Review of Exercise

Date: March 30, 2018; Code by Théophile Blard; Review by Joeri Bekker

Overall impression

The implementation is very basic, which is exactly what we asked. There are a few PEP8 violations (tabs instead of spaces, camelCase functions) but overall the code is understandable. The autocomplete part in the view made it a bit messy and the lack of relational integrity between cities and hotels made the exercise easier and less interesting.

The lack of a Git repository makes it impossible to see the timeline and the time in which the project was done. The Python and Django quality are not of Junior level yet.

Mandatory

Below, each mandatory part of the assignment is evaluated.

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)

Main

- Import leads to all data imported into Django models [+]
- No foreign key relations [-]

Extra

- No __str__ method on Hotel model [-]
- Custom primary key fields are not necessary or recommended [-]

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.

Main

- Use of management command [+]
- Importing locally works [+]
- Importing remotely works [+] (apologies for the remote endpoint going down)

Extra

- Simple code, proper separation of local and remote imports [+]

Create a view/template that uses the imported data to: a) Choose a city (any of those that are in the dataset) and b) Show all hotels in the city that is selected

Main

- View works [+]
- Use of form [+]

Extra

- Use of function-based views (+/-)
- Incorrect field used in form (multiple cities can be selected) or incorrect queryset/query param which doesn't take multiple cities into account [-]

Add unit tests to prove the quality of your code. Be pragmatic in what you test and how you test it.

Main

- Not done [-]

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.

Main

- Documentation included [+]

Optional

Below, each optional part of the assignment is evaluated.

Create a Django admin interface so that there is a backend in which the data can be viewed and edited.

Main

- Not done [-]

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.

Main

- Results are not retrieved using async [-]
- [Bonus] auto complete works async [+]

Extra

- Use of standard components in jQuery [+]
- Use of the name instead of the ID of the city, also skip the form validation [-]
- Auto complete enabled via hardcoded constant (not a setting, not a different URL) [-]

The data needs to be imported every day, write a cronjob that does this.

Main

- Cronjob included in documentation [+]

Extra

- The lack of foreign keys makes this trivial. Also, there was no thought behind the situation to keep cities even if they are missing from the import. Or if a city is missing, what happens to the hotels [-]

Make use of a (D)VCS system and allow us to see your in-between commits.

Main

- Not done [-]

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.

Main

- Documentation with instructions [+]
- Missing requirements.txt file. No mention of which libraries to install (requests) [-]

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.

Main

- Not done [-]