Lisa's Grocery Store

version 1.0

Lisa Kugler

July 28, 2015

Contents

Welcome to Lisa's Grocery Store's documentation! 1
- General Description 1
- Getting Started 1
- Project Folders and Files 2
- Python modules 2
  - Python Modules 2
    - application.py 2
    - database_setup.py 5
    - load_grocery_store.py 6
- API Endpoints 6
- Indices and tables 7
- Contact 7

Index 9

Python Module Index 11

Welcome to Lisa's Grocery Store's documentation!

Lisa's Grocery Store is an implementation of the Item Catalog project specified in the Udacity Full Stack Web Developer Nanodegree.

Seealso

https://github.com/LisaJK/grocerystore/blob/master/README.txt

Seealso

https://www.udacity.com/course/full-stack-web-developer-nanodegree--nd004

General Description

"Lisa's Grocery Store" is a web application that provides a list of products within a variety of product categories. It integrates user registration and authentication via Google or Facebook. Authenticated users have the ability to post, edit, or delete their own products.

Additional Functionality:

- API Endpoints: Apart from the required JSON endpoints, the app has also an implementation of XML and Atom endpoints.
- CRUD: Read: A product image field is added which can be read from the database and displayed on the page.
- CRUD: Create: A product image field can be included when a new product is created in the database.
- CRUD: Update: For already existing products, product images can be added, changed and deleted.
- CRUD: Delete: The function is implemented using POST requests and nonces to prevent cross-site request forgeries (CSRF).
- Comments: Comments are (hopefully ;-)) thorough and concise.

Getting Started

1. Install Vagrant and Virtual Box as described in the course materials of the "Full Stack Foundations".
2. Clone the repository from GitHub $ git clone https://github.com/LisaJK/grocerystore.git
3. Launch the Vagrant VM.
4. Move to the directory "/vagrant/catalog"
5. Run the application within the VM by typing "python application.py" in the console.
6. Access the application by visiting "http://localhost:8000" on your browser.

Note

The app was developed and tested with Chrome Version 43.0.2357.134 m. It was also tested with Firefox 39.0. In IE you might have to add https://accounts.google.com to your trusted sites to enable Google Login (Facebook resp.).

Project Folders and Files

1. README.txt:
   File containing all necessary information how to download and run the project.
2. application.py:
   This python file contains the server-side implementation of the app.
3. database_setup.py:
   This python file contains the setup of the database "grocery_store.db".
4. load_grocery_store.py:
   Run this python file to add some initial users, product categories and products to the database "grocery_store_db".
5. templates folder:
   This folder contains all Flask templates of the app.
   - category.html
   - deleteCategory.html
   - deleteProduct.html
   - editCategory.html
   - editProduct.html
   - layout.html
   - login.html
   - newCategory.html
   - newProduct.html
   - product.html
   - products.html
   - showCategory.html
   - showGroceryStore.html
   - showProduct.html
6. static folder:
   - styles.css: CSS file containing the styles applied to the html files.
   - vegetables-752156_1280.jpg: background image.
7. uploads folder:
   Empty folder used to store product images uploaded for products.
8. docs folder:
   A folder docs containing the documentation created using Sphinx.
9. JSON files containing OAuth2.0 parameters: - client_secret_fb.json - client_secret_g.json

Python modules

Python Modules

application.py

This module contains the server functionality of Lisa's Grocery Store.

application.ALLOWED_EXTENSIONS = *set(['gif', 'jpeg', 'JPG', 'jpg', 'png'])*
uploaded images can have the extension included in this set.

application.FB_APP_ID = *u'1610860372517361'*
str: app id assigned by Facebook and saved in 'client_secret_fb.json'.

application.FB_APP_SECRET = *u'a3585e18fc02dc3f9496c6c7837dcf4f'*
str: app secret assigned by Facebook and saved in 'client_secret_fb.json'.

application.GOOGLE_CLIENT_ID = *u'130832505461-33h5klvebue7cqnh8ojer6drf67e7js3.apps.googleusercontent.com'*
str: client id assigned by Google and saved in 'client_secret_g.json'.

application.UPLOAD_FOLDER = *'./uploads'*
str: name of the folder where uploaded images can be found.

application.allowed_file (filename)
Returns True if the filename has an allowed extension.

application.buildProductXML (product)
Builds up the XML tree for a product.

application.categoriesAtom ()
Atom API to get info about all categories.

application.categoriesJSON ()
JSON API to get info about all categories.

application.categoriesXML ()
XML API to get info about all categories.

application.categoryAtom (category_name)
Atom API to get info about a given category.

application.categoryJSON (category_name)
JSON API to get info about a given category.

application.categoryXML (category_name)
XML API to get info about a given category.

application.createLoginOutput ()
Creates the login output for the login page after a successful login via Google or Facebook.

application.createUser (login_session)
Create a new user in the database.

application.createXMLResponse (tree)
Creates the xml response out of the given tree.

application.csrf_protect ()
If a POST does not contain a csrf token or contains a wrong csrf token a Forbidden is raised.

application.deleteCategory (category_name)
Delete the given category and its connected products.
If not logged in yet, the user is redirected to the login page. If the request method is POST, the category and all products of the category are deleted. The images of the deleted products are also deleted in the upload folder. If the request method is GET, the delete category page is rendered if the user is the owner of the category.

application.deleteProduct (category_name, product_name)
Delete a given product of a category.
If not logged in yet, the user is redirected to the login page. If the request method is POST, the product is deleted. The image of the deleted product is also deleted in the upload folder. If the request method is GET, the delete product page is rendered if the user is the owner of the product.

application.editCategory (category_name)
Edit a given category.

If not logged in yet, the user is redirected to the login page. If the request method is POST, the category is updated with the returned values. If the request method is GET, the edit category page is rendered if the user is the owner of the category.

application.editProduct (product_name)
Edit a given product.
If not logged in yet, the user is redirected to the login page. If the request method is POST, the product is updated with the returned values. The image of the product is also updated in the upload folder. If the request method is GET, the edit product page is rendered if the user is the owner of the product.

application.fbconnect ()
Login the user by Facebook Sign In.
First, the short term token returned by the POST is echanged into a long term access token. Then get the user info and the picture and store the user data in the login session for later use.

application.fbdisconnect ()
Revoke a current fb access token and reset the login_session.

application.gconnect ()
Login the user by Google Sign In.
First, obtain the authorization code from the POST, update the authorization code into a credentials object and check if the access token within the credentials object is valid. After having checked that the access token is used for the intended user and app, get the user info and store the user data for later use in the login session.

application.gdisconnect ()
Revoke a current google access token and reset the login_session.

application.generate_csrf_token ()
Creates a csrf token.

application.getUserID (email)
Returns the user id connected to the given email or None if the email is not existing in the database yet.

application.getUserInfo (user_id)
Returns a user of the given user_id.

application.getUsername ()
Returns the username if already in the login session, otherwise None.

application.get_random_string ()
Creates an uppercase random string

application.login ()
Login function which renders the login page and starts the login process.

application.logout ()
Logout a user (Google or Facebook).

application.newCategory ()
Create a new category.
If not logged in yet, the user is redirected to the login page. If the request method is POST, a new category is created in the database with the returned values. If the request method is GET, the new category page is rendered.

application.newProduct ()
Create a new product.
If not logged in yet, the user is redirected to the login page. If the request method is POST, a new category is created in the database with the returned values. The image is stored in the upload folder. If the request method is GET, the new product page is rendered.

application.productAtom (category_name, product_name)
Atom API to get info about a given product and category.

application.productJSON (category_name, product_name)
JSON API to get info about a given product and category.

application.productXML (category_name, product_name)
XML API to get info about a given product and category.

application.productsAtom ()

Atom API to get info about all products.

application.productsJSON ()
JSON API to get info about all products.

application.productsXML ()
XML API to get info about all products.

application.resetUserSession (result)
resets the user data of the login session.

application.showCategory (category_name)
Show the category and the products of the category.

application.showGroceryStore ()
Show the grocerystore with all categories and all products existing in the database.

application.showProduct (category_name, product_name)
Show one given product of a given category.

application.storeUserData (username, email, picture, ext_user_id, google_credentials=None, fb_access_token=None)
Stores the user data given by Google or Facebook in the login session.

application.uploads (filename)
Locates the given file in the upload directory and shows it in the browser.

database_setup.py

Database setup for Lisa's Grocery Store. The database consists of three tables representing three objects:

- User: A user is created after login and can create, update or delete categories and products.
- Category: A category can be created, updated and deleted. It is always owned by one user.
- Product: A product can be created, updated and deleted. It is always owned by one user and assigned to one category.

class database_setup.Category (**kwargs)
Bases: sqlalchemy.ext.declarative.api.Base
Class representing a Category.

description
description (Column): a description of the category.

name
name (Column): the category name (primary key).

serialize
Return object data in easily serializeable format.

user
relation to the user who is the owner.

user_id
user_id (Column): reference to the owner of the category.

class database_setup.Product (**kwargs)
Bases: sqlalchemy.ext.declarative.api.Base
Class representing an product.

category

category_name
category_name (Column): the category the product belongs to.

description
description (Column): a description of the product.

image_file_name
image_file_name (str): file name of an image of the product. The image is stored in the upload folder.

name
name (Column): the product name, primary key.

price
price (Column): the price of the product.

serialize
Return object data in easily serializeable format.

user
relation to the user who is the owner.

user_id
user_id (Column): the id of the owner of the product.

class database_setup.User (**kwargs)
Bases: sqlalchemy.ext.declarative.api.Base
Class representing a user.

email
email (Column): the email of the user.

id
id (Column): the internal user id used as primary key.

name
name (Column): the user name.

picture
picture (Column): link to a picture of the user.

serialize
Return object data in easily serializeable format.

load_grocery_store.py

Populates the database of Lisa's Grocery Store with a few Users, Categories and Products.

load_grocery_store.DBSession = *sessionmaker(class_='Session',autoflush=True, bind=Engine(sqlite:///grocery_store.db), autocommit=False, expire_on_commit=True)*
A DBSession() instance establishes all conversations with the database and represents a "staging zone" for all the objects loaded into the database session object. Any change made against the objects in the session won't be persisted into the database until you call session.commit(). If you're not happy about the changes, you can revert all of them back to the last commit by calling session.rollback().

load_grocery_store.engine = *Engine(sqlite:///grocery_store.db)*
Bind the engine to the metadata of the Base class so that the declaratives can be accessed through a DBSession instance.

API Endpoints

JSON:

- http://localhost:8000/grocerystore/categories/JSON

- http://localhost:8000/grocerystore/products/JSON
- http://localhost:8000/grocerystore/<category_name>/products/JSON
- http://localhost:8000/grocerystore/<category_name>/<product_name>/JSON

XML:

- http://localhost:8000/grocerystore/categories/XML
- http://localhost:8000/grocerystore/products/XML
- http://localhost:8000/grocerystore/<category_name>/products/XML
- http://localhost:8000/grocerystore/<category_name>/<product_name>/XML

Atom:

- http://localhost:8000/grocerystore/categories/Atom
- http://localhost:8000/grocerystore/products/Atom
- http://localhost:8000/grocerystore/<category_name>/products/Atom
- http://localhost:8000/grocerystore/<category_name>/<product_name>/Atom

Indices and tables

- genindex
- modindex
- search

Contact

lisa.kugler@googlemail.com

Index

A

ALLOWED_EXTENSIONS (in module application)

allowed_file() (in module application)

application (module)

B

buildProductXML() (in module application)

C

categoriesAtom() (in module application)

categoriesJSON() (in module application)

categoriesXML() (in module application)

Category (class in database_setup)

category (database_setup.Product attribute)

category_name (database_setup.Product attribute)

categoryAtom() (in module application)

categoryJSON() (in module application)

categoryXML() (in module application)

createLoginOutput() (in module application)

createUser() (in module application)

createXMLResponse() (in module application)

csrf_protect() (in module application)

D

database_setup (module)

DBSession (in module load_grocery_store)

deleteCategory() (in module application)

deleteProduct() (in module application)

description (database_setup.Category attribute)

(database_setup.Product attribute)

E

editCategory() (in module application)

editProduct() (in module application)

email (database_setup.User attribute)

engine (in module load_grocery_store)

F

FB_APP_ID (in module application)

FB_APP_SECRET (in module application)

fbconnect() (in module application)

fbdisconnect() (in module application)

G

gconnect() (in module application)

gdisconnect() (in module application)

generate_csrf_token() (in module application)

get_random_string() (in module application)

getUserID() (in module application)

getUserInfo() (in module application)

getUsername() (in module application)

GOOGLE_CLIENT_ID (in module application)

I

id (database_setup.User attribute)

image_file_name (database_setup.Product attribute)

L

load_grocery_store (module)

login() (in module application)

logout() (in module application)

N

name (database_setup.Category attribute)

(database_setup.Product attribute)

(database_setup.User attribute)

newCategory() (in module application)

newProduct() (in module application)

P

picture (database_setup.User attribute)

price (database_setup.Product attribute)

Product (class in database_setup)

productAtom() (in module application)

productJSON() (in module application)

productsAtom() (in module application)

productsJSON() (in module application)

productsXML() (in module application)

productXML() (in module application)

R

resetUserSession() (in module application)

S

serialize (database_setup.Category attribute)

(database_setup.Product attribute)
(database_setup.User attribute)
showCategory() (in module application)
showGroceryStore() (in module application)
showProduct() (in module application)
storeUserData() (in module application)

U

UPLOAD_FOLDER (in module application)
uploads() (in module application)
User (class in database_setup)
user (database_setup.Category attribute)
(database_setup.Product attribute)
user_id (database_setup.Category attribute)
(database_setup.Product attribute)

Python Module Index

a

application

d

database_setup

l

load_grocery_store