Django Globus Portal Framework

Release 0.4.8

Nickolaus Saint

Mar 29, 2023

INSTALLATION:

1 Installation and Setup **3**
- 1.1 Globus Auth . . . 3
- 1.2 Settings . . . 4

2 Portal Configuration **7**
- 2.1 Index Creation and Ingest . . . 7
- 2.2 Configuring Facets . . . 10
- 2.3 Built-in Fields . . . 11
- 2.4 Templates . . . 13
- 2.5 Advanced: Multiple Indices . . . 16
- 2.6 Previewing Content . . . 17
- 2.7 Custom Search Views . . . 19
- 2.8 Facet Modifiers . . . 21
- 2.9 Search Settings Reference . . . 22
- 2.10 Customizing URLs . . . 25
- 2.11 Requiring Login . . . 27

3 Settings **29**
- 3.1 General Settings . . . 29
- 3.2 Auth Settings . . . 29
- 3.3 Search Settings . . . 30
- 3.4 Templates . . . 30
- 3.5 Under the Hood . . . 31

4 Settings Example **33**

5 Local Credentials **37**

6 Deployment **39**
- 6.1 Docker . . . 39

7 Indices and tables **41**

Globus Portal Framework is a collection of tools for quickly bootstrapping a portal for Globus Search. Use the guide below to get your portal running with Globus Auth and your custom search data. After that, you can make your data more accessible with Globus Transfer, or extend it how you want with your custom workflow.

CHAPTER

ONE

INSTALLATION AND SETUP

Install the packages below to get started. Globus Portal Framework requires Python 3.7 and higher.

```
pip install django-admin django-globus-portal-framework
```

Django-admin will help you bootstrap your initial portal while Globus Portal Framework will add tooling for Globus services, Search, Transfer, and Auth.

Run the `django-admin` command below to create your project layout.

```
django-admin startproject myportal
cd myportal
```

This will create your basic Django project with the following project structure:

```
myportal/
  db.sqlite3
  manage.py
  myportal/
    settings.py
    urls.py
```

1.1 Globus Auth

Globus Auth will allow your users to login to the portal, storing their tokens in the database for Search, Transfer, or any other custom functionality you want to implement.

You'll need a Client ID from Globus. Follow these instructions from the Globus Auth Developer Guide.

When you register your application with Globus, make sure it has the following settings:

- **Redirect URL** – `http://localhost:8000/complete/globus/`
- **Native App** – *Unchecked*

1.2 Settings

Next, you will need to modify your `myportal/settings.py` file to enable user auth and Globus Portal Framework components. You can copy-paste the individual settings below, or use our *Settings Example* for a complete `settings.py` file reference.

```
# Your portal credentials for a Globus Auth Flow
SOCIAL_AUTH_GLOBUS_KEY = 'Put your Client ID here'
SOCIAL_AUTH_GLOBUS_SECRET = 'Put your Client Secret Here'

# This is a general Django setting if views need to redirect to login
# https://docs.djangoproject.com/en/3.2/ref/settings/#login-url
LOGIN_URL = '/login/globus'

# This dictates which scopes will be requested on each user login
SOCIAL_AUTH_GLOBUS_SCOPE = [
    'urn:globus:auth:scope:search.api.globus.org:search',
]

# Installed apps tells Django which packages to load on startup
INSTALLED_APPS = [
    ...
    'globus_portal_framework',
    'social_django',
]

# Middleware provides exception handling
MIDDLEWARE = [
    ...
    'globus_portal_framework.middleware.ExpiredTokenMiddleware',
    'globus_portal_framework.middleware.GlobusAuthExceptionMiddleware',
    'social_django.middleware.SocialAuthExceptionMiddleware',
]

# Authentication backends setup OAuth2 handling and where user data should be
# stored
AUTHENTICATION_BACKENDS = [
    'globus_portal_framework.auth.GlobusOpenIdConnect',
    'django.contrib.auth.backends.ModelBackend',
]

# The context processor below provides some basic context to all templates
TEMPLATES = [
  {
      'BACKEND': 'django.template.backends.django.DjangoTemplates',
      'DIRS': [],
      'APP_DIRS': True,
      'OPTIONS': {
          'context_processors': [
              ...
              'globus_portal_framework.context_processors.globals',
          ],
      },
```

(continues on next page)

(continued from previous page)

```
  },
]
```

Add the base URLs for Globus Portal Framework in your `myportal/urls.py` file. These will provide a starting point for your Globus Portal. You may keep or discard any existing paths in your `urlpatterns`.

```
from django.urls import path, include

urlpatterns = [
    # Provides the basic search portal
    path('', include('globus_portal_framework.urls')),
    # Provides Login urls for Globus Auth
    path('', include('social_django.urls', namespace='social')),
]
```

Now run your server to see your Globus Portal. Migrate will setup your database, which will be used in the next section when adding Globus Auth. The second command will run your Globus Portal.

Note: Make sure you use http://localhost:8000 in your browser, so your URL matches the callback URL for your Globus App at developers.globus.org. A mismatch will cause an error when logging in.

```
python manage.py migrate
python manage.py runserver localhost:8000
```

You should now be able to view a portal at http://localhost:8000/

CHAPTER

TWO

PORTAL CONFIGURATION

With the base portal, you can take it in a number of different directions. Globus Portal Framework was primarily designed around Globus Search, however other configurations are also possible.

2.1 Index Creation and Ingest

The Globus Search Index stores the metadata for your search potral, and Globus Portal Framework queries and presents this information in a way that's convenient for users. No search metadata is stored directly on the portal, rather the portal is only a graphical interface for constructing search queries for users and presenting the information in a digestable fashion.

Before any work can be done on a portal, the search index must first be created in Globus Search and metadata ingested into it. Once the search index can be queried for information, the portal can be configured to display results for users.

2.1.1 Creating the Index

There are a few different tutorials on creating a search index and ingesting data into it. If you want a deeper dive beyond the basics, see the resouces below:

- **Metadata Search and Discovery**
  - A fast and friendly tutorial on the basics of Globus Search
  - Run interactively on jupyter.demo.globus.org!
- **Gladier Flows Tutorial**
  - An automated approach to ingesting metadata into an index
  - Run interactively on jupyter.demo.globus.org!
- **Searchable Files Demo**
  - A deeper dive into maintaing a Globus Search index

The Globus CLI can now be used to manage index settings. Use the following to get started:

```
pipx install globus-cli
globus search index create myindex "A description of my index"
```

Should return something that looks like:

```
Index ID:     3e2525cc-e8c1-49cd-bef5-a9566770d74c
Display Name: myindex
Status:       open
```

Take note of the Index ID UUID returned by this command, this will be used later to point your portal at your new search index.

2.1.2 Ingesting Metadata

Note: See the reference ingest document for a real-world example. The document below is oversimplified for readablility.

Metadata within Globus Search is unstructured and can be tailored to the specific needs of the project. In simple terms, a search result is a JSON document with a "subject" containing user defined content. See the `simple-ingest-doc.json` below:

```
{
    "ingest_type": "GMetaList",
    "ingest_data": {
        "gmeta": [
            {
                "id": "metadata",
                "subject": "globus://ddb59af0-6d04-11e5-ba46-22000b92c6ec/share/godata/
↪file1.txt",
                "visible_to": ["public"],
                "content": {
                    "title": "File Number 1",
                    "url": "globus://ddb59af0-6d04-11e5-ba46-22000b92c6ec/share/godata/
↪file1.txt",
                    "author": "Data Researcher",
                    "tags": ["globus", "tutorial", "file"],
                    "date": "2022-11-15T12:31:28.560098",
                    "times_accessed": 23974,
                    "original_collection_name": "Globus Tutorial Endpoint 1"
                }
            }
        ]
    }
}
```

The document can be ingested into your index above with the following:

```
globus search ingest my-index-uuid simple-ingest-doc.json
```

Working from inside out, everything under the `content` block is completely defined by the user. Each new ingested field Globus Search detects will be scanned and indexed, and can be searched upon immediately after ingest. `visible_to` defines access, and `subject` is a unique identifier for the search result. `id` defines different independent sub-categories under `subject`.

Warning: Field types in Globus Search may only be set once the first time you ingest a new field. Types may not change for the lifetime of the index. Setting new types requires the index to be either reset or recreated. Both require emailing support at support@globus.org.

For example above: "author" and "url" are both string types, and future ingest for those fields must also be strings. Non-string values will raise an error if the types change.

That's it for the actual metadata. The outer envelope of the sample above `ingest_data` and `gmeta` defines the document as a search ingest document. See the ingest documentation for more info.

2.1.3 Portal Configuration

Copy-paste the following `SEARCH_INDEXES` dictionary in `myportal/settings.py` to define one or more search indices. Use the UUID of the index you created in *Index Creation and Ingest*.

```
# List of search indices managed by the portal
SEARCH_INDEXES = {
    'my-index-slug': {
        'name': 'My Search Index',
        'uuid': 'my-index-uuid',
    }
}
```

The configuration above consists of three pieces of information:

- `my-index-slug` – The slug for your index. This will map to the browser url and can be any reasonable value.
- `name` – The name for your index. This shows up in some templates and can be any value.
- `uuid` – The UUID of your index. This can be found with the `globus search index list` command line with the Globus CLI.

You should now have enough information to run your new portal. If the Django server is already running, make sure to refresh your webpage, otherwise start the server.

```
python manage.py runserver localhost:8000
```

Your index name should show up on the index selection page at `http://localhost:8000`, and the search record should now show up at `http://localhost:8000/my-index-slug/`. The existing record can be edited by re-ingesting the same subject with different content, or new records can be created by changing the subject.

Next, we will add facets to this portal.

2.1.4 Authenticated Search

If search records contain anything other than `public` inside `visible_to`, users will need to login to view records. Make sure you have Globus Auth setup, and you have a search scope set in your `myportal/settings.py` file.

```
SOCIAL_AUTH_GLOBUS_SCOPE = [
    ...
    'urn:globus:auth:scope:search.api.globus.org:search',
]
```

It's common for different Globus Groups to be set on various records on the `visible_to` field. Records will simply not show up for users who do not have access.

2.2 Configuring Facets

Facets are used to filter and query your search index. To configure them, add a new field called `facets` to `SEARCH_INDEXES`. A basic example is below:

```
SEARCH_INDEXES = {
    'index-slug': {
        'name': 'My Search Index',
        'uuid': 'my-search-index-uuid',
        'facets': [
          {
            'name': 'Tags',
            'field_name': 'tags'
          },
        ],
    }
}
```

Now, the next time the portal is ran, the new `Tags` facet will show up for any search records matched in the query. Given a record with content that looks like the following:

```
{
  "title": "File Number 1",
  "url": "globus://ddb59af0-6d04-11e5-ba46-22000b92c6ec/share/godata/file1.txt",
  "author": "Data Researcher",
  "tags": ["globus", "tutorial", "file"],
  "date": "2022-11-15T12:31:28.560098"
}
```

The default search page should show the `Tags` facet on the left side with each value, `globus`, `tutorial`, and `file`. Each additional search record will add to this list, and repeating numbers will increment the number for each value.

Note: Only results with matching fields (`tags` above) will show up in the results. By default, facets which match no record are not shown.

New sets of `{'name': ..., 'field_name': ...}`, can be added to the list of `facets` in order to provide more filtering options. In all cases, while `name` can be any string value, `field_name` must be a component found in the `content` section of at least one of the ingested search records.

See *Search Settings Reference* for more information on different facet types and options.

2.2.1 Filters

By default, Django Globus Portal Framework shows facets on the left side of the search page with check marks. Checking any item on the left will cause the portal to filter on the information given.

Filtering at this time is automatically handled by the portal for all facet types.

2.3 Built-in Fields

Search Fields take raw search metadata from Globus Search and expose them for use by templates. Commonly, raw data from Globus Search needs a bit more processing before it can be viewed in templates. Examples include parsing dates or generating links to the Globus Webapp.

The Django Globus Portal Framework includes some built-in templates which will automatically be rendered if given the right field names. Some of these include:

- Title – A title for a given subject, shown on the search results page and the detail page.
- Globus App Link – A link to the file on https://app.globus.org.
- HTTPS URL – A direct-download link to the file.

First, let's take a look at the metadata once more:

```
{
    "ingest_type": "GMetaList",
    "ingest_data": {
        "gmeta": [
            {
                "id": "metadata",
                "subject": "globus://ddb59af0-6d04-11e5-ba46-22000b92c6ec/share/godata/
↪file1.txt",
                "visible_to": ["public"],
                "content": {
                    "title": "File Number 1",
                    "url": "globus://ddb59af0-6d04-11e5-ba46-22000b92c6ec/share/godata/
↪file1.txt",
                    "author": "Data Researcher",
                    "tags": ["globus", "tutorial", "file"],
                    "date": "2022-11-15T12:31:28.560098",
                    "times_accessed": 23974,
                    "original_collection_name": "Globus Tutorial Endpoint 1"
                }
            }
        ]
    }
}
```

Create an empty `myportal/fields.py` file next to `settings.py`, and copy-paste the following code.

```
import os
from urllib.parse import urlsplit, urlunsplit, urlencode


def title(result):
    """The title for this Globus Search subject"""
    return result[0]["title"]


def globus_app_link(result):
    """A Globus Webapp link for the transfer/sync button on the detail page"""
    url = result[0]["url"]
```

(continues on next page)

(continued from previous page)

```
    parsed = urlsplit(url)
    query_params = {
        "origin_id": parsed.netloc,
        "origin_path": os.path.dirname(parsed.path),
    }
    return urlunsplit(
        ("https", "app.globus.org", "file-manager", urlencode(query_params), "")
    )


def https_url(result):
    """Add a direct download link to files over HTTPS"""
    path = urlsplit(result[0]["url"]).path
    return urlunsplit(("https", "g-71c9e9.10bac.8443.data.globus.org", path, "", ""))
```

Here the `result[0]` variable encapsulates the information of a given search record, and can be used to access any component of the metadata `content` such as `title` and `url`.

To propagate `myportal/fields.py` throughout your portal, configure fields for your search index by adding `fields` to your `SEARCH_INDEXES`:

```
from myportal import fields

SEARCH_INDEXES = {
    "index-slug": {
        "uuid": "my-search-index-uuid",
        ...  # Previous fields hidden for brevity
        "fields": [
            # Calls a function with your search record as a parameter
            ("title", fields.title),
            ("globus_app_link", fields.globus_app_link),
            ("https_url", fields.https_url)
        ],
    }
}
```

You should notice the following changes the next time you run your server:

- **The Search Page**
  - "File Number 1" now shows up as the title
- **The Detail Page**
  - The "Transfer/Sync" buttons are now functional

Continue on to cover custom Templates.

2.4 Templates

Templates in Globus Portal Framework are an extension of the Django Template system, consisting of a basic set of included templates to make starting a portal quick and easy. A list of all Globus Portal Framework templates can be found under the Github template repo.

Templates follow a strict directory layout, file paths must match exactly for templates to be rendered correctly. Ensure `myportal` above matches your project name, and your `templates` directory is created in the correct location. Globus Portal Framework templates match the following directory structure:

```
myportal/
    manage.py
    myportal/
        templates/
          globus-portal-framework/
            v2/
              detail-overview.html
              components/
                  search-results.html
```

If you want to browse the original templates, you can find them by browsing the source template directory on github.

2.4.1 Customizing Search Results

Override *search-results.html* by creating the following file. Make sure the template directory matches exactly.

Note: If no changes to the search page take effect, double check your `TEMPLATES` setting in your `settings.py` file. Ensure a template path is set, or add one with `'DIRS': [BASE_DIR / 'myportal' / 'templates']`.

```
{# myportal/templates/globus-portal-framework/v2/components/search-results.html #}
<div>
  {% for result in search.search_results %}
  <div class="card my-3">
    <div class="card-header">
      <h3 class="search-title">
        <a href="{% url 'detail' globus_portal_framework.index result.subject %}">{
↪{result.title|default:'Result'}}</a>
      </h3>
    </div>
    <div class="card-body">
      <table class="table table-sm borderless">
        <tbody>
        <tr>
          {% for item in result.search_highlights %}
          <th>{{item.title}}</th>
          {% endfor %}
        </tr>
        <tr>
          {% for item in result.search_highlights %}
          {% if item.type == "date" %}
          <th>{{item.value | date:"DATETIME_FORMAT"}}</th>
```

(continues on next page)

(continued from previous page)

```
          {% else %}
          <th>{{item.value}}</th>
          {% endif %}
          {% endfor %}
        </tr>
        </tbody>
      </table>
    </div>
  </div>
  {% endfor %}
</div>
```

Reloading the page should result in empty search results. Don't worry, we will fix those in a minute!

Let's review some template context above:

- `myportal/templates/globus-portal-framework/v2/components/search-results.html` – is the path you need to override the base template. This tells Django to replace the existing template with the new *search-results.html* file
- `search.search_results` – is context provided by the search view. It contains information on the response from the Globus Search query
- `{% url 'detail' globus_portal_framework.index result.subject %}` – builds the detail page for viewing specific information about a search result
- `result (temp var)` – contains both raw search information, in addition to any fields defined in `SEARCH_RESULTS.myindex.fields`.
- `result.search_highlights` – is a field that doesn't exist yet, let's create it!

Now to fix search results to make them show up properly. The new field `search_highlights` is needed to pick relavent information to show on the search page. Add the following to your `fields.py` file:

```
import datetime
from typing import List, Mapping, Any

def search_highlights(result: List[Mapping[str, Any]]) -> List[Mapping[str, dict]]:
    """Prepare the most useful pieces of information for users on the search results␣
↪page."""
    search_highlights = list()
    for name in ["author", "date", "tags"]:
        value = result[0].get(name)
        value_type = "str"

        # Parse a date if it's a date. All dates expected isoformat
        if name == "date":
            value = datetime.datetime.fromisoformat(value)
            value_type = "date"
        elif name == "tags":
            value = ", ".join(value)

        # Add the value to the list
        search_highlights.append(
            {
                "name": name,
```

(continues on next page)

(continued from previous page)

```
                "title": name.capitalize(),
                "value": value,
                "type": value_type,
            }
        )
    return search_highlights
```

And add the new setting in `settings.py`

```
"fields": [
      ...
      ("search_highlights", fields.search_highlights),
  ],
```

Search results will now look much nicer!

2.4.2 Customizing the Detail Page

Modifying the result detail page will be similar to adding search highlights above with some differences. The approach begins the same way, by creating a file that shadows the name of the original.

```
{% extends 'globus-portal-framework/v2/detail-overview.html' %}

{% block detail_search_content %}

<h3 class="text-center mb-5">General Info</h3>
<div class="row">
  <div class="col-md-6">
    {% include 'globus-portal-framework/v2/components/detail-dc-metadata.html' %}
  </div>
  <div class="col-md-6">
    {% include 'globus-portal-framework/v2/components/detail-general-metadata.html' %}
  </div>
</div>

{% endblock %}
```

Make sure the filename is `myportal/templates/globus-portal-framework/v2/components/search-results.html` Let's review some differences in this template:

- `extends` - This template builds on the existing template instead of replacing it
- `block` - Tells Django to replace this specific content with our own
- `include` - Include some additional templates to render some specific data
  - DC Metadata - A template to render metadata in Datacite Format
  - General Metadata - A template to render any project-specific metadata

The dc and general project metadata templates help render commonly desired fields for the detail page. Their use is entierly optional. They require fields named *dc* and *project_metadata* respectively, see the following new fields below.

```
def dc(result):
    """Render metadata in datacite format, Must confrom to the datacite spec"""
```

(continues on next page)

(continued from previous page)

```
    date = datetime.datetime.fromisoformat(result[0]['date'])
    return {
        "formats": ["text/plain"],
        "creators": [{"creatorName": result[0]['author']}],
        "contributors": [{"contributorName": result[0]['author']}],
        "subjects": [{"subject": s for s in result[0]['tags']}],
        "publicationYear": date.year,
        "publisher": "Organization",
        "dates": [{"date": date,
                   "dateType": "Created"}],
        "titles": [{"title": result[0]['title']}],
        "version": "1",
        "resourceType": {
            "resourceTypeGeneral": "Dataset",
            "resourceType": "Dataset"
        }
    }


def project_metadata(result):
    """Render any project-specific metadata for this project. Does not conform to
    a spec and can be of any type, although values should be generally human readable."""
    project_metadata_names = ['times_accessed', 'original_collection_name']
    return {k: v for k, v in result[0].items() if k in project_metadata_names}
```

Add the fields to settings.py.

```
  "fields": [
        ...
        ("dc", fields.dc),
        ("project_metadata", fields.dc),
    ],

And the detail page will now be much nicer.
```

2.5 Advanced: Multiple Indices

If you have multiple search indices and want to re-use the same search views with different templates, you can set the `template_override_dir` for a given index.

```
SEARCH_INDEXES = {
    'myindex': {
        ...
        'template_override_dir': 'myportal',
    }
}
```

You need to create a directory for the `template_override_dir` name you choose, and place all of your templates within that directory. Your structure should look like this:

```
myportal/
    manage.py
    myportal/
        templates/
          myportal/  # <-- Create this folder, move all index-specific templates under it
            globus-portal-framework/
              v2/
                components/
                    detail-nav.html
                    search-facets.html
                    search-results.html
                search.html
                detail-overview.html
                detail-transfer.html
```

For any views where multi-index templates are supported, Globus Portal Framework will first attempt to find the index specific template, then will back-off to the 'standard' template without your project prefix. For example, if you define two templates called "myportal/templates/globus-portal-framework/v2/components/search-results.html" and "myportal/templates/myportal/globus-portal-framework/v2/components/search-results.html", when your user visits the "myportal" index Globus Portal Framework will first try to load "myportal/templates/myportal/globus-portal-framework/v2/components/search-results.html", then fall back to the other template if it does not exist.

You can extend this behavior yourself with the "index_template" templatetag.

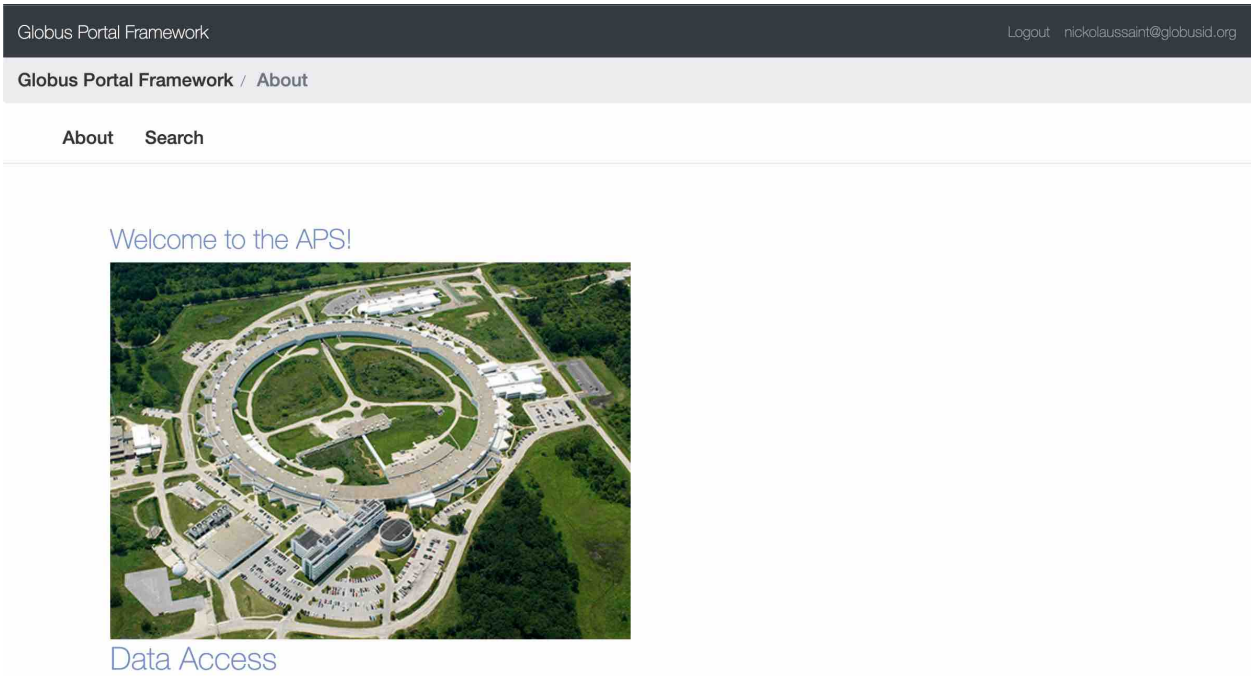
```
{# Include at the top of the page #}
{% load index_template %}

{# Use this to check for a 'template override' for this search index #}
{% index_template 'globus-portal-framework/v2/components/search-results.html' as it_
↪search_results %}
{% include it_search_results %}
```

You can always view the DGPF template source for a reference.

2.6 Previewing Content

Most Globus Connect Server Collections support HTTPS as a way to access files on a Globus Collection. For portals in Django Globus Portal Framework, this prodives a way to display content directly in overview pages or search results. For example, if each search result in a science portal embeds an image that conveys the quality of the dataset, it can significantly improve the search experience for users. In addition, static resources can be used to improve the look of any static info pages, like in the example below:



2.6.1 Displaying Public Images

For public images, the process is only a matter of constructing a valid link to the Globus Collection, and the image will be displayed when the page loads. Most GCSv5.4 collections support an HTTPS URL for this exact purpose. The domain name for a collection can be found by searching collections on the Globus Webapp or via the Globus CLI:

```
$ globus collection show my-collection-uuid
Display Name:                my-collection
...
HTTPS URL:                   https://g-7581c.fd635.8444.data.globus.org
```

Links to files on a collection contain the HTTPS URL as a domain name in addition to path to the desired file. Note the path must be correct, and the directory on the collection must be shared with Public. The URL can be used on HTML img tags like the example below:

```
<img src="https://g-7581c.fd635.8443.data.globus.org/portal/public/APS_aerial.jpeg"␣
↪width="50%">
```

2.6.2 Non-Image Content

Files on a Globus Collection can be accessed via GET requests. Typically this involves some javascript code to make the GET request, then dynamically attaching it to an HTML tag. Numerous examples exist online for making the request and attaching response text to elements.

2.6.3 Private Content Over HTTPS

Accessing non-public data from a collection over HTTPS is a much more involved process, and requires making the request to the file using an authorized access token for the GCS Collection. More info on making authorized requests to a Globus Collection can be found here.

Django Globus Portal Framework can be configured to request the data_access scope above by adding the scope to the `SOCIAL_AUTH_GLOBUS_SCOPE` variable in `settings.py`. The token can be loaded for each user using the following Django view below (**Remember to login again to populate the token for your user**):

```
from django.http import JsonResponse
from globus_portal_framework.gclients import load_globus_access_token

def https_access(request):
    # The resource server for collections is typicaly the collection uuid
    token = load_globus_access_token(request.user, 'globus_collection_uuid')
    return JsonResponse({'http_access_token': token})
```

Warning: Special care should be taken when exposing user tokens to the browser. With added flexibilty comes increased risk. Don't expose tokens which are not needed by front-end applications.

HTTP GET requests using javascript can request private documents from the GCS server by first requesting the data_access token from the server backend (view above), then making a second request to the GCS collection with the access token set as the Authorization header as described in the Accessing Data section of the GCS Docs.

Note that for binary content like images, the HTML `<img>` tag cannot provide the authorization header, meaning the content must be fetched first using javascript and attached manually.

Better support for accessing private documents on Globus Collections is planned at some point in the future.

2.7 Custom Search Views

Note: **Note**: Most portals shouldn't need these changes, it's only necessary if you need finer control over the low level components that make requests to the Globus Search service. Try customizing your search templates [here first](https://github.com/globusonline/django-globus-portal-framework/wiki/Customizing-Fields-and-Templates).

At some point, you may need more control over the context for your search templates. An example may be rendering graphs based on the statistics returned by facets. The examples below include writing your own 'search' view, and wiring it up to override the default search view provided by Globus Portal Framework. This requires two changes:

- views.py – Writing a custom view to capture a user search parameters and return context to the portal
- urls.py – Apply the custom view to one or more of your indices

In your project's urls.py:

```
from django.urls import path, include
from globus_portal_framework.urls import register_custom_index
from exalearn.views import mysearch

# Register a new custom index converter.
register_custom_index('custom_search', ['myindex'])
```

(continues on next page)

(continued from previous page)

```
urlpatterns = [
    # Override 'search' url with custom search view
    path('<custom_search:index>/', mysearch, name='search'),
    # Provides the basic search portal
    path('', include('globus_portal_framework.urls')),
    path('', include('globus_portal_framework.urls_debugging')),
    path('', include('social_django.urls', namespace='social')),
]
```

The URLs are still very similar to the old ones, except for the *register_custom_index* line. This registers a new [Django URL Converter](https://docs.djangoproject.com/en/2.2/topics/http/urls/#registering-custom-path-converters) for the indices you include in the second argument. This does a couple things:

- Only the indices you include will use the new functionality
- Unrelated URLs won't match as an 'index', such as if a bot searches for '/robots.txt'. Only URLs which map to the indices you include in the list will be matched.

In your project's views.py:

```
from django.shortcuts import render
from globus_portal_framework.gsearch import post_search, get_search_query, get_search_
↪filters, get_template

def mysearch(request, index):
    query = get_search_query(request)
    filters = get_search_filters(request)
    context = {'search': post_search(index, query, filters, request.user,
                                     request.GET.get('page', 1))}
    return render(request, get_template(index, 'search.html'), context)
```

With the above, we're using a number of components to prepare and process the search:

- *get_search_query* fetches the user's query from the query params on the request
- *get_search_filters* fetches any filters in query params that should be applied to the search
- *post_search* prepares and sends the request to Globus Search, in addition to processing the results based on configuration defined in settings.SEARCH_INDEXES
- *get_template* will attempt to grab an overridden custom index template if it exists (templates/myindex/search.html), grab a standard overridden template (templates/search.html), or simply render the basic Globus Portal Framework search.html template

2.7.1 We Need To Go Deeper

It's possible to access the lowest level mechanism to modify requests made to Globus Search. Again, most use-cases shouldn't require this, but it may be necessary if you need to utilize Globus Search feature not provided by Globus Portal Framework. If this is the case, consider opening an issue, so we can provide the feature for others.

Advanced search views.py

```
from django.shortcuts import render
from globus_portal_framework.gsearch import (
```

(continues on next page)

(continued from previous page)

```
    get_search_query, get_search_filters,
    process_search_data, get_facets, get_template, get_index
)
from globus_portal_framework.gclients import load_search_client


def my_advanced_search(request, index):
    index_data = get_index(index)
    search_cli = load_search_client(request.user)
    query = get_search_query(request)
    filters = get_search_filters(request)
    data = {'q': query,
            'filters': filters}
    result = search_cli.post_search(index_data['uuid'], data)
    search_data = {
        'search_results': process_search_data(index_data.get('fields', []),
                                              result.data['gmeta']),
        'facets': get_facets(result, index_data.get('facets', []),
                             filters, index_data.get('filter_match')),
    }
    context = {'search': search_data}
    return render(request, get_template(index, 'search.html'), context)
```

The custom search function above allows for extended flexibility in what gets sent to Globus Search and the resulting context you want rendered in your templates. There are a few new components we're using:

- *get_index* Will search settings.SEARCH_INDEXES for your index, and return data associated with it.
- *load_search_client* Will fetch the base globus_sdk.SearchClient class loaded with an authorizer for the current user (Or nothing, if the user is logged out).
- *process_search_data* applies the *fields* defined in settings.SEARCH_INDEXES to the search data returned by Globus Search.
- *get_facets* processes the facet data returned by Globus Search, and prepares the context so that users can filter on those facets on their next action.

2.8 Facet Modifiers

Modify facet data before it is passed to templates for rendering. This is handy if you re-use the same views for many indices, and want to keep them the same. Configure facet modifiers by setting the *facet_modifiers* on your search index configuration ([See Example](https://github.com/globusonline/django-globus-portal-framework/wiki/Configuring-Facets)). Globus Portal Framework comes with a few built-in facet modifiers:

```
'facet_modifiers': [
    'globus_portal_framework.modifiers.facets.drop_empty',
    'globus_portal_framework.modifiers.facets.sort_terms',
    'globus_portal_framework.modifiers.facets.sort_terms_numerically',
    'globus_portal_framework.modifiers.facets.reverse',
],
```

Each of these modifiers will be applied to facets in the order they are defined.

2.8.1 Custom Facet Modifiers

You can add your own modifiers to the list:

```
'facet_modifiers': [
    'globus_portal_framework.modifiers.facets.drop_empty',
    'myapp.modifiers.drop_small_buckets',
    'myapp.modifiers.do_the_thing',
],
```

Each entry in the list is an import string to a Python callable. Each callable needs to take a single argument for the list of facets, and return the new modified list of facets. Modifying the *facets* parameter won't cause issues.

Define the function below in a module that matches the import string above. The function below should be defined in a module called *myapp/modifiers.py*

```
def drop_small_buckets(facets):
    """Drop any buckets on facets with small values. This prevents
    users from gaining insights about search data with carefully crafted
    filtering."""
    for facet in facets:
        if not facet.get('buckets'):
            continue
        facet['buckets'] = [b for b in facet['buckets'] if b['count'] > 5]
    return facets
```

2.9 Search Settings Reference

This reference contains all search related settings that go in `settings.py`

2.9.1 Search Indices

Configure your Globus Search indexes with the `SEARCH_INDEXES` variable in your *myproject/settings.py* file. Below are all of the main top-level fields. The following match directly with Globus Search fields, and match directly with Globus Search.

- facets
- sort
- boosts
- bypass_visible_to
- result_format_version

See more information in the Globus Search documentation

Field Name	Description
name	The title of this search index.
uuid	The Globus Search UUID for this Globus Search Index
fields	User defined functions for processing metadata returned by Globus Searches
facets	Display stats on search results, provide corresponding filters for future Searches
facet_modifiers	Change how facets are displayed. See *Facet Modifiers*
sort	Sort results of a Globus Search
boosts	Increase or decrease values of fields
filter_match	Default filtering on 'term' facets. 'match-any' or 'match-all' supported
template_override_dir	Directory for using different custom templates per-index on a multi-index portal
result_format_version	Version of Search Result documents to return
bypass_visible_to	Show all search records regardless visible_to permission (index admins only)

Search Settings Example

```
SEARCH_INDEXES = {
    'myportal': {
        'name': 'My Science Portal',
        'uuid': '5e83718e-add0-4f06-a00d-577dc78359bc',
        'fields': [
            'my_title'
        ],
        'facets': [
            {
                'field_name': 'foo.bar.baz',
            }
        ],
        'sort': [
            {
                'field_name': 'path.to.date',
                'order': 'asc'
            }
        ]
        'boosts': [
            {
                'field_name': 'author',
                'factor': 5
            }
        ],
        'filter_match': 'match-all',
        'template_override_dir': 'myportal',
        'bypass_visible_to': True,
    }
}
```

2.9.2 Configuring Facets

Field information for each list entry defined in "facets" is described here. Information within each object is checked and mostly forwarded on to Globus Search. See more information in the Globus Search documentation

Field Name	Type	Description
name	String	Title for this facet
field_name	String	The search metadata field where this facet should be applied
type	String	Type of facet. Supported: terms, date_histogram, numeric_histogram, sum, avg
size	Integer	Number of 'buckets' to return
histogram_range	Object	Contains 'low' and 'high' number or date to specify range bounds
date_interval	String	Date Unit to use. Supported: years, months, days, hours, minutes, seconds

For more information on how facets can be displayed, See *Facet Modifiers*

Facet Setting Example

The following is an example SEARCH_INDEXES inside `myproject/settings.py`

```
SEARCH_INDEXES = {
    'myportal': {
        'name': 'My Portal',
        'uuid': '5e83718e-add0-4f06-a00d-577dc78359bc',
        'fields': [],
        'facets': [
            {
                'name': 'Term Facets',
                'field_name': 'mybooks.genre',
            },
            {
                'name': 'Dates',
                'field_name': 'dc.dates.date',
                'type': 'date_histogram',
                'date_interval': 'day',
            },
            {
                'name': 'File Sizes',
                'field_name': 'files.length',
                'type': 'numeric_histogram',
                'histogram_range': {'low': 0, 'high': 10000}
            },
        ],
        'facet_modifiers': [
            'globus_portal_framework.modifiers.facets.drop_empty',
        ],
        'filter_match': 'match-all',
    }
}
```

2.10 Customizing URLs

Below is reference information on basic configuration, in addition to more advanced customization for broader use-cases. Both info for *urls.py* and template url names are listed.

2.10.1 Reference URLs

The base URLs for Globus Portal Framework are below:

```
path('<index:index>/', search, name='search'),
path('<index:index>/detail-preview/<subject>/',
     detail_preview, name='detail-preview'),
path('<index:index>/detail-preview/<subject>/<endpoint>/<path:url_path>/',
     detail_preview, name='detail-preview'),
path('<index:index>/detail-transfer/<subject>', detail_transfer,
     name='detail-transfer'),
path('<index:index>/detail/<subject>/', detail, name='detail'),
```

You can reverse these URLs via templates with the following:

```
<a href="{% url 'search' 'myindex'  %}">Link to myindex search page</a>
<a href="{% url 'detail' 'myindex' 'my-globus-search-subject' %}">Link to record detail␣
↪page</a>
<a href="{% url 'detail-preview' 'myindex' 'my-globus-search-subject' %}">Link to record␣
↪preview page</a>
<a href="{% url 'detail-transfer' 'myindex' 'my-globus-search-subject' %}">Link to␣
↪record transfer page</a>
```

2.10.2 Configuring Built-in URLs

Globus Portal Framework comes built-in with a top level URL you can use to get started. Below is all you need to start.

```
from django.urls import path, include

urlpatterns = [
    # Provides the basic search portal
    path('', include('globus_portal_framework.urls')),
    # (OPTIONAL) Provides debugging for your Globus Search Index result data
    path('', include('globus_portal_framework.urls_debugging')),
    # (RECOMMENDED) Provides Login urls for Globus Auth
    path('', include('social_django.urls', namespace='social')),
]
```

2.10.3 Configuring Custom URLs

You can customize URLs in a couple of different ways. The first simple way is to keep the view the same but change the URL mapping scheme. This is handy if you want to make your URLs look different. For example, you want your search page to be *http://myportal.org/science-index/data/*, where 'data' is a new customized URL.

The second customization is adding a custom view instead of the standard DGPF one. You can add custom views for one index while keeping the URLs for other indices the same.

Remapping URLs

```
from django.urls import path

from globus_portal_framework.views import search
from myportal.views import advanced_search

urlpatterns = [
    path('<index:index>/data', search, name='search'),
    path('<index:index>/advanced-search', advanced_search, name='advanced-search'),
    path('', include('globus_portal_framework.urls')),
    path('', include('globus_portal_framework.urls_debugging')),
    path('', include('social_django.urls', namespace='social')),
]
```

The URLs above remaps the standard 'search' view to a custom URL. Make sure the new mapping is above the other Globus Portal Framework URLs so it takes precedence. There is also a custom 'advanced-search' url above, so all indices can use a different custom view for different types of searches.

However, there is a potential problem here. These URLs force ALL urls to use the new '/data' URL and '/advanced-search' url. What if you only want one index to use Advanced searches?

2.10.4 Remapping Custom Index URLs

We can register a custom index to use the view we want, and it won't affect URLs for other indices.

```
from django.urls import path
from globus_portal_framework.urls import register_custom_index
from myportal.views import advanced_search
# You can register more than one string to match your index. In this
# case, we may have another Globus Search index we want to match as a
# test index. In that case, the test index will re-use all of the prod
# index views.
register_custom_index('my_index', ['my-index', 'my-test-index'])

urlpatterns = [
    path('<my_index>/advanced-search', advanced_search, name='advanced-search'),
    path('', include('globus_portal_framework.urls')),
    path('', include('globus_portal_framework.urls_debugging')),
    path('', include('social_django.urls', namespace='social')),
]
```

Now, *https://my-index/advanced-search* will call *advanced_search()* and all other views will call the regular *search()* view and have the search url *https://my-other-index/*.

2.11 Requiring Login

If you need to reach out to Globus Services, such as Transfer, users will need to be pre-authenticated so the portal can use their tokens. Django has built-in functions to check this, but needs some tuning to work with Python-Social-Auth.

Make sure you have a working portal with Globus Auth. If not, review the tutorial documentation and make sure you can login with Globus.

2.11.1 Settings

First, tell Django where your login link is. For Python Social Auth, the link below will work fine.

```
LOGIN_URL = '/login/globus'
```

2.11.2 Views

Now, you can define your views like this:

```
from django.shortcuts import render
from django.contrib.auth.decorators import login_required
from globus_portal_framework.gclients import load_transfer_client


@login_required
def my_view(request, index):
    tc = load_transfer_client(request.user)
    mypaths = tc.operation_ls('ddb59aef-6d04-11e5-ba46-22000b92c6ec', path='/share/godata
↪')
    context = {'mypaths': mypaths}
    return render(request, 'mypaths.html', context)
```

If your user encounters *my_view* above, the *@login_required* decorator will redirect them to the *LOGIN_URL* defined in your *settings.py*

2.11.3 Disabling Links Requiring Login

If you want to prevent unauthenticated users from even navigating to your views in the first place, you can disable links in templates.

```
<nav>
  <ul class="nav nav-tabs">
    <li class="nav-item">
      <a class="nav-link" href="{% url 'my-landing-page' globus_portal_framework.index %}
↪">About {{project_title}}</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="{% url 'my-projects-page' globus_portal_framework.index
↪%}">Projects</a>
    </li>
    <li class="nav-item">
```

(continues on next page)

(continued from previous page)

```
      <a class="nav-link" href="{% url 'my-search' globus_portal_framework.index %}">
↪Search</a>
    </li>
    <li class="nav-item">
      <a class="nav-link
        {% if not request.user.is_authenticated %}
        disabled
        {% endif %}
      " href="{% url 'my-files' globus_portal_framework.index %}">View My Files</a>
    </li>
  </ul>
</nav>
```

In this example using Bootstrap, the "View My Files" link will be disabled.

CHAPTER

THREE

SETTINGS

Here is a full list of settings allowed by Globus Portal Framework. Adding these settings to your own *settings.py* file will override the defaults below.

3.1 General Settings

General settings should apply to most portals.

```
# Configure the general title for your project
PROJECT_TITLE = 'My Project'
```

3.2 Auth Settings

```
# Tells Django where your projects login url is
# Useful for using the ``@login_required`` decorator on custom views
LOGIN_URL = '/login/globus'

# Get your keys at 'developers.globus.org'
# Login is managed primarily by python-social-auth
SOCIAL_AUTH_GLOBUS_KEY = '<your_Globus_Auth_Client_ID>'
SOCIAL_AUTH_GLOBUS_SECRET = '<your_Globus_Auth_Client_Secret>'

# Tells Django how to authenticate users
AUTHENTICATION_BACKENDS = [
    'globus_portal_framework.auth.GlobusOpenIdConnect',
    'django.contrib.auth.backends.ModelBackend',
]

# Can be used to customize Gloubs Auth.
# setting access_type to offline enables refresh tokens.
# WARNING: This can be dangerous.
SOCIAL_AUTH_GLOBUS_AUTH_EXTRA_ARGUMENTS = {
    'access_type': 'offline',
}

# Set scopes what user tokens to request from Globus Auth
SOCIAL_AUTH_GLOBUS_SCOPE = [
```

(continues on next page)

(continued from previous page)

```
    'urn:globus:auth:scope:search.api.globus.org:search',
    'urn:globus:auth:scope:transfer.api.globus.org:all',
    'urn:globus:auth:scope:groups.api.globus.org:view_my_groups_and_memberships'
]
```

3.3 Search Settings

- SEARCH_INDEXES – The main listing of search indexes in your portal

```
# Number of search results that will display on the search page before paginating
SEARCH_RESULTS_PER_PAGE = 10
# Max number of pages to display
SEARCH_MAX_PAGES = 10

# Default query if no user search or saved session search.
# Note: This will be slow for an index with a lot of search data.
DEFAULT_QUERY = '*'
# Filtering behavior to use for searching across indices.
# Note: Can be overrided by per-index settings.
DEFAULT_FILTER_MATCH = FILTER_MATCH_ALL
```

3.4 Templates

```
# Setting for which Globus Portal Framework template set you should use.
# Mostly for backwards compatibility, but allows for a fully custom set of
# templates.
BASE_TEMPLATES = 'globus-portal-framework/v2/'

# General Template settings. Full example listed for reference, but only
# the last three context_processors are relevant
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                # Social Django context processors for login
                'social_django.context_processors.backends',
                'social_django.context_processors.login_redirect',
                # Globus Portal Framework general context for search indices
                # and other general context per-template.
                'globus_portal_framework.context_processors.globals',
            ],
```

(continues on next page)

(continued from previous page)

```
        },
    },
]
```

3.5 Under the Hood

Modify default client loading behavior. Typically only used in [DGPF Confidential Client](https://github.com/globus/dgpf-confidential-client) ` GLOBUS_CLIENT_LOADER = 'globus_portal_framework.gclients.load_globus_client' `

CHAPTER

FOUR

SETTINGS EXAMPLE

Below is a reference Django `settings.py` file for Django Globus Portal Framework. It contains the basics for running a new portal.

Note: The project name used is `myproject`, but this needs to be updated to the name you chose when running the initial `django-admin startproject myproject` command.

```
"""
Django settings for myportal project.

Generated by 'django-admin startproject' using Django 3.2.8.

For more information on this file, see
https://docs.djangoproject.com/en/3.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.2/ref/settings/
"""
import logging
from pathlib import Path

log = logging.getLogger(__name__)

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/

# SECURITY WARNING: keep all secret keys used in production secret!
# You can generate a secure secret key with `openssl rand -hex 32`
SECRET_KEY = 'django-insecure-47f(ub2qs-n!b@&&)tis&l$&qf1%^@&jy-95jx!bahqrm^19m2'
# Your portal credentials for enabling user login via Globus Auth
SOCIAL_AUTH_GLOBUS_KEY = ''
SOCIAL_AUTH_GLOBUS_SECRET = ''

# This is a general Django setting if views need to redirect to login
# https://docs.djangoproject.com/en/3.2/ref/settings/#login-url
LOGIN_URL = '/login/globus'
```

(continues on next page)

(continued from previous page)

```
# This dictates which scopes will be requested on each user login
SOCIAL_AUTH_GLOBUS_SCOPE = [
    'urn:globus:auth:scope:search.api.globus.org:search',
]

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'globus_portal_framework',
    'social_django',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'globus_portal_framework.middleware.ExpiredTokenMiddleware',
    'globus_portal_framework.middleware.GlobusAuthExceptionMiddleware',
    'social_django.middleware.SocialAuthExceptionMiddleware',
]

# Authentication backends setup OAuth2 handling and where user data should be
# stored
AUTHENTICATION_BACKENDS = [
    'globus_portal_framework.auth.GlobusOpenIdConnect',
    'django.contrib.auth.backends.ModelBackend',
]

ROOT_URLCONF = 'myportal.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'myportal' / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
```

(continues on next page)

(continued from previous page)

```
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                'globus_portal_framework.context_processors.globals',
            ],
        },
    },
]

WSGI_APPLICATION = 'myportal.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


LOGGING = {
    'version': 1,
    'handlers': {
        'stream': {'level': 'DEBUG', 'class': 'logging.StreamHandler'},
    },
    'loggers': {
        'django': {'handlers': ['stream'], 'level': 'INFO'},
        'globus_portal_framework': {'handlers': ['stream'], 'level': 'INFO'},
        'myportal': {'handlers': ['stream'], 'level': 'INFO'},
    },
}

# Internationalization
# https://docs.djangoproject.com/en/3.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
```

(continues on next page)

(continued from previous page)

```
# https://docs.djangoproject.com/en/3.2/howto/static-files/

STATIC_URL = '/static/'

# Default primary key field type
# https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

try:
    from .local_settings import *
except ImportError:
    expected_path = Path(__file__).resolve().parent / 'local_settings.py'
    log.warning(f'You should create a file for your secrets at {expected_path}')
```

CHAPTER

FIVE

LOCAL CREDENTIALS

For simplicity, the tutorial has you adding secret credentials directly to a *settings.py* file. This isn't ideal if using version control to track general settings. A common workaround is to keep a `local_settings.py` in the same directory not tracked by git, and importing them into your main `settings.py` file at the very bottom.

Example `local_settings.py` file:

```
# local_settings.py
SOCIAL_AUTH_GLOBUS_KEY = '<YOUR APP CLIENT ID>'
SOCIAL_AUTH_GLOBUS_SECRET = '<YOUR APP SECRET>'
DEBUG = True
```

Then, at the bottom of `settings.py`:

```
# Override any settings here if a local_settings.py file exists
try:
    from .local_settings import *  # noqa
except ImportError:
    pass
```

Your normal settings.py file can now be safely tracked within version control.

CHAPTER

SIX

DEPLOYMENT

Checkout the Django Deployment Checklist for useful tips on your deployment.

6.1 Docker

Note: We don't support docker yet, but this is a planned feature!

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search