
UriRegistry Documentation

Release 0.1.3

Onroerend Erfgoed

Aug 16, 2022

CONTENTS

1	Introduction	3
2	Architecture	5
2.1	Schema	5
2.2	pyramid_urireferencer	6
3	Services	7
3.1	UriRegistry	7
3.2	Pyramid_urireferencer	9
4	Development	11
4.1	Setting up a development environment	11
4.2	Configuring a UriRegistry	11
4.3	Testing	12
4.4	Adding pyramid_urireferencer to an application	12
5	API Documentation	15
5.1	uriregistry	15
5.2	pyramid_urireferencer	17
6	History	21
6.1	0.3.1 (16-08-2022)	21
6.2	0.3.0 (21-02-2022)	21
6.3	0.2.0 (07-12-2020)	21
6.4	0.1.2 (08-06-2017)	21
6.5	0.1.1 (04-08-2015)	21
6.6	0.1.0 (11-06-2015)	22
7	Indices and tables	23
	Python Module Index	25
	HTTP Routing Table	27
	Index	29

This project aims to provide a central location for keeping track of different resources through their URI's. It has been designed to answer one simple question: in which applications is a certain resource (ie. a URI) being used?

INTRODUCTION

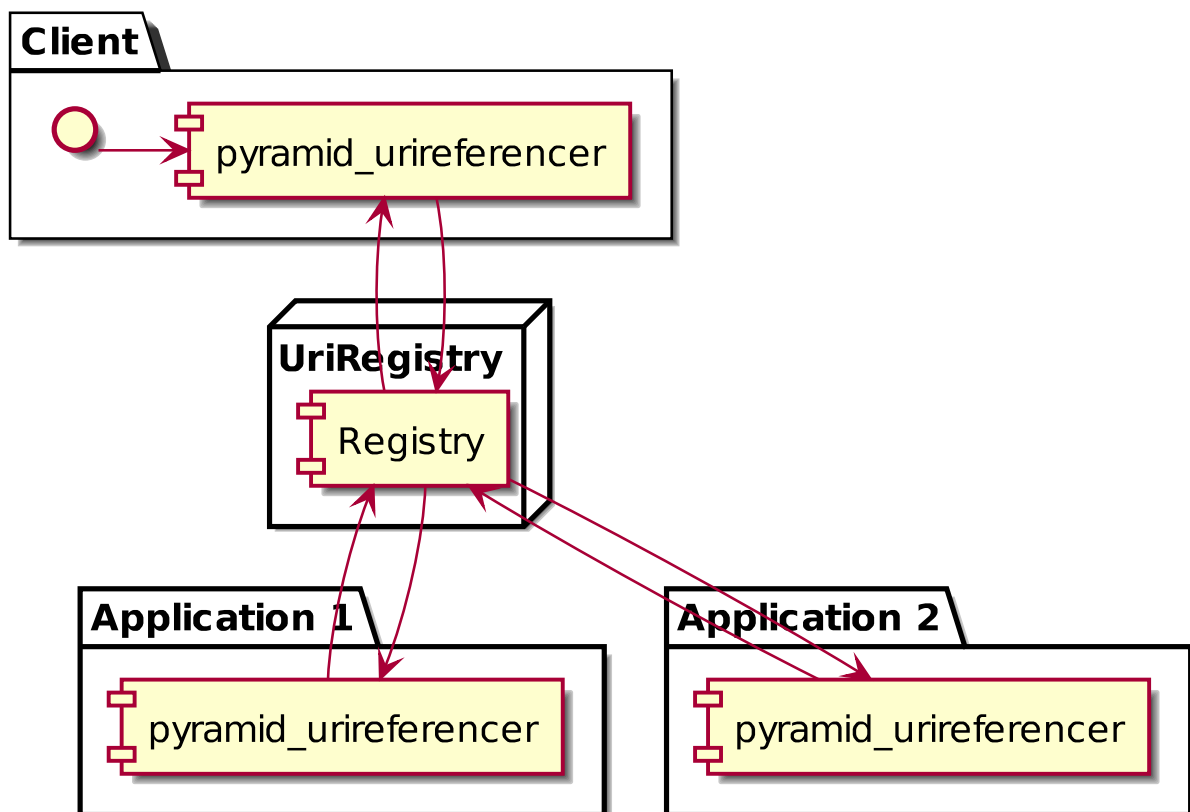
This project, UriRegistry, aims to solve one (and only one) problem that can arise when working with distributed systems. Is one of my resources being used somewhere? This can be especially important when we want to delete a certain resource. While in typical RESTful fashion, we cannot guarantee a client that a certain resource will keep on existing indefinitely, it can be a good thing to let our end-users know that the resource is currently in use somewhere.

The basic idea is quite simple. Every application has an endpoint that can be used by clients to ask if a certain URI is in use in that application. The application replies if that is the case and if so with some information where that might be the case.

Finally, there's a separate application, the UriRegistry, that can serve as a single point of access. To ensure that a client does not need to know all the applications that might be using one of its resources, we use a central registry. This central registry knows which URI is used in which application, queries the applications in question, gathers and tallies the results and presents these to the client. The client itself only needs to know where the central registry is. The registry takes care of the rest.

ARCHITECTURE

2.1 Schema



A client queries the registry with a certain URI, eg. *http://id.erfgoed.net/foo/bar*. The registry checks if it knows any applications that might be using this URI. It discovers that two applications could possibly be using it. Both applications are queried. In each application a `pyramid_urireferencer.referenceer.AbstractReferencer` has been configured that can check if an incoming URI is in use in the application. The results are sent back to the registry. The registry tallies the results and aggregates them. A final response is sent back to the client.

2.2 pyramid_urireferencer

This plugin will expose a service at */references*. This service endpoint will take a single parameter, *uri*. A full request looks like eg. */references?uri=http://id.erfgoed.net/besluiten/1*. Within the application, a check will be executed to see if the application keeps references to this particular URI.

The plugin also provides a method `pyramid_urireferencer.referencer.Referencer.is_referenced()` that can be used to contact the central registry to see if a certain URI is in use somewhere. This method requires a function `pyramid_urireferencer.referencer.Referencer.get_uri()` to determine the uri of the current request.

SERVICES

3.1 UriRegistry

The central UriRegistry has a single endpoint that can be called.

GET /references

Query the registry to see if and possibly where a URI is in use.

Structure of a response:

- **query_uri** - the URI we're looking for
- **success** - **Did the registry succeed in querying the underlying** services. Will be *True* if all requests succeeded, else *False*.
- **has_references** - Will be *True* as soon as one application has at least one reference to the item in question.
- **count** - How many references were found in total?
- **applications** - A list of all applications that were queried and the results they returned
 - **title** - A title for the application
 - **service_url** - Url of the application's references service
 - **uri** - Uri of the application. Does not need to be a http uri.
 - **success** - Will be *True* if the request for this application succeeded, else *False*.
 - **has_references** - Will be *True* if at least one reference was found. If the request failed, this will be *None*. Not *False*.
 - **count** - Returns the number of references found. If the request failed (success==`False`), this will be *None*.
 - **items** - A list of resources that have a reference to the URI in question. For performance reasons, a maximum of 5 resources is allowed. If the request failed, this will be *None*.
 - * **title** - Title of the resource
 - * **uri** - Uri of the resource

Example request:

```
GET /references?uri=http://id.erfgoed.net/foobar/2
Host: uriregistry.org
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "query_uri": "http://id.erfgoed.net/foobar/2",
  "success": true,
  "has_references": true,
  "count": 8,
  "applications": [
    {
      "count": 8,
      "title": "app1",
      "success": true,
      "has_references": true,
      "uri": "http://www.erfgoed.net",
      "service_url": "http://www.erfgoed.net/references",
      "items": [
        {
          "name": "itemname1",
          "uri": "http://www.erfgoed.net/baz/1"
        }, {
          "name": "itemname_2",
          "uri": "http://www.erfgoed.net/baz/10"
        }, {
          "name": "itemname_3",
          "uri": "http://www.erfgoed.net/baz/14"
        }, {
          "name": "itemname_4",
          "uri": "http://www.erfgoed.net/baz/34"
        }
      ],
    }, {
      "count": null,
      "title": "app2",
      "success": false,
      "has_references": null,
      "service_url": "http://something.erfgoed.net/references",
      "uri": "http://something.erfgoed.net",
      "items": null
    }
  ],
}
```

Query Parameters

- **uri** – The uri of the resource the client wants information on. Required.

Status Codes

- **200 OK** – The service has a valid answer
- **400 Bad Request** – There's something wrong with the request, eg. no URI parameter present.

3.2 Pyramid_urireferencer

Every application that implements *pyramid_urireferencer* has the same endpoint as the central registry, but with a slightly different response set.

GET /references

Query the application to see if and possibly where a certain URI is in use.

Example request:

```
GET /references?uri=http://id.erfgoed.net/foobar/2
Host: www.erfgoed.net
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "count": 8,
  "title": "app1",
  "success": true,
  "has_references": true,
  "uri": "http://www.erfgoed.net",
  "service_url": "http://www.erfgoed.net/references",
  "items": [
    {
      "name": "itemname1",
      "uri": "http://www.erfgoed.net/baz/1"
    }, {
      "name": "itemname2",
      "uri": "http://www.erfgoed.net/baz/10"
    }, {
      "name": "itemname3",
      "uri": "http://www.erfgoed.net/baz/14"
    }, {
      "name": "itemname4",
      "uri": "http://www.erfgoed.net/baz/34"
    }
  ]
}
```

Query Parameters

- **uri** – The uri of the resource the client wants information on. Required.

Status Codes

- **200 OK** – The service has a valid answer
- **400 Bad Request** – There's something wrong with the request, eg. no URI parameter present.

DEVELOPMENT

4.1 Setting up a development environment

Check out the code.

```
$ git clone https://github.com/OnroerendErfgoed/uriregistry.git
```

Create a virtual environment (require virtualenvwrapper).

```
# Create a new environment
$ mkvirtualenv uriregistry
# Activate an existing environment
$ workon uriregistry
```

Install requirements.

```
$ pip install -r requirements-dev.txt
$ python setup.py develop
```

Run the application with the sample config `sample.yaml`.

```
$ pserve development.ini
```

Point your browser at <http://localhost:6543> to see it in action!

4.2 Configuring a UriRegistry

Your UriRegistry can be configured with a YAML file. By default, a file `sample.yaml` in the `uriregistry` package is used, but you can change this withing your own `development.ini`.

```
uriregistry.config = %(here)s/myapp.yaml
```

In this config file you specify which applications can be called by the registry when looking for URI's in use. You can also specify for each URI template in what application it might be found.

```
applications:
- uri: http://localhost:5555
  name: app1
  service_url: http://localhost:5555/references
- uri: http://localhost:2222
```

(continues on next page)

(continued from previous page)

```
name: app2
service_url: http://localhost:2222/references
uri_templates:
- match_uri: http://id.erfgoed.net/foobar/\d+
  applications:
    - http://localhost:5555
    - http://localhost:2222
- match_uri: http://id.erfgoed.net/bar/\w+
  applications:
    - http://localhost:5555
- match_uri: http://id.erfgoed.net/foo/.+
  applications:
    - http://localhost:2222
```

4.3 Testing

Tests are run with `pytest`. We support the last python 2.x release and the two most current python 3.x release. To make testing easier, use `tox`.

```
# Run all tests for all environments
$ tox
# No coverage
$ py.test
# Coverage
$ py.test --cov uriregistry --cov-report term-missing tests
```

4.4 Adding `pyramid_urireferencer` to an application

When you want to add an application to the network of applications, you need to include the `pyramid_urireferencer` library. Add it to your `requirements.txt` and `setup.py` requirements.

Add the library to your application by including the following in your main:

```
config.include('pyramid_urireferencer')
```

Now you need to configure your application. Edit your `development.ini` and add two configuration options.

```
# settings for the urireferencer
# A dotted name indicating where your referencer can be found
urireferencer.referencer = myapp.referencer.MyReferencer
# The url pointing towards your own UriRegistry
urireferencer.registry_url = http://localhost:6543
```

Of course, you also need to write this referencer. To do this, create an object that implements the abstract `pyramid_urireferencer.referencer.AbstractReferencer`. Depending on your needs it might be easier to extend the `pyramid_urireferencer.referencer.Referencer`. This class already has a `is_referenced()` method. But the method requires a function `get_uri()` to determine the uri of the current request. The `get_uri()` still needs to be implemented. The referencer also requires you to implement the `references()` method.


```
from pyramid_urirefencer.referencer import Referencer
from pyramid_urirefencer.models import ApplicationResponse

class DemoReferencer(Referencer):

    def get_uri(self, request):
        id = request.matchdict['id']
        if request.data_manager.get(aid).type == 'cirkel':
            return request.registry.settings['cirkel.uri'].format(id)
        else:
            return request.registry.settings['square.uri'].format(id)

    def references(self, uri):
        try:
            # Generate a demo response
            has_references = True
            count = 8
            items = []
            for x in range(1, 5):
                items.append(Item("itemname_" + str(x), "http://demo_uri/" + str(x)))
            success = True
        except:
            has_references = None
            count = None
            items = None
            success = False
        return ApplicationResponse(
            'My application',
            'http://app.me',
            'http://app.me/references',
            success,
            has_references,
            count,
            items
        )
```


API DOCUMENTATION

5.1 uriregistry

`uriregistry._load_configuration(path)`

Load the configuration for the UriRegistry.

Parameters `path` (*str*) – Path to the config file in YAML format.

Returns A dict with the config options.

`uriregistry._parse_settings(settings)`

Parse the relevant settings for this application.

Parameters `settings` (*dict*) –

`uriregistry.main(global_config, **settings)`

This function returns a Pyramid WSGI application.

Parameters `global_config` (*pyramid.config.Configurator*) –

5.1.1 Models module

`class uriregistry.models.Application(uri, title, service_url)`

Represents the config for an application.

Parameters

- **uri** (*string*) – A uri that identifies the application
- **title** (*string*) – A title for the application
- **service_url** (*string*) – The url for the service that can be queried

`class uriregistry.models.UriTemplate(match_uri, applications)`

Represents the config for a certain uri template.

Parameters

- **match_uri** (*string*) – A regex that needs to be matched.
- **applications** (*list*) – A list of application uri's.

`matches(uri)`

Does the URI match this template?

Parameters `uri` (*string*) – URI to be matched

Return type boolean

5.1.2 Registry module

class `uriregistry.registry.UriRegistry`(*applications=[]*, *uris=[]*)
Central registry that tracks uris and the applications they are being used in.

get_applications(*uri*)
Get all applications that might have a reference to this URI.

Parameters *uri* (*string*) – Uri for which the applications need to be found.

`uriregistry.registry._build_uri_registry`(*registry*, *registryconfig*)

Parameters

- **registry** (*pyramid.registry.Registry*) – Pyramid registry
- **registryconfig** (*dict*) – UriRegistry config in dict form.

Return type `uriregistry.registry.UriRegistry`

`uriregistry.registry.get_uri_registry`(*registry*)
Get the `uriregistry.registry.UriRegistry` attached to this pyramid application.

Return type `uriregistry.registry.UriRegistry`

5.1.3 Views module

`uriregistry.views._get_registry_response`(*application_responses*, *uri*)
Generate the final response by aggregating all the application responses.

Parameters

- **application_responses** (*list*) – All `pyramid_urirefencer.models.ApplicationResponse` instances.
- **uri** (*str*) – Uri that was evaluated
- **base_uri** (*str*) – Base uri of the uri that was evaluated

Returns `pyramid_urirefencer.models.RegistryResponse` with all information the registry has collected

5.1.4 Utils module

`uriregistry.utils.query_application`(*app*, *uri*)
Checks if a certain app has references to a URI.

Parameters

- **uriregistry.models.Application** – The application to be evaluated
- **uri** – The uri that has to be checked

Rtype `pyramid_urirefencer.models.ApplicationResponse`

5.2 pyramid_urireferencer

`pyramid_urireferencer._add_referencer(registry)`

Gets the Referencer from config and adds it to the registry.

`pyramid_urireferencer.get_referencer(registry)`

Get the referencer class

Return type `pyramid_urireferencer.referencer.AbstractReferencer`

`pyramid_urireferencer.includeme(config)`

this function adds some configuration for the application

5.2.1 Models module

class `pyramid_urireferencer.models.ApplicationResponse(title, uri, service_url, success, has_references, count, items)`

Represents what a certain application will send back to the registry when asked if a certain uri is used by the application.

Parameters

- **title** (*string*) – Title of the application
- **uri** (*string*) – A uri for the application, not guaranteed to be a http url.
- **service_url** (*string*) – The url that answered the question
- **success** (*boolean*) – Was the querie successful?
- **has_references** (*boolean*) – Were any references found?
- **count** (*int*) – How many references were found?
- **items** (*list*) – A list of items that have a reference to the uri under survey. Limited to 5 items for performance reasons.

static `load_from_json(data)`

Load a `ApplicationResponse` from a dictionary or string (that will be parsed as json).

class `pyramid_urireferencer.models.Item(title, uri)`

A single item that holds a reference to the queried uri.

Parameters

- **title** (*string*) – Title of the item.
- **uri** (*string*) – Uri of the item.

static `load_from_json(data)`

Load a `Item` from a dictionary ot string (that will be parsed as json)

class `pyramid_urireferencer.models.RegistryResponse(query_uri, success, has_references, count, applications)`

Represents what the registry will send back to a client when asked if a certain uri is used somewhere.

Parameters

- **query_uri** (*string*) – Uri of the resource unser survey.
- **success** (*boolean*) – Were all the queries successful?
- **has_references** (*boolean*) – Were any references found?

- **count** (*int*) – How many references were found?
- **applications** (*list*) – A list of application results.

static load_from_json(*data*)

Load a RegistryResponse from a dictionary or a string (that will be parsed as json).

5.2.2 Protected resources module

This module is used when blocking operations on a certain uri that might be used in external applications. .. version-added:: 0.4.0

pyramid_urirefencer.protected_resources.protected_operation(*fn*)

Use this decorator to prevent an operation from being executed when the related uri resource is still in use. The parent_object must contain:

- **a request**
 - with a registry.queryUtility(IReferencer)

Raises

- **pyramid.httpexceptions.HTTPConflict** – Signals that we don't want to delete a certain URI because it's still in use somewhere else.
- **pyramid.httpexceptions.HTTPInternalServerError** – Raised when we were unable to check that the URI is no longer being used.

pyramid_urirefencer.protected_resources.protected_operation_with_request(*fn*)

Use this decorator to prevent an operation from being executed when the related uri resource is still in use. The request must contain a registry.queryUtility(IReferencer) :raises pyramid.httpexceptions.HTTPConflict: Signals that we don't want to

delete a certain URI because it's still in use somewhere else.

Raises **pyramid.httpexceptions.HTTPInternalServerError** – Raised when we were unable to check that the URI is no longer being used.

pyramid_urirefencer.protected_resources.protected_view(*view, info*)

allows adding *protected=True* to a view_config`

5.2.3 Referencer module

class pyramid_urirefencer.referencer.AbstractReferencer

This is an abstract class that defines what a Referencer needs to be able to handle.

It does two things:

- Check if a uri is being used in this application and report on this.
- Check if a certain uri is being used in another application by query a central registry. * this requires a function [`get_uri\(\)`](#) to determine the uri of the current request

abstract get_uri(*request*)

This method extracts a uri from the request. This is the uri that needs to be checked.

Parameters request – **pyramid.request.Request** with useful configuration information and connections of the application (registry, route_url, session) to determine the references

Return type string uri: URI of the resource we need to check for

abstract is_referenced(uri)

This method checks if a certain uri is being referenced from resources in other applications.

Parameters uri (string) – URI of the resource that needs to be checked

Return type `pyramid_urireferencer.models.RegistryResponse`

abstract references(uri, request)

This method checks if a certain uri is being referenced by any other resource within this application.

Parameters

- **uri** (string) – URI of the resource we need to check for
- **request** – `pyramid.request.Request` with useful configuration information and connections of the application (registry, route_url, session) to determine the references

Return type `pyramid_urireferencer.models.ApplicationResponse`

class `pyramid_urireferencer.referencer.Referencer`(registry_url, **kwargs)

This is an implementation of the `AbstractReferencer` that adds a generic `is_referenced()` method and plain methods: `references()` and `get_uri()`

is_referenced(uri)

This method checks if a certain uri is being referenced from resources in other applications.

Parameters uri (string) – URI of the resource that needs to be checked

Return type `pyramid_urireferencer.models.RegistryResponse`

5.2.4 Renderers module

`pyramid_urireferencer.renderers.application_adapter(obj, request)`

Adapter for rendering a `pyramid_urireferencer.models.ApplicationResponse` to json.

Parameters obj (`pyramid_urireferencer.models.ApplicationResponse`) – The response to be rendered.

Return type dict

`pyramid_urireferencer.renderers.registry_adapter(obj, request)`

Adapter for rendering a `pyramid_urireferencer.models.RegistryResponse` to json.

Parameters obj (`pyramid_urireferencer.models.RegistryResponse`) – The response to be rendered.

Return type dict

5.2.5 Views module

HISTORY

6.1 0.3.1 (16-08-2022)

- Avoid “max_workers must be greater than 0”-Exception in case no applications are listed in the configurarion (#97)

6.2 0.3.0 (21-02-2022)

- Improved performance (#8)
- Pyramid 2.0 upgrade (#85)

6.3 0.2.0 (07-12-2020)

- Library update(#69)
- Py3 compatibel maken (#66)

6.4 0.1.2 (08-06-2017)

- Make compatible with Python 3.5
- Update pyramid_urireferencer to 0.6.0 (#9)

6.5 0.1.1 (04-08-2015)

- Update pyramid_urireferencer to 0.4.0

6.6 0.1.0 (11-06-2015)

- Initial version
- Works with pyramid_urireferencer 0.2.0.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

p

- `pyramid_urireferencer`, [17](#)
- `pyramid_urireferencer.models`, [17](#)
- `pyramid_urireferencer.protected_resources`, [18](#)
- `pyramid_urireferencer.referencer`, [18](#)
- `pyramid_urireferencer.renderers`, [19](#)
- `pyramid_urireferencer.views`, [19](#)

U

- `uriregistry`, [15](#)
- `uriregistry.models`, [15](#)
- `uriregistry.registry`, [16](#)
- `uriregistry.utils`, [16](#)
- `uriregistry.views`, [16](#)

HTTP ROUTING TABLE

/references

GET /references, 9

Symbols

`_add_referencer()` (in module `pyramid_urireferencer`), 17
`_build_uri_registry()` (in module `uriregistry.registry`), 16
`_get_registry_response()` (in module `uriregistry.views`), 16
`_load_configuration()` (in module `uriregistry`), 15
`_parse_settings()` (in module `uriregistry`), 15

A

`AbstractReferencer` (class in `pyramid_urireferencer.referencer`), 18
`Application` (class in `uriregistry.models`), 15
`application_adapter()` (in module `pyramid_urireferencer.renderers`), 19
`ApplicationResponse` (class in `pyramid_urireferencer.models`), 17

G

`get_applications()` (`uriregistry.registry.UriRegistry` method), 16
`get_referencer()` (in module `pyramid_urireferencer`), 17
`get_uri()` (`pyramid_urireferencer.referencer.AbstractReferencer` method), 18
`get_uri_registry()` (in module `uriregistry.registry`), 16

I

`includeme()` (in module `pyramid_urireferencer`), 17
`is_referenced()` (`pyramid_urireferencer.referencer.AbstractReferencer` method), 19
`is_referenced()` (`pyramid_urireferencer.referencer.Referencer` method), 19
`Item` (class in `pyramid_urireferencer.models`), 17

L

`load_from_json()` (`pyramid_urireferencer.models.ApplicationResponse`

`static method`), 17
`load_from_json()` (`pyramid_urireferencer.models.Item` `static method`), 17
`load_from_json()` (`pyramid_urireferencer.models.RegistryResponse` `static method`), 18

M

`main()` (in module `uriregistry`), 15
`matches()` (`uriregistry.models.UriTemplate` method), 15
module
`pyramid_urireferencer`, 17
`pyramid_urireferencer.models`, 17
`pyramid_urireferencer.protected_resources`, 18
`pyramid_urireferencer.referencer`, 18
`pyramid_urireferencer.renderers`, 19
`pyramid_urireferencer.views`, 19
`uriregistry`, 15
`uriregistry.models`, 15
`uriregistry.registry`, 16
`uriregistry.utils`, 16
`uriregistry.views`, 16

P

`protected_operation()` (in module `pyramid_urireferencer.protected_resources`), 18
`protected_operation_with_request()` (in module `pyramid_urireferencer.protected_resources`), 18
`protected_view()` (in module `pyramid_urireferencer.protected_resources`), 18
`pyramid_urireferencer` module, 17
`pyramid_urireferencer.models` module, 17
`pyramid_urireferencer.protected_resources` module, 18
`pyramid_urireferencer.referencer` module, 18

pyramid_urirefencer.renderers
 module, 19
pyramid_urirefencer.views
 module, 19

Q

query_application() (*in module uriregistry.utils*), 16

R

Referencer (*class in pyramid_urirefencer.referencer*),
 19
references() (*pyramid_urirefencer.referencer.AbstractReferencer*
 method), 19
registry_adapter() (*in module pyra-*
 mid_urirefencer.renderers), 19
RegistryResponse (*class in pyra-*
 mid_urirefencer.models), 17

U

uriregistry
 module, 15
UriRegistry (*class in uriregistry.registry*), 16
uriregistry.models
 module, 15
uriregistry.registry
 module, 16
uriregistry.utils
 module, 16
uriregistry.views
 module, 16
UriTemplate (*class in uriregistry.models*), 15