Contents

INTRODUCTION 1

Working of Project DOI 1

Requirements 2

Elements Of Bibliographical Data 2

Obtaining the Data 3

UML Class Diagram 3

Sequence Diagram 5

doi_app module 5

project_doi package 6

Subpackages 6

project_doi.scrapyspider package 6

Subpackages 6

project_doi.scrapyspider.spiders package 6

Submodules 6

project_doi.scrapyspider.spiders.ieee_doi module 6

project_doi.scrapyspider.spiders.springer_doi module 7

project_doi.scrapyspider.spiders.wiley_doi module 8

Module contents 9

Submodules 9

project_doi.scrapyspider.items module 9

project_doi.scrapyspider.middlewares module 10

project_doi.scrapyspider.pipelines module 11

project_doi.scrapyspider.settings module 11

Module contents 11

Submodules 11

project_doi.api module 11

project_doi.config module 11

project_doi.database module 12

project_doi.export module 13

project_doi.models module 13

project_doi.scrape module 15

Module contents 16

project_doi.scrapyspider package 16

Subpackages 16

project_doi.scrapyspider.spiders package 16

Submodules 16

project_doi.scrapyspider.spiders.ieee_doi module 16

project_doi.scrapyspider.spiders.springer_doi module 17

project_doi.scrapyspider.spiders.wiley_doi module 18

Module contents 18

Submodules 18
project_doi.scrapyspider.items module 18
project_doi.scrapyspider.middlewares module 20
project_doi.scrapyspider.pipelines module 21
project_doi.scrapyspider.settings module 21
Module contents 21
project_doi.scrapyspider.spiders package 21
Submodules 21
project_doi.scrapyspider.spiders.ieee_doi module 21
project_doi.scrapyspider.spiders.springer_doi module 22
project_doi.scrapyspider.spiders.wiley_doi module 22
Module contents 23
Indices and tables **23**
Index **25**
Python Module Index **29**

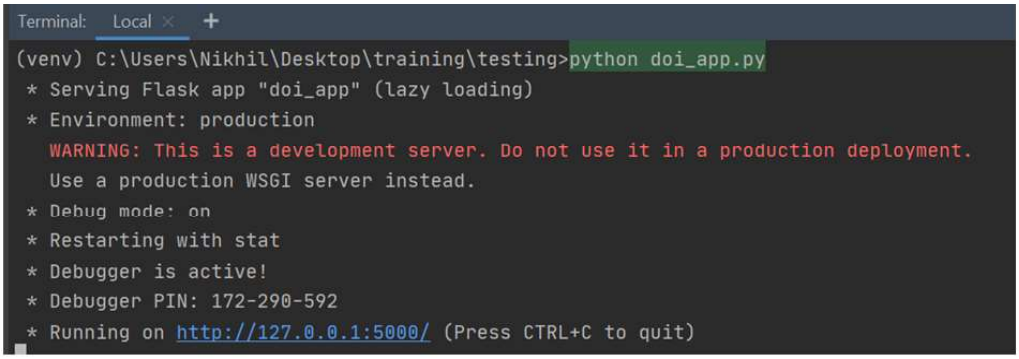
INTRODUCTION

The project is an api for extracting bibliographical data of publications published in ieee.org, springer.com, onlinewiley.com using Digital Object Identifier (DOI). Bibliographical data belonging to books, journal articles, book chapters and conference papers can be obtained using the respective DOI's. Users can input multiple DOI's, can upload csv and json files containing DOI's can be uploaded. Users can also download the bibliographical data in json and biibtex formats

Fig 1. User can upload files with DOI's or input DOI's

Working of Project DOI

Running the server : At first the project files should be pulled from the git repository to run the application. The link of the repository for Project DOI is https://mygit.th-deg.de/nb15766/project_doi_ Once the files are pulled to the local system the website should be made to a functioning site. For this we need to run the application locally in our personal computer or laptop. For this we need to access the local host and the website up and running. We need the local computer should be pre-installed with python, Flask and Scrapy. The commands can be access through command prompt or any command windows of choice. During the sample run we used command prompt. Once the files are obtained, run the command python doi_app.py after setting the correct path as shown in Fig 2.



Once the file path is obtained, we give the command to run the server which is “python doi_app.py”. If the command is executed successfully a development server is created with IP address http://127.0.0.1.:5000/ as shown in Fig. 2. This confirm that the website can be accessed through a web browser of choice through this IP address. the development server can be quitted by using CNTRL+C. The IP address is given in the browser which will direct to the home page as show in Fig 1

Requirements

- crochet==1.12.0
- Flask==1.1.2
- Scrapy==2.1.0
- SQLAlchemy==1.3.17
- bibtexparser~=1.1.0
- alembic~=1.4.2
- Flask-Classful==0.14.2
- Sphinx==3.1.2

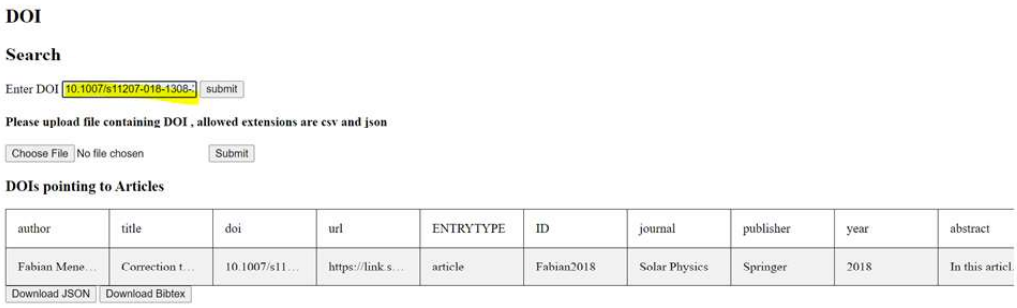
Elements Of Bibliographical Data

The following data for the different types of publication can be obtained.

BOOK	JOURNAL ARTICLE	CONFERENCE PAPER	CHAPTER
ENTRYTYPE	ENTRYTYPE	ENTRYTYPE	ENTRYTYPE
ID	ID	ID	ID
ABSTRACT	ABSTRACT	ABSTRACT	ABSTRACT
AUTHOR	AUTHOR	AUTHOR	AUTHOR
CHAPTERS	DOI	DOI	DOI
DOI	JOURNAL	PUBLISHER	PUBLISHER
ISBN	PUBLISHER	TIMESTAMP	TIMESTAMP
PUBLISHER	TIMESTAMP	TITLE	TITLE
TITLE	TITLE	URL	URL
URL	URL	YEAR	YEAR
	YEAR		

Obtaining the Data

Enter the DOI in the text area and press the submit button.



UML Class Diagram

In class diagrams, we work with the following elements: Class: A class represents a relevant concept from the domain, a set of persons, objects, or ideas that are depicted in our website.

Attribute: An attribute of a class represents a characteristic of a class that is of interest for the user of the website.

Operation: In a UML class diagram, we can add operations to classes and interfaces. An operation is a method or function that can be executed by an instance of a class or interface.

Visibility (+ and -): Use visibility markers to signify who can access the information contained within a class. Private visibility, denoted with (-) sign, hides information from anything outside the class partition. Public visibility, denoted with a (+) sign, allows all other classes to view the marked information.

Generalization (→): Generalization is a relationship between two classes. Is shows strong relation.

Association (■): An association indicates that objects of one class have a relationship with objects of another class.

Composition (Not-Shared Association): There is a strong lifecycle dependency between the two classes.

Multiplicity: A multiplicity allows for statements about the number of objects that are involved in an association.

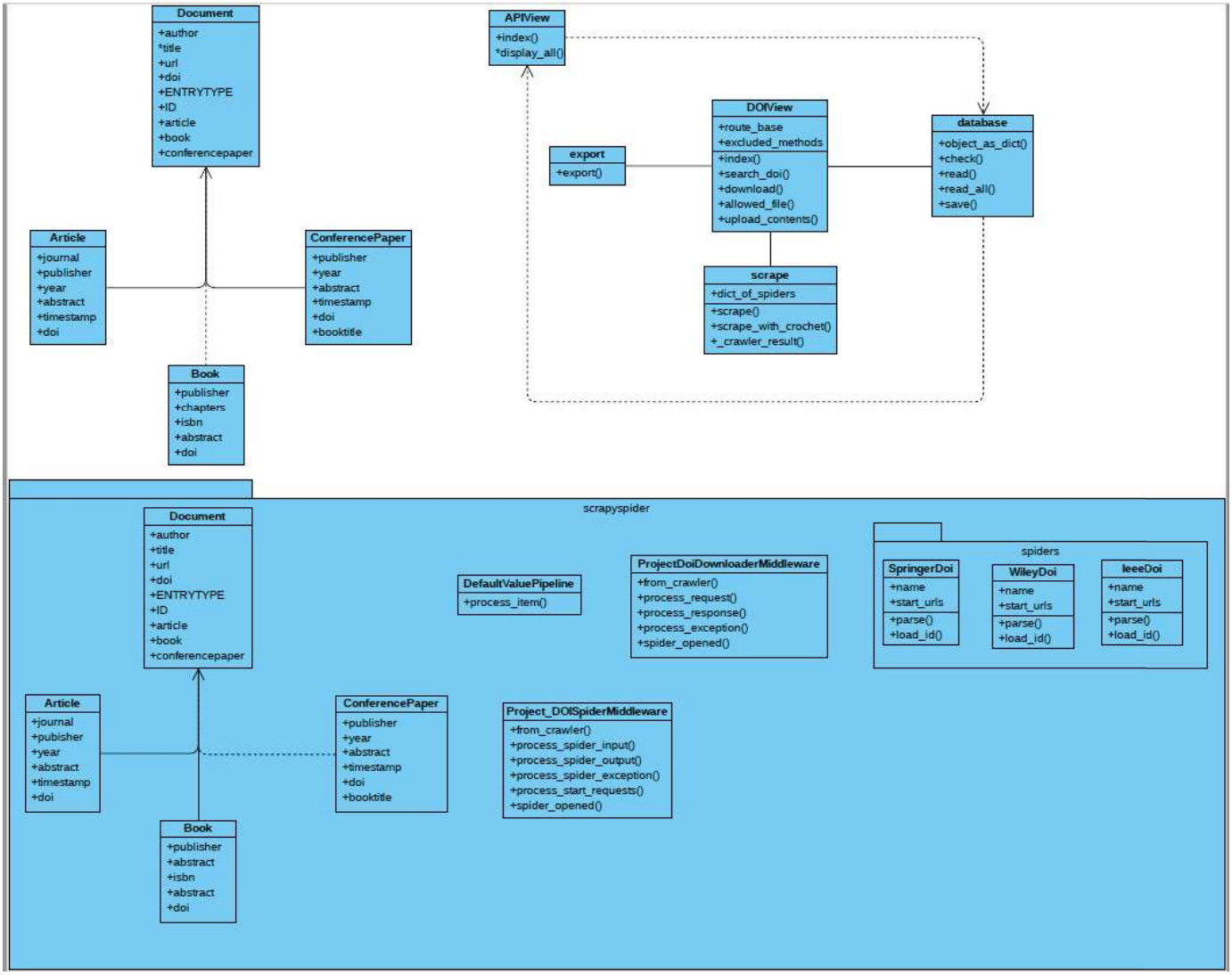


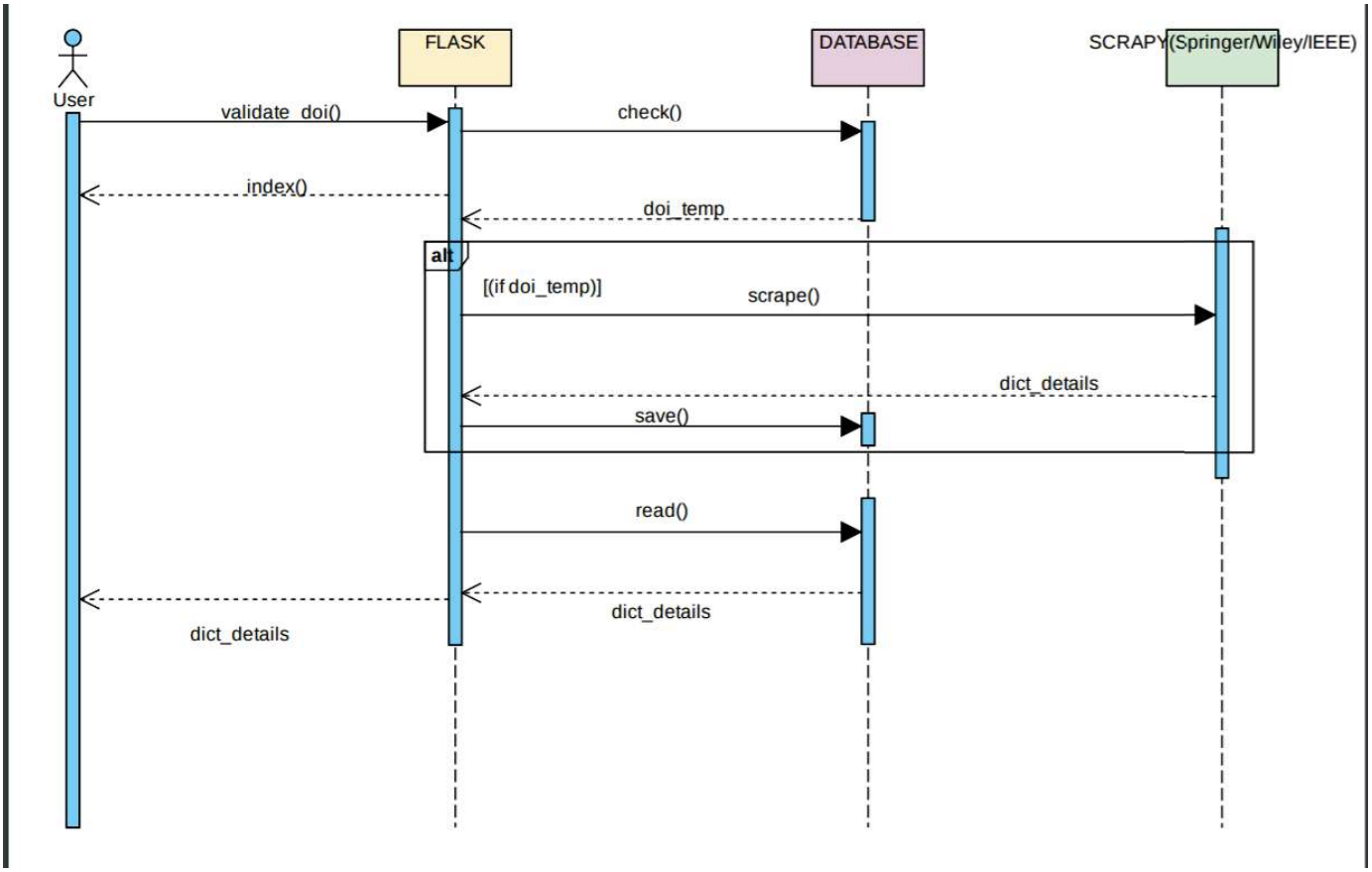
Fig above shows the complete UML class diagram for our project. Document class has three sub classes, Book, Article and ConferencePaper, which contain the rest of the bibliographical data, which are particular for the publications. DOIView Class is the core class of the project. The input is obtained from the user. After validating the input, the list of DOIs first searched in the database. If the DOI is present in the database, the data is read and displayed and made available for download in json and bibtex formats. DOI's can be uploaded by csv and json format. The DOI's are parsed from the file and converted into a list. After validating the DOI's the spiders are called one after another depending on the website of publication. If the DOI/s is not present in the database, the Scrape class object is initialized. Using the url obtained from the doi. get_real_url_from_doi(), the particular spider can be called . Currently bibliographical data of papers published by ieee, springer and wiley can be accessed. Consider some synchronous do-one-thing-after-the-other application code that wants to use event-driven Twisted-using code.We have two threads at a minimum: the application thread(s) and the reactor thread. There are also multiple layersnof code involved in this interaction

> Twisted code: Should only be called in reactor thread. This may be code from the Twisted package itself, or more likely code you have written that is built on top of Twisted. @wait_for/@run_in_reactor wrappers: The body of the functions runs in the reactor thread… but the caller should be in the application thread.The application code: Runs in the application thread(s), expects synchronous/blocking calls. dispatcher.connect will connect to the dispatcher that will kind of loop the code between these two functions.crawl_runner.crawl will connect to the our particular spider function based on the domain name in our scrapy file and after each yield will pass to the crawler_result function. The setting.py is applied to the crawl runner.

The data is extracted from the meta data present in the website. From springer and wiley , the data is obtained from the meta tags. For the papers published by ieee, the data is obtained from the javascript variable (global.document.metadata) present in the website using regex. ID is created by concatenating author and year.

Sequence Diagram

A sequence diagram is a graphic depiction of the interactions among the elements of our application. User is an actor. Activity of the actor can define by using use case. A user provides input DOI and the data is obtained by scraping if the input is a valid doi.



doi_app module

class doi_app.**DOIView**
Bases: **flask_classful.FlaskView**

Create application

- This creates an instance of the flask app and runs it
- Doi is entered by the user , as text or as csv or json files
- Check the validity of the doi
- Check whether doi is present in the database if not the respected spider is called based on the domain name

allowed_file (filename)

Check whether the file extensions present in the Allowed extensions.
Allowed extensions are json and csv.

Parameters: **filename** – the name of the uploaded file
Returns: true if the file type is present in the ALLOWED_EXTENSIONS object

base_args = *['/']*

download (file_bool)

Function to facilitate download.

Any existing temporary files in the download path is removed.

Parameters: **file_bool** – True if the requested file type is JSON , False for Bibtex

Returns: file

`excluded_methods` = *['allowed_file', 'upload_contents']*

`index` ()

By Default Flask will come into this when we run the file.

Returns: search_doi.html file in search folder in templates folder.

`route_base` = *'/'*

`search_doi` ()

Get Input from the user , validate and return the bibliographical details
After clicking the submit button or search button , flask comes here. The values of DOI ‘s are obtained from the user either as string separated by comma or as a json or csv file. Uploaded files are saved in the Upload folder. The DOI ‘s are parsed and saved as a list , removed the duplicate ones. Validated the DOI ‘s by checking the correct format of DOI provided by DOI.org . The url link is obtained from doi.get_url_from_doi(doi). Check the database for the details for each doi. If DOI ‘s are not present in the database, the domains are saved as a list and Scrape object is called. The data corresponds to the DOI ‘s are obtained.

Returns: html page containing the bibliographical data

`upload_contents` (extension, path)

DOI ‘s returned as list after parsing the uploaded files.
Allowed extensions are json and csv. The file is removed after parsing the contents.

Parameters:

- **extension** – The file type of the uploaded file, csv or json
- **path** – path of the uploaded file

Returns: the parsed contents of the file as a list

project_doi package

Subpackages

project_doi.scrapyspider package

Subpackages

project_doi.scrapyspider.spiders package

Submodules

project_doi.scrapyspider.spiders.ieee_doi module

class `project_doi.scrapyspider.spiders.ieee_doi.`**`IeeeDoi`** (*args, **kwargs)

Bases: `scrapy.spiders.Spider`

Spider class implementation for ieeexplore.ieee.org

`name` = *'ieee'*

`parse` (response)

Parse the page

- The type of the publication is found out from meta tag og:type.
- The fields are extracted from the web-page from javascript variable global.document.metadata, selector is the response object itself and loaded into Article or Book or ConferencePaper Item depend on the contentType
- The javascript variable is extracted using regex r"global.document.metadata=(.+?);" and saved as a json object
- title,
- author,
- Journal,
- publisher,
- year,
- abstract,
- doi,
- timestamp,
- url,
- booktitle,
- ENTRYTYPE,
- ID, (The ID populated from the function load_id))

Returns: Itemloader (item= Article or Conference paper depending on the type)

`project_doi.scrapyspider.spiders.ieee_doi.`**`load_id`** (loader)

ID is created by adding author and year

Parameters: **loader** – ItemLoader

Returns: ID (first name of author+year)

project_doi.scrapyspider.spiders.springer_doi module

class `project_doi.scrapyspider.spiders.springer_doi.`**`SpringerDoi`** (*args, **kwargs)

Bases: **`scrapy.spiders.Spider`**

Spider class implementation for link.springer.com

`name` = *'springer'*

`parse` (response)

Parse the page.

- The type of the publication is found out from meta tag og:type
- The fields are extracted from the web-page from meta tag , selector is the response object itself and loaded into Article Item
- title,(//div[@class='page-title']/h1/text())
- author, (//span[@class='authors-affiliations__name']/text())
- Journal,
- publisher,(//span[@id='publisher-name']/text())
- chapters, (//span[@class='c-tabs__deemphasize']/text())
- year, (//meta[@name='citation_publication_date']/@content)

- abstract, (//meta[@name='description']/@content)
- doi, (//input[@name='doi']/@value)
- timestamp, (//meta[@name='citation_publication_date']/@content)
- url, (//meta[@property='og:url']/@content)
- booktitle, (//meta[@name='citation_inbook_title']/@content)
- ENTRYTYPE, (//meta[@property='og:type']/@content)
- ID, (The ID populated from the function load_id))

Returns: Itemloader (item= Article or Conference paper depending on the type)

project_doi.scrapyspider.spiders.springer_doi.**load_id** (loader)

ID is created by concatenating author and year.

Parameters: **loader** – ItemLoader

Returns: ID (first name of author+year)

project_doi.scrapyspider.spiders.wiley_doi module

class project_doi.scrapyspider.spiders.wiley_doi.**WileyDoi** (*args, **kwargs)

Bases: **scrapy.spiders.Spider**

Spider class implementation for onlinelibrary.wiley.com

name = *'wiley'*

parse (response)

Parse the page.

- The type of the publication is found out from meta tag og:type
- The fields are extracted from the web-page from meta tag selector is the response object itself and loaded into Article Item
- title, (//meta[@name='citation_title']/@content)
- author, (//meta[@name='citation_author']/@content)
- author, (//*[@id='a1_Ctrl']/span/text())
- journal, (//meta[@name='citation_journal_title']/@content)
- publisher, (//meta[@name='citation_publisher']/@content)
- year, (//meta[@name='citation_online_date']/@content)
- abstract, (normalize-space(//meta[@property='og:description']/@content))
- doi, (//meta[@name='citation_doi']/@content)
- timestamp, (//meta[@name='citation_online_date']/@content)
- url, (//meta[@property='og:url']/@content)
- booktitle, (//meta[@name='citation_book_title']/@content)
- ENTRYTYPE, (//meta[@property='og:type']/@content)
- ID, (The ID populated from the function load_id))

Returns: Itemloader (item= Article or Conference paper depending on the type)

project_doi.scrapyspider.spiders.wiley_doi.**load_id** (loader)

ID is created by concatenating author and year.

Parameters: **loader** – ItemLoader

Returns: ID (first name of author+year)

Module contents

Submodules

project_doi.scrapyspider.items module

class `project_doi.scrapyspider.items.`**`Article`**(*args, **kwargs)
Bases: **`project_doi.scrapyspider.items.Document`**
Class that represents fields for the item corresponding to an article.
Inherited from class Document. The following attributes of a DOI are stored in this table:

- journal (the name of the journal in which the article is published)
- publisher (name of the publisher of the article)
- year (the year of publication)
- abstract (the abstract of the publication)
- timestamp (the published date of the article)

`fields` = *{'ENTRYTYPE': {'input_processor': <scrapy.loader.processors.MapCompose object>}, 'ID': {}, 'abstract': {}, 'author': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.Join object>}, 'doi': {}, 'journal': {}, 'publisher': {'output_processor': <scrapy.loader.processors.Join object>}, 'timestamp': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.TakeFirst object>}, 'title': {}, 'url': {}, 'year': {'input_processor': <scrapy.loader.processors.MapCompose object>}}*

class `project_doi.scrapyspider.items.`**`Book`**(*args, **kwargs)
Bases: **`project_doi.scrapyspider.items.Document`**
Class that represents fields for the item corresponding to a book.
Inherited from class Document. The following attributes of a DOI are stored in this table:

- publisher (name of the publisher of the article)
- chapters (the number of chapters included in the book)
- isbn (the isbn of the book)
- abstract (the abstract of the publication)

`fields` = *{'ENTRYTYPE': {'input_processor': <scrapy.loader.processors.MapCompose object>}, 'ID': {}, 'ISBN': {}, 'abstract': {}, 'author': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.Join object>}, 'chapters': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.TakeFirst object>}, 'doi': {}, 'publisher': {'output_processor': <scrapy.loader.processors.Join object>}, 'title': {}, 'url': {}}*

class `project_doi.scrapyspider.items.`**`ConferencePaper`**(*args, **kwargs)
Bases: **`project_doi.scrapyspider.items.Document`**
Class that represents fields for the item corresponding to a Conference paper.

Inherited from class Document. The following attributes of a DOI are stored in this table:

- booktitle (the name of the journal in which the article is published)
- publisher (name of the publisher of the article)
- year (the year of publication)
- abstract (the abstract of the publication)
- timestamp (the published date of the article)

fields = *{'ENTRYTYPE': {'input_processor': <scrapy.loader.processors.MapCompose object>}, 'ID': {}, 'abstract': {'input_processor': <scrapy.loader.processors.MapCompose object>}, 'author': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.Join object>}, 'booktitle': {}, 'doi': {}, 'publisher': {'output_processor': <scrapy.loader.processors.Join object>}, 'timestamp': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.TakeFirst object>}, 'title': {}, 'url': {}, 'year': {'input_processor': <scrapy.loader.processors.MapCompose object>}}*

class project_doi.scrapyspider.items.**Document** (*args, **kwargs)
Bases: **scrapy.item.Item**

Defines your fields for Item

- author (author of the publication)
- title (title of the publication)
- doi (DOI of the publication)
- url (the resolved URL)
- ENTRYTYPE (The type of article, book, article, paper(Conference Paper)
- ID (ID created using the first name of an author and the year of publication or in case of book the name of the author)

fields = *{'ENTRYTYPE': {'input_processor': <scrapy.loader.processors.MapCompose object>}, 'ID': {}, 'author': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.Join object>}, 'doi': {}, 'title': {}, 'url': {}}*

project_doi.scrapyspider.items.**date_convert** (value)

The date is parsed to python datetime (default: current time)

Parameters: **value** – date as string
Returns: datetime.datetime

project_doi.scrapyspider.items.**filter_number** (value)

The digits are extracted for chapters

Parameters: **value** – string
Returns: integer

project_doi.scrapyspider.middlewares module

class project_doi.scrapyspider.middlewares.**MyprojectspDownloaderMiddleware**
Bases: **object**

classmethod **from_crawler** (crawler)

process_exception (request, exception, spider)

process_request (request, spider)

process_response (request, response, spider)

spider_opened (spider)

class project_doi.scrapyspider.middlewares.**MyprojectspSpiderMiddleware**
Bases: **object**

classmethod **from_crawler** (crawler)

process_spider_exception (response, exception, spider)

process_spider_input (response, spider)

process_spider_output (response, result, spider)

process_start_requests (start_requests, spider)

spider_opened (spider)

project_doi.scrapyspider.pipelines module

class project_doi.scrapyspider.pipelines.**DefaultValuesPipeline**
Bases: **object**

process_item (item, spider)

project_doi.scrapyspider.settings module

Module contents

Submodules

project_doi.api module

class project_doi.api.**ApiView**
Bases: **flask_classful.FlaskView**

Create application
This creates an instance of the ApiView and runs it

base_args = *['/api/']*

display_all ()

Displays the data returned as a json_object.

Returns: json

index ()

By Default Flask will come into this when we run the file.

- The can be accessed by 125.0.0.5000/api?doi=
- User can either search for a particular doi or can get all the data available in the database

route_base = *None*

project_doi.config module

class project_doi.config.**Config**
Bases: **object**

Config
Contains all the configurations for flask:

- SQLALCHEMY_DATABASE_URI = 'sqlite:///app.db'
- SQLALCHEMY_TRACK_MODIFICATIONS = False
- SECRET_KEY = 'dev'

- ALLOWED_EXTENSIONS = {‘csv’, ‘json’}
- DICT_OF_SPIDERS = {‘springer’: SpringerDoi, ‘wiley’: WileyDoi, ‘ieee’: IeeeDoi}
- UPLOAD_FOLDER = os.path.join(basedir, ‘uploads’)

ALLOWED_EXTENSIONS = *{'csv', 'json'}*

DICT_OF_SPIDERS = *{'ieee': <class 'project_doi.scrapyspider.spiders.ieee_doi.IeeeDoi'>, 'springer': <class 'project_doi.scrapyspider.spiders.springer_doi.SpringerDoi'>, 'wiley': <class 'project_doi.scrapyspider.spiders.wiley_doi.WileyDoi'>}*

SECRET_KEY = *'dev'*

SQLALCHEMY_DATABASE_URI = *'sqlite:///app.db'*

SQLALCHEMY_TRACK_MODIFICATIONS = *False*

UPLOAD_FOLDER = *'C:\\Users\\Nikhil\\Desktop\\training\\testing\\project_doi\\uploads'*

project_doi.database module

project_doi.database.**check**(dois)

Check Table Document for DOI ‘s and returns the list of DOI ‘s which are not present.

Parameters: **dois** – the list of DOI ‘s entered by user

Return doi_temp: the list of DOI ‘s not present in the database

project_doi.database.**object_as_dict**(obj)

The rows returned from table converted to dictionary with keys as column heading.

Parameters: **obj** – rows fron table as object

Returns: dictionary of rows

project_doi.database.**read**(dois)

Go through each of the tables and returns the bibliographical data as a dictionary.
The rows returned from each table are saved to keys in the dictionary corresponding to the their entry type or the table name

Parameters: **dois** – the list of dois’ entered by the user

Return out_db: the dictionary of bibliographical data with respect to the list of details. the dictionary has three keys, book, article, paper . Each of them pointing to list of bibliographical data based on the type of articles(ENTRYTYPE).

project_doi.database.**read_all**()

To read all the entries present in the database.

Returns: dictionary containing each rows , separated based on the type of the entry

project_doi.database.**save**(item)

Function to write the scraped data to the database.

- The item is saved according to the ENTRYTYPE.
- The item with ENTRYTYPE as book is saved to the Book
- The item with ENTRYTYPE as paper or chapter is saved to the ConferencePaper
- The item with ENTRYTYPE other than the above mentioned are saved to article

Parameters: **item** – Item retrieved using scrapy

project_doi.export module

project_doi.export.**export** (data, file_bool)

Convert the data to json or bibtex and write to a temporary file.

Parameters:
- **data** – The data containing the bibliographical details
- **file_bool** – True for the file to be exported as json False for bibtex format

Returns: the filename and the mimetype

project_doi.models module

class project_doi.models.**Article** (**kwargs)
Bases: **project_doi.models.Document**

Class that represents DOI details corresponding to an article.
Inherited from class Document. The following attributes of a DOI are stored in this table:

- journal (the name of the journal in which the article is published)
- publisher (name of the publisher of the article)
- year (the year of publication)
- abstract (the abstract of the publication)
- timestamp (the published date of the article)

ENTRYTYPE

ID

abstract

article

author

book

conferencepaper

doi

journal

publisher

timestamp

title

url

year

class project_doi.models.**Book** (**kwargs)
Bases: **project_doi.models.Document**

Class that represents DOI details corresponding to a book.
Inherited from class Document. The following attributes of a DOI are stored in this table:

- publisher (name of the publisher of the article)
- chapters (the number of chapters included in the book)
- isbn (the isbn of the book)
- abstract (the abstract of the publication)

ENTRYTYPE

ID

abstract

article

author

book

chapters

conferencepaper

document

doi

isbn

publisher

title

url

class project_doi.models.**ConferencePaper** (**kwargs)

Bases: **project_doi.models.Document**

Class that represents DOI details corresponding to a Conference paper.

Inherited from class Document. The following attributes of a DOI are stored in this table:

- booktitle (the name of the journal in which the article is published)
- publisher (name of the publisher of the article)
- year (the year of publication)
- abstract (the abstract of the publication)
- timestamp (the published date of the article)

ENTRYTYPE

ID

abstract

article

author

book

booktitle

conferencepaper

document

doi

publisher

timestamp

title

url

year

class project_doi.models.**Document** (**kwargs)
Bases: **sqlalchemy.ext.declarative.api.Model**

Class that represents basic DOI details

The following attributes of a DOI are stored in this table:

- author (author of the publication)
- title (title of the publication)
- doi (DOI of the publication)
- url (the resolved URL)
- ENTRYTYPE (The type of article, book, article, paper(Conference Paper)
- ID (ID created using the first name of an author and the year of publication or in case of book the name of the author)

ENTRYTYPE

ID

article

author

book

conferencepaper

doi

title

url

project_doi.scrape module

class project_doi.scrape.**Scrape**
Bases: **object**

Class that represents connection between flask and scrapy.

- run crawler in twisted reactor synchronously
- Initialize CrawlRunner()

crawl_runner = <*scrapy.crawler.CrawlerRunner object*>

dict_of_spiders = *{}*

scrape (domain, dict_of_spiders)

run crawler in twisted reactor synchronously.

Parameters:
- **domain** – the list of domains
- **dict_of_spiders:{'springer'** – SpringerDoi, 'wiley': WileyDoi, 'ieee': IeeeDoi}

scrape_with_crochet (domain)

signal fires when single item is processed and calls _crawler_result to save that item.
Consider some synchronous do-one-thing-after-the-other application code that wants to use event-driven Twisted-using code. We have two threads at a minimum: the application thread(s) and the reactor thread. There are also multiple layers of code involved in this interaction
Twisted code: Should only be called in reactor thread. This may be code from the Twisted package itself, or more likely code you have written that is built on top of Twisted.
@wait_for/@run_in_reactor wrappers: The body of the functions runs in the reactor thread… but the caller should be in the application thread.
The application code: Runs in the application thread(s), expects synchronous/blocking calls. dispatcher.connect will connect to the dispatcher that will kind of loop the code between these two functions. crawl_runner.crawl will connect to the our particular spider function based on the domain name, in our scrapy file and after each yield will pass to the crawler_result function. The setting.py is applied to the crawl runner.

Parameters: **domain** – the domain to crawl
Returns: a twisted.internet.defer.Deferred

Module contents

project_doi.scrapyspider package

Subpackages

project_doi.scrapyspider.spiders package

Submodules

project_doi.scrapyspider.spiders.ieee_doi module

class project_doi.scrapyspider.spiders.ieee_doi.**IeeeDoi** (*args, **kwargs)
Bases: **scrapy.spiders.Spider**
Spider class implementation for ieeexplore.ieee.org

name = *'ieee'*

parse (response)

Parse the page

- The type of the publication is found out from meta tag og:type.
- The fields are extracted from the web-page from javascript variable global.document.metadata, selector is the response object itself and loaded into Article or Book or ConferencePaper Item depend on the contentType
- The javascript variable is extracted using regex r"global.document.metadata=(.+?);" and saved as a json object

- title,
- author,
- Journal,
- publisher,
- year,
- abstract,
- doi,
- timestamp,
- url,
- booktitle,
- ENTRYTYPE,
- ID, (The ID populated from the function load_id))

Returns: Itemloader (item= Article or Conference paper depending on the type)

project_doi.scrapyspider.spiders.ieee_doi.**load_id** (loader)

ID is created by adding author and year

Parameters: **loader** – ItemLoader

Returns: ID (first name of author+year)

project_doi.scrapyspider.spiders.springer_doi module

class project_doi.scrapyspider.spiders.springer_doi.**SpringerDoi** (*args, **kwargs)

Bases: **scrapy.spiders.Spider**

Spider class implementation for link.springer.com

name = *'springer'*

parse (response)

Parse the page.

- The type of the publication is found out from meta tag og:type
- The fields are extracted from the web-page from meta tag , selector is the response object itself and loaded into Article Item
- title,(//div[@class='page-title']/h1/text())
- author, (//span[@class='authors-affiliations__name']/text())
- Journal,
- publisher,(//span[@id='publisher-name']/text())
- chapters, (//span[@class='c-tabs__deemphasize']/text())
- year, (//meta[@name='citation_publication_date']/@content)
- abstract, (//meta[@name='description']/@content)
- doi, (//input[@name='doi']/@value)
- timestamp, (//meta[@name='citation_publication_date']/@content)
- url, (//meta[@property='og:url']/@content)
- booktitle, (//meta[@name='citation_inbook_title']/@content)
- ENTRYTYPE, (//meta[@property='og:type']/@content)

- ID, (The ID populated from the function load_id))

Returns: Itemloader (item= Article or Conference paper depending on the type)

project_doi.scrapyspider.spiders.springer_doi.**load_id** (loader)

ID is created by concatenating author and year.

Parameters: **loader** – ItemLoader

Returns: ID (first name of author+year)

project_doi.scrapyspider.spiders.wiley_doi module

class project_doi.scrapyspider.spiders.wiley_doi.**WileyDoi** (*args, **kwargs)

Bases: **scrapy.spiders.Spider**

Spider class implementation for onlinelibrary.wiley.com

name = *'wiley'*

parse (response)

Parse the page.

- The type of the publication is found out from meta tag og:type
- The fields are extracted from the web-page from meta tag selector is the response object itself and loaded into Article Item
- title, (//meta[@name='citation_title']/@content)
- author, (//meta[@name='citation_author']/@content)
- author, (//*[@id='a1_Ctrl']/span/text())
- journal, (//meta[@name='citation_journal_title']/@content)
- publisher, (//meta[@name='citation_publisher']/@content)
- year, (//meta[@name='citation_online_date']/@content)
- abstract, (normalize-space(//meta[@property='og:description']/@content))
- doi, (//meta[@name='citation_doi']/@content)
- timestamp, (//meta[@name='citation_online_date']/@content)
- url, (//meta[@property='og:url']/@content)
- booktitle, (//meta[@name='citation_book_title']/@content)
- ENTRYTYPE, (//meta[@property='og:type']/@content)
- ID, (The ID populated from the function load_id))

Returns: Itemloader (item= Article or Conference paper depending on the type)

project_doi.scrapyspider.spiders.wiley_doi.**load_id** (loader)

ID is created by concatenating author and year.

Parameters: **loader** – ItemLoader

Returns: ID (first name of author+year)

Module contents

Submodules

project_doi.scrapyspider.items module

class project_doi.scrapyspider.items.**Article** (*args, **kwargs)
Bases: **project_doi.scrapyspider.items.Document**
Class that represents fields for the item corresponding to an article.
Inherited from class Document. The following attributes of a DOI are stored in this table:

- journal (the name of the journal in which the article is published)
- publisher (name of the publisher of the article)
- year (the year of publication)
- abstract (the abstract of the publication)
- timestamp (the published date of the article)

fields = *{'ENTRYTYPE': {'input_processor': <scrapy.loader.processors.MapCompose object>}, 'ID': {}, 'abstract': {}, 'author': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.Join object>}, 'doi': {}, 'journal': {}, 'publisher': {'output_processor': <scrapy.loader.processors.Join object>}, 'timestamp': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.TakeFirst object>}, 'title': {}, 'url': {}, 'year': {'input_processor': <scrapy.loader.processors.MapCompose object>}}*

class project_doi.scrapyspider.items.**Book** (*args, **kwargs)
Bases: **project_doi.scrapyspider.items.Document**
Class that represents fields for the item corresponding to a book.
Inherited from class Document. The following attributes of a DOI are stored in this table:

- publisher (name of the publisher of the article)
- chapters (the number of chapters included in the book)
- isbn (the isbn of the book)
- abstract (the abstract of the publication)

fields = *{'ENTRYTYPE': {'input_processor': <scrapy.loader.processors.MapCompose object>}, 'ID': {}, 'ISBN': {}, 'abstract': {}, 'author': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.Join object>}, 'chapters': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.TakeFirst object>}, 'doi': {}, 'publisher': {'output_processor': <scrapy.loader.processors.Join object>}, 'title': {}, 'url': {}}*

class project_doi.scrapyspider.items.**ConferencePaper** (*args, **kwargs)
Bases: **project_doi.scrapyspider.items.Document**
Class that represents fields for the item corresponding to a Conference paper.

Inherited from class Document. The following attributes of a DOI are stored in this table:

- booktitle (the name of the journal in which the article is published)
- publisher (name of the publisher of the article)
- year (the year of publication)
- abstract (the abstract of the publication)
- timestamp (the published date of the article)

fields = *{'ENTRYTYPE': {'input_processor': <scrapy.loader.processors.MapCompose object>}, 'ID': {}, 'abstract': {'input_processor': <scrapy.loader.processors.MapCompose object>}, 'author': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.Join object>}, 'booktitle': {}, 'doi': {}, 'publisher': {'output_processor': <scrapy.loader.processors.Join object>}, 'timestamp': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.TakeFirst object>}, 'title': {}, 'url': {}, 'year': {'input_processor': <scrapy.loader.processors.MapCompose object>}}*

class project_doi.scrapyspider.items.**Document** (*args, **kwargs)

Bases: `scrapy.item.Item`

Defines your fields for Item

- author (author of the publication)
- title (title of the publication)
- doi (DOI of the publication)
- url (the resolved URL)
- ENTRYTYPE (The type of article, book, article, paper(Conference Paper)
- ID (ID created using the first name of an author and the year of publication or in case of book the name of the author)

fields = *{'ENTRYTYPE': {'input_processor': <scrapy.loader.processors.MapCompose object>}, 'ID': {}, 'author': {'input_processor': <scrapy.loader.processors.MapCompose object>, 'output_processor': <scrapy.loader.processors.Join object>}, 'doi': {}, 'title': {}, 'url': {}}*

project_doi.scrapyspider.items.**date_convert** (value)

The date is parsed to python datetime (default: current time)

Parameters: **value** – date as string

Returns: datetime.datetime

project_doi.scrapyspider.items.**filter_number** (value)

The digits are extracted for chapters

Parameters: **value** – string

Returns: integer

project_doi.scrapyspider.middlewares module

class project_doi.scrapyspider.middlewares.**MyprojectspDownloaderMiddleware**

Bases: `object`

classmethod **from_crawler** (crawler)

process_exception (request, exception, spider)

process_request (request, spider)

process_response (request, response, spider)

spider_opened (spider)

class project_doi.scrapyspider.middlewares.**MyprojectspSpiderMiddleware**

Bases: `object`

classmethod **from_crawler** (crawler)

process_spider_exception (response, exception, spider)

process_spider_input (response, spider)

process_spider_output (response, result, spider)

process_start_requests (start_requests, spider)

spider_opened (spider)

project_doi.scrapyspider.pipelines module

class project_doi.scrapyspider.pipelines.**DefaultValuesPipeline**
Bases: **object**

process_item (item, spider)

project_doi.scrapyspider.settings module

Module contents

project_doi.scrapyspider.spiders package

Submodules

project_doi.scrapyspider.spiders.ieee_doi module

class project_doi.scrapyspider.spiders.ieee_doi.**IeeeDoi** (*args, **kwargs)
Bases: **scrapy.spiders.Spider**
Spider class implementation for ieeexplore.ieee.org

name = *'ieee'*

parse (response)

Parse the page

- The type of the publication is found out from meta tag og:type.
- The fields are extracted from the web-page from javascript variable global.document.metadata, selector is the response object itself and loaded into Article or Book or ConferencePaper Item depend on the contentType
- The javascript variable is extracted using regex r"global.document.metadata=(.+?);" and saved as a json object
- title,
- author,
- Journal,
- publisher,
- year,
- abstract,
- doi,
- timestamp,
- url,
- booktitle,
- ENTRYTYPE,
- ID, (The ID populated from the function load_id))

Returns: Itemloader (item= Article or Conference paper depending on the type)

project_doi.scrapyspider.spiders.ieee_doi.**load_id** (loader)

ID is created by adding author and year

Parameters: **loader** – ItemLoader

Returns: ID (first name of author+year)

project_doi.scrapyspider.spiders.springer_doi module

class project_doi.scrapyspider.spiders.springer_doi.**SpringerDoi** (*args, **kwargs)

Bases: **scrapy.spiders.Spider**

Spider class implementation for link.springer.com

name = *'springer'*

parse (response)

Parse the page.

- The type of the publication is found out from meta tag og:type
- The fields are extracted from the web-page from meta tag , selector is the response object itself and loaded into Article Item
- title,(//div[@class='page-title']/h1/text())
- author, (//span[@class='authors-affiliations__name']/text())
- Journal,
- publisher,(//span[@id='publisher-name']/text())
- chapters, (//span[@class='c-tabs__deemphasize']/text())
- year, (//meta[@name='citation_publication_date']/@content)
- abstract, (//meta[@name='description']/@content)
- doi, (//input[@name='doi']/@value)
- timestamp, (//meta[@name='citation_publication_date']/@content)
- url, (//meta[@property='og:url']/@content)
- booktitle, (//meta[@name='citation_inbook_title']/@content)
- ENTRYTYPE, (//meta[@property='og:type']/@content)
- ID, (The ID populated from the function load_id))

Returns: Itemloader (item= Article or Conference paper depending on the type)

project_doi.scrapyspider.spiders.springer_doi.**load_id** (loader)

ID is created by concatenating author and year.

Parameters: **loader** – ItemLoader

Returns: ID (first name of author+year)

project_doi.scrapyspider.spiders.wiley_doi module

class project_doi.scrapyspider.spiders.wiley_doi.**WileyDoi** (*args, **kwargs)

Bases: **scrapy.spiders.Spider**

Spider class implementation for onlinelibrary.wiley.com

name = *'wiley'*

parse (response)

Parse the page.

- The type of the publication is found out from meta tag og:type
- The fields are extracted from the web-page from meta tag selector is the response object itself and loaded into Article Item
- title, (//meta[@name='citation_title']/@content)
- author, (//meta[@name='citation_author']/@content)
- author, (//*[@id='a1_Ctrl']/span/text())
- journal, (//meta[@name='citation_journal_title']/@content)
- publisher, (//meta[@name='citation_publisher']/@content)
- year, (//meta[@name='citation_online_date']/@content)
- abstract, (normalize-space(//meta[@property='og:description']/@content))
- doi, (//meta[@name='citation_doi']/@content)
- timestamp, (//meta[@name='citation_online_date']/@content)
- url, (//meta[@property='og:url']/@content)
- booktitle, (//meta[@name='citation_book_title']/@content)
- ENTRYTYPE, (//meta[@property='og:type']/@content)
- ID, (The ID populated from the function load_id))

Returns: Itemloader (item= Article or Conference paper depending on the type)

`project_doi.scrapyspider.spiders.wiley_doi.`**`load_id`** (loader)

ID is created by concatenating author and year.

Parameters: **loader** – ItemLoader

Returns: ID (first name of author+year)

Module contents

Indices and tables

- **genindex**
- **modindex**
- **search**

Index

A

abstract (project_doi.models.Article attribute)

(project_doi.models.Book attribute)

(project_doi.models.ConferencePaper attribute)

ALLOWED_EXTENSIONS (project_doi.config.Config attribute)

allowed_file() (doi_app.DOIView method)

ApiView (class in project_doi.api)

Article (class in project_doi.models)

(class in project_doi.scrapyspider.items)

article (project_doi.models.Article attribute)

(project_doi.models.Book attribute)

(project_doi.models.ConferencePaper attribute)

(project_doi.models.Document attribute)

author (project_doi.models.Article attribute)

(project_doi.models.Book attribute)

(project_doi.models.ConferencePaper attribute)

(project_doi.models.Document attribute)

B

base_args (doi_app.DOIView attribute)

(project_doi.api.ApiView attribute)

Book (class in project_doi.models)

(class in project_doi.scrapyspider.items)

book (project_doi.models.Article attribute)

(project_doi.models.Book attribute)

(project_doi.models.ConferencePaper attribute)

(project_doi.models.Document attribute)

booktitle (project_doi.models.ConferencePaper attribute)

C

chapters (project_doi.models.Book attribute)

check() (in module project_doi.database)

ConferencePaper (class in project_doi.models)

(class in project_doi.scrapyspider.items)

conferencepaper (project_doi.models.Article attribute)

(project_doi.models.Book attribute)

(project_doi.models.ConferencePaper attribute)

(project_doi.models.Document attribute)

Config (class in project_doi.config)

crawl_runner (project_doi.scrape.Scrape attribute)

D

date_convert() (in module project_doi.scrapyspider.items)

DefaultValuesPipeline (class in project_doi.scrapyspider.pipelines)

DICT_OF_SPIDERS (project_doi.config.Config attribute)

dict_of_spiders (project_doi.scrape.Scrape attribute)

display_all() (project_doi.api.ApiView method)

Document (class in project_doi.models)

(class in project_doi.scrapyspider.items)

document (project_doi.models.Book attribute)

(project_doi.models.ConferencePaper attribute)

doi (project_doi.models.Article attribute)

(project_doi.models.Book attribute)

(project_doi.models.ConferencePaper attribute)

(project_doi.models.Document attribute)

doi_app

module

DOIView (class in doi_app)

download() (doi_app.DOIView method)

E

ENTRYTYPE (project_doi.models.Article attribute)

(project_doi.models.Book attribute)

(project_doi.models.ConferencePaper attribute)

(project_doi.models.Document attribute)

excluded_methods (doi_app.DOIView attribute)

export() (in module project_doi.export)

F

fields (project_doi.scrapyspider.items.Article attribute)

(project_doi.scrapyspider.items.Book attribute)

(project_doi.scrapyspider.items.ConferencePaper attribute)

(project_doi.scrapyspider.items.Document attribute)

filter_number() (in module project_doi.scrapyspider.items)

from_crawler() (project_doi.scrapyspider.middlewares.MyprojectspDownloaderMiddleware class method)

(project_doi.scrapyspider.middlewares.MyprojectspSpiderMiddleware class method)

I

ID (project_doi.models.Article attribute)

(project_doi.models.Book attribute)

(project_doi.models.ConferencePaper attribute)

(project_doi.models.Document attribute)

IeeeDoi (class in project_doi.scrapyspider.spiders.ieee_doi)

index() (doi_app.DOIView method)

(project_doi.api.ApiView method)

isbn (project_doi.models.Book attribute)

J

journal (project_doi.models.Article attribute)

L

load_id() (in module project_doi.scrapyspider.spiders.ieee_doi)

(in module project_doi.scrapyspider.spiders.springer_doi)

(in module project_doi.scrapyspider.spiders.wiley_doi)

M

module

doi_app

project_doi

project_doi.api

project_doi.config

project_doi.database

project_doi.export

project_doi.models

project_doi.scrape

project_doi.scrapyspider

project_doi.scrapyspider.items

project_doi.scrapyspider.middlewares

project_doi.scrapyspider.pipelines

project_doi.scrapyspider.settings

project_doi.scrapyspider.spiders

project_doi.scrapyspider.spiders.ieee_doi

project_doi.scrapyspider.spiders.springer_doi

project_doi.scrapyspider.spiders.wiley_doi

MyprojectspDownloaderMiddleware (class in project_doi.scrapyspider.middlewares)

MyprojectspSpiderMiddleware (class in project_doi.scrapyspider.middlewares)

N

name (project_doi.scrapyspider.spiders.ieee_doi.IeeeDoi attribute)

(project_doi.scrapyspider.spiders.springer_doi.SpringerDoi attribute)

(project_doi.scrapyspider.spiders.wiley_doi.WileyDoi attribute)

O

object_as_dict() (in module project_doi.database)

P

parse() (project_doi.scrapyspider.spiders.ieee_doi.IeeeDoi method)

(project_doi.scrapyspider.spiders.springer_doi.SpringerDoi method)

(project_doi.scrapyspider.spiders.wiley_doi.WileyDoi method)

process_exception() (project_doi.scrapyspider.middlewares.MyprojectspDownloaderMiddleware method)

process_item() (project_doi.scrapyspider.pipelines.DefaultValuesPipeline method)

process_request() (project_doi.scrapyspider.middlewares.MyprojectspDownloaderMiddleware method)

process_response() (project_doi.scrapyspider.middlewares.MyprojectspDownloaderMiddleware method)

process_spider_exception() (project_doi.scrapyspider.middlewares.MyprojectspSpiderMiddleware method)

process_spider_input() (project_doi.scrapyspider.middlewares.MyprojectspSpiderMiddleware method)

process_spider_output() (project_doi.scrapyspider.middlewares.MyprojectspSpiderMiddleware method)

process_start_requests() (project_doi.scrapyspider.middlewares.MyprojectspSpiderMiddleware method)

project_doi

module

project_doi.api

module

project_doi.config

module

project_doi.database

module

project_doi.export

module

project_doi.models

module

project_doi.scrape

module

project_doi.scrapyspider
module
project_doi.scrapyspider.items
module
project_doi.scrapyspider.middlewares
module
project_doi.scrapyspider.pipelines
module
project_doi.scrapyspider.settings
module
project_doi.scrapyspider.spiders
module
project_doi.scrapyspider.spiders.ieee_doi
module
project_doi.scrapyspider.spiders.springer_doi
module
project_doi.scrapyspider.spiders.wiley_doi
module
publisher (project_doi.models.Article attribute)
(project_doi.models.Book attribute)
(project_doi.models.ConferencePaper attribute)

R

read() (in module project_doi.database)
read_all() (in module project_doi.database)
route_base (doi_app.DOIView attribute)
(project_doi.api.ApiView attribute)

S

save() (in module project_doi.database)
Scrape (class in project_doi.scrape)
scrape() (project_doi.scrape.Scrape method)
scrape_with_crochet() (project_doi.scrape.Scrape method)
search_doi() (doi_app.DOIView method)
SECRET_KEY (project_doi.config.Config attribute)
spider_opened() (project_doi.scrapyspider.middlewares.MyprojectspDownloaderMiddleware method)
(project_doi.scrapyspider.middlewares.MyprojectspSpiderMiddleware method)
SpringerDoi (class in project_doi.scrapyspider.spiders.springer_doi)
SQLALCHEMY_DATABASE_URI (project_doi.config.Config attribute)
SQLALCHEMY_TRACK_MODIFICATIONS (project_doi.config.Config attribute)

T

timestamp (project_doi.models.Article attribute)
(project_doi.models.ConferencePaper attribute)
title (project_doi.models.Article attribute)
(project_doi.models.Book attribute)
(project_doi.models.ConferencePaper attribute)
(project_doi.models.Document attribute)

U

upload_contents() (doi_app.DOIView method)
UPLOAD_FOLDER (project_doi.config.Config attribute)
url (project_doi.models.Article attribute)
(project_doi.models.Book attribute)
(project_doi.models.ConferencePaper attribute)
(project_doi.models.Document attribute)

W

WileyDoi (class in project_doi.scrapyspider.spiders.wiley_doi)

Y

year (project_doi.models.Article attribute)
(project_doi.models.ConferencePaper attribute)

Python Module Index

d

doi_app

p

project_doi
project_doi.api
project_doi.config
project_doi.database
project_doi.export
project_doi.models
project_doi.scrape
project_doi.scrapyspider
project_doi.scrapyspider.items
project_doi.scrapyspider.middlewares
project_doi.scrapyspider.pipelines
project_doi.scrapyspider.settings
project_doi.scrapyspider.spiders
project_doi.scrapyspider.spiders.ieee_doi
project_doi.scrapyspider.spiders.springer_doi
project_doi.scrapyspider.spiders.wiley_doi