Tangent MicroServices Documentation

Release 1

Tangent Solutions

March 10, 2015
## Contents

1 Getting Started  
1.1 Micro Services Projects .......................... 3

2 Service Registry 

3 Writing a MicroService with Python (Django)  
3.1 Typical Project Layout .............................. 7  
3.2 Toolset ........................................... 7  
3.3 Project Setup ...................................... 7  
3.4 Build the Database ................................ 10  
3.5 Initial Data ....................................... 10  
3.6 Writing some Code ................................ 10  
3.7 Authentication .................................... 11  
3.8 Documenting ...................................... 11  
3.9 Testing ........................................... 12  
3.10 Continuous Integration with Jenkins .......... 12

4 Standards and Conventions  
4.1 Resource Naming .................................. 15  
4.2 Service Requirements ............................. 15

5 Indices and tables ................................. 17
Contents:
Getting Started

Every project within the MicroServices suite has an individual read me file with specific instructions on getting started with the project.

1.1 Micro Services Projects

- User Service
CHAPTER 2

Service Registry

- UserService
- HoursService
CHAPTER 3

Writing a MicroService with Python (Django)

3.1 Typical Project Layout

api
    migrations
        __init__.py
        _init_.py
    admin.py
    api.py
    models.py
    tests.py
    views.py

projectservice
    __init__.py
    settings.py
    urls.py
    wsgi.py

data
    initial.json
    .gitignore
    db.sqlite3
    manage.py
    README.md
    requirements.txt

3.2 Toolset

- Documentation: Sphinx + ReadTheDocs.
- Django Rest Framework

3.3 Project Setup

Instructions

1. Setup and activate your virtual environment
   
   virtualenv env
   source env/bin/activate
2. Create a requirements.txt file with the following and run it

   ```
django==1.7
gunicorn
requests
djangorestframework==3
django-rest-swagger
django-filter

## dev requirements
sphinx
sphinx_rtd_theme
mock
responses
ipdb
ipython

## Test and quality analysis
pylint
coverage
django-jenkins
django-extensions
django-cors-headers

## custom libs:
-e git://github.com/TangentMicroServices/PythonAuthenticationLib.git#egg=tokenauth
```

Run the requirements file using:

   ```
pip install -r requirements.txt
```

3. Create the python project

   ```
django-admin.py startproject projectservice .
```

Note:

- projectservice all lowercase
- note that . at the end: so it creates it in the current directory

4. Check that your structure is as follows:

   ```
LICENSE
README.md
manage.py
requirements.txt
projectservice
    __init__.py
    settings.py
    urls.py
    wsgi.py
```

5. Create an API app:

   ```
python manage.py startapp api
```

6. Create api.py in the api app:
touch api/api.py

7. Add the following to settings.py:

```python
# CUSTOM AUTH
AUTHENTICATION_BACKENDS = (  
    'django.contrib.auth.backends.ModelBackend',  
    'tokenauth.authbackends.TokenAuthBackend'  
)

## REST
REST_FRAMEWORK = {  
    'DEFAULT_PERMISSION_CLASSES': {  
        'rest_framework.permissions.IsAuthenticated',  
    },  
    'DEFAULT_AUTHENTICATION_CLASSES': {  
        ## we need this for the browsable API to work  
        'rest_framework.authentication.SessionAuthentication',  
        'tokenauth.authbackends.RESTTokenAuthBackend',  
    }  
}

# Services:

## Service base urls without a trailing slash:
USER_SERVICE_BASE_URL = 'http://staging.userservice.tangentme.com'

JENKINS_TASKS = {  
    'django_jenkins.tasks.run_pylint',  
    'django_jenkins.tasks.with_coverage',  
    # 'django_jenkins.tasks.run_sloccount',  
    # 'django_jenkins.tasks.run_graphmodels'
}

PROJECT_APPS = {  
    'api',
}

8. Update INSTALLED_APPS in settings.py:

INSTALLED_APPS = {  
    ...
    
    ## 3rd party  
    'rest_framework',  
    'rest_framework_swagger',
    
    ## custom  
    'tokenauth',  
    'api',
    
    # testing etc:  
    'django_jenkins',  
    'django_extensions',  
    'corsheaders',
}

9. Update MIDDLEWARE_CLASSES in settings.py:
MIDDLEWARE_CLASSES = (

    ## add this:
    'tokenauth.middleware.TokenAuthMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.middleware.common.CommonMiddleware',
)

Note: Note that CorsMiddleware needs to come before Django’s CommonMiddleware if you are using Django’s USE_ETAGS = True setting, otherwise the CORS headers will be lost from the 304 not-modified responses, causing errors in some browsers.

10. Update settings.py with the following setting at the bottom

    CORS_ORIGIN_ALLOW_ALL = True

3.4 Build the Database

1. Sync the database:

    python manage.py syncdb

Note: Make the username admin and password a by default

2. Perform any migrations if necessary:

    python manage.py makemigrations
    python manage.py migrate

3.5 Initial Data

1. Login to the admin panel and create some test data

2. Dump the data:

    python manage.py dumpdata > data/initial.json

3. Run the data to test that it works:

    python manage.py loaddata data/initial.json

3.6 Writing some Code

Create some end points using - Django REST Framework.

Note: To include a Swagger API explorer for your API. Add:

```
url(r'^api-explorer/', include('rest_framework_swagger.urls')),
```

to urls.py. for more info on using Swagger with Django Rest Framework, see:
Warning: The following code is for the hours service using entry. Rename accordingly.

1. In models.py add the following:

   ```python
   from django.contrib.auth.models import User
   ...
   class Entry(models.Model):
       user = models.ForeignKey(User)
       title = models.CharField(max_length=200)
   ```

2. In api.py add the following:

   ```python
   from rest_framework import viewsets, routers, serializers
   from rest_framework.decorators import detail_route
   from rest_framework.response import Response
   ...
   class EntryViewSet(viewsets.ModelViewSet):
       model = Entry
       serializer_class=EntrySerializer

       hours_router = routers.DefaultRouter()
       hours_router.register('entry', EntryViewSet)
   ```

3. In urls.py add the following:

   ```python
   from api.api import hours_router
   ...
   urlpatterns = patterns(''
       url(r'^$', include(hours_router.urls)),
   )
   ```

4. python manage.py runserver

### 3.7 Authentication

#### 3.8 Documenting

1. Build the documentation in Sphinx

   ```bash
   sphinx-quickstart
   ```

   This will create a folder called /docs and the structure should like this this:

   Makefile
   make.bat
   build/
   source/
   _static
   _templates
   conf.py
   index.rst

2. Add /docs/build/ to .gitignore file
3. Write your own documentation as you go - RST Docs.
4. Update the readme file with instructions on how to setup the project

**Warning:** The following code is for the hours service. Rename accordingly.

```python
# HoursService

[[Build Status](http://jenkins.tangentme.com/buildStatus/icon?job=Build HoursService)]

A Service for time tracking

## Setting Up

1. Start and activate environment
   ```
   Virtualenv env
   source env/bin/activate
   ```

1. Run the requirements
   ```
   pip install -r requirements.txt
   ```

1. Install the database
   ```
   python manage.py syncdb
   ```

1. Run the initial data (if required - this is test data only)
   ```
   python manage.py loaddata data/initial.json
   ```

1. Run the tests to ensure the project is up and running correctly
   ```
   python manage.py test
   ```

### 3.9 Testing

#### 3.9.1 Unit Tests

Uint tests can be run with:

```python
python manage.py test
``` 

#### 3.9.2 Integration Tests

Integration tests should be stored in files matching the pattern `*_ITCase.py`. They can be run with:

```python
python manage.py test --pattern="*_ITCase.py"
```

### 3.10 Continious Integration with Jenkins

Requirements
Instructions

1. Install requirements:
   - pip install -r requirements.txt
   - pip install pylint
   - pip install coverage
   - pip install django-jenkins
   - pip install django-extensions

2. Configure settings.py:

   ```python
   JENKINS_TASKS = (   
      'django_jenkins.tasks.run_pylint',
      'django_jenkins.tasks.with_coverage',
      # 'django_jenkins.tasks.run_sloccount',
      # 'django_jenkins.tasks.run_graphmodels'
   )

   ## Apps to run analysis over:
   PROJECT_APPS = (   
      'api',
   )
```

3. Run:
   - `./manage.py jenkins`

This will:

- Run tests (build junit report)
- Generate coverage report (cobertura)
- Run pylint (generate checkstyle report)

All files are generated in the `reports` directory
4.1 Resource Naming

4.2 Service Requirements

A MicroService is required to provide the following resources to be considered stable:

- A minimum of setup instructions so that a developer can check the project out, get it running and run the tests
- Up-to-date documentation on REST API
- Unit test coverage > 80%
- Integration tests against any other Services on which this service is dependent
- Default data that is generated on the staging version of the Service (for other services to integration test against).
  * This should be documented
Indices and tables

- genindex
- modindex
- search