Case: Integrating third parties

Introduction

For this case we'll make use of a common situation at Maykin Media: integrating third parties in a piece of software. These third parties expose data or API's which are used in the services that we offer our clients.

In this case, to keep the exercise small, an application has to be built that imports CSV hotel data. Of course CSV data is flat, while in the case of data models in Django one wants to make use of relations. So during the import of the data those relations have to be restored. Then in the end a small front-end application has to be built that allows users to lookup hotel data. See the exercise description for further details.

Environment

Use Python 3 and the latest version of Django.

Data

For the exercise you will need to work with two CSV files, in this case these are files that are updated every day and exposed through a webserver using authenticated HTTP:

URL : http://rachel.maykinmedia.nl/djangocase/city.csv and http://rachel.maykinmedia.nl/djangocase/hotel.csv
Username : python-demo
Password : claw30_bumps

The files are also attached alongside this document, should importing over authenticated HTTP not work out somehow.

Exercise

On the functional level there are a number of mandatory and optional parts in this exercise. Please have a look at the additional demands as well, which go into the technical demands.

Mandatory

- Make sure that all import data ends up in Django models and, while importing, make sure that relations between the different models are taken care of (a hotel is situated in a certain city)
- There needs to be importing logic in your application for daily imports, importing the data by hand is not an option. Try to import the CSV data over authenticated HTTP, if that doesn't work out: import the CSV files directly.
- Create a view/template that uses the imported data to:
    - Choose a city (any of those that are in the dataset)
    - Show all hotels in the city that is selected

- Add unit tests to prove the quality of your code. Be pragmatic in what you test and how you test it.
- Document your code in a way that another developer could easily pick up the project to extend your system with new functionality. Again: be pragmatic.

Optional

- Create a Django admin interface so that there is a backend in which the data can be viewed and edited.
- Make use of JavaScript to do an asynchronous request for the hotel data and automatically display the data. A bonus here is having an auto-complete field to do the city selection instead of a pull down.
- The data needs to be imported every day, write a cronjob that does this.
- Make use of a (D)VCS system and allow us to see your in-between commits.
- Anything done to make it easier for us to deploy your application and see what it does (fixtures, use of zc.buildout, etc.) is a plus.
- Create an interface for hotel managers: Users have access to all hotels in a single city, can add, update and remove hotels within their city.

Wrapping up

If something doesn't work out, just continue with the next part. The additional points in the exercise are not required but definitely help in showing off your development skills.

If something is not clear you can e-mail us or give us a call.

Good luck,

Joeri & Alex

Maykin Media
http://www.maykinmedia.nl