Project: Catalog

Version: 1.0

Release: 12 March 2016

Python: Version 3.5

Pricing: Freeware!

Developed by: Andrew Gardiner
andrew@spondor.com.au

Catalog Application

This application is the developers answer to an assignment in the Udacity Full Stack Web Developer Nano Degree – Project 3.

The Catalog Application allows users to create categories. These categories can relate to anything. In the example provided, typically sports are the categories.

Users then create items that are linked to a category. For example, a category may be 'football' and items linked to that category may be 'boots, 'jumpers' and 'balls.

Another example for what such an application may be used for would be a catalog of books owned. Categories may be 'Grisham Books', 'Business Books' and 'Other Fiction'. Items under the 'Grisham Books' category may be, for example, 'A Time to Die', 'The Pelican Brief', and 'The Client'.

The application allows a user to peruse the information without being logged in. If a user is not logged in, then they cannot amend any of the data.

A user can log in through their 'Google' or 'Facebook' accounts. Once logged in a user can:

- create new categories;
- create new items. The user can create an item in a category even though they did not create the category;
- edit categories that they created; and
- edit or delete items that they created.

To provide a friendly experience picture files can be associated with a Category and each Item.

Background to the Setup

An Oracle virtual box manager was used. In this instance the "Oracle VM Virtual Box Manager" was installed. It was the first program run.

Next we needed to set up the vagrant virtual machine. Git Bash was used as the command editor. Within gitbash we needed to navigate to the fullstack/vagrant/catalog directory.

Once in the directory run the following commands:

- Vagrant up – This command powers on the virtual machine
- Vagrant ssh – This command logs in
- cd /vagrant – This command accesses the shared directory
- cd catalog – This command takes us to the catalog folder. Note there is no /.

At this stage we should see the following prompt:

- vagrant@vagrant-ubuntu-trusty-32:/vagrant/catalog$

To run the program type: python application.py

File Structure

The following files are included in, and form part of, the application.

File Name	Description
database_setup.py	This file was used to create the database that is relied upon. Further description of the tables created are set out below.
Catalog.db	This is the database file that is created as a result of running the database_setup.py file. This file is amended through the use of the application.py file.
populate_database.py	This is a python file that populated the Catalog.db file with a single user, a couple of categories and a number of items for each category. The file was very basic and did not include any picture files as the first version provided basic information only.
Application.py	This is the file that is run. The file is a python file and it links to the database and through a series of functions calls various html files. Further description of the functions within this file are set out below. All of the files listed below are run or accessed as a result of some call from within this file.
client_secrets.json	This is a file that the Google login process uses.
fb_client_secrets.json	This is a file that the Facebook login process uses.
Within the Static Folder (css files)	
Styles.css	The cascading style sheet file that creates the theme for the html files.
Within the Templates Folder (html files)	
header.html	A file that provides the standard header details including the application name and login details. This file is called in the header of each other html file.
login.html	A file that was created with assistance from the Udacity team and in particular the 'Authentication and Authorization: OAuth' course. The Google and Facebook code is incorporated.
menu.html & publicMenu.	The files that are the 'home' pages to the application. The file to be used is determined by whether a user is logged in.
categoryList.html, editCategoryList.html, and publicCategoryList.html	The files that have a category in the sidebar and each of that category's associated items in the main box. The file to be used is determined by whether a user is: • not logged in – publicCategoryList.html

	• logged in but not owner – categoryList.html; or • logged in and owner – editCategoryList.html.
newCategory.html	The file that allows a logged in user to create a new category.
editItem.html & publicItem.html	The files that display an item that has been clicked on in the main box. The associated category is displayed in the side bar. The file is determined by whether: • editItem.html - the user is logged in and owns the item; or • publicItem.html – otherwise.
newItem.html	The file that allows a logged in user to create a new item.
deleteItem.html	The file that allows the owner of that item to delete the item.

Database Structure

A file called database_setup.py is provided. That file was used to create the database that is relied upon. The database consists of three tables:

- The User table: This table stores the user's information as provided to us by the third party log in procedures. The fields are:
    - Id: the primary key;
    - Name: the users name;
    - Email: the unique email address; and
    - Picture: the picture of the user.
- The Category table: This table, as the name suggests, stores the information about the categories that are created. The fields are:
    - Id: the primary key;
    - Name: the name of the category. So as to be able to keep the program as clean as possible and to make url's more user friendly the name was defined to be unique;
    - Picture: a link to a picture that can be provided when the category is created or when it is edited;
    - User_id: a link to the User table. This entry lets us store which user created the category and as such who can edit the category; and
    - User: general relationship to the user.
- The Item table: This table stores the information about the items that are created that, if the app is used properly, relate to the category to which they are linked. The fields are:
    - Id: the primary key;
    - Name: the name of the item;
    - Description: a general description of the item can be provided;
    - Picture: a link to a picture that can be provided when the item is created or when it is edited;
    - Category_id: a link to the Category table. This entry lets us store which category the item is linked to;
    - Category: general relationship to the category table;
    - User_id: a link to the User table. This entry lets us store which user created the item and as such who can edit and delete the item; and
    - User: general relationship to the user.

The Application.py File

This is the main file that is run. By running the file, the url is effectively set at 'http://localhost:8000/'. As a result of using flask the use of the above url calls the function catalogList(). This function in turn calls the 'home' page html's for the application.

The 'home' pages are either the menu.html page if a user is already logged in or the publicMenu.html file if the user is not logged in.

These html pages then have a series of links that the user can click on. These links call various functions within the application.py file which in turn render various html files.

The functions within the Application.py file can be broken into a number of segments:

The Login Functions	
showLogin()	Create anti-forgery state token.
fbconnect()	Sets up a Facebook connection.
fbdisconnect()	Disconnects a Facebook connection.
gconnect()	Sets up a Google connection.
gdisconnect()	Disconnects a Google connection.
disconnect()	Disconnects based on the provider – calls fbdisconnect or gdisconnect.
General User Functions	
createUser(login_session)	Creates a user from the login session.
getUserID(email)	A function that uses the unique email to get the user id.
JSON Functions	
categoryJSON(category_name)	Outputs as a JSON based on the category.
categoryItemJSON(category_name, item_name)	Outputs as a JSON an item of a category.
XML Function	
categoryXML(category_name)	Outputs as a XML based on the category.
Catalog or Root Function	
catalogList()	This is the main function and the start point for the application. The /root directory points to the catalog directory.
Category Functions	
CategoryList(category_name)	This function calls a html file that shows a Category on the left of the screen and the items in that category on the right. Depending on the logged in status and ownership status of the user determines which html is called.
newCategory(), editCategory(category_id)	These functions call html files that allows the authorised users to create or edit category information.
getCategoryId(category_name)	A simple function to return the category id from a known category_name.
Item Functions	
newItem(), editItem(category_name,Item_name), & deleteItem(category_name,Item_name)	These functions call html files that allow authorised users to create, edit and delete items.

There are more specific comments in the application.py file that provide further notes as to how the functions achieve their purpose.

The reason for using the 'category_name' and 'item_name' parameters instead of the respective primary keys for those tables was to utilise these parameters in the url. As a result the url's are far more readable. For example the url when looking at a category called 'Soccer' having an id of 1 will be http://localhost:8000/catalog/Soccer/ instead of http://localhost:8000/catalog/1/.

Similarly, the item name is also used in the url instead of the associated id number from the id field of table item.

Conclusion and Disclaimer

The developer thanks Udacity for its continued refreshing approach to this course.

The developer accepts that this application could be tweaked to provide a better user experience. The time cost benefit of some of the tweaking was determined not to be as beneficial as moving on in the course. Having said that, the application was meant to, and the developer believes it does, exceed the basic specifications.