece of your GeoNode project's style by adding css rules to your site_base.css, but since GeoNode is based on Bootstrap, there are many pre-defined themes that you can simply drop into your project to get a whole new look. This is very similar to wordpress themes and is a powerful and easy way to change the look of your site without much effort.

Bootswatch

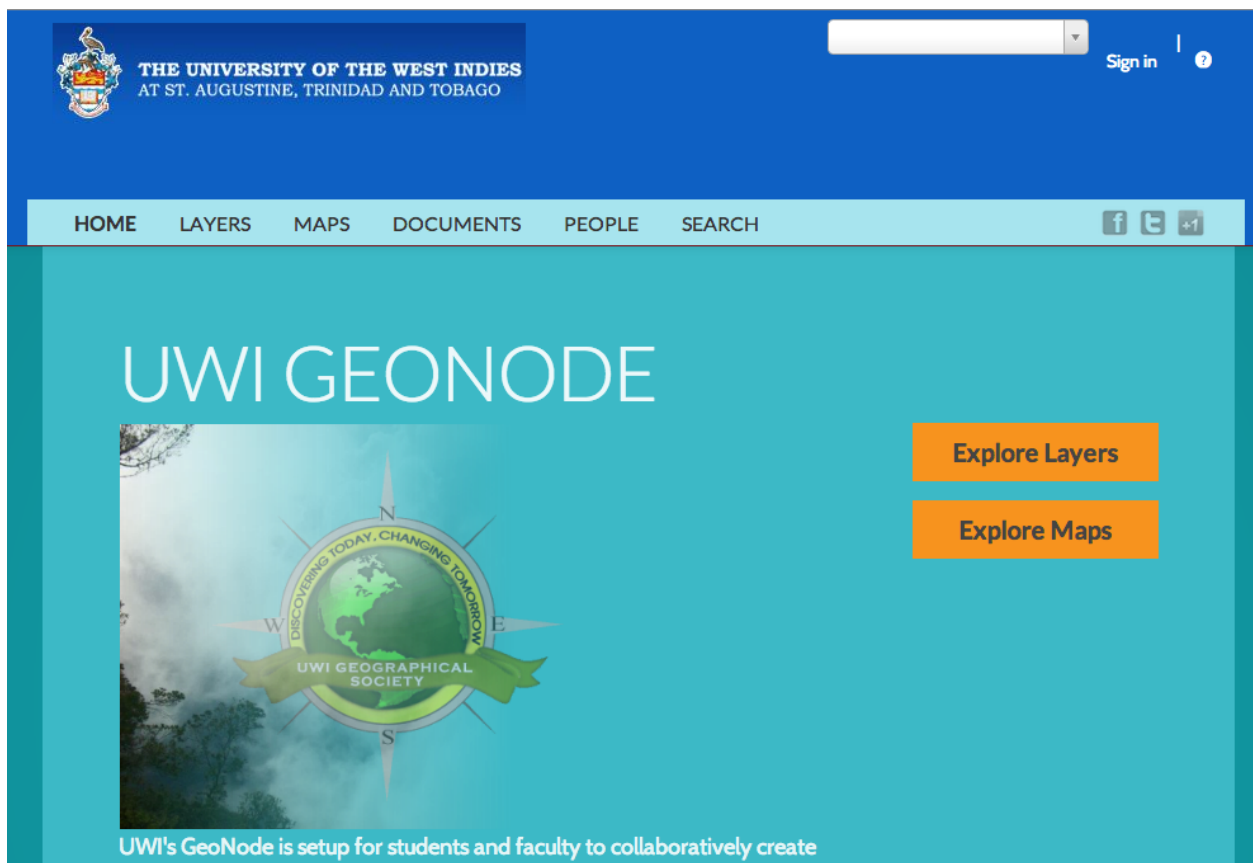
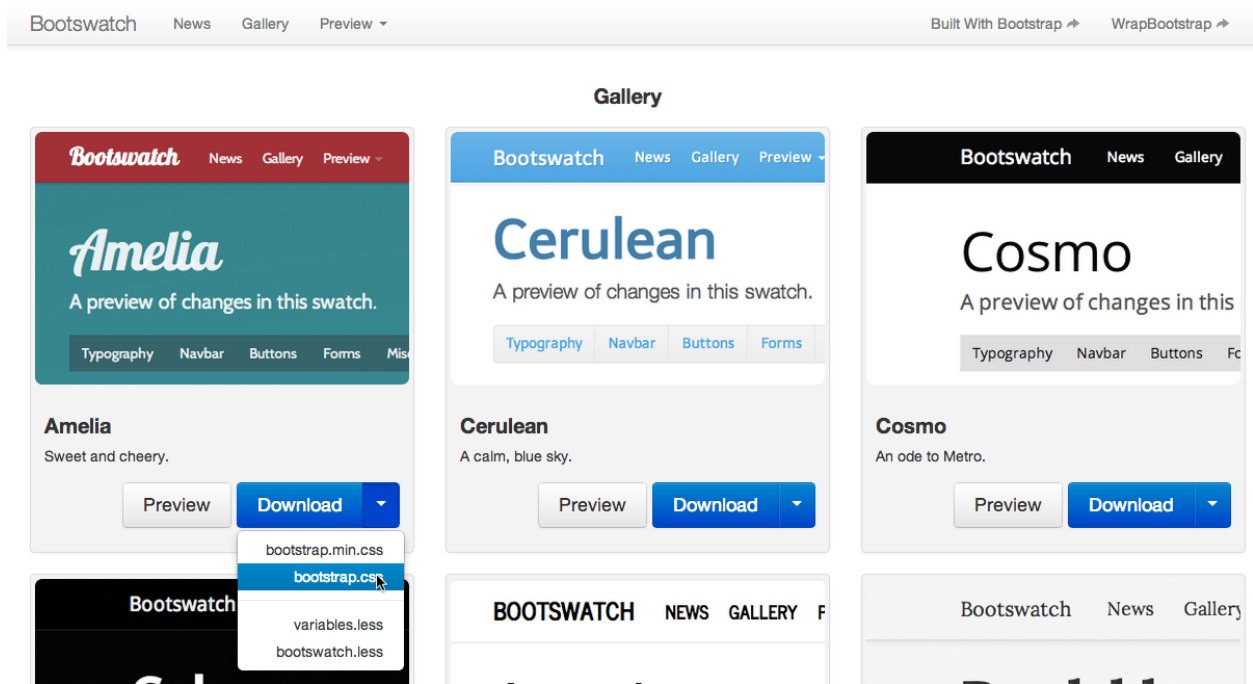
bootswatch.com is a site where you can download ready to use themes for your GeoNode project site. The following steps will show you how to use a theme from bootswatch in your own geonode site.

1. Visit bootswatch.com and select a theme (we will use Amelia for this example). Select the download bootstrap.css option in the menu:

1. Put this file in my_geonode/static/css
2. Update the site_base.html template to include this file. It should now look like this:

```
{% extends "base.html" %}
{% block extra_head %}
    <link href="{{ STATIC_URL }}css/site_base.css" rel="stylesheet"/>
    <link href="{{ STATIC_URL }}css/bootstrap.css" rel="stylesheet"/>
{% endblock %}
```

3. Restart the development server and visit your site:



Your GeoNode project site is now using the Amelia theme in addition to the changes you have made.

3.4 Adding Additional Django apps to your GeoNode Project

Since GeoNode is based on Django, your GeoNode project can be augmented and enhanced by adding additional third-party pluggable Django apps or by writing an app of your own.

This section of the workshop will introduce you to the Django pluggable app ecosystem, and walk you through the process of writing your own app and adding a blog app to your project.

3.4.1 Intro to Django Pluggable Apps

The Django app ecosystem provides a large number of apps that can be added to your project. Many are mature and used in many existing projects and sites, while others are under active early-stage development. Websites such as Django Packages provide an interface for discovering and comparing all the apps that can be plugged in to your Django project. You will find that some can be used with very little effort on your part, and some will take more effort to integrate.

<http://www.djangopackages.com/>

3.4.2 Adding your own Django App

Lets walk through the an example of the set of steps necessary to create a very basic django polling app to and add it to your GeoNode project. This section is an abridged version of the Django tutorial itself and it is strongly recommended that you go through this external tutorial along with this section as it provides much more background material and a significantly higher level of detail. You should become familiar with all of the information in the Django Tutorial as it is critical to your success as a GeoNode project developer.

Throughout this section, we will walk through the creation of a basic poll application.

It'll consist of two parts:

- A public site that lets people view polls and vote in them.
- An admin site that lets you add, change and delete polls.

Since we have already created our GeoNode project from a template project, we will start by creating our app structure and then adding models:

```
$ python manage.py startapp polls
```

That'll create a directory polls, which is laid out like this:

```
polls/  
  __init__.py  
  models.py  
  tests.py  
  views.py
```

This directory structure will house the poll application.

The first step in writing a database Web app in Django is to define your models – essentially, your database layout, with additional metadata.

In our simple poll app, we'll create two models: polls and choices. A poll has a question and a publication date. A choice has two fields: the text of the choice and a vote tally. Each choice is associated with a poll.

These concepts are represented by simple Python classes.

Edit the `polls/models.py` file so it looks like this:

```
from django.db import models

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    def __unicode__(self):
        return self.question

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice = models.CharField(max_length=200)
    votes = models.IntegerField()
    def __unicode__(self):
        return self.choice
```

That small bit of model code gives Django a lot of information. With it, Django is able to:

- Create a database schema (CREATE TABLE statements) for this app.
- Create a Python database-access API for accessing Poll and Choice objects.
- But first we need to tell our project that the polls app is installed.

Edit the `my_geonode/settings.py` file, and update the `INSTALLED_APPS` setting to include the string `'polls'`. So it'll look like this:

```
INSTALLED_APPS = (

    # Apps bundled with Django
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.admin',
    'django.contrib.sitemaps',
    'django.contrib.staticfiles',
    'django.contrib.messages',
    'django.contrib.humanize',

    #Third party apps

    # <snip>

    # GeoNode internal apps
    'geonode.maps',
    'geonode.upload',
    'geonode.layers',
    'geonode.people',
    'geonode.proxy',
    'geonode.security',
    'geonode.search',
    'geonode.catalogue',
    'geonode.documents',

    # My GeoNode apps
    'polls',

)
```

Now Django knows to include the polls app. Let's run another command:

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

The syncdb command runs the SQL from sqlall on your database for all apps in INSTALLED_APPS that don't already exist in your database. This creates all the tables, initial data and indexes for any apps you've added to your project since the last time you ran syncdb. syncdb can be called as often as you like, and it will only ever create the tables that don't exist.

GeoNode uses south for migrations ...

Next, let's add the Django admin configuration for our polls app so that we can use the Django Admin to manage the records in our database. Edit a new file called polls/admin.py and make it look like the this:

```
from polls.models import Poll
from django.contrib import admin

admin.site.register(Poll)
```

Run the development server and explore the polls app in the Django Admin by pointing your browser to <http://localhost:8000/admin/> and logging in with the credentials you specified when you first ran syncdb.

Django administration Welcome, admin. [Change password](#) / [Log out](#)

Site administration

Account	
Account deletions	+ Add Change
Accounts	+ Add Change
Signup codes	+ Add Change
Actstream	
Actions	+ Add Change
Follows	+ Add Change
Announcements	
Announcements	+ Add Change
Auth	
Groups	+ Add Change
Users	+ Add Change
Avatar	
Avatars	+ Add Change
Dialogos	
Comments	+ Add Change
Documents	
Contact roles	+ Add Change
Documents	+ Add Change
Layers	
Attributes	+ Add Change
Contact roles	+ Add Change
Layers	+ Add Change

Recent Actions
My Actions
None available

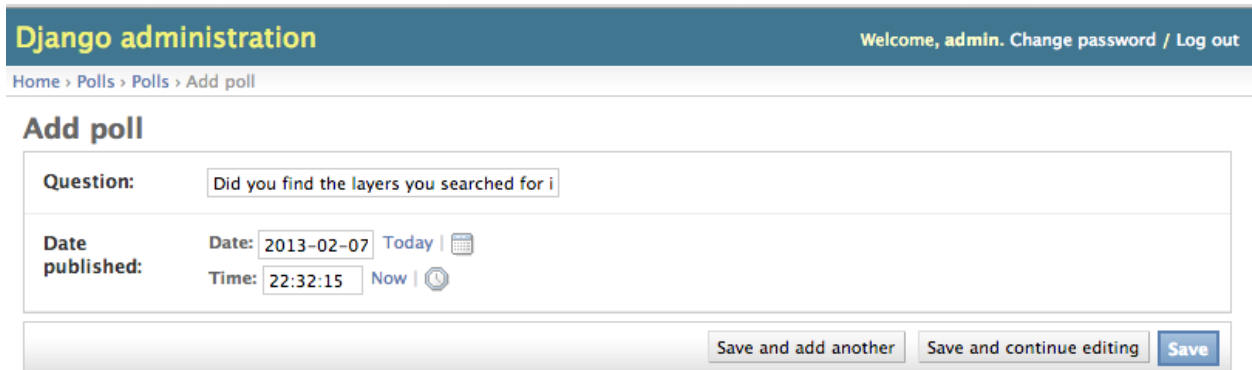
You can see all of the other apps that are installed as part of your GeoNode project, but we are specifically interested in our Polls app for now.

Maps		
Map layers	+ Add	✎ Change
Maps	+ Add	✎ Change
People		
Profiles	+ Add	✎ Change
Roles	+ Add	✎ Change
Polls		
Polls	+ Add	✎ Change
Relationships		
Relationship statuses	+ Add	✎ Change
Request		
Requests	+ Add	✎ Change
Security		
Generic object role mappings	+ Add	✎ Change
Object roles	+ Add	✎ Change
User object role mappings	+ Add	✎ Change
Sites		
Sites	+ Add	✎ Change
Taggit		
Tags	+ Add	✎ Change
Upload		
Upload files	+ Add	✎ Change
Uploads	+ Add	✎ Change

Next we will add a new poll via automatically generated admin form.



You can enter any sort of question you want for initial testing and select today and now for the publication date.



The next step is to configure the Choice model in the admin, but we will configure the choices to be editable inline with the Poll objects they are attached to. Edit the same polls/admin.py so it now looks like the following:

```
from polls.models import Poll, Choice
from django.contrib import admin

class ChoiceInline(admin.StackedInline):
    model = Choice
    extra = 3

class PollAdmin(admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['question']}),
        ('Date information', {'fields': ['pub_date'], 'classes': ['collapse']}),
    ]
    inlines = [ChoiceInline]

admin.site.register(Poll, PollAdmin)
```

This tells Django: “Choice objects are edited on the Poll admin page. By default, provide enough fields for 3 choices.”

You can now return to the Poll admin and either add a new poll or edit the one you already created and see that you can now specify the poll choices inline with the poll itself..

From here, we want to create views to display the polls inside our GeoNode project. A view is a “type” of Web page in your Django application that generally serves a specific function and has a specific template. In our poll application, we’ll have the following four views:

- Poll “index” page – displays the latest few polls.
- Poll “detail” page – displays a poll question, with no results but with a form to vote.

Django administration

Welcome, admin. Change password / Log out

Home > Polls > Polls > Did you find the layers you searched for in my GeoNode

Change poll

History

Question:

Did you find the layers you searched for i

Date information (Show)

Choices

Choice: #1

Choice:

Yes

Votes:

Choice: #2

Choice:

No

Votes:

Choice: #3

Choice:

Some, but not all

Votes:

+ Add another Choice

✖ Delete

Save and add another

Save and continue editing

Save

- Poll “results” page – displays results for a particular poll.
- Vote action – handles voting for a particular choice in a particular poll.

The first step of writing views is to design your URL structure. You do this by creating a Python module, called a URLconf. URLconfs are how Django associates a given URL with given Python code.

Lets start by adding our url configuration directly to the `urls.py` that already exists in your project `my_geonode/urls.py`. Edit this file and add the following lines after the rest of the existing imports around line 80:

```
url(r'^polls/$', 'polls.views.index'),
url(r'^polls/(?P<poll_id>\d+)/$', 'polls.views.detail'),
url(r'^polls/(?P<poll_id>\d+)/results/$', 'polls.views.results'),
url(r'^polls/(?P<poll_id>\d+)/vote/$', 'polls.views.vote'),
```

Note: Eventually we will want to move this set of url configurations inside the `urls` app itself, but for the sake of brevity in this workshop, we will put them in the main `urls.py` for now. You can consult the Django tutorial for more information on this topic.

Next lets write the views to drive the url patterns we configured above. Edit `polls/views.py` to that it looks like the following:

```
from django.template import Context, loader
from polls.models import Poll
from django.http import HttpResponse
from django.http import Http404
from django.shortcuts import render_to_response

def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    return render_to_response('polls/index.html',
        RequestContext(request, {'latest_poll_list': latest_poll_list}))

def detail(request, poll_id):
    try:
        p = Poll.objects.get(pk=poll_id)
    except Poll.DoesNotExist:
        raise Http404
    return render_to_response('polls/detail.html', RequestContext(request, {'poll': p}))

def results(request, poll_id):
    return HttpResponse("You're looking at the results of poll %s." % poll_id)

def vote(request, poll_id):
    return HttpResponse("You're voting on poll %s." % poll_id)
```

Note: We have only stubbed in the views for the results and vote pages and they are not very useful as is. We will revisit these later.

Now we have views in place, but we are referencing templates that do not yet exist. Lets add them by first creating a template directory in your polls app as `polls/templates/polls` and creating `polls/templates/polls/index.html` to look like the following:

```
{% if latest_poll_list %}
<ul>
{% for poll in latest_poll_list %}
<li><a href="/polls/{{ poll.id }}">{{ poll.question }}</a></li>
```



```

    {% endfor %}
  </ul>
{% else %}
  <p>No polls are available.</p>
{% endif %}

```

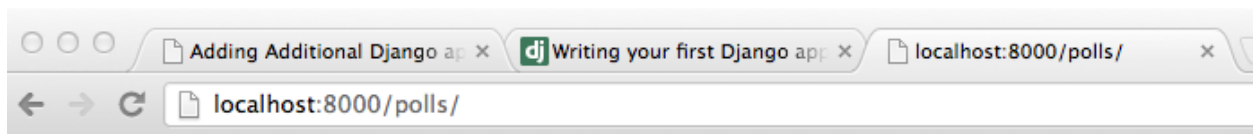
Next we need to create the template for the poll detail page. Create a new file at `polls/templates/polls/detail.html` to look like the following:

```

<h1>{{ poll.question }}</h1>
<ul>
{% for choice in poll.choice_set.all %}
  <li>{{ choice.choice }}</li>
{% endfor %}
</ul>

```

You can now visit <http://localhost:8000/polls/> in your browser and you should see the the poll question you created in the admin presented like this.



- [Did you find the layers you searched for in my GeoNode](#)

We actually really want our polls app to display as part of our GeoNode project with the same theme, so lets update the 2 templates we created above to make them extend from the `site_base.html` template we looked at in the last section. You will need to add the following 2 lines to the top of each file:

```

{% extends 'site_base.html' %}
{% block body %}

```

And close the block at the bottom of each file with:

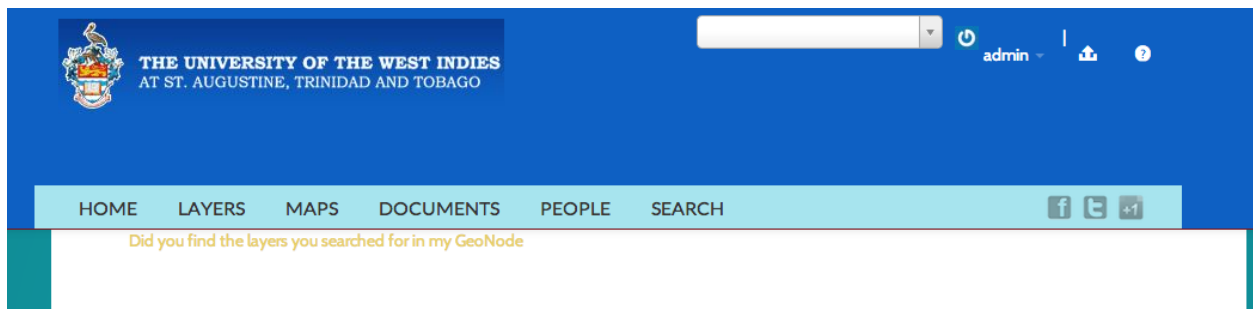
```

{% endblock %}

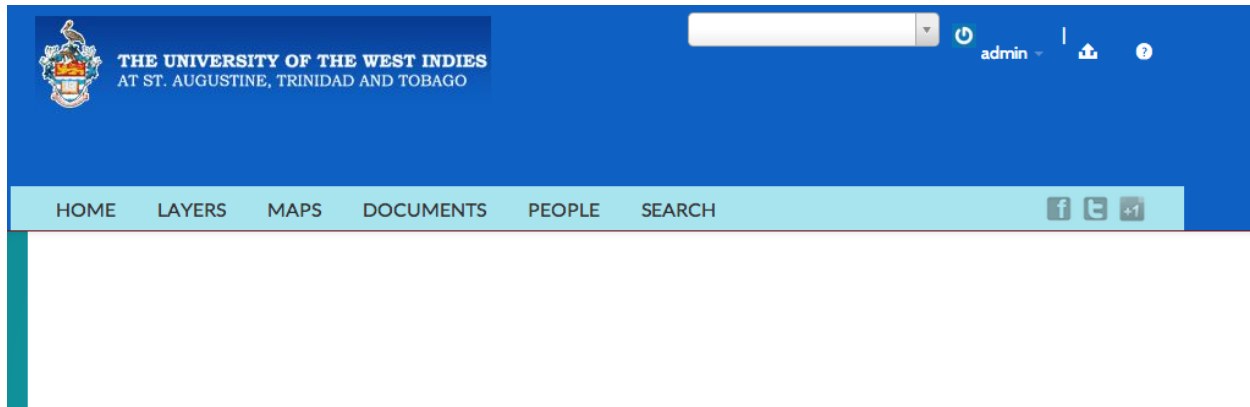
```

This tells Django to extend from the `site_base.html` template so your polls app has the same style as the rest of your GeoNode, and it specifies that the content in these templates should be rendered to the body block defined in GeoNode's `base.html` template that your `site_base.html` extends from.

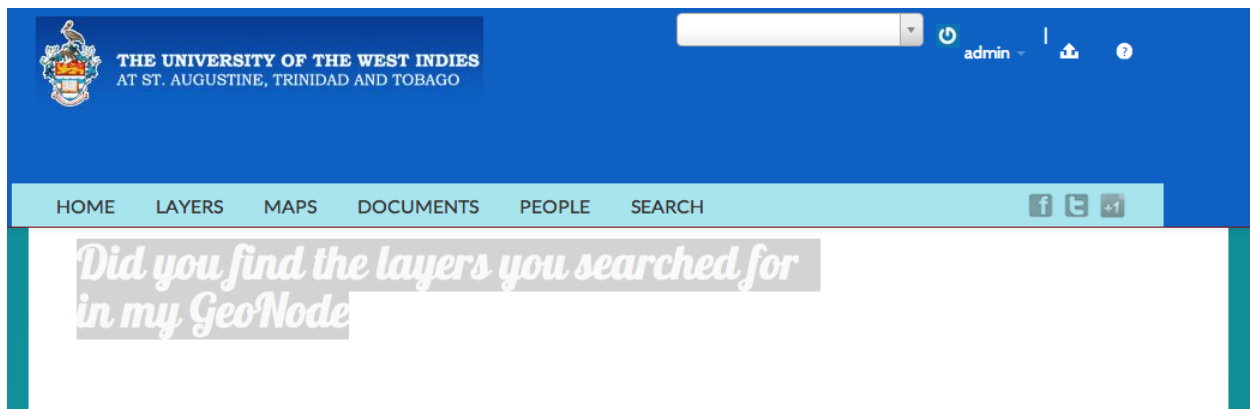
You can now visit the index page of your polls app and see that it is now wrapped in the same style as the rest of your geonode site.



If you click on a question from the list you will be taken to the poll detail page.



Looks like its empty, but in fact the text is there, but its been styled to white by the bootswatch theme we added in the last section. If you highlight the area where the text is, you will see that it is in fact there.



Now that you have walked through the basic steps to create a very minimal (and currently mostly useless) Django app and integrate it with your GeoNode project, you should pick up the Django tutorial at part 4 and follow it to add the form for actually accepting responses to your poll questions.

Again, its strongly recommended that you spend as much time as you need with the Django tutorial itself until you feel comfortable with all of the concepts it covers. They are the essential building blocks you will need to extend your GeoNode project by adding your own apps.

3.4.3 Adding a 3rd Party Blog App

3.4.4 Adding Other Apps

3.5 Integrating your Project with other Systems

3.5.1 Third Party Integration Overview

3.5.2 OGC Services

3.5.3 Google Earth

3.5.4 qGIS

3.5.5 OpenStreetMap

3.5.6 Wordpress

3.5.7 MapBox

SETTING UP A GEONODE DEVELOPMENT ENVIRONMENT

This module will lead you through the steps necessary to install a GeoNode development environment.

4.1 GeoNode Development Tools

4.2 Git Repository Setup

4.3 Installing GeoNode's Python Package

4.4 Pavement.py and Paver

4.5 Manually Deploying your Development Environment

LOADING DATA INTO A GEONODE

This module will walk you through the various options available to load data into your GeoNode from GeoServer, on the command-line or programatically. You can choose from among these techniques depending on what kind of data you have and how you have your geonode setup.

5.1 GeoServer Data Configuration

5.2 Using ogr2ogr to load data into GeoNode

5.3 Loading OSM Data into GeoNode

GEONODE APIS

6.1 OGC Services

6.2 GeoServer REST API

6.3 GeoServers Import and Print APIs

6.4 GeoNode's Ad-Hoc API

GEONODE DEBUGGING TECHNIQUES

7.1 Debugging GeoNode's Python Components

7.2 Debugging GeoNode in the Browser

7.3 Debugging GeoServer

GEONODE'S DEVELOPMENT PROCESS

8.1 GeoNode's Issue Tracking System

8.2 Testing in GeoNode

8.2.1 Unit Tests

8.2.2 Integration Tests

8.2.3 Javascript Tests

8.3 GeoNode's Patch Review Process

8.4 GeoNode Improvement Proposals

8.5 GeoNode's Roadmap Process

8.6 Development Resources