Reference guide

Comic search engine

1. Objective

The goal is to build a search engine for comics. The data is extracted from bdgest and more specifically the bedetheque subdomain. The web application has a nice look and feel, a wide range of search criterias and relevant visualizations.

2. Structure

The project consists of three main components: a MongoDB database, a Scrapy data extractor and a Flask web application. It is available at this GitHub repository. In order to clone the project, run the following command on your machine (you need to have git installed):

```
git clone https://github.com/nicolasvo95/OUAP-4314.git
```

The MongoDB database is NoSQL. The hierarchy is database, then collection, document and fields within a document. In the project, the database is called **bdgest**, the collections are **authors, series** and **comics**. The fields for each collection can be found here. Since the project's main language is Python, pymongo is used to query the database. The data extraction is done with the Scrapy framework. The web application is developed with Flask and the visualizations with HighchartsJS. The appearance and responsivity are obtained with MaterializeCSS.

3. Deployment

Concerning the deployment of the project, it is done via Docker Compose. Therefore, Docker and Docker Compose are required. See how to install them here. It allows for the deployment of all three containers at once, thus, meanwhile the scraped data is fed to the database, the web application can already be used.

If you already wish to deploy the project, simply run the following:

```
docker-compose up
```

If you wish to deploy in detached mode (i.e. in background):

```
docker-compose up -d
```

Check the logs with:

```
docker-compose logs
```

Check the status of the containers with:

```
docker-compose ps
```

Gracefully shut the containers down:

```
docker-compose down
```

N.B.: you may need to pull the Docker image for the mongo database. In that case, run the following:

```
docker pull mongo
```

The Docker compose first builds and runs the Mongo database, because both Scrapy and Flask containers interact with it. The specified **links** parameter allows us to connect the Scrapy and Flaks containers to the database with the name **mongo**. The **volumes** parameter allows for real-time modification of a container's code.

The MongoDB container does not need a specific Dockerfile as its image is directly pulled from the Dockerhub. The Scrapy and Flask projects both have a Dockerfile based on a Python 3 image. Their dependencies are installed via a pip install of the requirements.txt file located in their respective directories.

4. Scrapy

4.1. Settings

The settings.py script controls the way the scrap is done. In fact, websites have defense mechanisms such as temporary IP blacklisting in order to prevent excessive scraping. The file allows to set a delay between each new fetching of a page and many other limitations. For example, the definition of specific user agents and customized middleware behaviour. In the case of the project, it was found acceptable to simply put a limit between each new request as well as a modified download delay:

```
AUTOTHROTTLE_START_DELAY = 3
DOWNLOAD_DELAY = 2
```

4.2. Items

The items.py script defines the structure of the three items author, series and comic. Since bdgest already provides a distinct ID for authors, series and comics, they are also retrieved and used as ID for our database. In MongoDB, the **_id** field defines a document ID,

which by default is an automatically generated ObjectID. However it is not as convenient to read and manipulate.

4.3. Pipelines

The pipelines.py script defines how the items are inserted into the Mongo database. Indeed, each item has a specific destination collection. Upon receiving, the name of the item is simply checked:

```
if isinstance(item, AuthorsItem):
```

When it is checked that the item is not empty, it is inserted:

```
self.collection.insert_one(dict(item))
```

4.4. Spider

The authors.py script defines the actual spider. First, the Scrapy items are imported from the items.py file:

```
from ..items import AuthorsItem, SeriesItem, ComicsItem
```

Also the already existing databases are dropped for a fresh start:

```
client = MongoClient("mongo")
client.drop_database("bdgest")
```

The spider is named **authors**, its list of allowed domains consists simply of **bedetheque.com**. The list of start URLs consists of the pages referencing the authors by alphabetical order with the addition of a page for names starting by a special character. A tidy list comprehension covers all the pages:

```
start_urls =
['https://www.bedetheque.com/liste_auteurs_BD_{0}.html'.format(x) for x
in string.ascii_lowercase]
start_urls.append('https://www.bedetheque.com/liste_auteurs_BD_0.html')
```

The first **parse** function simply retrieves the URL for each author page from the pages listed in **start_urls**. Then a **yield** calls a Scrapy Request with the author page URL as parameter and the **parse_authors** function as callback.

parse_authors retrieves many informations about each author: first name, last name, nickname, birth date, death date, image, country and personal webpage. An additional full name is created by joining first name, last name and nickname. On the author page, the link

to each of his/her series can be found. It is the parameter of the next **yield** with **parse_series** as callback function.

parse_series retrieves informations about both series and comics. Indeed, the full list of series can be retrieved by ending the URL with __10000.html:

```
var_url = re.sub('.html$', '__10000.html', var_url) #show all comics of a series
```

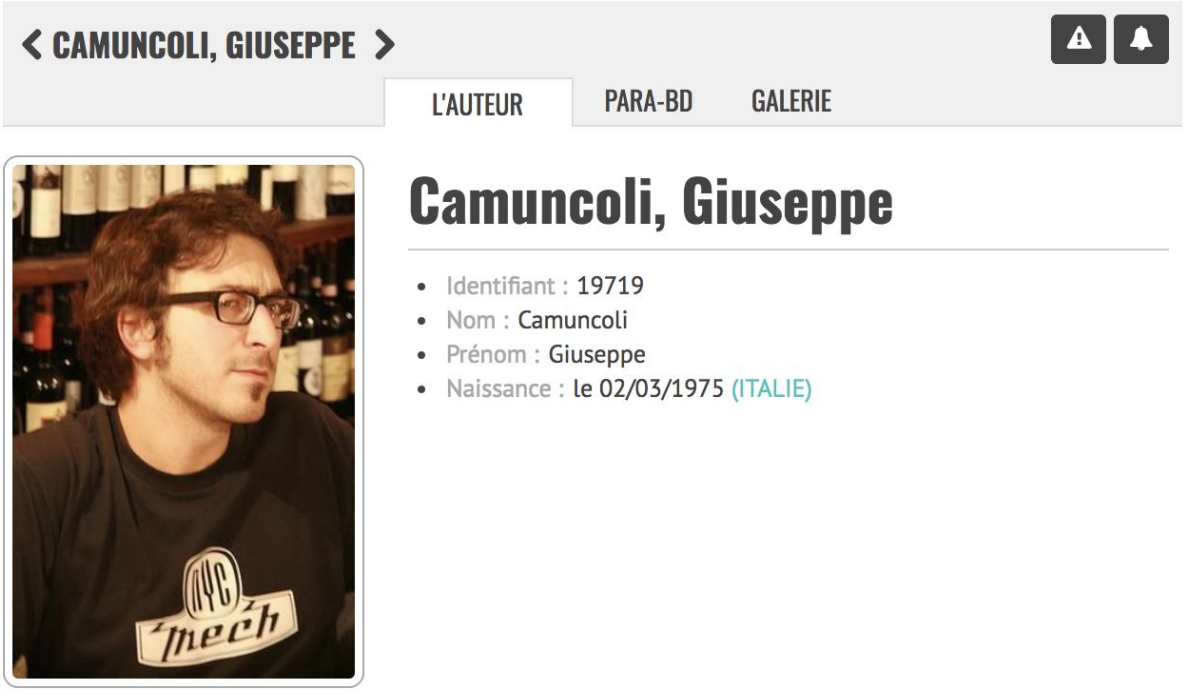
It also happens that for each series page, all the comics are displayed and with many informations, which in a first time meant we did not need to go further and actually fetch each comic's individual page.

parse_comics retrieves the information which could not be found on the series page: a description of the comic and its cover image.

When a comic is done being scraped, its item is sent to the pipeline. When all of a series' comics have been scraped, its item is sent to the pipeline. Finally when an author's series have all been scraped, its item is sent to the pipeline.

The idea behind the use of **yield** is to have the **parse** functions behave like **generators**. Therefore, when a series URL needs to be parsed, it can be done and when it is finished being parsed, we go back to the **parse_author** function but at its previous state and keep feeding the rest of series pages to be parsed.

Example of pages:



Autres collaborations	de	à	Rôle
Deadpool (Marvel France 4e série - 2013)	2015		Couverture
Deadpool (Marvel France 5e série - 2017)	2018		Couverture
Hellblazer (Brian Azzarello présente)	2017		Encrage
Marvel Heroes (Marvel France 3e série)	2011		Couverture
Marvel Icons (Marvel France 2e série)	2011		Couverture
Marvel Stars	2011		Couverture
Spider-Man (Marvel France 4e série - 2013)	2013		Couverture
Spider-Man (Marvel France 6e série - 2017)	2017		Encrage
Spider-Man : Spider-Island	2017		Encrage
Star Wars (Panini Comics - 2017)	2018		Couverture

SA BIBLIOGRAPHIE

Les tableaux synthéthiques ci-dessous peuvent être faussés par des éditions anniversaires (souvent post-mortem). Ils ne tiennent pas compte des autres pseudonymes sous lequel signe cet auteur.

Séries principales	de	à	Rôle
All Star Batman	2018		
All-New Amazing Spider-Man (Marvel Now!)	2017	2018	
All-New Spider-Man	2016	2017	
All-New X-Men (Marvel Now!)	2015		
Amazing Spider-Man (Marvel Now!)	2016	2017	
Astonishing X-Men (kiosque)	2010		
Batman Rebirth (DC Presse)	2018		
Dark Wolverine	2017		
Hellblazer (100% Vertigo)	2009		
Iron Man (3e série, Marvel France)	2013		
Justice League vs. Suicide Squad	2018		
Maori	2013	2014	
Marvel (Moustique)	2014		
Marvel Heroes Extra	2010		
Marvel Universe (Panini - 2007)	2008		
Moon Knight (100% Marvel - 2015)	2015		
Scorpions du Désert (Les)	2007		
Spider-Man (Marvel France 3e série - 2012)	2012	2013	
Spider-Man (Marvel France 4e série - 2013)	2013	2014	
Spider-Man (Marvel France 5e série - 2015)	2015		
Spider-Man (Marvel France 6e série - 2017)	2017		

Author page and its series

Maori

Acheter

POLAR · Série en cours · Europe · 2 albums · Français · 2013-2014 · 40233

2 ALBUMS · 1 AVIS · VENTE · PARA-BD · GALERIE

© Ankama Éditions - 2013

Genre : Polar
Parution : Série en cours
Tomes : 2
Identifiant : 40233
Origine : Europe
Langue : Français
Forum : Discuter de la série dans les forums

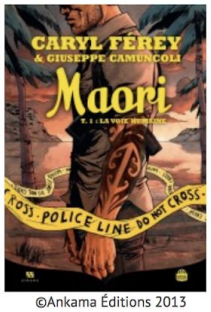
Proposer un mot clé

Jack Kenu est Maori, seul depuis le départ de sa femme, et accessoirement officier de police à Auckland, Nouvelle-Zélande. L'effondrement du pays, ruiné par la crise financière devenue mondiale, ne l'intéresse pas plus que la bataille électorale faisant rage entre Kirwan, Premier Ministre du parti conservateur, et le très controversé Witkaire, un député Maori adepte de "la Voie Humaine", programme s'appuyant sur une approche économique et sociétale radicalement novatrice. On découvre le corps d'une jeune femme sur une plage, le crâne fracassé. Sandra, la fille de Witkaire. Jack, chargé de l'enquête, apprend que Sandra a quitté l'école expérimentale de son père pour frayer avec des voyous Maoris, discréditant le discours émancipateur de Witkaire. Crime crapuleux, manipulation ? Jack Kenu n'est pas au bout de ses (mauvaises) surprises...

1 vote

LES CHRONIQUES BDGEST · LES PREVIEWS

DÉTAIL DES ALBUMS DE LA SÉRIE



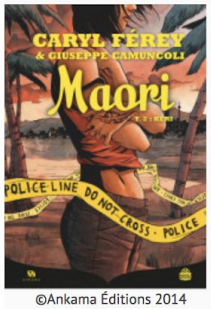
©Ankama Éditions 2013
Férey/Camuncoli

1 . La voie humaine.

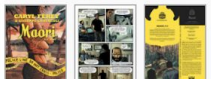
Collection · Wishlist · amazon · BDfugue · fnac · PRICE MINISTER

Evaluation : Note: 3.7/5 (9 votes)
Identifiant : 198920
Scénario : Férey, Caryl
Dessin : Camuncoli, Giuseppe
Couleurs : Richard, Stéphane
Dépot légal : 10/2013 (Parution le 10/10/2013)
Estimation : non coté
Editeur : Ankama Éditions
Collection : Hostile Holster
Format : Format normal
ISBN : 978-2-359-10367-0
Planches : 62
Autres infos : ★
Créé le : 07/10/2013 (modifié le 05/02/2014 13:13)

Info édition : noté Première édition



©Ankama Éditions 2014
Férey/Camuncoli



2 . Keri

Collection · Wishlist · amazon · BDfugue · fnac · PRICE MINISTER

Evaluation : Note: 4.5/5 (4 votes)
Identifiant : 210929
Scénario : Férey, Caryl
Dessin : Camuncoli, Giuseppe
Couleurs : Richard, Stéphane
Dépot légal : 03/2014 (Parution le 14/03/2014)
Estimation : non coté
Editeur : Ankama Éditions
Collection : Hostile Holster
Format : Format normal
ISBN : 978-2-359-10368-7
Planches : 62
Autres infos : ★
Créé le : 28/02/2014 (modifié le 11/03/2018 10:06)

LES 2 ALBUMS DE LA SÉRIE

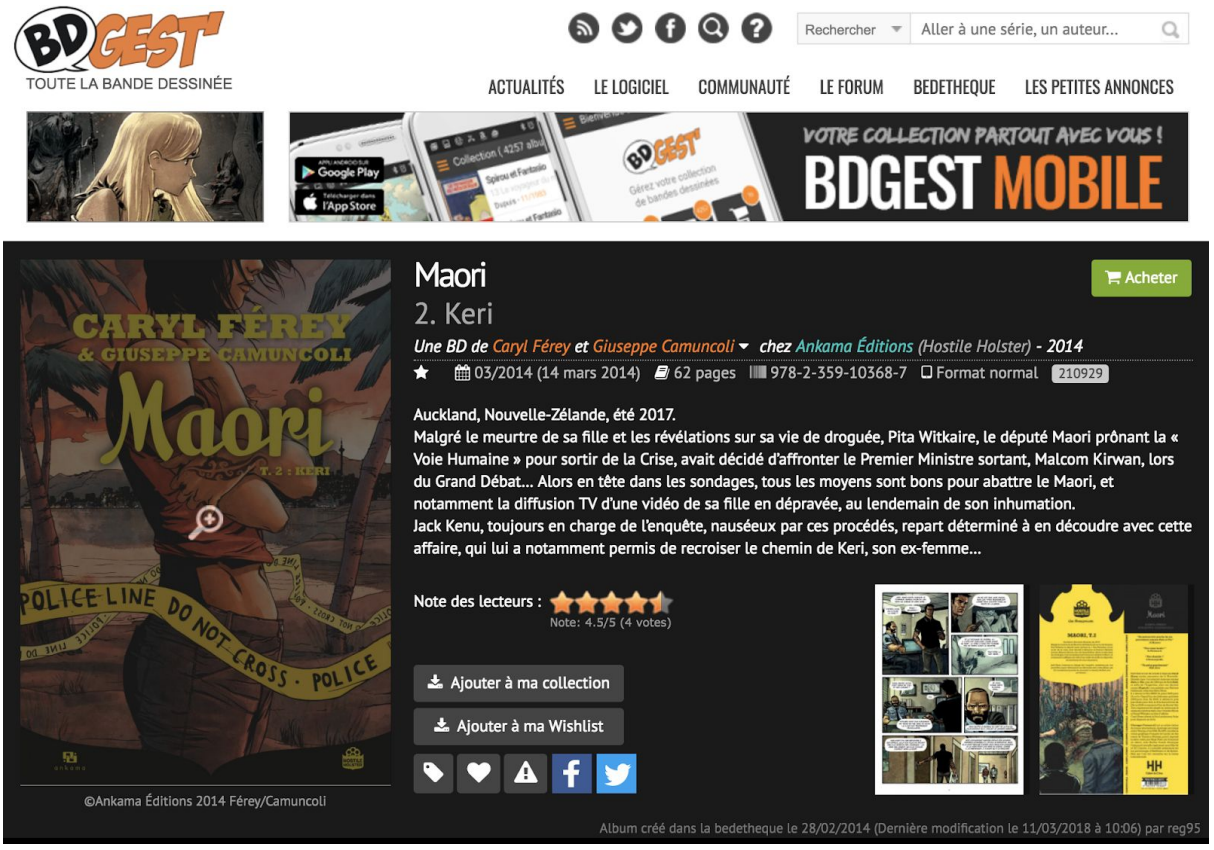
1. La voie humaine. 10/2013
2. Keri 03/2014

LES 2 ALBUMS DE LA SÉRIE

1. La voie humaine. 10/2013
2. Keri 03/2014

CONCOURS
NOBODY
Christian De Metter

[A series page and its comics down the page]



A comic page

5. Flask

Inside the app folder:

- views.py controls the routes, how the data is processed and which templates are rendered
- forms.py defines the structure of each form, i.e. the fields, their labels, their type

Inside the templates folder:

- macros.html contains the jinja macros which render the forms, tables of results and information cards
- dictionaries.html contains some variables so that they do not need to be redefined at many places
- base.html is the base template for all pages of the web application
- index.html defines the homepage and its three forms
- author.html, series.html and comic.html define the result pages for each author, series and comic respectively
- author_id.html defines the page of an author in particular
- series_id.html defines the page of a series in particular
- comic_id.html defines the page of a comic

The static folder contains the CSS and JS libraries responsible for the appearance and

mobile responsivity of the web app.

5.1. run.py

It is the file which launches the web application. The port can be specified as well as the use of multithreading.

```
app.run(debug=True, port=1000, host='0.0.0.0', threaded=True)
```

5.2. forms.py

The project uses the WTForms library known for efficiently formatting forms in Flask web applications. In this project, there is a form for each author, series and comic. Each form has many fields and labels which are neatly defined in the forms.py script.

5.3. views.py

The file first imports the previously defined forms:

```
from .forms import AuthorForm, SeriesForm, ComicForm
```

FIrst, the connection to the Mongo client and the database set to **bdgest**:

```
client = MongoClient("mongo")
db = client['bdgest']
```

A few functions were written in order to keep the code as *DRY* as possible:

- **series_by_id** and **author_by_id** simply return the name of the series or author knowing its ID
- **table_comic** and **table_series** are functions capable of returning lists of comics and series in a processed format. Not every field in the database are wanted for display on the website, therefore only the desired fields are kept:

```
document_updated.update({key: (document[key] if document.get(key) else
"") for key in ("scenario", "illustration", "editor", "legal_deposit")})
```

And special keys in the document starting with **redirect** help creating the redirections to other routes on the website, i.e. author and series specific page. It will be explained in depth in the macros.html template.

- **kardesh** is a function which resolves the task of searching through multiple collections (fun fact: kardesh means brother in Turkish). For example, in order to search for a series with a name “a” whose author has a name “b”, a first preliminary

query of the authors collection for a name starting with “b” returns the list of author IDs matching the criteria. The initial query is updated to answer the condition that its author IDs need to belong to this list of author IDs.

The routes are:

- **home** (template index.html): homepage with the three forms
- **author** (template author.html): result page of the author form. The birth date and death date are formatted to day-month-year.
- **series** (template series.html): it is a bit more efficient than the **author** route with more dictionaries to automatically fetch the results of the input fields and to properly build the Mongo query string. It essentially returns a list of series which answer the search.
- **comic** (template comic.html): similar to **series**
- **author_id** (template author_id.html): takes an **author_id** as argument. It outputs information about an author, his/her list of series and comics. Mongo aggregations are executed in order to get the data which feeds the visualizations in JS:

```
    pipeline = [
        {"$match": {"author_id": author_id}},
        {"$group": {"_id":"$genre", "count":{"$sum":1}}}
    ]
    output["pie"] = db.command('aggregate', 'series', pipeline=pipeline,
explain=False)["cursor"]["firstBatch"]
```

The aggregation with pymongo requires a defined pipeline which then has to be executed for a specific collection in the database. The output is in the **cursor** and **firstBatch** keys of the result dictionary. However, it was observed that this structure of the result dictionary is not the same for all versions of MongoDB, therefore, adjustments may need to be made.

- **series_id** (template series_id.html): simply displays information about the series and the list of comics belonging to the series in question, including a description when it is available
- **comic_id** (template comic_id.html): simply displays information about the comic in question, including a description when it is available

5.4. Templates

The **macros.html** template contains three functions:

- **render_form**: renders the form based on lists of labels and field names which are WTForm-formatted
- **render_table**: renders the tables of authors, series and comics. It treats regular links (a href) and redirections differently: if the variable starts with “redirect”, it links to a route, else it is a regular link to the outside world.
- **render_card**: renders a card with the information contained in most of the fields of an author, series or comic document. It takes a document and a dictionary of fields as

parameters. The dictionary of fields is defined in **dictionaries.html** where for each author, series and comic, pairs of desired keys and values are stored, in order to display the desired information with a nice formatting. It also renders a short description of a series or a comic when it is available in the database.

The **dictionaries.html** template contains some defined lists which are imported and used by other templates. The lists define the titles for the tables of authors, series and comics to be displayed. There are also dictionary containing the fields to be displayed by the **render_card** macro. Defining them here allows us to define them once for every single template which needs them. Therefore, the code is kept *DRY*.

The **base.html** template is used by all of the coming templates. It defines the head, i.e. the imports of CSS and JS libraries, the compatibility with mobile devices and also the navbar.

The **index.html** template defines the homepage. The three forms are loaded in most part with the **render_table** macro.

author.html, series.html and **comic.html** display the results of the search forms. **author_id.html, series_id.html, comic_id.html** display the individual pages of an author and of a series.