OCR-Projet 5

Release 1.0

Mehdi BICHARI

Apr 25, 2020

OVERVIEW

CHAPTER

ONE

INSTALLATION

1.1 Python:

In order to install OCR-Projet5 you need to use python 3+.

If you don't already have it, please refer to python's site.

1.2 Python's modules:

The modules needed by python are listed in the requirements.txt at the root of the project.

To install module, place yourself at the root of OCR_Projet5 and use the command:

```
pip3 install -r requirements.txt
```

1.3 MySQL:

The software uses MySQL to store data. To install it, please refer to MySQL's site.

Please refer to *Database_setup* to create your own database.

CHAPTER

TWO

DATABASE_SETUP

2.1 Database creation:

To create your own database, please refer to MySQL website

2.2 Change Database:

You can change the database used by the software by replacing the saying N while the programs asks you which database you want to use. If you say N, you will be asked to give the new database's name.

If you drop your database and want a new one, either if it has the same name, answer N while the program asks you which database you want to use. If you don't, the database won't be populated and the program won't works.

CHAPTER

THREE

CATEGORIES MANAGER

class CategoryManager

class controller.categories_manager.**CategoryManager**(*table: str = 'Categories'*)

Manager for category class

static convert_to_category(*categories: Dict*) → List[model.category.Category]

Convert list of dictionnaries to list of Category.

Parameters **categories** – list containing dictionnaries

Returns List[Category]

static get_from_openfoodfacts(*categories_url: str*) → Dict

Get categories from OpenFoodFacts.

Parameters **categories_url** – URL of categories on OpenFoodFact

Returns Dict

insert_in_database(*categories: List[model.category.Category], session: controller.session.Session*) → None

Put categories in user's database.

Parameters

- **categories** – List containing all categories on OpenFoodFacts
- **session** – Session

Returns None

CHAPTER

FOUR

DATABASE

User's database

class controller.database.**Database**(*session*, *database_name*)

Database used to store program's data

static _Database__convert_to_objects(*category_manager: controller.categories_manager.CategoryManager*, *object_dict: Dict*, *product_manager: controller.products_manager.ProductManager*, *store_manager: controller.stores_manager.StoreManager*) → Dict

Convert all dict to objects according to the key.

Parameters

- **category_manager** – CategoryManager
- **object_dict** – Dictionnary containing lines to convert
- **product_manager** – ProductManager
- **store_manager** – StoreManager

Returns Dict

_Database__create_tables(*session*) → None

User's database tables creation.

Parameters **session** – user's database connection

Returns None

_Database__get_all_data(*category_manager: controller.categories_manager.CategoryManager*, *product_manager: controller.products_manager.ProductManager*, *store_manager: controller.stores_manager.StoreManager*) → Dict

Get all data needed for the application from OpenFoodFacts.

Parameters

- **category_manager** – CategoryManager
- **product_manager** – ProductManager
- **store_manager** – StoreManager

Returns Dict

_Database__save_user_database() → None

Save user's database name in param file.

Returns None

populate(*category_manager: controller.categories_manager.CategoryManager*, *product_manager: controller.products_manager.ProductManager*, *store_manager: controller.stores_manager.StoreManager*) → None

Populate the user database with OpenFoodFacts data.

Parameters

- **category_manager** – categories from OFF
- **product_manager** – products from OOF
- **store_manager** – stores from OOF

Returns None

CHAPTER

FIVE

PRODUCTS MANAGER

Class ProductManager

class controller.products_manager.**ProductManager**(*table: str = 'Products'*)

Manager for class Product

static convert_to_products(*products: List*) → List[model.product.Product]

Convert list of dictionnaries to list of Product.

Parameters **products** – list containing dictionnaries

Returns List[Product]

static get_bad_products(*category: str*, *session: controller.session.Session*) → List[model.product.Product]

Get bad products from given category in user database.

Parameters

- **category** – Category to filter on
- **session** – Session

Returns List[Product]

static get_better_product(*product: model.product.Product*, *session: controller.session.Session*) → model.product.Product

Get a better product in replacement.

Parameters

- **product** – Product to replace
- **session** – Session

Returns None

static get_from_openfoodfact(*categories: List[str], openfoodfacts_url: str, parameters: Dict*) → List

Get all the products from given categories in OpenFoodFact.

Parameters

- **categories** – List of given categories to filter
- **openfoodfacts_url** – url of Openfoodfacts API
- **parameters** – base_params

Returns List

static get_saved_products(*session: controller.session.Session*) → List

Get all previously saved products.

Parameters **session** – Session

Returns List[Product]

insert_products_in_database(*products: List[model.product.Product], session: controller.session.Session*) → None

Insert products in user's database.

Parameters

- **products** – List of products to insert in database
- **session** – Session

Returns None

static save_product_replacement(*base_product: model.product.Product, replacement_product: model.product.Product, session: controller.session.Session*) → None

Save in database the product substitution.

Parameters

- **base_product** – Product to replace
- **replacement_product** – Product which replace
- **session** – Session

Returns None

CHAPTER

SIX

SESSION

This class allows us to create the connection with mysql

class controller.session.**Session**

Mysql Session

close()

Close mysql connection.

Returns None

connect()

Connect to mysql.

Returns None

database_exists(*database_name: str*) → bool

Check is database exists in mysql.

Parameters **database_name** – Name of the database to check

Returns bool

insert(*statement: str*, *data: List*) → None

Insert in user's database.

Parameters

- **statement** – statement to insert
- **data** – values inserted

static prepare_insert_statement(*table: str*, *columns: List*) → str

Prepare the insert statement.

Parameters

- **table** – Where to insert datas
- **columns** – Columns of table

select(*statement: str*, *filters: Optional[tuple] = None*) → List

Select in user's database.

Parameters

- **statement** – Statement to select
- **filters** – Filters

Returns List

CHAPTER

SEVEN

STORES MANAGER

Class StoreManager

class controller.stores_manager.**StoreManager**(*table: str = 'Stores'*)

Manager of stores

static convert_to_store(*stores: List*) → List[model.store.Store]

Convert list of dictionnaries to list of objects.

Parameters **stores** – list containing dictionnaries

Returns List[Store]

static get_from_openfoodfacts(*store_url: str*) → List

Method to get all the stores contained in OpenFoodFacts.

Parameters **store_url** – URL of stores on OpenFoodFact

Returns List

insert_in_database(*session: controller.session.Session, stores: List[model.store.Store]*) → None

Method to put all stores in database.

Parameters

- **session** – Session
- **stores** – list containing all stores on OpenFoodFacts

Returns None

CHAPTER

EIGHT

CATEGORY CLASS

Class category

class model.category.**Category**(*name: str*, *off_id: str*, *url: str*)

CHAPTER

NINE

PRODUCT CLASS

Class product

class model.product.**Product**(*brands: str*, *categories_tags: List*, *id: str*, *nutriscore_grade: str*, *packaging_tags: str*, *product_name_fr: str*, *stores_tags: List*, *url: str*)

CHAPTER

TEN

STORE CLASS

Class store

class model.store.**Store**(*name: str*, *url: str*)

CHAPTER

ELEVEN

USER INTERFACE

Module containing all user interface.

view.user_interface.**choose_category**() → tuple
Choose category from categories available.

Returns tuple (dict(categories), choice from user)

view.user_interface.**choose_product**(*products*) → int
Choose a product from bad product list.

Parameters **products** – List of products

Returns int

view.user_interface.**create_dict**(*list_to_transform: List*) → Dict
Transform list to dict with numbers as keys.

Parameters **list_to_transform** – List

Returns Dict

view.user_interface.**define_database**(*category_manager: controller.categories_manager.CategoryManager*, *product_manager: controller.products_manager.ProductManager*, *session: controller.session.Session*, *store_manager: controller.stores_manager.StoreManager*) → controller.session.Session
Format user's database to host application's data.

Parameters

- **category_manager** – CategoryManager
- **product_manager** – ProductManager
- **session** – Session
- **store_manager** – StoreManager

Returns Session

view.user_interface.**format_dict**(*dictionnary: Dict*)
Format dictionnaries and add a blank line after.

Parameters **dictionnary** – Dictionnary to format

Returns None

view.user_interface.**navigate**(*product_manager: controller.products_manager.ProductManager*, *session: controller.session.Session*) → controller.session.Session

Function to navigate inside the program.

Parameters

- **product_manager** – ProductManager
- **session** – Session

Returns Session

view.user_interface.**validate_choice**(*dict_choice: Dict*, *theme: str*) → int

Validate users choice through a dict a return the number of the answer.

Parameters

- **dict_choice** – Dict containing proposals
- **theme** – sentence to explain dict

Returns int

view.user_interface.**welcome**(*category_manager: controller.categories_manager.CategoryManager*, *product_manager: controller.products_manager.ProductManager*, *session: controller.session.Session*, *store_manager: controller.stores_manager.StoreManager*) → controller.session.Session

Function at the beginning of the program. It verifies if a database is already set in param.py and if it exists. If all is ok, navigate function is launched.

Parameters

- **category_manager** – CategoryManager
- **product_manager** – ProductManager
- **session** – Session
- **store_manager** – StoreManager

Returns Session

CHAPTER

TWELVE

MAIN FILE

Main program file. To launch first.

main.**main**()
    Main method of the program

CHAPTER

THIRTEEN

PARAM FILE

This file is used to store constants of the program

PYTHON MODULE INDEX

c

m

p

v

INDEX

Symbols

C

D

F

G

I

M

N

P

S

V

W