Developer’s Guide — Economist Webforms 1.0 documentation

[Economist Webforms](index.html)

Contents:

* [About](readme.html)
* [Developer’s Guide](#)
  + [Overview](#overview)
  + [Deployment](#deployment)
    - [Local](#local)
    - [Python Anywhere](#python-anywhere)
  + [Project structure](#project-structure)
    - [User interactions](#user-interactions)
  + [App](#app)
    - [Forms](#forms)
    - [Models](#models)
    - [Templates](#templates)
    - [Utilities](#module-app.utility)
    - [Views](#module-app.views)
  + [Maintenance](#maintenance)
    - [Unit tests](#unit-tests)
    - [Documentation](#documentation)
  + [Future work](#future-work)

[Economist Webforms](index.html)

* »
* Developer’s Guide
* [View page source](_sources/developer.rst.txt)

# [Developer’s Guide](#id1)[¶](#developer-s-guide)

author

Ryan Wolfslayer

date

July 29, 2020

Contents

* [Developer’s Guide](#developer-s-guide)
  + [Overview](#overview)
  + [Deployment](#deployment)
    - [Local](#local)
    - [Python Anywhere](#python-anywhere)
  + [Project structure](#project-structure)
    - [User interactions](#user-interactions)
  + [App](#app)
    - [Forms](#forms)
    - [Models](#models)
    - [Templates](#templates)
    - [Utilities](#module-app.utility)
    - [Views](#module-app.views)
  + [Maintenance](#maintenance)
    - [Unit tests](#unit-tests)
    - [Documentation](#documentation)
  + [Future work](#future-work)

## [Overview](#id2)[¶](#overview)

Many libraries and information centers now offer an institutional repository that houses research from the parent institution. These repositories allow researchers, associated with a given university, government agency, or think-tank, to disseminate working papers and early versions of their research. For those familiar with the open-access movement, institutional repositories are often key-players in freely distributing the most cutting-edge scholarship. Our library is associated with a government organization that employs over 200 Ph.D. economists, all of whom are encouraged to maintain an internal web page that pulls its data from our repository. To add or edit their publication list, economists currently need to contact someone from our library staff and request the edits. The current system is inefficient for both sides of the exchange. On the one hand, economists need to know already or lookup who to contact to make updates. For the library, these email requests are often free-form and lack much of the metadata we require to make a full entry into the repository system. When emails are ambiguous, a librarian or contractor will need to research the economist, paper title, journal publisher, and more to come up with a complete record. During high-output times of the year, it is not unusual to see as many as 20 requests per day, each of which takes several minutes to prepare. The aim of this product is two-fold. By creating a web-form, we can gently solicit the information we need from economists without overwhelming them with yet more paperwork. The human-computer interaction facilitated by this form should serve to remove ambiguity on both sides and pave the way for more opportunities to automate data entry.

## [Deployment](#id3)[¶](#deployment)

### [Local](#id4)[¶](#local)

To run a local version of this app install the dependencies from requirements.txt.

$ pip install -r requirements.txt

Next, run the main.py file to launch a Flask development server. By default the server is localhost:5000

$ python main.py

### [Python Anywhere](#id5)[¶](#python-anywhere)

There is an instance of the webform hosted on PythonAnywhere. To view the example, [click here](http://wryan14.pythonanywhere.com/). For information on launching a Flask application on PythonAnywhere, view [this documentation](https://help.pythonanywhere.com/pages/Flask/).

When deploying this Flask application on any server, you will want to change the local\_host variable (located at the toop of the app’s \_\_init\_\_.py file to reflect the correct domain name).

For example change:

# app/\_\_init\_\_.py  
local\_url = "http://localhost:5000"

to

# app/\_\_init\_\_.py  
# local\_url = "http://localhost:5000"  
local\_url = "http://wryan14.pythonanywhere.com"

## [Project structure](#id6)[¶](#project-structure)

.  
├── LICENSE  
├── README.md  
├── app  
│   ├── \_\_init\_\_.py  
│   ├── forms.py  
│   ├── models.py  
│   ├── static  
│   ├── templates  
│   ├── utility.py  
│   └── views.py  
├── config.py  
├── main.py  
├── requirements.txt  
├── spec.md  
├── tests  
│   ├── logs  
│   └── test\_utility.py  
└── todos.md

### [User interactions](#id7)[¶](#user-interactions)

**Add new publication to repository**

* The user arrives at the landing page and selects new publication
  + views.home
  + templates/welcome\_page.html
* The user is provided the option to autopopulate the form using the publication’s [DOI](https://library.uic.edu/help/article/1966/what-is-a-doi-and-how-do-i-use-them-in-citations)
  + views.newpub
  + templates/newpub.html
  + utitlity.CRef
  + forms.AuthorForm
  + forms.NewPublication
* After autopopulating most of the form, the user can make minor edits or submit as is
  + views.newpub
  + templates.newpub
* The data is stored in a database which will be queried by an automated system for review; the user is redirected to a success page
  + views.success\_new
  + templates/success\_new.html
  + models.Doc
  + models.Author

**Edit existing publication in repository**

* The user arrives at the landing page and selects edit
  + views.home
  + templates/welcome\_page.html
* The user selects the desired publication from a table (possibly using the filter feature)
  + views.dash\_app
  + utility.cdm\_pull
  + models.BeforeAuthor
  + models.BeforeDoc
* After finding the publication, user hits edit and is redirected to an edit form that is prepopulated with metadata
  + views.editpub
  + templates/editpub.html
* The user edits the desired field and is redirected to a success page that shows the recent changes
  + models.EditAuthor
  + models.EditDoc
* Changes are stored in the database using tables that store the metadata before and after the edit
  + views.success\_edit
  + templates/success\_edit.html
* The database is queried and reviewed by an automated process
  + models.BeforeAuthor
  + models.EditAuthor
  + models.BeforeDoc
  + models.EditDoc

**Update forthcoming publication** [UNDER CONSTRUCTION]

* The user arrives at the landng page and selects forthcoming
  + views.home
  + templates/welcome\_page.html
* The user selects the desired publication from a table, using the table’s filter function
  + [NEEDS BUILT]
* After finding the publication, user is redirected to a form that gathers metadata
  + views.updatepub
  + templates/updatepub.html
* The form is stored in a database and queried by an automated process
  + views.success\_update
  + templates/success\_update.html
  + [MODELS NEED BUILT]

## [App](#id8)[¶](#app)

### [Forms](#id9)[¶](#forms)

The forms are directly tied to the input fields within the templates. The following table should help you link the name of the form with the template name and output.

|  |  |  |
| --- | --- | --- |
| name | location(s) | image |
| DOI Form | * templates/newpub.html | _images/form1.png |
| AuthorForm | * forms.py * templates/newpub.html * templates/editpub.html * templates/updatepub.html | _images/authorform.png |
| NewPublication | * forms.py * templates/newpub.html | _images/form2.png |
| UpdatePublication | * forms.py * templates/editpub.html | _images/form3.png |
| UpdatePublicationStatus | * forms.py * templates/updatepub.html | _images/form4.png |

*class* app.forms.AuthorForm(*\*args*, *\*\*kwargs*)[[source]](_modules/app/forms.html#AuthorForm)[¶](#app.forms.AuthorForm)

Bases: wtforms.form.Form

Form that inherits from wtforms.Form class. Contains fields for author first\_name and last\_name

*class* app.forms.NewPublication(*\*args*, *\*\*kwargs*)[[source]](_modules/app/forms.html#NewPublication)[¶](#app.forms.NewPublication)

Bases: flask\_wtf.form.FlaskForm

Contains the form-fields required to add a new publication. The author FieldList allows for up to 30 authors.

*class* app.forms.UpdatePublication(*\*args*, *\*\*kwargs*)[[source]](_modules/app/forms.html#UpdatePublication)[¶](#app.forms.UpdatePublication)

Bases: flask\_wtf.form.FlaskForm

Contains the form-fields required to update a publication. The author FieldList allows for up to 30 authors.

*class* app.forms.UpdatePublicationStatus(*\*args*, *\*\*kwargs*)[[source]](_modules/app/forms.html#UpdatePublicationStatus)[¶](#app.forms.UpdatePublicationStatus)

Bases: flask\_wtf.form.FlaskForm

[IN PROGRESS] This will contain the form-fields required to udpate a forthcoming publication.

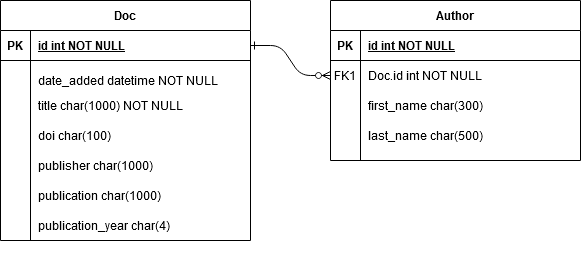
### [Models](#id10)[¶](#models)

The models determine how the database is structured. It is imporant to understand how the database is structured, so we can make proper SQL queries.

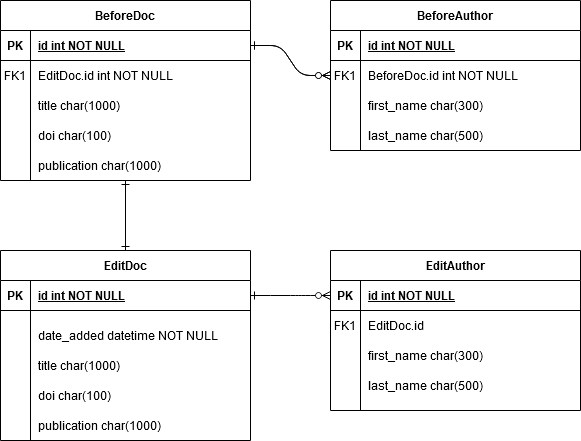
**ER Diagrams**

These diagrams show a quick break down of the tables. Each document can have many authors, but, for now, authors can only have one document. The connection between the BeforeDoc and EditDoc model is one-to-one, meaning the BeforeDoc can only have one EditDoc connection and visa versa.

*New Publication model*



*Edit Publication model*



**models.py**

*class* app.models.Author(*\*\*kwargs*)[[source]](_modules/app/models.html#Author)[¶](#app.models.Author)

Bases: sqlalchemy.ext.declarative.api.Model

Author information associated with Doc model and adding new publication.

Parameters

* **id** ([*int*](https://docs.python.org/3/library/functions.html#int)) – primary key (auto-generated)
* **doc\_id** ([*int*](https://docs.python.org/3/library/functions.html#int)) – foreign key from Doc model (Doc.id)
* **first\_name** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – First name of author associated with Doc.id publication
* **last\_name** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – Last name of author associated with Doc.id publication
* **doc** – Establishes backref connection to Doc model

*class* app.models.BeforeAuthor(*\*\*kwargs*)[[source]](_modules/app/models.html#BeforeAuthor)[¶](#app.models.BeforeAuthor)

Bases: sqlalchemy.ext.declarative.api.Model

Authors associated with BeforeDoc documents (i.e. documents prior to edits).

Parameters

* **id** ([*int*](https://docs.python.org/3/library/functions.html#int)) – primary key (auto generated)
* **doc\_id** ([*int*](https://docs.python.org/3/library/functions.html#int)) – foreign key for BeforeDoc model
* **first\_name** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – first name associated with document author prior to edits
* **last\_name** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – last name associated with document author prior to edits
* **before\_doc** – Establishes backref connection to BeforeDoc

*class* app.models.BeforeDoc(*\*\*kwargs*)[[source]](_modules/app/models.html#BeforeDoc)[¶](#app.models.BeforeDoc)

Bases: sqlalchemy.ext.declarative.api.Model

Model that captures state of a document object’s metadata prior to edits. This will be a useful point of comparison for later work with document metadata. (i.e. What did the user edit?)

Parameters

* **id** ([*int*](https://docs.python.org/3/library/functions.html#int)) – primary key (auto-generated)
* **doc\_id** ([*int*](https://docs.python.org/3/library/functions.html#int)) – foreign key for the EditDoc model, edit\_docs.id
* **title** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – title of document prior to edits
* **doi** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – Digital object identifier (DOI) prior to edits
* **publication** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – Entitiy publishing document object prior to edits
* **edit\_doc** – Establishes backref connection to EditDoc model

*class* app.models.Doc(*\*\*kwargs*)[[source]](_modules/app/models.html#Doc)[¶](#app.models.Doc)

Bases: sqlalchemy.ext.declarative.api.Model

Model associated with adding a new publication.

Parameters

* **id** ([*int*](https://docs.python.org/3/library/functions.html#int)) – primary key (auto-generated)
* **date\_added** (*datetime*) – date document added to database (useful for queries; auto-generated)
* **title** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – title of new document
* **doi** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – Digitial Object Identifier (DOI), should not contain <https://doi.org/>
* **publisher** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – Name of entity publishing new object
* **publication** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – Name of entity in which object is published
* **publication\_year** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – Year entity published (different from date\_added)

*class* app.models.EditAuthor(*\*\*kwargs*)[[source]](_modules/app/models.html#EditAuthor)[¶](#app.models.EditAuthor)

Bases: sqlalchemy.ext.declarative.api.Model

Authors associated with EditDoc model (i.e. authors after user edits)

Parameters

* **id** ([*int*](https://docs.python.org/3/library/functions.html#int)) – primary key (auto-generated)
* **doc\_id** ([*int*](https://docs.python.org/3/library/functions.html#int)) – foreign key for EditDoc model
* **first\_name** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – first name associated with document author after user edits
* **last\_name** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – last name associated with document author after user edits
* **before\_doc** – Establishes backref connection to EditDoc

*class* app.models.EditDoc(*\*\*kwargs*)[[source]](_modules/app/models.html#EditDoc)[¶](#app.models.EditDoc)

Bases: sqlalchemy.ext.declarative.api.Model

Model that captures the state of a document after a user submits edits. Changes can later be identified by comparing EditDoc metadata to BeforeDoc metadata.

Parameters

* **id** ([*int*](https://docs.python.org/3/library/functions.html#int)) – primary key (auto-generated)
* **date\_added** (*datetime*) – date document added to database (useful for queries; auto-generated)
* **title** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – title of document after edits
* **doi** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – Digital Object Identifier after edits
* **publication** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – Name of entity in which document is published after edits

### [Templates](#id11)[¶](#templates)

Templates use Jinja to create the HTML views used in the application.

|  |  |  |
| --- | --- | --- |
| file | description | image |
| base.html | This template contains the navigation bar,  connects the JS and CSS to the project,  and implements the JS script needed to  adjust the number of authors per publication  when editing records. | _images/base_html.png |
| welcome\_page.html | Navigation page used to direct users to the  new, edit, and forthcoming processes. | _images/welcome_html.png |
| newpub.html | Connects to   * view.py - newpub * forms.py - AuthorForm and NewPublication * models.py - Doc and Author | _images/newpub_html.png |
| editpub.html | Connects to   * views.py - editpub * forms.py - AuthorForm and UpdatePublication * models.py - BeforeAuthor, BeforeDoc * models.py - EditAuthor, EditDoc | _images/editpub_html.png |
| updatepub.html | Connects to   * views.py - updatepub * forms - updatepublicationstatus | _images/updatepub_html.png |
| \_\_init\_\_.py | Combines components form the application,  and allows for the app to be fully imported  into main.py. This file also contains the  dash table used to select records for edits. | _images/dash_table.png |
| success\_new.html | Reads from   * sqlite database * models.py Doc and Author | _images/success_new_html.png |
| success\_edit.html | Reads from   * sqlite database * models.py BeforeAuthor and BeforeDoc * models.py EditAuthor and EditDoc | _images/success_edit_html.png |
| success\_update.html | Under construction | _images/success_update_html.png |

### [Utilities](#id12)[¶](#module-app.utility)

*class* app.utility.CRef(*doi*)[[source]](_modules/app/utility.html#CRef)[¶](#app.utility.CRef)

Bases: [object](https://docs.python.org/3/library/functions.html#object)

Leverages the crossref.restful API to gather metadata requried to submit a new publication to the web form. The crossref schema can be found at <https://data.crossref.org/schemas/crossref_query_input2.0.xsd>.

This class parses in a way that satisfies the needs of the Economist Webform project. If you need different metadata, modify this class to target different fields.

Parameters

**doi** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – Digital object identifier (doi); URL like string associated with a publication registered with CrossRef

Attributes:

doi (str)

same as param

res (obj)

response to the crossref.restful API call

title (str)

title of the document returned in API call

publisher(str)

name of publisher of document from API call

journal\_name (str)

publication which houses document returned in API call

year (str)

year returned document was published

author\_find()[[source]](_modules/app/utility.html#CRef.author_find)[¶](#app.utility.CRef.author_find)

Method to find and parse a list of authors returned in crossref.restful API response.

Attributes:

author\_list (list)

list of authors [‘lastname, firstname’,] from API response

app.utility.cdm\_api\_trans()[[source]](_modules/app/utility.html#cdm_api_trans)[¶](#app.utility.cdm_api_trans)

Pythonic approach to running XSLT transformation. The transformation takes CONTENTdm’s API response and converts the XML into HTML tables, which, in turn, can be converted to a pandas dataframe.

app.utility.cdm\_pull(*query\_string=None*)[[source]](_modules/app/utility.html#cdm_pull)[¶](#app.utility.cdm_pull)

Calls CONTENTdm’s API and runs the cdm\_api\_trans XSLT transformation against te results. The function then converts the HTML tables from cdm\_api\_trans to a pandas dataframe, and returns the output.

Parameters

**query\_string** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – string that will return CDM API response with desired parameters

*Note* During the prototype stage of this project, this function will read in dummy data that replicates the CDM API response, instead of calling a live API. Make sure dummydata.xml is located in the static directory.

### [Views](#id13)[¶](#module-app.views)

app.views.cleartable()[[source]](_modules/app/views.html#cleartable)[¶](#app.views.cleartable)

Clears null values from the edit\_docs database; not intended for public use. For an unknown reason, the database model adds a blank row in-between each submit. To make the database more friendly to developers, this function does some cleanup.

Returns:

* ./templates/welcome\_page.html

app.views.dash\_app()[[source]](_modules/app/views.html#dash_app)[¶](#app.views.dash_app)

This dash app presents a Dash table of fields that a user can select. Once selected, the data is stored into the flask.session and used in the editpub process.

app.views.doc(*dir=''*, *filename='index.html'*)[[source]](_modules/app/views.html#doc)[¶](#app.views.doc)

Serves sphinx documentation

app.views.editpub()[[source]](_modules/app/views.html#editpub)[¶](#app.views.editpub)

Edit Publication page. If the users selects a table row from dash\_app template, this view saves the stored flask session data into the BeforeDoc database model. Any changes submitted to this view are then stored in the EditDoc model.

Returns:

* ./templates/editpub.html
* ./templates/success\_edit.html (upon submit)

app.views.home()[[source]](_modules/app/views.html#home)[¶](#app.views.home)

Welcome page that provides options to navigate to new, edit, or forthcoming functionality.

Returns:

* ./templates/welcome\_page.html

app.views.newpub()[[source]](_modules/app/views.html#newpub)[¶](#app.views.newpub)

New publication page where users can fill in the NewPublication form, and save form to Doc database table.

Returns:

* ./templates/newpub.html
* ./templates/success\_new.html (upon submit)

app.views.success\_edit()[[source]](_modules/app/views.html#success_edit)[¶](#app.views.success_edit)

Confirmation page after form submission for edit items

app.views.success\_new()[[source]](_modules/app/views.html#success_new)[¶](#app.views.success_new)

Confirmation page after form submission for new items

app.views.success\_update()[[source]](_modules/app/views.html#success_update)[¶](#app.views.success_update)

Confirmation page after form submission for update item

app.views.updatepub()[[source]](_modules/app/views.html#updatepub)[¶](#app.views.updatepub)

Renders the forthcoming process.

Returns:

* ./templates/updatepub.html
* ./templates/success\_update.html (upon submit)

View still in progress

## [Maintenance](#id14)[¶](#maintenance)

### [Unit tests](#id15)[¶](#unit-tests)

The tests directory contains a few unit tests that are designed to ensure the utility.py code is working correctly. It is essential to keep these tests up-to-date, so any development work on utility.py should also include modifications to files in the tests directory. Especially when working with third-party data, it is useful to regularly run tests to make sure the expected results are being returned. If CrossRef, for example, decides to change their schema, reformat, or is down for maintenance, tests can help identify the issue early on.

### [Documentation](#id16)[¶](#documentation)

All project documentation is created with Sphinx and is housed in the project’s static folder: (app/static/documentation). The project is using Sphinx’s auto-document feature for docstrings on existing classes and functions. Changes to existing docstrings will update automatically, but new classes, methods, and functions will need to be inserted into the documentation. After making changes, you should run the Sphinx make html command from the root of the documentation folder and push the changes to GitHub, and potentially pull the changes to PythonAnywhere or some other production server.

For additional information on Sphinx review the [Sphinx documentation](https://www.sphinx-doc.org/en/master/usage/quickstart.html).

## [Future work](#id17)[¶](#future-work)

Most of the forthcoming process needs to be constructed.

* Build a forthcoming table with Dash data tables; similar to the one that exists for the Edit process
* Build a database model for the forthcoming process

Permanent location

* Modify form to work with new server in permanent location
* Build queries to process data collected from the forms
* Build automated process to properly format data for inclusion in CONTENTdm repository
* The database will likely need to be switched from SQLite to Postgresql

Bugs and Appearance

* Improve formatting for smaller screens i.e. mobile apps

[Previous](readme.html)

© Copyright 2020, Ryan Wolfslayer

Built with [Sphinx](http://sphinx-doc.org/) using a [theme](https://github.com/rtfd/sphinx_rtd_theme) provided by [Read the Docs](https://readthedocs.org).