guillotina Documentation

Release 3.3.14

Ramon Navarro Bosch & Asko Soukka & Nathan Van Gheem

Jul 21, 2018

Contents

(REST Resource Application Server)

Guillotina is the only full-featured Python AsyncIO REST Resource Application Server designed for high-performance, horizontally scaling solutions. It is a high performance web server based on many of the technologies and lessons learned from Plone, Pyramid, Django and others all while utilizing Python's great AsyncIO library.

Using Python's AsyncIO, it works well with micro-service oriented environments.

Features:

- REST JSON API
- Built-in authentication/authorization, built-in JWT support
- Hierarchical data/url structure
- Permissions/roles/groups
- Fully customizable permission/roles/groups based on hierarchical data structure
- Robust customizable component architecture and configuration syntax
- Content types, dynamic behaviors
- Built-in CORS support

CHAPTER 1

Detailed Documentation

1.1 About

As the Web evolves, so do the frameworks that we use to work with the Web. Guillotina is part of that evolution, providing an asynchronous web server with a rich, REST-ful API to build your web applications around.

It is designed for building JavaScript applications. It is an API framework, not a typical template-based rendering framework like most web frameworks (Django/Pyramid/Plone). What we mean by this is that Guillotina will not generate HTML out-of-the-box for you. It is designed to be consumed by JavaScript applications that do the HTML rendering.

Features:

- REST JSON API
- Built-in authentication/authorization, built-in JWT support
- Hierarchical data/url structure
- Permissions/roles/groups
- Fully customizable permission/roles/groups based on hierarchical data structure
- Robust customizable component architecture and configuration syntax
- Content types, dynamic behaviors
- Built-in CORS support
- JSON schema support
- PostgreSQL and CockroachDB drivers
- Blobs

Guillotina is built on the lessons learned from great technologies of the open source projects Plone, Zope, Pyramid and Django.

Inspirations:

- Plone/Zope's hierarchical data model
- Pyramid's decorator-based auto-discovery application configuration syntax
- Django's global application settings style syntax
- Zope's component architecture for building robustly customizable applications
- Plone/Zope's security model
- JSON Schema

Lessons Learned (from said inspired frameworks):

- Trade-offs for the sake of performance is okay
- Too many complex dependencies causes difficulties in management and upgrades
- It's okay to fork dependency packages

1.1.1 History lesson

In the beginning, there was *bobo*.

bobo was what Jim Fulton called his initial idea of mapping objects to web urls. It's an old idea. A beautiful idea. The developers of Guillotina think it's the best possible way to conceptualize most content-centric APIs and organization of how your web applications think about data or their APIs.

Think about this simple example. Assuming you have the following dictionary:

```
{
  "foo": {
    "bar": {}
  }
}
```

The corresponding urls for a site based off this dictionary would be:

- http://localhost:8080/
- http://localhost:8080/foo
- http://localhost:8080/foo/bar

And so on... It's a simple way to build APIs from data around a hierarchy (or tree).

Extrapolating this concept, Jim Fulton also built the ZODB package. This was a complete database built on serializing Python objects using the pickle library. Then, frameworks like Zope (and eventually Plone), used this database and the *bobo* style of publishing objects to URLs to build a framework and CMS around.

Forked dependency packages

Guillotina has eaten a few packages that would have otherwise been dependencies.

The reasons for forking are:

- Required to support asyncio
- Provide tighter fit for framework
- Make installations less painful and error-prone

- Groking framework is easier when there is one package to import from

Forks:

- parts of the ZODB data model: we're on a relational storage model now
- plone.behavior
- zope.security
- zope.schema
- zope.component/zope.configuration
- zope.dublincore
- zope.i18n
- zope.lifecycleevent
- zope.location
- zope.event

1.1.2 What it isn't

- Guillotina is not a replacement for Plone
- Guillotina is not a re-implementation of Plone
- Guillotina does not implement all the features and APIs of Plone

It could some day with the *guillotina_cms* package but replacement of Plone is not the goal of Guillotina.

1.2 Quickstart

How to quickly get started using `guillotina`.

This tutorial will assume usage of virtualenv. You can use your own preferred tool for managing your python environment.

This tutorial assumes you have postgresql running

Setup the environment:

```
virtualenv .
```

Install `guillotina`:

```
./bin/pip install guillotina
```

Generate configuration file (requires `cookiecutter`):

```
./bin/pip install cookiecutter
./bin/g create --template=configuration
```

Finally, run the server:

```
./bin/g
```

The server should now be running on http://0.0.0.0:8080

Then, use Postman, `curl` or whatever tool you prefer to interact with the REST API.

Modify the configuration in `config.yaml` to customize server setttings.

1.2.1 Postgresql installation instructions

If you do not have a postgresql database server installed, you can use docker to get one running quickly.

Example docker run command:

```
docker run -e POSTGRES_DB=guillotina -e POSTGRES_USER=guillotina -p 127.0.0.
↪1:5432:5432 postgres:9.6
```

1.2.2 Creating a container

Guillotina containers are the building block of all other content. A container is where you place all other content for your application. Only containers can be created inside databases.

Let's create one:

```
  curl -X POST -H "Accept: application/json" --user root:root -H "Content-
↪Type: application/json" -d '{
    "@type": "Container",
    "title": "Guillotina 1",
    "id": "guillotina",
    "description": "Description"
  }' "http://127.0.0.1:8080/db/"
```

and create content inside the container:

```
  curl -X POST -H "Accept: application/json" --user root:root -H "Content-
↪Type: application/json" -d '{
    "@type": "Item",
    "title": "News",
    "id": "news"
  }' "http://127.0.0.1:8080/db/guillotina/"
```

1.3 REST API Reference

Contents:

1.3.1 Application

GET Get application data **permission: guillotina.AccessContent**http

```
GET / HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/ -H 'Accept: application/json' --user root:root
```

httpie

```
http http://localhost:8080/ Accept:application/json -a root:root
```

python-requests

```
requests.get('http://localhost:8080/', headers={'Accept': 'application/json'}
↪, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{
    "@type": "Application",
    "databases": [
        "db"
    ],
    "static_directory": [],
    "static_file": [
        "favicon.ico"
    ]
}
```

GET @apidefinition Get API Definition **permission: guillotina.GetContainers**http

```
GET /@apidefinition HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/@apidefinition -H 'Accept: application/json' --
↪user root:root
```

httpie

```
http http://localhost:8080/@apidefinition Accept:application/json -a␣
↪root:root
```

python-requests

```
requests.get('http://localhost:8080/@apidefinition', headers={'Accept':
↪'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5
```

```
{
    "guillotina.interfaces.content.IApplication": {
        "GET": {
            "context": [
                "guillotina.interfaces.content.IApplication",
                "guillotina.interfaces.content.ITraversable",
                "zope.interface.Interface"
            ],
            "description": "Retrieves serialization of application",
            "method": "GET",
            "module": "guillotina.api.app.get",
            "permission": "guillotina.AccessContent",
            "responses": {
                "200": {
                    "description": "Application data",
                    "schema": {
                        "$ref": "#/definitions/Application"
                    }
                }
            },
            "summary": "Get application data"
        },
        "PUT": {
            "context": [
                "guillotina.interfaces.content.IApplication",
                "guillotina.interfaces.content.ITraversable",
                "zope.interface.Interface"
            ],
            "ignore": true,
            "method": "PUT",
            "permission": "guillotina.MountDatabase"
        },
        "endpoints": {
            "@apidefinition": {
                "GET": {
                    "context": [
                        "guillotina.interfaces.content.IApplication",
                        "guillotina.interfaces.content.ITraversable",
                        "zope.interface.Interface"
                    ],
                    "description": "Retrieves information on API
↪configuration",
                    "method": "GET",
                    "module": "guillotina.api.app.get_api_definition",
                    "name": "@apidefinition",
                    "permission": "guillotina.GetContainers",
                    "summary": "Get API Definition"
                }
            }
        }
    },
    "guillotina.interfaces.content.IContainer": {
        "DELETE": {
            "context": [
                "guillotina.interfaces.content.IContainer",
                "guillotina.interfaces.content.IResource",
                "guillotina.interfaces.content.ILocation",
                "guillotina.interfaces.content.IAsyncContainer",
```

```
                "guillotina.interfaces.content.ITraversable",
                "guillotina.component.interfaces.ISite",
                "guillotina.component.interfaces.IPossibleSite",
                "zope.interface.Interface"
            ],
            "method": "DELETE",
            "permission": "guillotina.DeleteContainers",
            "summary": "Delete container"
        },
        "endpoints": {
            "@addons": {
                "DELETE": {
                    "context": [
                        "guillotina.interfaces.content.IContainer",
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "guillotina.interfaces.content.IAsyncContainer",
                        "guillotina.interfaces.content.ITraversable",
                        "guillotina.component.interfaces.ISite",
                        "guillotina.component.interfaces.IPossibleSite",
                        "zope.interface.Interface"
                    ],
                    "method": "DELETE",
                    "module": "guillotina.api.addons.uninstall",
                    "name": "@addons",
                    "parameters": [
                        {
                            "in": "body",
                            "name": "body",
                            "schema": {
                                "$ref": "#/definitions/Addon"
                            }
                        }
                    ],
                    "permission": "guillotina.ManageAddons",
                    "summary": "Uninstall an addon from container"
                },
                "GET": {
                    "context": [
                        "guillotina.interfaces.content.IContainer",
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "guillotina.interfaces.content.IAsyncContainer",
                        "guillotina.interfaces.content.ITraversable",
                        "guillotina.component.interfaces.ISite",
                        "guillotina.component.interfaces.IPossibleSite",
                        "zope.interface.Interface"
                    ],
                    "method": "GET",
                    "module": "guillotina.api.addons.get_addons",
                    "name": "@addons",
                    "permission": "guillotina.ManageAddons",
                    "responses": {
                        "200": {
                            "description": "Get list of available and␣
↪installed addons",
                            "schema": {
                                "$ref": "#/definitions/AddonResponse"
```

```
                            }
                        }
                    },
                    "summary": "List available addons"
                },
                "POST": {
                    "context": [
                        "guillotina.interfaces.content.IContainer",
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "guillotina.interfaces.content.IAsyncContainer",
                        "guillotina.interfaces.content.ITraversable",
                        "guillotina.component.interfaces.ISite",
                        "guillotina.component.interfaces.IPossibleSite",
                        "zope.interface.Interface"
                    ],
                    "method": "POST",
                    "module": "guillotina.api.addons.install",
                    "name": "@addons",
                    "parameters": [
                        {
                            "in": "body",
                            "name": "body",
                            "schema": {
                                "$ref": "#/definitions/Addon"
                            }
                        }
                    ],
                    "permission": "guillotina.ManageAddons",
                    "summary": "Install addon to container"
                }
            },
            "@registry": {
                "GET": {
                    "context": [
                        "guillotina.interfaces.content.IContainer",
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "guillotina.interfaces.content.IAsyncContainer",
                        "guillotina.interfaces.content.ITraversable",
                        "guillotina.component.interfaces.ISite",
                        "guillotina.component.interfaces.IPossibleSite",
                        "zope.interface.Interface"
                    ],
                    "method": "GET",
                    "name": "@registry",
                    "permission": "guillotina.ReadConfiguration",
                    "responses": {
                        "200": {
                            "description": "Successuflly registered interface
↪",
                            "schema": {
                                "properties": {
                                    "value": {
                                        "type": "object"
                                    }
                                }
                            },
```

```
                            "type": "object"
                        }
                    },
                    "summary": "Read container registry settings"
                },
                "PATCH": {
                    "context": [
                        "guillotina.interfaces.content.IContainer",
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "guillotina.interfaces.content.IAsyncContainer",
                        "guillotina.interfaces.content.ITraversable",
                        "guillotina.component.interfaces.ISite",
                        "guillotina.component.interfaces.IPossibleSite",
                        "zope.interface.Interface"
                    ],
                    "method": "PATCH",
                    "name": "@registry",
                    "parameters": {
                        "in": "body",
                        "name": "body",
                        "schema": {
                            "properties": {
                                "value": {
                                    "required": true,
                                    "type": "any"
                                }
                            }
                        },
                        "type": "object"
                    },
                    "permission": "guillotina.WriteConfiguration",
                    "responses": {
                        "200": {
                            "description": "Successuflly wrote configuration"
                        }
                    },
                    "summary": "Update registry setting"
                },
                "POST": {
                    "context": [
                        "guillotina.interfaces.content.IContainer",
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "guillotina.interfaces.content.IAsyncContainer",
                        "guillotina.interfaces.content.ITraversable",
                        "guillotina.component.interfaces.ISite",
                        "guillotina.component.interfaces.IPossibleSite",
                        "zope.interface.Interface"
                    ],
                    "method": "POST",
                    "name": "@registry",
                    "parameters": [
                        {
                            "in": "body",
                            "name": "body",
                            "schema": {
                                "properties": {
```

```
                                    "initial_values": {
                                        "required": false,
                                        "type": "object"
                                    },
                                    "interface": {
                                        "required": true,
                                        "type": "string"
                                    }
                                }
                            },
                            "type": "object"
                        }
                    ],
                    "permission": "guillotina.RegisterConfigurations",
                    "responses": {
                        "200": {
                            "description": "Successuflly registered interface
↪"
                        }
                    },
                    "summary": "Register a new interface to for registry␣
↪settings"
                }
            },
            "@types": {
                "GET": {
                    "context": [
                        "guillotina.interfaces.content.IContainer",
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "guillotina.interfaces.content.IAsyncContainer",
                        "guillotina.interfaces.content.ITraversable",
                        "guillotina.component.interfaces.ISite",
                        "guillotina.component.interfaces.IPossibleSite",
                        "zope.interface.Interface"
                    ],
                    "method": "GET",
                    "name": "@types",
                    "permission": "guillotina.AccessContent",
                    "responses": {
                        "200": {
                            "description": "Result results on types",
                            "schema": {
                                "properties": {}
                            }
                        }
                    },
                    "summary": "Read information on available types"
                }
            },
            "@user": {
                "GET": {
                    "context": [
                        "guillotina.interfaces.content.IContainer",
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "guillotina.interfaces.content.IAsyncContainer",
                        "guillotina.interfaces.content.ITraversable",
```

```
                        "guillotina.component.interfaces.ISite",
                        "guillotina.component.interfaces.IPossibleSite",
                        "zope.interface.Interface"
                    ],
                    "method": "GET",
                    "module": "guillotina.api.user.get_user_info",
                    "name": "@user",
                    "permission": "guillotina.AccessContent",
                    "responses": {
                        "200": {
                            "description": "Get information on the user",
                            "schema": {
                                "properties": {}
                            }
                        }
                    },
                    "summary": "Get information on the currently logged in
↪user"
                }
            },
            "@ws": {
                "GET": {
                    "context": [
                        "guillotina.interfaces.content.IContainer",
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "guillotina.interfaces.content.IAsyncContainer",
                        "guillotina.interfaces.content.ITraversable",
                        "guillotina.component.interfaces.ISite",
                        "guillotina.component.interfaces.IPossibleSite",
                        "zope.interface.Interface"
                    ],
                    "method": "GET",
                    "name": "@ws",
                    "permission": "guillotina.AccessContent",
                    "summary": "Make a web socket connection"
                }
            },
            "@wstoken": {
                "GET": {
                    "context": [
                        "guillotina.interfaces.content.IContainer",
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "guillotina.interfaces.content.IAsyncContainer",
                        "guillotina.interfaces.content.ITraversable",
                        "guillotina.component.interfaces.ISite",
                        "guillotina.component.interfaces.IPossibleSite",
                        "zope.interface.Interface"
                    ],
                    "method": "GET",
                    "name": "@wstoken",
                    "permission": "guillotina.AccessContent",
                    "responses": {
                        "200": {
                            "description": "The new token",
                            "schema": {
                                "properties": {
```

```
                                    "token": {
                                        "required": true,
                                        "type": "string"
                                    }
                                }
                            }
                        }
                    },
                    "summary": "Return a web socket token"
                }
            }
        }
    },
    "guillotina.interfaces.content.IDatabase": {
        "DELETE": {
            "context": [
                "guillotina.interfaces.content.IDatabase",
                "guillotina.interfaces.content.ITraversable",
                "guillotina.interfaces.content.IAsyncContainer",
                "zope.interface.Interface"
            ],
            "ignore": true,
            "method": "DELETE",
            "permission": "guillotina.UmountDatabase"
        },
        "GET": {
            "context": [
                "guillotina.interfaces.content.IDatabase",
                "guillotina.interfaces.content.ITraversable",
                "guillotina.interfaces.content.IAsyncContainer",
                "zope.interface.Interface"
            ],
            "method": "GET",
            "permission": "guillotina.GetContainers",
            "responses": {
                "200": {
                    "description": "Get a list of containers",
                    "schema": {
                        "properties": {
                            "containers": {
                                "items": {
                                    "type": "string"
                                },
                                "type": "array"
                            }
                        }
                    }
                }
            },
            "summary": "Get list of containers"
        },
        "POST": {
            "context": [
                "guillotina.interfaces.content.IDatabase",
                "guillotina.interfaces.content.ITraversable",
                "guillotina.interfaces.content.IAsyncContainer",
                "zope.interface.Interface"
            ],
```

```
            "description": "Creates a new container on the database",
            "method": "POST",
            "parameters": [
                {
                    "in": "body",
                    "name": "body",
                    "schema": {
                        "$ref": "#/definitions/BaseResource"
                    }
                }
            ],
            "permission": "guillotina.AddContainer",
            "responses": {
                "200": {
                    "description": "Container result",
                    "schema": {
                        "$ref": "#/definitions/BaseResource"
                    }
                }
            },
            "summary": "Create a new Container"
        }
    },
    "guillotina.interfaces.content.IResource": {
        "DELETE": {
            "context": [
                "guillotina.interfaces.content.IResource",
                "guillotina.interfaces.content.ILocation",
                "zope.interface.Interface"
            ],
            "method": "DELETE",
            "permission": "guillotina.DeleteContent",
            "responses": {
                "200": {
                    "description": "Successuflly deleted resource"
                }
            },
            "summary": "Delete resource"
        },
        "GET": {
            "context": [
                "guillotina.interfaces.content.IResource",
                "guillotina.interfaces.content.ILocation",
                "zope.interface.Interface"
            ],
            "method": "GET",
            "permission": "guillotina.ViewContent",
            "responses": "guillotina.api.content.get_content_json_schema_
↪responses",
            "summary": "Retrieves serialization of resource"
        },
        "OPTIONS": {
            "context": [
                "guillotina.interfaces.content.IResource",
                "guillotina.interfaces.content.ILocation",
                "zope.interface.Interface"
            ],
            "method": "OPTIONS",
```

```
            "permission": "guillotina.AccessPreflight",
            "summary": "Get CORS information for resource"
        },
        "PATCH": {
            "context": [
                "guillotina.interfaces.content.IResource",
                "guillotina.interfaces.content.ILocation",
                "zope.interface.Interface"
            ],
            "method": "PATCH",
            "parameters": "guillotina.api.content.patch_content_json_schema_
↪parameters",
            "permission": "guillotina.ModifyContent",
            "responses": {
                "200": {
                    "description": "Resource data",
                    "schema": {
                        "$ref": "#/definitions/Resource"
                    }
                }
            },
            "summary": "Modify the content of this resource"
        },
        "POST": {
            "context": [
                "guillotina.interfaces.content.IResource",
                "guillotina.interfaces.content.ILocation",
                "zope.interface.Interface"
            ],
            "method": "POST",
            "parameters": [
                {
                    "in": "body",
                    "name": "body",
                    "schema": {
                        "$ref": "#/definitions/AddableResource"
                    }
                }
            ],
            "permission": "guillotina.AddContent",
            "responses": {
                "200": {
                    "description": "Resource data",
                    "schema": {
                        "$ref": "#/definitions/ResourceFolder"
                    }
                }
            },
            "summary": "Add new resouce inside this container resource"
        },
        "endpoints": {
            "@all_permissions": {
                "GET": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
```

```
                    "method": "GET",
                    "module": "guillotina.api.content.all_permissions",
                    "name": "@all_permissions",
                    "permission": "guillotina.SeePermissions",
                    "responses": {
                        "200": {
                            "description": "All the permissions defined on␣
↪this resource",
                            "schema": {
                                "$ref": "#/definitions/AllPermissions"
                            }
                        }
                    },
                    "summary": "See all permission settings for this resource
↪"
                }
            },
            "@async-catalog-reindex": {
                "POST": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "POST",
                    "name": "@async-catalog-reindex",
                    "permission": "guillotina.ReindexContent",
                    "responses": {
                        "200": {
                            "description": "Successfully initiated reindexing
↪"
                        }
                    },
                    "summary": "Asynchronously reindex entire container␣
↪content"
                }
            },
            "@behaviors": {
                "DELETE": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "DELETE",
                    "module": "guillotina.api.behaviors.default_delete",
                    "name": "@behaviors",
                    "parameters": [
                        {
                            "in": "body",
                            "name": "body",
                            "schema": {
                                "$ref": "#/definitions/Behavior"
                            }
                        }
                    ],
                    "permission": "guillotina.ModifyContent",
                    "responses": {
```

```
                        "200": {
                            "description": "Successfully removed behavior"
                        },
                        "201": {
                            "description": "Behavior not assigned here"
                        }
                    },
                    "summary": "Remove behavior from resource"
                },
                "GET": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "GET",
                    "module": "guillotina.api.behaviors.default_get",
                    "name": "@behaviors",
                    "permission": "guillotina.AccessContent",
                    "responses": {
                        "200": {
                            "description": "A listing of behaviors for␣
↪content",
                            "schema": {
                                "$ref": "#/definitions/BehaviorsResponse"
                            }
                        }
                    },
                    "summary": "Get information on behaviors for this␣
↪resource"
                },
                "PATCH": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "PATCH",
                    "module": "guillotina.api.behaviors.default_patch",
                    "name": "@behaviors",
                    "parameters": [
                        {
                            "in": "body",
                            "name": "body",
                            "schema": {
                                "$ref": "#/definitions/Behavior"
                            }
                        }
                    ],
                    "permission": "guillotina.ModifyContent",
                    "responses": {
                        "200": {
                            "description": "Successfully added behavior"
                        },
                        "201": {
                            "description": "Behavior already assigned here"
                        }
                    },
```

```
                    "summary": "Add behavior to resource"
                }
            },
            "@canido": {
                "GET": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "GET",
                    "module": "guillotina.api.content.can_i_do",
                    "name": "@canido",
                    "parameters": [
                        {
                            "in": "query",
                            "name": "permission",
                            "required": true,
                            "type": "string"
                        }
                    ],
                    "permission": "guillotina.AccessContent",
                    "responses": {
                        "200": {
                            "description": "Successuflly changed permission"
                        }
                    }
                }
            },
            "@catalog": {
                "DELETE": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "DELETE",
                    "module": "guillotina.api.search.catalog_delete",
                    "name": "@catalog",
                    "permission": "guillotina.ManageCatalog",
                    "responses": {
                        "200": {
                            "description": "Successfully deleted catalog"
                        }
                    },
                    "summary": "Delete search catalog"
                },
                "POST": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "POST",
                    "module": "guillotina.api.search.catalog_post",
                    "name": "@catalog",
                    "permission": "guillotina.ManageCatalog",
                    "responses": {
```

```
                        "200": {
                            "description": "Successfully initialized catalog"
                        }
                    },
                    "summary": "Initialize catalog"
                }
            },
            "@catalog-reindex": {
                "POST": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "POST",
                    "name": "@catalog-reindex",
                    "permission": "guillotina.ReindexContent",
                    "responses": {
                        "200": {
                            "description": "Successfully reindexed content"
                        }
                    },
                    "summary": "Reindex entire container content"
                }
            },
            "@download": {
                "GET": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "GET",
                    "name": "@download",
                    "permission": "guillotina.ViewContent",
                    "traversed_service_definitions": {
                        "{field_name}": {
                            "parameters": [
                                {
                                    "description": "Name of file field",
                                    "in": "path",
                                    "name": "field_name",
                                    "required": true
                                }
                            ],
                            "responses": {
                                "200": {
                                    "description": "Successfully updated
↪content"
                                }
                            },
                            "summary": "Download the content of a file"
                        }
                    }
                }
            },
            "@search": {
                "GET": {
```

```
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "GET",
                    "module": "guillotina.api.search.search_get",
                    "name": "@search",
                    "parameters": [
                        {
                            "in": "query",
                            "name": "q",
                            "required": true,
                            "type": "string"
                        }
                    ],
                    "permission": "guillotina.SearchContent",
                    "responses": {
                        "200": {
                            "description": "Search results",
                            "schema": {
                                "$ref": "#/definitions/SearchResults"
                            },
                            "type": "object"
                        }
                    },
                    "summary": "Make search request"
                },
                "POST": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "POST",
                    "module": "guillotina.api.search.search_post",
                    "name": "@search",
                    "parameters": [
                        {
                            "in": "body",
                            "name": "body",
                            "schema": {
                                "properties": {}
                            }
                        }
                    ],
                    "permission": "guillotina.RawSearchContent",
                    "responses": {
                        "200": {
                            "description": "Search results",
                            "schema": {
                                "$ref": "#/definitions/SearchResults"
                            },
                            "type": "object"
                        }
                    },
                    "summary": "Make a complex search query"
                }
```

```
            },
            "@sharing": {
                "GET": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "GET",
                    "module": "guillotina.api.content.sharing_get",
                    "name": "@sharing",
                    "permission": "guillotina.SeePermissions",
                    "responses": {
                        "200": {
                            "description": "All the sharing defined on this␣
↪resource",
                            "schema": {
                                "$ref": "#/definitions/ResourceACL"
                            }
                        }
                    },
                    "summary": "Get sharing settings for this resource"
                },
                "POST": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "POST",
                    "name": "@sharing",
                    "parameters": [
                        {
                            "in": "body",
                            "name": "body",
                            "schema": {
                                "$ref": "#/definitions/Permissions"
                            },
                            "type": "object"
                        }
                    ],
                    "permission": "guillotina.ChangePermissions",
                    "responses": {
                        "200": {
                            "description": "Successuflly changed permission"
                        }
                    },
                    "summary": "Change permissions for a resource"
                },
                "PUT": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "PUT",
                    "name": "@sharing",
                    "parameters": [
```

```
                        {
                            "in": "body",
                            "name": "body",
                            "schema": {
                                "$ref": "#/definitions/Permissions"
                            },
                            "type": "object"
                        }
                    ],
                    "permission": "guillotina.ChangePermissions",
                    "responses": {
                        "200": {
                            "description": "Successuflly replaced permissions
↪"
                        }
                    },
                    "summary": "Replace permissions for a resource"
                }
            },
            "@tusupload": {
                "HEAD": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "HEAD",
                    "name": "@tusupload",
                    "permission": "guillotina.ModifyContent",
                    "traversed_service_definitions": {
                        "{field_name}": {
                            "parameters": [
                                {
                                    "description": "Name of file field",
                                    "in": "path",
                                    "name": "field_name",
                                    "required": true
                                }
                            ],
                            "responses": {
                                "200": {
                                    "description": "Successfully patched data
↪",
                                    "headers": {
                                        "Access-Control-Expose-Headers": {
                                            "type": "string"
                                        },
                                        "Tus-Resumable": {
                                            "type": "string"
                                        },
                                        "Upload-Offset": {
                                            "type": "integer"
                                        }
                                    }
                                }
                            },
                            "summary": "TUS endpoint"
                        }
```

```
                    }
                },
                "OPTIONS": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "OPTIONS",
                    "name": "@tusupload",
                    "permission": "guillotina.AccessPreflight",
                    "traversed_service_definitions": {
                        "{field_name}": {
                            "parameters": [
                                {
                                    "description": "Name of file field",
                                    "in": "path",
                                    "name": "field_name",
                                    "required": true
                                }
                            ],
                            "responses": {
                                "200": {
                                    "description": "Successfully returned␣
↪tus info",
                                    "headers": {
                                        "Tus-Extension": {
                                            "type": "string"
                                        },
                                        "Tus-Max-Size": {
                                            "type": "integer"
                                        },
                                        "Tus-Resumable": {
                                            "type": "string"
                                        },
                                        "Tus-Version": {
                                            "type": "string"
                                        }
                                    }
                                }
                            },
                            "summary": "TUS endpoint"
                        }
                    }
                },
                "PATCH": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "PATCH",
                    "name": "@tusupload",
                    "permission": "guillotina.ModifyContent",
                    "traversed_service_definitions": {
                        "{field_name}": {
                            "parameters": [
                                {
```

```
                                    "description": "Name of file field",
                                    "in": "path",
                                    "name": "field_name",
                                    "required": true
                                },
                                {
                                    "in": "headers",
                                    "name": "Upload-Offset",
                                    "required": true,
                                    "type": "integer"
                                },
                                {
                                    "in": "headers",
                                    "name": "CONTENT-LENGTH",
                                    "required": true,
                                    "type": "integer"
                                }
                            ],
                            "responses": {
                                "204": {
                                    "description": "Successfully patched data
↪",
                                    "headers": {
                                        "Upload-Offset": {
                                            "type": "integer"
                                        }
                                    }
                                }
                            },
                            "summary": "TUS endpoint"
                        }
                    }
                },
                "POST": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
                    ],
                    "method": "POST",
                    "name": "@tusupload",
                    "permission": "guillotina.ModifyContent",
                    "traversed_service_definitions": {
                        "{field_name}": {
                            "parameters": [
                                {
                                    "description": "Name of file field",
                                    "in": "path",
                                    "name": "field_name",
                                    "required": true
                                },
                                {
                                    "in": "headers",
                                    "name": "Upload-Offset",
                                    "required": true,
                                    "type": "integer"
                                },
                                {
```

```
                                    "in": "headers",
                                    "name": "UPLOAD-LENGTH",
                                    "required": true,
                                    "type": "integer"
                                },
                                {
                                    "in": "headers",
                                    "name": "UPLOAD-MD5",
                                    "required": false,
                                    "type": "string"
                                },
                                {
                                    "in": "headers",
                                    "name": "UPLOAD-EXTENSION",
                                    "required": false,
                                    "type": "string"
                                },
                                {
                                    "in": "headers",
                                    "name": "TUS-RESUMABLE",
                                    "required": true,
                                    "type": "string"
                                },
                                {
                                    "in": "headers",
                                    "name": "UPLOAD-METADATA",
                                    "required": false,
                                    "type": "string"
                                }
                            ],
                            "responses": {
                                "204": {
                                    "description": "Successfully patched data
↪",
                                    "headers": {
                                        "Access-Control-Expose-Headers": {
                                            "type": "string"
                                        },
                                        "Location": {
                                            "type": "string"
                                        },
                                        "Tus-Resumable": {
                                            "type": "string"
                                        }
                                    }
                                }
                            },
                            "summary": "TUS endpoint"
                        }
                    }
                }
            },
            "@upload": {
                "PATCH": {
                    "context": [
                        "guillotina.interfaces.content.IResource",
                        "guillotina.interfaces.content.ILocation",
                        "zope.interface.Interface"
```

```
                    ],
                    "method": "PATCH",
                    "name": "@upload",
                    "permission": "guillotina.ModifyContent",
                    "traversed_service_definitions": {
                        "{field_name}": {
                            "parameters": [
                                {
                                    "description": "Name of file field",
                                    "in": "path",
                                    "name": "field_name",
                                    "required": true
                                }
                            ],
                            "responses": {
                                "200": {
                                    "description": "Successfully updated
↪content"
                                }
                            },
                            "summary": "Update the content of a file"
                        }
                    }
                }
            }
        },
    "guillotina.interfaces.content.IStaticDirectory": {
        "GET": {
            "context": [
                "guillotina.interfaces.content.IStaticDirectory",
                "guillotina.interfaces.content.ITraversable",
                "zope.interface.Interface"
            ],
            "method": "GET",
            "permission": "guillotina.AccessContent"
        }
    },
    "guillotina.interfaces.content.IStaticFile": {
        "GET": {
            "context": [
                "guillotina.interfaces.content.IStaticFile",
                "zope.interface.Interface"
            ],
            "method": "GET",
            "permission": "guillotina.AccessContent"
        }
    }
}
```


1.3.2 Database

GET Get list of containers **permission: guillotina.GetContainers**http

```
GET /db HTTP/1.1
Accept: application/json
```

```
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db -H 'Accept: application/json' --user␣
↪root:root
```

httpie

```
http http://localhost:8080/db Accept:application/json -a root:root
```

python-requests

```
requests.get('http://localhost:8080/db', headers={'Accept': 'application/json
↪'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{
    "@type": "Database",
    "containers": [
        "container"
    ]
}
```

POST Create a new Container **permission: guillotina.AddContainer**http

```
POST /db HTTP/1.1
Accept: application/json
Host: localhost:8080
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
    "@type": "Container",
    "id": "container",
    "title": "Container"
}
```

curl

```
curl -i -X POST http://localhost:8080/db -H 'Accept: application/json' -H
↪'Content-Type: application/json' --data-raw '{"@type": "Container", "id":
↪"container", "title": "Container"}' --user root:root
```

httpie

```
echo '{
  "@type": "Container",
  "id": "container",
  "title": "Container"
}' | http POST http://localhost:8080/db Accept:application/json Content-
↪Type:application/json -a root:root
```

python-requests

```
requests.post('http://localhost:8080/db', headers={'Accept': 'application/
↪json', 'Content-Type': 'application/json'}, json={'@type': 'Container', 'id
↪': 'container', 'title': 'Container'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{
    "error": {
        "message": "Duplicate id",
        "type": "NotAllowed"
    }
}
```

1.3.3 Container

GET Retrieves serialization of resource **permission: guillotina.ViewContent**http

```
GET /db/container HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db/container -H 'Accept: application/json' --
↪user root:root
```

httpie

```
http http://localhost:8080/db/container Accept:application/json -a root:root
```

python-requests

```
requests.get('http://localhost:8080/db/container', headers={'Accept':
↪'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{
    "@id": "http://localhost:8080/db/container",
    "@type": "Container",
    "UID": "5121b7b53e4847c8bb8b281cf873a9e0",
    "__behaviors__": [],
    "__name__": "container",
    "creation_date": "2017-08-03T16:53:18.314861-05:00",
    "items": [],
    "length": 0,
```

```
    "modification_date": "2017-08-03T16:53:18.314861-05:00",
    "parent": {},
    "title": "Container",
    "type_name": "Container"
}
```

POST Add new resouce inside this container resource **permission: guillotina.AddContent**http

```
POST /db/container HTTP/1.1
Accept: application/json
Host: localhost:8080
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
    "@type": "Folder",
    "id": "folder",
    "title": "My Folder"
}
```

curl

```
curl -i -X POST http://localhost:8080/db/container -H 'Accept: application/
↪json' -H 'Content-Type: application/json' --data-raw '{"@type": "Folder",
↪"id": "folder", "title": "My Folder"}' --user root:root
```

httpie

```
echo '{
  "@type": "Folder",
  "id": "folder",
  "title": "My Folder"
}' | http POST http://localhost:8080/db/container Accept:application/json␣
↪Content-Type:application/json -a root:root
```

python-requests

```
requests.post('http://localhost:8080/db/container', headers={'Accept':
↪'application/json', 'Content-Type': 'application/json'}, json={'@type':
↪'Folder', 'id': 'folder', 'title': 'My Folder'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: http://localhost:8080/db/container/folder
Server: Python/3.6 aiohttp/2.2.5

{
    "@id": "http://localhost:8080/db/container/folder",
    "@type": "Folder",
    "UID": "691c83337ab74a85a162fbec877d614c",
    "__behaviors__": [],
    "__name__": "folder",
    "creation_date": "2017-08-03T16:54:08.479606-05:00",
    "guillotina.behaviors.dublincore.IDublinCore": {
        "contributors": [
```

```
            "root"
        ],
        "creation_date": "2017-08-03T16:54:08.479606-05:00",
        "creators": [
            "root"
        ],
        "description": null,
        "effective_date": null,
        "expiration_date": null,
        "modification_date": "2017-08-03T16:54:08.479606-05:00",
        "publisher": null,
        "tags": null,
        "title": null
    },
    "items": [],
    "length": 0,
    "modification_date": "2017-08-03T16:54:08.479606-05:00",
    "parent": {
        "@id": "http://localhost:8080/db/container",
        "@type": "Container"
    },
    "title": "My Folder",
    "type_name": "Folder"
}
```


DELETE

permission: guillotina.DeleteContainershttp

```
DELETE /db/container HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i -X DELETE http://localhost:8080/db/container -H 'Accept: application/
↪json' --user root:root
```

httpie

```
http DELETE http://localhost:8080/db/container Accept:application/json -a␣
↪root:root
```

python-requests

```
requests.delete('http://localhost:8080/db/container', headers={'Accept':
↪'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/1.3.3

null
```

GET @addons

permission: guillotina.ManageAddonshttp

```
GET /db/container/@addons HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db/container/@addons -H 'Accept: application/
↪json' --user root:root
```

httpie

```
http http://localhost:8080/db/container/@addons Accept:application/json -a␣
↪root:root
```

python-requests

```
requests.get('http://localhost:8080/db/container/@addons', headers={'Accept
↪': 'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/1.3.3

{
    "available": [
        {
            "id": "govsocial",
            "title": "govsocial Addon"
        }
    ],
    "installed": []
}
```

POST @addons Install addon to container **permission: guillotina.ManageAddons**http

```
POST /db/container/@addons HTTP/1.1
Accept: application/json
Host: localhost:8080
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
    "id": "myaddon"
}
```

curl

```
curl -i -X POST http://localhost:8080/db/container/@addons -H 'Accept:␣
↪application/json' -H 'Content-Type: application/json' --data-raw '{"id":
↪"myaddon"}' --user root:root
```

httpie

```
echo '{
  "id": "myaddon"
}' | http POST http://localhost:8080/db/container/@addons Accept:application/
↪json Content-Type:application/json -a root:root
```

python-requests

```
requests.post('http://localhost:8080/db/container/@addons', headers={'Accept
↪': 'application/json', 'Content-Type': 'application/json'}, json={'id':
↪'myaddon'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{
    "error": {
        "message": "Property 'id' is required to be valid",
        "type": "RequiredParam"
    }
}
```

DELETE @addons Uninstall an addon from container **permission: guillotina.ManageAddons**http

```
DELETE /db/container/@addons HTTP/1.1
Accept: application/json
Host: localhost:8080
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
    "id": "myaddon"
}
```

curl

```
curl -i -X DELETE http://localhost:8080/db/container/@addons -H 'Accept:␣
↪application/json' -H 'Content-Type: application/json' --data-raw '{"id":
↪"myaddon"}' --user root:root
```

httpie

```
echo '{
  "id": "myaddon"
}' | http DELETE http://localhost:8080/db/container/@addons␣
↪Accept:application/json Content-Type:application/json -a root:root
```

python-requests

```
requests.delete('http://localhost:8080/db/container/@addons', headers={
↪'Accept': 'application/json', 'Content-Type': 'application/json'}, json={
↪'id': 'myaddon'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{
    "error": {
        "message": "Property 'id' is required to be valid",
        "type": "RequiredParam"
    }
}
```

GET @registry Read container registry settings **permission: guillotina.ReadConfiguration**http

```
GET /db/container/@registry/guillotina.documentation.testmodule.ISchema.foo␣
↪HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db/container/@registry/guillotina.
↪documentation.testmodule.ISchema.foo -H 'Accept: application/json' --user␣
↪root:root
```

httpie

```
http http://localhost:8080/db/container/@registry/guillotina.documentation.
↪testmodule.ISchema.foo Accept:application/json -a root:root
```

python-requests

```
requests.get('http://localhost:8080/db/container/@registry/guillotina.
↪documentation.testmodule.ISchema.foo', headers={'Accept': 'application/json
↪'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{
    "value": "New foobar value"
}
```

POST @registry Register a new interface to for registry settings **permission: guillotina.RegisterConfigurations**http

```
POST /db/container/@registry HTTP/1.1
Accept: application/json
Host: localhost:8080
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
    "initial_values": {
```

```
        "foo": "bar"
    },
    "interface": "guillotina.documentation.testmodule.ISchema"
}
```

curl

```
curl -i -X POST http://localhost:8080/db/container/@registry -H 'Accept:␣
↪application/json' -H 'Content-Type: application/json' --data-raw '{
↪"initial_values": {"foo": "bar"}, "interface": "guillotina.documentation.
↪testmodule.ISchema"}' --user root:root
```

httpie

```
echo '{
  "initial_values": {
    "foo": "bar"
  },
  "interface": "guillotina.documentation.testmodule.ISchema"
}' | http POST http://localhost:8080/db/container/@registry␣
↪Accept:application/json Content-Type:application/json -a root:root
```

python-requests

```
requests.post('http://localhost:8080/db/container/@registry', headers={
↪'Accept': 'application/json', 'Content-Type': 'application/json'}, json={
↪'initial_values': {'foo': 'bar'}, 'interface': 'guillotina.documentation.
↪testmodule.ISchema'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 201 Created
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{}
```

PATCH @registry Update registry setting **permission: guillotina.WriteConfiguration**http

```
PATCH /db/container/@registry/guillotina.documentation.testmodule.ISchema.
↪foo HTTP/1.1
Accept: application/json
Host: localhost:8080
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
    "value": "New foobar value"
}
```

curl

```
curl -i -X PATCH http://localhost:8080/db/container/@registry/guillotina.
↪documentation.testmodule.ISchema.foo -H 'Accept: application/json' -H
↪'Content-Type: application/json' --data-raw '{"value": "New foobar value"}
↪' --user root:root
```

httpie

```
echo '{
  "value": "New foobar value"
}' | http PATCH http://localhost:8080/db/container/@registry/guillotina.
↪documentation.testmodule.ISchema.foo Accept:application/json Content-
↪Type:application/json -a root:root
```

python-requests

```
requests.patch('http://localhost:8080/db/container/@registry/guillotina.
↪documentation.testmodule.ISchema.foo', headers={'Accept': 'application/json
↪', 'Content-Type': 'application/json'}, json={'value': 'New foobar value'},
↪ auth=('root', 'root'))
```

response

```
HTTP/1.1 204 No Content
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5
```

GET @registry/[dotted-name:string]

permission: guillotina.ReadConfigurationhttp

```
GET /db/container/@registry/guillotina.documentation.testmodule.ISchema.foo␣
↪HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db/container/@registry/guillotina.
↪documentation.testmodule.ISchema.foo -H 'Accept: application/json' --user␣
↪root:root
```

httpie

```
http http://localhost:8080/db/container/@registry/guillotina.documentation.
↪testmodule.ISchema.foo Accept:application/json -a root:root
```

python-requests

```
requests.get('http://localhost:8080/db/container/@registry/guillotina.
↪documentation.testmodule.ISchema.foo', headers={'Accept': 'application/json
↪'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/1.3.3

{
    "value": "New foobar value"
}
```

PATCH @registry/[dotted-name:string]

permission: guillotina.WriteConfigurationhttp

```
PATCH /db/container/@registry/guillotina.documentation.testmodule.ISchema.
↪foo HTTP/1.1
Accept: application/json
Host: localhost:8080
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
    "value": "New foobar value"
}
```

curl

```
curl -i -X PATCH http://localhost:8080/db/container/@registry/guillotina.
↪documentation.testmodule.ISchema.foo -H 'Accept: application/json' -H
↪'Content-Type: application/json' --data-raw '{"value": "New foobar value"}
↪' --user root:root
```

httpie

```
echo '{
  "value": "New foobar value"
}' | http PATCH http://localhost:8080/db/container/@registry/guillotina.
↪documentation.testmodule.ISchema.foo Accept:application/json Content-
↪Type:application/json -a root:root
```

python-requests

```
requests.patch('http://localhost:8080/db/container/@registry/guillotina.
↪documentation.testmodule.ISchema.foo', headers={'Accept': 'application/json
↪', 'Content-Type': 'application/json'}, json={'value': 'New foobar value'},
↪ auth=('root', 'root'))
```

response

```
HTTP/1.1 204 No Content
Content-Type: application/json
Server: Python/3.6 aiohttp/1.3.3
```

GET @types Read information on available types **permission: guillotina.AccessContent**http

```
GET /db/container/@types HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db/container/@types -H 'Accept: application/
↪json' --user root:root
```

httpie

```
http http://localhost:8080/db/container/@types Accept:application/json -a␣
↪root:root
```

python-requests

```
requests.get('http://localhost:8080/db/container/@types', headers={'Accept':
↪'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

[
    {
        "$schema": "http://json-schema.org/draft-04/schema#",
        "definitions": {
            "IDublinCore": {
                "invariants": [],
                "properties": {
                    "contributors": {
                        "description": "The unqualified Dublin Core
↪'Contributor' element values",
                        "items": {
                            "type": "string"
                        },
                        "title": "Contributors",
                        "type": "array"
                    },
                    "creation_date": {
                        "description": "The date and time that an object is␣
↪created. \nThis is normally set automatically.",
                        "title": "Creation Date",
                        "type": "datetime"
                    },
                    "creators": {
                        "description": "The unqualified Dublin Core 'Creator
↪' element values",
                        "items": {
                            "type": "string"
                        },
                        "title": "Creators",
                        "type": "array"
                    },
                    "description": {
                        "description": "The first unqualified Dublin Core
↪'Description' element value.",
                        "title": "Description",
                        "type": "string"
                    },
                    "effective_date": {
                        "description": "The date and time that an object␣
↪should be published. ",
                        "title": "Effective Date",
                        "type": "datetime"
                    },
                    "expiration_date": {
```

```
                        "description": "The date and time that the object␣
↪should become unpublished.",
                        "title": "Expiration Date",
                        "type": "datetime"
                    },
                    "modification_date": {
                        "description": "The date and time that the object␣
↪was last modified in a\nmeaningful way.",
                        "title": "Modification Date",
                        "type": "datetime"
                    },
                    "publisher": {
                        "description": "The first unqualified Dublin Core
↪'Publisher' element value.",
                        "title": "Publisher",
                        "type": "string"
                    },
                    "tags": {
                        "description": "The unqualified Dublin Core 'Tags'␣
↪element values",
                        "items": {
                            "type": "string"
                        },
                        "title": "Tags",
                        "type": "array"
                    },
                    "title": {
                        "description": "The first unqualified Dublin Core
↪'Title' element value.",
                        "title": "Title",
                        "type": "string"
                    }
                },
                "required": [
                    "title",
                    "description",
                    "creation_date",
                    "modification_date",
                    "effective_date",
                    "expiration_date",
                    "creators",
                    "tags",
                    "publisher",
                    "contributors"
                ],
                "type": "object"
            }
        },
        "invariants": [],
        "properties": {
            "IDublinCore": [
                {
                    "$ref": "#/definitions/IDublinCore"
                }
            ],
            "__behaviors__": {
                "description": "Dynamic behaviors for the content type",
                "title": "Enabled behaviors",
```

```
                "type": "array"
            },
            "__name__": {
                "description": "The object can be looked up from the parent
↪'s sublocations using this name.",
                "title": "The name within the parent",
                "type": "string"
            },
            "title": {
                "description": "Title of the Resource",
                "title": "Title",
                "type": "string"
            },
            "type_name": {
                "type": "string"
            }
        },
        "required": [
            "type_name"
        ],
        "title": "Item",
        "type": "object"
    },
    {
        "$schema": "http://json-schema.org/draft-04/schema#",
        "definitions": {
            "IDublinCore": {
                "invariants": [],
                "properties": {
                    "contributors": {
                        "description": "The unqualified Dublin Core
↪'Contributor' element values",
                        "items": {
                            "type": "string"
                        },
                        "title": "Contributors",
                        "type": "array"
                    },
                    "creation_date": {
                        "description": "The date and time that an object is␣
↪created. \nThis is normally set automatically.",
                        "title": "Creation Date",
                        "type": "datetime"
                    },
                    "creators": {
                        "description": "The unqualified Dublin Core 'Creator
↪' element values",
                        "items": {
                            "type": "string"
                        },
                        "title": "Creators",
                        "type": "array"
                    },
                    "description": {
                        "description": "The first unqualified Dublin Core
↪'Description' element value.",
                        "title": "Description",
                        "type": "string"
```

```
                    },
                    "effective_date": {
                        "description": "The date and time that an object␣
↪should be published. ",
                        "title": "Effective Date",
                        "type": "datetime"
                    },
                    "expiration_date": {
                        "description": "The date and time that the object␣
↪should become unpublished.",
                        "title": "Expiration Date",
                        "type": "datetime"
                    },
                    "modification_date": {
                        "description": "The date and time that the object␣
↪was last modified in a\nmeaningful way.",
                        "title": "Modification Date",
                        "type": "datetime"
                    },
                    "publisher": {
                        "description": "The first unqualified Dublin Core
↪'Publisher' element value.",
                        "title": "Publisher",
                        "type": "string"
                    },
                    "tags": {
                        "description": "The unqualified Dublin Core 'Tags'␣
↪element values",
                        "items": {
                            "type": "string"
                        },
                        "title": "Tags",
                        "type": "array"
                    },
                    "title": {
                        "description": "The first unqualified Dublin Core
↪'Title' element value.",
                        "title": "Title",
                        "type": "string"
                    }
                },
                "required": [
                    "title",
                    "description",
                    "creation_date",
                    "modification_date",
                    "effective_date",
                    "expiration_date",
                    "creators",
                    "tags",
                    "publisher",
                    "contributors"
                ],
                "type": "object"
            }
        },
        "invariants": [],
        "properties": {
```

```
            "IDublinCore": [
                {
                    "$ref": "#/definitions/IDublinCore"
                }
            ],
            "__behaviors__": {
                "description": "Dynamic behaviors for the content type",
                "title": "Enabled behaviors",
                "type": "array"
            },
            "__name__": {
                "description": "The object can be looked up from the parent
↪'s sublocations using this name.",
                "title": "The name within the parent",
                "type": "string"
            },
            "title": {
                "description": "Title of the Resource",
                "title": "Title",
                "type": "string"
            },
            "type_name": {
                "type": "string"
            }
        },
        "required": [
            "type_name"
        ],
        "title": "Folder",
        "type": "object"
    },
    {
        "$schema": "http://json-schema.org/draft-04/schema#",
        "definitions": {},
        "invariants": [],
        "properties": {
            "__behaviors__": {
                "description": "Dynamic behaviors for the content type",
                "title": "Enabled behaviors",
                "type": "array"
            },
            "__name__": {
                "description": "The object can be looked up from the parent
↪'s sublocations using this name.",
                "title": "The name within the parent",
                "type": "string"
            },
            "title": {
                "description": "Title of the Resource",
                "title": "Title",
                "type": "string"
            },
            "type_name": {
                "type": "string"
            }
        },
        "required": [
            "type_name"
```

```
        ],
        "title": "Container",
        "type": "object"
    }
]
```

1.3.4 Folder

GET Retrieves serialization of resource **permission: guillotina.ViewContent**http

```
GET /db/container/folder HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db/container/folder -H 'Accept: application/
↪json' --user root:root
```

httpie

```
http http://localhost:8080/db/container/folder Accept:application/json -a␣
↪root:root
```

python-requests

```
requests.get('http://localhost:8080/db/container/folder', headers={'Accept':
↪'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{
    "@id": "http://localhost:8080/db/container/folder",
    "@type": "Folder",
    "UID": "691c83337ab74a85a162fbec877d614c",
    "__behaviors__": [],
    "__name__": "folder",
    "creation_date": "2017-08-03T16:54:08.479606-05:00",
    "guillotina.behaviors.dublincore.IDublinCore": {
        "contributors": [
            "root"
        ],
        "creation_date": "2017-08-03T16:54:08.479606-05:00",
        "creators": [
            "root"
        ],
        "description": null,
        "effective_date": null,
        "expiration_date": null,
        "modification_date": "2017-08-03T16:54:08.479606-05:00",
        "publisher": null,
```

```
        "tags": null,
        "title": null
    },
    "items": [],
    "length": 0,
    "modification_date": "2017-08-03T16:54:08.479606-05:00",
    "parent": {
        "@id": "http://localhost:8080/db/container",
        "@type": "Container"
    },
    "title": "My Folder",
    "type_name": "Folder"
}
```

POST Add new resouce inside this container resource **permission: guillotina.AddContent**http

```
POST /db/container/folder HTTP/1.1
Accept: application/json
Host: localhost:8080
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
    "@type": "Item",
    "id": "item",
    "title": "My Item"
}
```

curl

```
curl -i -X POST http://localhost:8080/db/container/folder -H 'Accept:␣
↪application/json' -H 'Content-Type: application/json' --data-raw '{"@type
↪": "Item", "id": "item", "title": "My Item"}' --user root:root
```

httpie

```
echo '{
  "@type": "Item",
  "id": "item",
  "title": "My Item"
}' | http POST http://localhost:8080/db/container/folder Accept:application/
↪json Content-Type:application/json -a root:root
```

python-requests

```
requests.post('http://localhost:8080/db/container/folder', headers={'Accept
↪': 'application/json', 'Content-Type': 'application/json'}, json={'@type':
↪'Item', 'id': 'item', 'title': 'My Item'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: http://localhost:8080/db/container/folder/item
Server: Python/3.6 aiohttp/2.2.5

{
```

```
    "@id": "http://localhost:8080/db/container/folder/item",
    "@type": "Item",
    "UID": "4431ba1d1b2f4442ae6ac40c06c2be4c",
    "__behaviors__": [],
    "__name__": "item",
    "creation_date": "2017-08-03T16:54:08.818671-05:00",
    "guillotina.behaviors.dublincore.IDublinCore": {
        "contributors": [
            "root"
        ],
        "creation_date": "2017-08-03T16:54:08.818671-05:00",
        "creators": [
            "root"
        ],
        "description": null,
        "effective_date": null,
        "expiration_date": null,
        "modification_date": "2017-08-03T16:54:08.818671-05:00",
        "publisher": null,
        "tags": null,
        "title": null
    },
    "modification_date": "2017-08-03T16:54:08.818671-05:00",
    "parent": {
        "@id": "http://localhost:8080/db/container/folder",
        "@type": "Folder"
    },
    "title": "My Item",
    "type_name": "Item"
}
```


PATCH Modify the content of this resource **permission: guillotina.ModifyContent**http

```
PATCH /db/container/folder HTTP/1.1
Accept: application/json
Host: localhost:8080
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
    "title": "My Folder Updated"
}
```

curl

```
curl -i -X PATCH http://localhost:8080/db/container/folder -H 'Accept:␣
↪application/json' -H 'Content-Type: application/json' --data-raw '{"title
↪": "My Folder Updated"}' --user root:root
```

httpie

```
echo '{
  "title": "My Folder Updated"
}' | http PATCH http://localhost:8080/db/container/folder Accept:application/
↪json Content-Type:application/json -a root:root
```

python-requests

```
requests.patch('http://localhost:8080/db/container/folder', headers={'Accept
↪': 'application/json', 'Content-Type': 'application/json'}, json={'title':
↪'My Folder Updated'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 204 No Content
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5
```

DELETE Delete container **permission: guillotina.DeleteContainers**http

```
DELETE /db/container HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i -X DELETE http://localhost:8080/db/container -H 'Accept: application/
↪json' --user root:root
```

httpie

```
http DELETE http://localhost:8080/db/container Accept:application/json -a␣
↪root:root
```

python-requests

```
requests.delete('http://localhost:8080/db/container', headers={'Accept':
↪'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

null
```

GET @all_permissions See all permission settings for this resource **permission: guillotina.SeePermissions**http

```
GET /db/container/folder/@all_permissions HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db/container/folder/@all_permissions -H
↪'Accept: application/json' --user root:root
```

httpie

```
http http://localhost:8080/db/container/folder/@all_permissions␣
↪Accept:application/json -a root:root
```

python-requests

```
requests.get('http://localhost:8080/db/container/folder/@all_permissions',␣
↪headers={'Accept': 'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

[
    {
        "folder": {
            "prinperm": [],
            "prinrole": [
                {
                    "principal": "root",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                }
            ],
            "roleperm": []
        }
    },
    {
        "container": {
            "prinperm": [],
            "prinrole": [
                {
                    "principal": "root",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "principal": "root",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                }
            ],
            "roleperm": []
        }
    },
    {
        "(no name)": {
            "prinperm": [
                {
                    "permission": "guillotina.AccessContent",
                    "principal": "root",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.AddContainer",
                    "principal": "root",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.DeleteContainers",
                    "principal": "root",
```

```
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.GetAPIDefinition",
                    "principal": "root",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.GetContainers",
                    "principal": "root",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.GetDatabases",
                    "principal": "root",
                    "setting": "Allow"
                }
            ]
        }
    },
    {
        "system": {
            "prinperm": [],
            "prinrole": [],
            "roleperm": [
                {
                    "permission": "guillotina.AccessPreflight",
                    "role": "guillotina.Anonymous",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.AccessPreflight",
                    "role": "guillotina.Authenticated",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.Public",
                    "role": "guillotina.Anonymous",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.Public",
                    "role": "guillotina.Authenticated",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ViewContent",
                    "role": "guillotina.Reader",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ViewContent",
                    "role": "guillotina.Reviewer",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ViewContent",
                    "role": "guillotina.Owner",
```

```
            "setting": "Allow"
        },
        {
            "permission": "guillotina.ViewContent",
            "role": "guillotina.Editor",
            "setting": "Allow"
        },
        {
            "permission": "guillotina.AccessContent",
            "role": "guillotina.Reader",
            "setting": "Allow"
        },
        {
            "permission": "guillotina.AccessContent",
            "role": "guillotina.Reviewer",
            "setting": "Allow"
        },
        {
            "permission": "guillotina.AccessContent",
            "role": "guillotina.Owner",
            "setting": "Allow"
        },
        {
            "permission": "guillotina.AccessContent",
            "role": "guillotina.Editor",
            "setting": "Allow"
        },
        {
            "permission": "guillotina.AccessContent",
            "role": "guillotina.ContainerAdmin",
            "setting": "Allow"
        },
        {
            "permission": "guillotina.DeleteContent",
            "role": "guillotina.Owner",
            "setting": "Allow"
        },
        {
            "permission": "guillotina.AddContent",
            "role": "guillotina.Owner",
            "setting": "Allow"
        },
        {
            "permission": "guillotina.ModifyContent",
            "role": "guillotina.Owner",
            "setting": "Allow"
        },
        {
            "permission": "guillotina.ModifyContent",
            "role": "guillotina.Editor",
            "setting": "Allow"
        },
        {
            "permission": "guillotina.ChangePermissions",
            "role": "guillotina.Owner",
            "setting": "Allow"
        },
        {
```

```
                    "permission": "guillotina.SeePermissions",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ReindexContent",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ReindexContent",
                    "role": "guillotina.Editor",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ManageAddons",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ReadConfiguration",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.WriteConfiguration",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.RegisterConfigurations",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ManageCatalog",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.RawSearchContent",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.DeleteContainers",
                    "role": "guillotina.ContainerDeleter",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.SearchContent",
                    "role": "guillotina.Member",
                    "setting": "Allow"
                }
            ]
        }
    }
]
```

GET @behaviors Get information on behaviors for this resource **permission: guillotina.AccessContent**http

```
GET /db/container/folder/@behaviors HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db/container/folder/@behaviors -H 'Accept:␣
↪application/json' --user root:root
```

httpie

```
http http://localhost:8080/db/container/folder/@behaviors Accept:application/
↪json -a root:root
```

python-requests

```
requests.get('http://localhost:8080/db/container/folder/@behaviors', headers=
↪{'Accept': 'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{
    "available": [],
    "dynamic": [],
    "guillotina.behaviors.dublincore.IDublinCore": {
        "invariants": [],
        "properties": {
            "contributors": {
                "description": "The unqualified Dublin Core 'Contributor'␣
↪element values",
                "items": {
                    "type": "string"
                },
                "title": "Contributors",
                "type": "array"
            },
            "creation_date": {
                "description": "The date and time that an object is created.␣
↪\nThis is normally set automatically.",
                "title": "Creation Date",
                "type": "datetime"
            },
            "creators": {
                "description": "The unqualified Dublin Core 'Creator'␣
↪element values",
                "items": {
                    "type": "string"
                },
                "title": "Creators",
                "type": "array"
```

```
            },
            "description": {
                "description": "The first unqualified Dublin Core
↪'Description' element value.",
                "title": "Description",
                "type": "string"
            },
            "effective_date": {
                "description": "The date and time that an object should be␣
↪published. ",
                "title": "Effective Date",
                "type": "datetime"
            },
            "expiration_date": {
                "description": "The date and time that the object should␣
↪become unpublished.",
                "title": "Expiration Date",
                "type": "datetime"
            },
            "modification_date": {
                "description": "The date and time that the object was last␣
↪modified in a\nmeaningful way.",
                "title": "Modification Date",
                "type": "datetime"
            },
            "publisher": {
                "description": "The first unqualified Dublin Core 'Publisher
↪' element value.",
                "title": "Publisher",
                "type": "string"
            },
            "tags": {
                "description": "The unqualified Dublin Core 'Tags' element␣
↪values",
                "items": {
                    "type": "string"
                },
                "title": "Tags",
                "type": "array"
            },
            "title": {
                "description": "The first unqualified Dublin Core 'Title'␣
↪element value.",
                "title": "Title",
                "type": "string"
            }
        },
        "required": [
            "title",
            "description",
            "creation_date",
            "modification_date",
            "effective_date",
            "expiration_date",
            "creators",
            "tags",
            "publisher",
            "contributors"
```

```
        ],
        "type": "object"
    },
    "static": [
        "guillotina.behaviors.dublincore.IDublinCore"
    ]
}
```

GET @sharing Get sharing settings for this resource **permission: guillotina.SeePermissions**http

```
GET /db/container/folder/@sharing HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db/container/folder/@sharing -H 'Accept:␣
↪application/json' --user root:root
```

httpie

```
http http://localhost:8080/db/container/folder/@sharing Accept:application/
↪json -a root:root
```

python-requests

```
requests.get('http://localhost:8080/db/container/folder/@sharing', headers={
↪'Accept': 'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{
    "inherit": [
        {
            "@id": "http://localhost:8080/db/container",
            "prinperm": {},
            "prinrole": {
                "root": {
                    "guillotina.ContainerAdmin": "Allow",
                    "guillotina.Owner": "Allow"
                }
            },
            "roleperm": {}
        }
    ],
    "local": {
        "prinperm": {},
        "prinrole": {
            "root": {
                "guillotina.Owner": "Allow"
            }
        },
```

```
        "roleperm": {}
    }
}
```

1.3.5 Item

GET Retrieves serialization of resource **permission: guillotina.ViewContent**http

```
GET /db/container/folder/item HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db/container/folder/item -H 'Accept:␣
↪application/json' --user root:root
```

httpie

```
http http://localhost:8080/db/container/folder/item Accept:application/json -
↪a root:root
```

python-requests

```
requests.get('http://localhost:8080/db/container/folder/item', headers={
↪'Accept': 'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{
    "@id": "http://localhost:8080/db/container/folder/item",
    "@type": "Item",
    "UID": "4431ba1d1b2f4442ae6ac40c06c2be4c",
    "__behaviors__": [],
    "__name__": "item",
    "creation_date": "2017-08-03T16:54:08.818671-05:00",
    "guillotina.behaviors.dublincore.IDublinCore": {
        "contributors": [
            "root"
        ],
        "creation_date": "2017-08-03T16:54:08.818671-05:00",
        "creators": [
            "root"
        ],
        "description": null,
        "effective_date": null,
        "expiration_date": null,
        "modification_date": "2017-08-03T16:54:08.818671-05:00",
        "publisher": null,
        "tags": null,
        "title": null
```

```
    },
    "modification_date": "2017-08-03T16:54:08.818671-05:00",
    "parent": {
        "@id": "http://localhost:8080/db/container/folder",
        "@type": "Folder"
    },
    "title": "My Item",
    "type_name": "Item"
}
```

PATCH Modify the content of this resource **permission: guillotina.ModifyContent**http

```
PATCH /db/container/folder/item HTTP/1.1
Accept: application/json
Host: localhost:8080
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
    "title": "My Item Updated"
}
```

curl

```
curl -i -X PATCH http://localhost:8080/db/container/folder/item -H 'Accept:␣
↪application/json' -H 'Content-Type: application/json' --data-raw '{"title
↪": "My Item Updated"}' --user root:root
```

httpie

```
echo '{
  "title": "My Item Updated"
}' | http PATCH http://localhost:8080/db/container/folder/item␣
↪Accept:application/json Content-Type:application/json -a root:root
```

python-requests

```
requests.patch('http://localhost:8080/db/container/folder/item', headers={
↪'Accept': 'application/json', 'Content-Type': 'application/json'}, json={
↪'title': 'My Item Updated'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 204 No Content
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5
```

DELETE Delete resource **permission: guillotina.DeleteContent**http

```
DELETE /db/container/folder/item HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i -X DELETE http://localhost:8080/db/container/folder/item -H 'Accept:␣
↪application/json' --user root:root
```

httpie

```
http DELETE http://localhost:8080/db/container/folder/item␣
↪Accept:application/json -a root:root
```

python-requests

```
requests.delete('http://localhost:8080/db/container/folder/item', headers={
↪'Accept': 'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

null
```

GET @all_permissions See all permission settings for this resource **permission: guillotina.SeePermissions**http

```
GET /db/container/folder/item/@all_permissions HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db/container/folder/item/@all_permissions -H
↪'Accept: application/json' --user root:root
```

httpie

```
http http://localhost:8080/db/container/folder/item/@all_permissions␣
↪Accept:application/json -a root:root
```

python-requests

```
requests.get('http://localhost:8080/db/container/folder/item/@all_permissions
↪', headers={'Accept': 'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

[
    {
        "item": {
            "prinperm": [],
            "prinrole": [
                {
                    "principal": "root",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                }
            ],
```

```
            "roleperm": []
        }
    },
    {
        "folder": {
            "prinperm": [],
            "prinrole": [
                {
                    "principal": "root",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                }
            ],
            "roleperm": []
        }
    },
    {
        "container": {
            "prinperm": [],
            "prinrole": [
                {
                    "principal": "root",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "principal": "root",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                }
            ],
            "roleperm": []
        }
    },
    {
        "(no name)": {
            "prinperm": [
                {
                    "permission": "guillotina.AccessContent",
                    "principal": "root",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.AddContainer",
                    "principal": "root",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.DeleteContainers",
                    "principal": "root",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.GetAPIDefinition",
                    "principal": "root",
                    "setting": "Allow"
                },
                {
```

```
                    "permission": "guillotina.GetContainers",
                    "principal": "root",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.GetDatabases",
                    "principal": "root",
                    "setting": "Allow"
                }
            ]
        }
    },
    {
        "system": {
            "prinperm": [],
            "prinrole": [],
            "roleperm": [
                {
                    "permission": "guillotina.AccessPreflight",
                    "role": "guillotina.Anonymous",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.AccessPreflight",
                    "role": "guillotina.Authenticated",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.Public",
                    "role": "guillotina.Anonymous",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.Public",
                    "role": "guillotina.Authenticated",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ViewContent",
                    "role": "guillotina.Reader",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ViewContent",
                    "role": "guillotina.Reviewer",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ViewContent",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ViewContent",
                    "role": "guillotina.Editor",
                    "setting": "Allow"
                },
                {
```

```
                    "permission": "guillotina.AccessContent",
                    "role": "guillotina.Reader",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.AccessContent",
                    "role": "guillotina.Reviewer",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.AccessContent",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.AccessContent",
                    "role": "guillotina.Editor",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.AccessContent",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.DeleteContent",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.AddContent",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ModifyContent",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ModifyContent",
                    "role": "guillotina.Editor",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ChangePermissions",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.SeePermissions",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ReindexContent",
                    "role": "guillotina.Owner",
                    "setting": "Allow"
```

```
                },
                {
                    "permission": "guillotina.ReindexContent",
                    "role": "guillotina.Editor",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ManageAddons",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ReadConfiguration",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.WriteConfiguration",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.RegisterConfigurations",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.ManageCatalog",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.RawSearchContent",
                    "role": "guillotina.ContainerAdmin",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.DeleteContainers",
                    "role": "guillotina.ContainerDeleter",
                    "setting": "Allow"
                },
                {
                    "permission": "guillotina.SearchContent",
                    "role": "guillotina.Member",
                    "setting": "Allow"
                }
            ]
        }
    }
]
```

GET @behaviors Get information on behaviors for this resource **permission: guillotina.AccessContent**http

```
GET /db/container/folder/item/@behaviors HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db/container/folder/item/@behaviors -H
↪'Accept: application/json' --user root:root
```

httpie

```
http http://localhost:8080/db/container/folder/item/@behaviors␣
↪Accept:application/json -a root:root
```

python-requests

```
requests.get('http://localhost:8080/db/container/folder/item/@behaviors',␣
↪headers={'Accept': 'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{
    "available": [],
    "dynamic": [],
    "guillotina.behaviors.dublincore.IDublinCore": {
        "invariants": [],
        "properties": {
            "contributors": {
                "description": "The unqualified Dublin Core 'Contributor'␣
↪element values",
                "items": {
                    "type": "string"
                },
                "title": "Contributors",
                "type": "array"
            },
            "creation_date": {
                "description": "The date and time that an object is created.␣
↪\nThis is normally set automatically.",
                "title": "Creation Date",
                "type": "datetime"
            },
            "creators": {
                "description": "The unqualified Dublin Core 'Creator'␣
↪element values",
                "items": {
                    "type": "string"
                },
                "title": "Creators",
                "type": "array"
            },
            "description": {
                "description": "The first unqualified Dublin Core
↪'Description' element value.",
                "title": "Description",
                "type": "string"
            },
            "effective_date": {
                "description": "The date and time that an object should be␣
↪published. ",
```

```
                "title": "Effective Date",
                "type": "datetime"
            },
            "expiration_date": {
                "description": "The date and time that the object should␣
↪become unpublished.",
                "title": "Expiration Date",
                "type": "datetime"
            },
            "modification_date": {
                "description": "The date and time that the object was last␣
↪modified in a\nmeaningful way.",
                "title": "Modification Date",
                "type": "datetime"
            },
            "publisher": {
                "description": "The first unqualified Dublin Core 'Publisher
↪' element value.",
                "title": "Publisher",
                "type": "string"
            },
            "tags": {
                "description": "The unqualified Dublin Core 'Tags' element␣
↪values",
                "items": {
                    "type": "string"
                },
                "title": "Tags",
                "type": "array"
            },
            "title": {
                "description": "The first unqualified Dublin Core 'Title'␣
↪element value.",
                "title": "Title",
                "type": "string"
            }
        },
        "required": [
            "title",
            "description",
            "creation_date",
            "modification_date",
            "effective_date",
            "expiration_date",
            "creators",
            "tags",
            "publisher",
            "contributors"
        ],
        "type": "object"
    },
    "static": [
        "guillotina.behaviors.dublincore.IDublinCore"
    ]
}
```


GET @sharing Get sharing settings for this resource **permission: guillotina.SeePermissions**http

```
GET /db/container/folder/item/@sharing HTTP/1.1
Accept: application/json
Host: localhost:8080
Authorization: Basic cm9vdDpyb290
```

curl

```
curl -i http://localhost:8080/db/container/folder/item/@sharing -H 'Accept:␣
↪application/json' --user root:root
```

httpie

```
http http://localhost:8080/db/container/folder/item/@sharing␣
↪Accept:application/json -a root:root
```

python-requests

```
requests.get('http://localhost:8080/db/container/folder/item/@sharing',␣
↪headers={'Accept': 'application/json'}, auth=('root', 'root'))
```

response

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Python/3.6 aiohttp/2.2.5

{
    "inherit": [
        {
            "@id": "http://localhost:8080/db/container/folder",
            "prinperm": {},
            "prinrole": {
                "root": {
                    "guillotina.Owner": "Allow"
                }
            },
            "roleperm": {}
        },
        {
            "@id": "http://localhost:8080/db/container",
            "prinperm": {},
            "prinrole": {
                "root": {
                    "guillotina.ContainerAdmin": "Allow",
                    "guillotina.Owner": "Allow"
                }
            },
            "roleperm": {}
        }
    ],
    "local": {
        "prinperm": {},
        "prinrole": {
            "root": {
                "guillotina.Owner": "Allow"
            }
        },
        "roleperm": {}
```

```
    }
}
```

1.4 Installation/Configuration/Deployment

production

Contents:

1.4.1 Installation

Guillotina is an installable python package which can be installed with `pip`, `easy_install` or `buildout`.

Additionally, Guillotina provides docker images.

Running

Installing Guillotina provide the `g` executable. To run the server, simply:

```
g serve
```

Read command options for details of options.

1.4.2 Configuration

`guillotina` and its addons define a global configuration that is used. All of these settings are configurable by providing a JSON configuration file to the start script.

By default, the startup script looks for a `config.yaml` file. You can use a different file by using the `-c` option for the script like this: `./bin/guillotina -c myconfig.yaml`.

Databases

Guillotina uses PostgreSQL out-of-the-box.

To configure available databases, use the `databases` option. Configuration options map 1-to-1 to database setup:

```
---
databases:
  - db:
      storage: postgresql
      dsn:
        scheme: postgres
        dbname: guillotina
        user: postgres
        host: localhost
        password: ''
        port: 5432
      read_only: false
```

Currently supported database drivers are:

- `postgresql`
- `cockroach`

Static files

```
static:
  favicon.ico: static/favicon.ico
  static_files: module_name:static
```

These files will then be available on urls `/favicon.ico` and `/static_files`.

JavaScript Applications

We can also serve JS apps from guillotina. These will allow routing on your JS application without any extra configuration by returning the base directory `index.html` for every sub directory in the url.

Once there is SSR support in Python, guillotina will integrate with it through this as well.

```
jsapps:
  app: path/to/app
```

Root user password

```
root_user:
  password: root
```

CORS

```
cors:
  allow_origin:
    - "*"
  allow_methods:
    - GET
    - POST
    - DELETE
    - HEAD
    - PATCH
  allow_headers:
    - "*"
  expose_headers:
    - "*"
  allow_credentials: true
  max_age: 3660
```

Applications

To extend/override Guillotina, the `applications` configuration allows you to specify which to enable.

```
applications:
  - guillotina_elasticsearch
```

Async utilities

```
utilities:
  -
    provides: guillotina.interfaces.ICatalogUtility
    factory: guillotina_elasticsearch.utility.ElasticSearchUtility
    settings: {}
```

Middleware

`guillotina` is built on `aiohttp` which provides support for middleware. You can provide an array of dotted names to use for your application.

```
middlewares:
  - guillotina_myaddon.Middleware
```

aiohttp settings

You can pass `aiohttp_settings` to configure the aiohttp server.

```
aiohttp_settings:
  client_max_size: 20971520
```

JWT Settings

If you want to enable JWT authentication, you'll need to configure the JWT secret in Guillotina.

```
jwt:
  secret: foobar
  algorithm: HS256
```

Miscellaneous settings

- `port` (number): Port to bind to. *defaults to* `8080`
- `access_log_format` (string): Customize access log format for aiohttp. *defaults to* `None`
- `store_json` (boolean): Serialize object into json field in database. *defaults to* `true`
- `host` (string): Where to host the server. *defaults to* `"0.0.0.0"`
- `port` (number): Port to bind to. *defaults to* `8080`
- `conflict_retry_attempts` (number): Number of times to retry database conflict errors. *defaults to* `3`
- `cloud_storage` (string): Dotted path to cloud storage field type. *defaults to* `"guillotina.interfaces.IDBFileField"`

Transaction strategy

Guillotina provides a few different modes to operate in to customize the level of performance versus consistency. The setting used for this is `transaction_strategy` which defaults to `resolve`.

Even though we have different transaction strategies that provide different voting algorithms to decide if it's a safe write, all write operations STILL make sure that the object committed matches the transaction it was retrieved with. If not, a conflict error is detected and the request is retried. So even if you choose the transaction strategy with no database transactions, there is still a level of consistency so that you know you will only modify an object that is consistent with the one retrieved from the database.

Example configuration:

```
databases:
  - db:
      storage: postgresql
      transaction_strategy: resolve
      dsn:
        scheme: postgres
        dbname: guillotina
        user: postgres
        host: localhost
        password: ''
        port: 5432
```

Available options:

- `none`: No db transaction, no conflict resolution. Fastest but most dangerous mode. Use for importing data or if you need high performance and do not have multiple writers.
- `tidonly`: The same as `none` with no database transaction; however, we still use the database to issue us transaction ids for the data committed. Since no transaction is used, this is potentially just as safe as any of the other strategies just as long as you are not writing to multiple objects at the same time — in those cases, you might be in an inconsistent state on `tid` conflicts.
- `dbresolve`: Use db transaction but do not perform any voting when writing(no conflict resolution).
- `dbresolve_readcommitted`: Same as no vote; however, db transaction only started at commit phase. This should provide better performance; however, you'll need to consider the side affects of this for reading data.
- `simple`: Detect concurrent transactions and error if another transaction id is committed to the db ahead of the current transaction id. This is the safest mode to operate in but you might see conflict errors.
- `resolve`: Same as simple; however, it allows commits when conflicting transactions are writing to different objects.
- `resolve_readcommitted`: Same as resolve however, db transaction only started at commit phase. This should provide better performance; however, you'll need to consider that side affects of this for reading data.

Warning: not all storages are compatible with all transaction strategies.

Connection class

The default asyncpg connection class has some overhead. Guillotina provides a way to override it with a custom class or a provided lighter one:

```
pg_connection_class: guillotina.db.storages.pg.LightweightConnection
```

1.4.3 Production

Nginx front

It's very common to run the API using `nginx` with a `proxy_pass` in front, so there is an option to define the URL for the generated URLs inside the api:

Adding the header:

```
X-VirtualHost-Monster https://example.com/api/
```

will do a rewrite of the URLs.

Sample configuration on `nginx`:

```
    location /api/ {
        proxy_set_header X-VirtualHost-Monster $scheme://$http_host/api/
        proxy_pass http://api.guillotina.svc.cluster.local:80/;
    }
```

1.4.4 Logging

Logging configuration is built into `guillotina`'s configuration syntax.

If the `logging` setting is provided, it is simply passed to Python's `dict` config method: https://docs.python.org/3.6/library/logging.config.html#logging-config-dictschema

Example guillotina configuration

To log errors for guillotina for example:

```
{
  "logging": {
    "version": 1,
    "formatters": {
      "brief": {
        "format": "%(message)s"
      },
      "default": {
        "format": "%(asctime)s %(levelname)-8s %(name)-15s %(message)s",
        "datefmt": "%Y-%m-%d %H:%M:%S"
      }
    },
    "handlers": {
      "file": {
        "class": "logging.handlers.RotatingFileHandler",
        "formatter": "default",
        "filename": "logconfig.log",
        "maxBytes": 1024,
        "backupCount": 3
      }
    },
```

```
    "loggers": {
      "guillotina": {
        "level": "DEBUG",
        "handlers": ["file"],
        "propagate": 0
      }
    }
  }
}
```

Request logging example

```
{
  "logging": {
    "version": 1,
    "formatters": {
      "default": {
        "format": "%(message)s"
      }
    },
    "handlers": {
      "file": {
        "class": "logging.handlers.RotatingFileHandler",
        "formatter": "default",
        "filename": "access.log",
        "maxBytes": 1024,
        "backupCount": 3
      }
    },
    "loggers": {
      "aiohttp.access": {
        "level": "INFO",
        "handlers": ["file"],
        "propagate": 0
      }
    }
  }
}
```

Available Loggers

- guillotina
- aiohttp.access
- aiohttp.client
- aiohttp.internal
- aiohttp.server
- aiohttp.web
- aiohttp.websocket

1.5 Awesome Guillotina

Some useful applications to get you started:

- guillotina_swagger: Automatic swagger generation
- guillotina_elasticsearch: Elasticsearch integration
- guillotina_dbusers: Users/Groups stored as content
- guillotina_mailer: Mailer utilities
- guillotina_rediscache: Cache database with redis
- guillotina_s3storage: S3 file storage
- guillotina_gcloudstorage: GCloud file storage
- guillotina_statsd: statsd metrics
- guillotina_prometheus: prometheus stats

Where to find more packages:

- Guillotina Web
- Onna

1.6 Developer documentation

Contents:

1.6.1 Narrative

In these narrative docs, we'll go through creating a todo application.

Installation

```
pip install guillotina
```

Generating the initial application

Guillotina comes with a `cookiecutter` template for creating a base application.

First, install `cookiecutter` if it isn't already installed.

```
pip install cookiecutter
```

Then, run the generate command:

```
guillotina create --template=application
```

Enter `guillotina_todo` for package_name.

Then, install your package:

```
cd guillotina_todo
python setup.py develop
```

Configuring

The scaffold produces an initial `config.yaml` configuration file for you.

You can inspect and customize your configuration. Most notable is the database configuration. If you want to run a development `postgresql` server, the scaffold ships with a `Makefile` that provides a command to run a postgresql docker: `make run-postgres`.

Creating to do type

Types consist of an interface (schema) using the excellent `zope.interface` package and a class that uses that interface.

Create a `content.py` file with the following:

```
from guillotina import configure
from guillotina import schema
from guillotina import interfaces
from guillotina import content


class IToDo(interfaces.IItem):
    text = schema.Text()


@configure.contenttype(
    type_name="ToDo",
    schema=IToDo)
class ToDo(content.Item):
    """
    Our ToDo type
    """
```

Then, we want to make sure our content type configuration is getting loaded, so add this to your `__init__.py` `includeme` function:

```
    configure.scan('guillotina_todo.content')
```

Running

You run you application by using the guillotina command runner again:

```
guillotina serve -c config.yaml
```

Creating your todo list

Create container first:

```
curl -X POST --user root:root \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -d '{
  "@type": "Container",
  "title": "ToDo List",
  "id": "todo",
  "description": "My todo list"
  }' "http://127.0.0.1:8080/db/"
```

Install your todo list application:

```
curl -X POST \
  --user root:root \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -d '{
  "id": "guillotina_todo"
  }' "http://127.0.0.1:8080/db/todo/@addons"
```

Add todo items:

```
curl -X POST \
  --user root:root \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -d '{
  "@type": "ToDo",
  "text": "Get milk"
  }' "http://127.0.0.1:8080/db/todo"
```

```
curl -X POST \
  --user root:root \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -d '{
  "@type": "ToDo",
  "text": "Do laundry"
  }' "http://127.0.0.1:8080/db/todo"
```

Get a list of todo items:

```
curl -H "Accept: application/json" --user root:root "http://127.0.0.1:8080/
↪db/todo"
```

1.6.2 Security

Security for every operation is managed against three definitions (in order of priority):

- Local
- Global
- Code

Locally can be defined:

- A user/group has a permission in this object but not children

- A user/group has a permission in this object and its children
- A user/group is forbidden permission in this object and its children
- A user/group has a role on this object but not its children
- A user/group has a role on this object and its children
- A user/group is forbidden a role on this object and its children
- A role has a permission on this object and its children
- A role has a permission on this object and its children
- A role is forbidden permission in this object and its children

Globally:

- A user/group has this Role
- A user/group has this Permission

Code:

- A user/group has this Role
- A user/group has this Permission
- A Role has this Permission

Roles

There are two kind of roles: Global and Local. The ones that are defined to be local can't be used globally and vice-versa. On indexing, the global roles are the ones that are indexed for security in addition to the flat user/group information from each resource.

Python helper functions

```
# Code to get the global roles that have access_content to an object
from guillotina.security.utils import get_roles_with_access_content
get_roles_with_access_content(obj)

# Code to get the user list that have access content to an object
from guillotina.security.utils import get_principals_with_access_content
get_principals_with_access_content(obj)


# Code to get all the security info
from guillotina.security.utils import settings_for_object
settings_for_object(obj)

# Code to get the Interaction object ( security object )
from guillotina.interfaces import IInteraction

interaction = IInteraction(request)

# Get the list of global roles for a user and some groups
interaction.global_principal_roles(principal, groups)
```

```
# Get if the authenticated user has permission on a object
interaction.check_permission(permission, obj)
```

REST APIs

Get all the endpoints and their security

`[GET] APPLICATION_URL/@apidefinition` (you need `guillotina.GetContainers` permission)

Get the security info for a resource (with inherited info)

`[GET] RESOURCE/@sharing` (you need `guillotina.SeePermissions` permission)

Modify the local roles/permission for a resource

`[POST] RESOURCE/@sharing` (you need `guillotina.ChangePermissions` permission)

```
{
"prinperm": [
  {
    "principal": "foobar",
    "permission": "guillotina.ModifyContent",
    "setting": "Allow"
  }
],
"prinrole": [
  {
    "principal": "foobar",
    "role": "guillotina.Owner",
    "setting": "Allow"
  }
],
"roleperm": [
  {
    "permission": "guillotina.ModifyContent",
    "role": "guillotina.Member",
    "setting": "Allow"
  }
]
}
```

The different types are:

- Allow: you set it on the resource and the children will inherit
- Deny: you set it on the resource and the children will inherit
- AllowSingle: you set it on the resource and the children will not inherit
- Unset: you remove the setting

1.6.3 Roles

`guillotina` implements robust ACL security.

An overview of our security features are:

- Users are given roles and groups
- Roles are granted permissions
- Groups are granted roles
- Roles can be granted to users on specific objects

Requests security

By default request has participation of anonymous user plus the ones added by auth plugins

Databases, Application and static files objects

Databases and static files have a specific permission system. They don't have roles by default and the permissions are specified to root user

- guillotina.AddContainer
- guillotina.GetContainers
- guillotina.DeleteContainers
- guillotina.AccessContent
- guillotina.GetDatabases

Anonymous user has on DB/StaticFiles/StaticDirectories/Application object :

- guillotina.AccessContent

Roles in guillotina container objects

Defined at:

- guillotina/permissions.py

Content Related

guillotina.Anonymous

- guillotina.AccessPreflight

guillotina.Member

- guillotina.AccessContent

guillotina.Reader

- guillotina.AccessContent
- guillotina.ViewContent

guillotina.Editor

- guillotina.AccessContent
- guillotina.ViewContent
- guillotina.ModifyContent
- guillotina.ReindexContent

guillotina.Reviewer

guillotina.Owner

- guillotina.AccessContent
- guillotina.ViewContent
- guillotina.ModifyContent
- guillotina.DeleteContent
- guillotina.AddContent
- guillotina.ChangePermissions
- guillotina.SeePermissions
- guillotina.ReindexContent

Container/App Roles

guillotina.ContainerAdmin

- guillotina.AccessContent
- guillotina.ManageAddons
- guillotina.RegisterConfigurations
- guillotina.WriteConfiguration
- guillotina.ReadConfiguration
- guillotina.ManageCatalog

guillotina.ContainerDeleter

- guillotina.DeletePortal

Default roles on Guillotina Container

They are stored in annotations using `IRolePermissionMap`.

Created objects set the `guillotina.Owner` role to the user who created it.

Default groups on Guillotina Container

Managers

RootParticipation

There is a `root` user who has permissions to all containers:

DB/APP permissions are defined on factory/content.py

1.6.4 Applications

Applications are used to provide additional functionality to guillotina.

Community Addons

Some useful addons to use in your own development:

- guillotina_elasticsearch: Index content in elastic search
- guillotina_pgcatalog: Index content in postgresql
- guillotina_dbusers: Store and authenticate users in the database
- guillotina_swagger: Automatic swagger support
- guillotina_mailer: async send mail

Creating

An application is a Python package that implements an entry point to tell guillotina to load it.

If you're not familiar with how to build Python applications, please read documentation on building packages before you continue.

In this example, `guillotina_myaddon` is your package module.

Initialization

Your `config.yaml` file will need to provide the application name in the `applications` array for it to be initialized.

```
applications:
    - guillotina_myaddon
```

Configuration

Once you create a `guillotina` application, there are two primary ways for it to hook into `guillotina`.

Call the `includeme` function

Your application can provide an `includeme` function at the root of the module and `guillotina` will call it with the instance of the `root` object.

```
def includeme(root):
  # do initialization here...
  pass
```

Load `app_settings`

If an `app_settings` dict is provided at the module root, it will automatically merge the global `guillotina` `app_settings` with the module's. This allows you to provide custom configuration.

1.6.5 Add-ons

Addons are integrations that can be installed or uninstalled against a Guillotina container. `guillotina` applications can potentially provide many addons. If you have not read the section on applications, please read that before you come here. The only way to provide addons is to first implement a `guillotina` application.

Creating an add-on

Create an addon installer class in an `install.py` file in your `guillotina` application:

```
from guillotina.addons import Addon
from guillotina import configure

@configure.addon(
    name="myaddon",
    title="My addon")
class MyAddon(Addon):

    @classmethod
    def install(cls, container, request):
        # install code
        pass

    @classmethod
    def uninstall(cls, container, request):
        # uninstall code
        pass
```

Note: Scanning

If your service modules are not imported at run-time, you may need to provide an additional scan call to get your services noticed by *guillotina*.

In your application *__init__.py* file, you can simply provide a *scan* call like:

```
from guillotina import configure
def includeme(root):
    configure.scan('my.package')
```

Layers

Your addon can also install layers for your application to lookup views and adapters from:

```
from guillotina.addons import Addon
from guillotina import configure
from guillotina.interfaces import ILayers

LAYER = 'guillotina_myaddon.interfaces.ILayer'

@configure.addon(
    name="myaddon",
    title="My addon")
class MyAddon(Addon):

    @classmethod
    def install(cls, container, request):
      registry = request.container_settings
      registry.for_interface(ILayers).active_layers |= {
          LAYER
      }

    @classmethod
    def uninstall(cls, container, request):
      registry = request.container_settings
      registry.for_interface(ILayers).active_layers -= {
        LAYER
      }
```

1.6.6 Services

Services provide responses to API endpoint requests. A service is the same as a "view" that you might see in many web frameworks.

The reason we're using the convention "service" is because we're focusing on creating API endpoints.

Defining a service

A service can be as simple as a function in your application:

```
from guillotina import configure
from guillotina.interfaces import IContainer

@configure.service(context=IContainer, name='@myservice', method='GET',
```

```
                    permission='guillotina.AccessContent')
async def my_service(context, request):
    return {
        'foo': 'bar'
    }
```

The most simple way to define a service is to use the decorator method shown here.

As long as your application imports the module where your service is defined, your service will be loaded for you.

In this example, the service will apply to a GET request against a container, `/zodb/guillotina/@myservice`.

Note: Scanning

If your service modules are not imported at run-time, you may need to provide an additional scan call to get your services noticed by *guillotina*.

In your application *__init__.py* file, you can simply provide a *scan* call like:

```
from guillotina import configure
def includeme(root):
    configure.scan('my.package')
```

Class-based services

For more complex services, you might want to use class-based services.

With the class-based approach, the example above will look like this:

```
from guillotina import configure
from guillotina.interfaces import IContainer
from guillotina.api.service import Service

@configure.service(context=IContainer, name='@myservice', method='GET',
                   permission='guillotina.AccessContent')
class MyService(Service):
    async def __call__(self):
        return {
            'foo': 'bar'
        }
```

Special cases

I want that my service is accessible no matter the content

You can define in the service configuration with `allow_acces=True`

```
@service(
    context=IResource, name='@download',
    method='GET', permission='guillotina.Public',
    allow_access=True)
```

```
async def my_service(context, request):
    pass
```

1.6.7 Content types

Content types allow you to provide custom schemas and content to your services.

Out-of-the-box, `guillotina` ships with simple `Container`, `Folder` and `Item` content types. The `Container` content type is the main content type to hold your data in. It is the starting point for applications and other things to operate within.

The `Folder` type allows someone to add items inside of it. Both types only have simple Dublin Core fields by default.

Defining content types

A content type consists of a class and optionally, a schema to define the custom fields you want your class to use.

A simple type will look like this::

```
from guillotina import configure
from guillotina.content import Folder
from guillotina.interfaces import IItem
from guillotina import schema

class IMySchema(IItem):
    foo = schema.Text()

@configure.contenttype(
    type_name="MyType",
    schema=IMySchema,
    behaviors=["guillotina.behaviors.dublincore.IDublinCore"])
class MyType(Folder):
    pass
```

This example creates a simple schema and assigns it to the `MyType` content type.

Note: Scanning

If your service modules are not imported at run-time, you may need to provide an additional scan call to get your services noticed by *guillotina*.

In your application *__init__.py* file, you can simply provide a *scan* call like:

```
from guillotina import configure
def includeme(root):
    configure.scan('my.package')
```

1.6.8 Behaviors

Besides having static content types definitions with their schema, there is the concept of *behaviors*. This allows us to provide functionality across content types, using specific marker interfaces to create adapters

and subscribers based on that behavior and not the content type.

Definition of a behavior

If you want to have a shared behavior based on some fields and operations that needs to be shared across different content types, you can define them on a `guillotina.schema` interface:

```
    from zope.interface import Interface
    from zope.interface import provider
    from guillotina.schema import Textline

    class IMyLovedBehavior(Interface):
        text = Textline(
            title=u'Text line field',
            required=False
        )

        text2 = Textline(
            title=u'Text line field',
            required=False
        )
```

Once you define the schema you can define a specific marker interface that will be applied to the objects that has this behavior:

```
    class IMarkerBehavior(Interface):
        """Marker interface for content with attachment."""
```

Finally the instance class that implements the schema can be defined in case you want to enable specific operations. Or you can use `guillotina.behaviors.instance.AnnotationBehavior` as the default annotation storage.

For example, in case you want to have a class that stores the field as content and not as annotations:

```
from guillotina.behaviors.properties import ContextProperty
from guillotina.behaviors.instance import AnnotationBehavior
from guillotina.interfaces import IResource
from guillotina import configure

@configure.behavior(
    title="Attachment",
    provides=IMyLovedBehavior,
    marker=IMarkerBehavior,
    for_=IResource)
class MyBehavior(AnnotationBehavior):
    """If attributes
    """
    text = ContextProperty(u'attribute', ())
```

In this example `text` will be stored on the context object and `text2` as a annotation.

Static behaviors

With behaviors you can define them as static for specific content types:

```
from guillotina import configure
from guillotina.interfaces import IItem
from guillotina.content import Item

@configure.contenttype(
    type_name="MyItem",
    schema=IItem,
    behaviors=["guillotina.behaviors.dublincore.IDublinCore"])
class MyItem(Item):
    pass
```

Note: Scanning

If your service modules are not imported at run-time, you may need to provide an additional scan call to get your services noticed by *guillotina*.

In your application *__init__.py* file, you can simply provide a *scan* call like:

```
from guillotina import configure
def includeme(root):
    configure.scan('my.package')
```

Create and modify content with behaviors

For the deserialization of the content you will need to pass on the POST/PATCH operation the behavior as a object on the JSON.

CREATE an ITEM with the expires : POST on parent:

```
    {
        "@type": "Item",
        "guillotina.behaviors.dublincore.IDublinCore": {
            "expires": "1/10/2017"
        }
    }
```

MODIFY an ITEM with the expires : PATCH on the object:

```
    {
        "guillotina.behaviors.dublincore.IDublinCore": {
            "expires": "1/10/2017"
        }
    }
```

Get content with behaviors

On the serialization of the content you will get the behaviors as objects on the content.

GET an ITEM : GET on the object:

```
    {
        "@id": "http://localhost:8080/zodb/guillotina/item1",
```

```
        "guillotina.behaviors.dublincore.IDublinCore": {
            "expires": "2017-10-01T00:00:00.000000+00:00",
            "modified": "2016-12-02T14:14:49.859953+00:00",
        }
    }
```

Dynamic Behaviors

`guillotina` offers the option to have content that has dynamic behaviors applied to them.

Which behaviors are available on a context

We can know which behaviors can be applied to a specific content.

`GET CONTENT_URI/@behaviors`:

```
    {
        "available": ["guillotina.behaviors.attachment.IAttachment"],
        "static": ["guillotina.behaviors.dublincore.IDublinCore"],
        "dynamic": [],
        "guillotina.behaviors.attachment.IAttachment": { },
        "guillotina.behaviors.dublincore.IDublinCore": { }
    }
```

This list of behaviors is based on the `for` statement on the configure of the behavior. The list of static ones are the ones defined on the content type definition on the configure. The list of dynamic ones are the ones that have been assigned.

Add a new behavior to a content

We can add a new dynamic behavior to a content using a `PATCH` operation on the object with the `@behavior` attribute, or in a small `PATCH` operation to the `@behavior` entry point with the value to add.

MODIFY an ITEM with the expires : PATCH on the object:

```
    {
        "guillotina.behaviors.dublincore.IDublinCore": {
            "expires": "1/10/2017"
        }
    }
```

MODIFY behaviors : PATCH on the object/@behaviors:

```
    {
        "behavior": "guillotina.behaviors.dublincore.IDublinCore"
    }
```

Delete a behavior to a content

We can add a new dynamic behavior to a content by a `DELETE` operation to the `@behavior` entry point with the value to remove.

DELETE behaviors : DELETE on the object/@behaviors:

```
{
    "behavior": "guillotina.behaviors.dublincore.IDublinCore"
}
```

Out-of-the-box Behaviors

Guillotina comes with a couple behaviors:

- `guillotina.behaviors.dublincore.IDublinCore`: Dublin core field
- `guillotina.behaviors.attachment.IAttachment`: Provide file field

1.6.9 Interfaces

`guillotina` uses interfaces to abstract and define various things including content. Interfaces are useful when defining API contracts, using inheritance, defining schema/behaviors and being able to define which content your services are used for.

In the services example, you'll notice the use of `context=IContainer` for the service decorator configuration. In that case, it is used to tell `guillotina` that the service is only defined for a container object.

Common interfaces

Interfaces you will be interested in defining services for are:

- `guillotina.interface.IDatabase`: A database contains the container objects
- `guillotina.interface.IContainer`: Container content object
- `guillotina.interface.IResource`: Base interface for all content
- `guillotina.interface.IContainer`: Base interface for content that can contain other content
- `guillotina.interface.IRegistry`: Registry object interface
- `guillotina.interface.IDefaultLayer`: Layers are an interface applied to the request object. IDefaultLayer is the base default layer applied to the request object.

1.6.10 Commands

You can provide your own CLI commands for guillotina through a simple interface.

Available commands

- `serve`: run the HTTP REST API server (this is the default command if none given)
- `shell`: drop into a shell with root object to manually work with
- `create`: use cookiecutter to generate guillotina applications
- `initialize-db`: databases are automatically initialized; however, you can use this command to manually do it

- `testdata`: populate the database with test data from wikipedia
- `run`: run a python script. The file must have a function `async def run(container):`

Command Options

- –
  - `--config`: path to configuration file. `defaults to config.(yaml|json)`
  - `--profile`: profile Guillotina while it's running
  - `--profile-output`: where to save profiling output
  - `--monitor`: run with aiomonitor `requires aiomonitor`
  - `--line-profiler`: use line_profiler `requires line_profiler`
  - `--line-profiler-matcher`: fnmatch of module/function to profile `requires line_profiler`
  - `--line-profiler-output`: to store output in a file `requires line_profiler`
- serve:
  - `--host`: host to bind to
  - `--port`: port to bind to
  - `--reload`: auto reload on code changes. `requires aiohttp_autoreload`
- shell
- create
  - `--template`: name of template to use
  - `--overwrite`: overwrite existing file
  - `--output`: where to save the file
- initialize-db
- testdata
  - `--per-node`: How many items to import per node
  - `--depth`: How deep to make the nodes
- run
  - `--script`: path to script to run with `run` async function

Running commands

Guillotina provides two binaries to run commands through, `bin/guillotina` and a shortcut, `bin/g`.

To run a command, it's just a positional argument on the running command::

```
bin/g shell
```

Creating commands

guillotina provides a simple API to write your own CLI commands.

Here is a minimalistic example:

```
from guillotina.commands import Command
class MyCommand(Command):

    def get_parser(self):
        parser = super(MyCommand, self).get_parser()
        # add command arguments here...
        return parser

    def run(self, arguments, settings, app):
        pass
```

Then, just add your command to your application's `app_settings` in the `__init__.py`:

```
app_settings = {
    "commands": {
        "mycommand": "my.package.commands.MyCommand"
    }
}
```

1.6.11 Application Configuration

`guillotina` handles configuration application customizations and extension mostly with decorators in code.

This page is meant to be a reference to the available decorators and options to those decorators.

service

`@configure.service`

- *context*: Content type interface this service is registered against. Example: IContainer: *required*
- *method*: HTTP method this service works against. Default is `GET`
- *permission*: Permission this service requires. Default is configure default_permission setting
- *layer*: Layer this service is registered for. Default is `IDefaultLayer`
- *name*: This is used as part of the uri. Example `@foobar` -> `/mycontent/@foobar`. Leave empty to be used for base uri of content -> `/mycontent`.

content type

`@configure.contenttype`

- *type_name*: Name of the content type: *required*
- *schema*: Interface schema to use for type: *required*

- *add_permission*: Permission required to add content. Defaults to `guillotina.AddContent`
- *allowed_types*: List of types allowed to be added inside this content assuming it is a Folder type. Defaults to allowing all types.

behavior

@configure.behavior

- *title*: Name of behavior
- *provides*: Interface this behavior provides
- *marker*: Marker interface to apply to utilized instance's behavior
- *for_*: Content type this behavior is available for

addon

@configure.addon

- *name*: *required*
- *title*: *required*

adapter

@configure.adapter

- *for_*: Type or list of types this adapter adapts: *required*
- *provides*: Interface this adapter provides: required
- *name*: Your adapter can be named to be looked up by name
- *factory*: To use without decorator syntax, this allows you to register adapter of class defined elsewhere

subscriber

@configure.subscriber

- *for_*: Type or list of types this subscriber is for: *required*
- *handler*: A callable object that handles event, this allows you to register subscriber handler defined elsewhere
- *factory*: A factory used to create the subscriber instance
- *provides*: Interface this adapter provides–must be used along with factory

utility

@configure.utility

- *provides*: Interface this utility provides
- *name*: Name of utility
- *factory*: A factory used to create the subscriber instance

permission

configure.permission

- *id*
- *title*
- *description*

role

configure.role

- *id*
- *title*
- *description*

grant

configure.grant

- *role*: ID of role
- *principal*: ID of principal to grant to
- *permission*: ID of permission to grant
- *permissions*: List of permission IDs to grant to

grant_all

configure.grant_all

- *principal*: ID of principal
- *role*: ID of role

1.6.12 Overriding Configuration

`guillotina` applications can override default `guillotina` configuration.

If multiple `guillotina` applications configure conflicting configurations, `guillotina` chooses the configuration according to the order the `guillotina` applications that are included.

1.6.13 Design

This section is meant to explain and defend the design of `guillotina`.

JavaScript application development focus

One of the main driving factors behind the development of `guillotina` is to streamline the development of custom web applications.

Some of the technologies we support in order to be a great web application development platform are:

- Everything is an API endpoint
- JWT
- Web sockets
- Configuration is done with JSON
- URL to object-tree data model

Speed

A primary focus of `guillotina` is speed. We take shortcuts and may use some ugly or less-well conceptually architected solutions in some areas in order to gain speed improvements.

Some of the decisions we made affect how applications and addons are designed. Mainly, we try to stay light on the amount of data we're loading from the database where possible and we try to lower the number of lookups we do in certain scenarios.

That being said, `guillotina` is not a barebones framework. It provides a lot of functionality so it will never be as fast as say Pyramid.

"There are no solutions. There are only trade-offs." - Thomas Sowell

Asynchronous

`guillotina` is asynchronous from the ground up, built on top of `aiohttp` using Python 3.6's asyncio features.

Practically speaking, being built completely on asyncio compatible technologies, `guillotina` does not block for network IO to the database, index catalog, redis, etc or whatever you've integrated.

Additionally, we have support for async utilities that run in the same async loop and async content events.

Finally, we also support web sockets OOTB.

Security

`Guillotina` uses the same great security infrastructure that Plone has been using for the last 15 years which allows you to define permissions, roles, groups, users and customize all of them contextually based on where the content is located in your container.

Style

Stylistically, `guillotina` pulls ideas from the best web frameworks:

- YAML/JSON configuration
- Pyramid-like idioms and syntax where it makes sense
- Functions + decorators over classes

1.6.14 Persistence

There are three kinds of objects that are considered on the system:

- Tree objects: objects are resources that implement `guillotina.interfaces.IResource`. This object has a `__name__` and a `__parent__` property that indicate the id on the tree and the link to the parent. By themselves they don't have access totheir children, they need to interact with the transaction object to get them.
- Nested: objects that are linked at some attribute inside the Tree object, this object are serialized with the main object and may lead to conflicts if there are lots of this kind of objects. It can belong to a field that is an object
- Annotations: objects that are associated with tree objects. These can be any type of data. In Guillotina, the main source of annotation objects are behaviors.

Saving objects

If you're manually modifying objects in services(or views) without using the serialization adapters, you need to register the object to be saved to the database. To do this, just use the `_p_register()` method.

```
@configure.service(
    method='PATCH', name='@dosomething')
async def matching_service(context, request):
    context.foobar = 'foobar'
    context._p_register()
```

Transactions

Guillotina automatically manages transactions for you in services; however, if you have long running services and need to flush data to the database, you can manually manage transactions as well.

```
from guillotina.transactions import get_tm

tm = get_tm()
await tm.commit()  # commit current transaction
await tm.begin()  # start new one
```

There is also an async context manager:

```
from guillotina.transactions import managed_transaction

async with managed_transaction() as txn:
    # modify objects
```

1.6.15 Blobs

`guillotina` provides basic blob file persistency support. These blobs are still stored in the database.

Registering a blobs

Blobs must be registered with and stored on a resource object. This is so we can do garbage collection on the blobs that were created for resources.

```
from guillotina.blob import Blob

blob = Blob(resource)
resource.blob = blob
blobfi = blob.open('w')

await blobfi.async_write(b'foobar')
assert await blobfi.async_read() == b'foobar'
```

Guillotina automatically reads and writes chunks of blob data from the database.

1.6.16 Router

Guillotina uses `aiohttp` for it's webserver. In order to route requests against Guillotina's traversal url structure, Guillotina provides it's own router that does traversal: `guillotina.traversal.router`.

How URLs are routed

Guillotina's content is structured like a file system. Objects are routed to URL paths. HTTP verbs are provided against those objects on those paths. Additional services(or views depending on terminology) are provided with URL path parts that start with `@`, for example, the `@move` endpoint.

Route matching

With Guillotina, you can also route custom sub paths off a registered service. Guillotina is primarily for routing objects to urls; however, this feature is used to provide additional parameters to the service.

An example of where this is used is for file services: `/db/container/item/@upload/file`.

Registering custom route parts

```
@configure.service(
    method='GET', permission='guillotina.AccessContent',
    name='@match/{foo}/{bar}')
async def matching_service(context, request):
    return request.matchdict  # will return {'foo': 'foo', 'bar': 'bar'}
```

Providing your own router

Guillotina allows you to provide your own customized router using the `router` settings.

Here is an example router that provides `/v1` and `/v2` type url structure:

```
from guillotina import configure
from guillotina.content import Resource
from guillotina.interfaces import IContainer
from guillotina.interfaces import IDefaultLayer
from guillotina.interfaces import IRequest
from guillotina.interfaces import IResource
from guillotina.traversal import TraversalRouter
```

```
from guillotina.traversal import traverse
from zope.interface import alsoProvides


class IV1Layer(IDefaultLayer):
    pass


class IV2Layer(IDefaultLayer):
    pass


@configure.service(method='GET', name='@foobar',
                   permission='guillotina.AccessContent',
                   layer=IV1Layer)
async def v1_service(context, request):
    return {
        'version': '1'
    }


@configure.service(method='GET', name='@foobar',
                   permission='guillotina.AccessContent',
                   layer=IV2Layer)
async def v2_service(context, request):
    return {
        'version': '2'
    }


@configure.contenttype(type_name="VersionRouteSegment")
class VersionRouteSegment(Resource):

    type_name = 'VersionRouteSegment'

    def __init__(self, name, parent):
        super().__init__()
        self.__name__ = self.id = name
        self.__parent__ = parent


class MyRouter(TraversalRouter):
    async def traverse(self, request: IRequest) -> IResource:
        resource, tail = await super().traverse(request)
        if len(tail) > 0 and tail[0] in ('v1', 'v2') and IContainer.
↪providedBy(resource):
            segment = VersionRouteSegment(tail[0], resource)
            if tail[0] == 'v1':
                alsoProvides(request, IV1Layer)
            elif tail[0] == 'v2':
                alsoProvides(request, IV2Layer)

            if len(tail) > 1:
                # finish traversal from here
                return await traverse(request, segment, tail[1:])
            else:
                resource = segment
                tail = tail[1:]
```

```
        return resource, tail


app_settings = {
    # provide custom application settings here...
    'router': MyRouter
}
```

1.6.17 Exceptions

Exceptions during the rendering of API calls are wrapped, logged and provided generic http status codes by default.

Guillotina provides a mechanism for customizing the status codes and type of responses given depending on the exception type.

Custom exception response

```
from aiohttp.web_exceptions import HTTPPreconditionFailed
from guillotina import configure
from guillotina.interfaces import IErrorResponseException

import json


@configure.adapter(
    for_=json.decoder.JSONDecodeError,
    provides=IErrorResponseException)
def json_decode_error_response(exc, error='', eid=None):
    return HTTPPreconditionFailed(
        reason=f'JSONDecodeError: {eid}')
```

1.6.18 Fields

Guillotina uses schemas to define content types and behaviors. These schemas consist of field definitions.

Available fields

- guillotina.schema.Bool
- guillotina.schema.Bytes
- guillotina.schema.Choice: validates against vocabulary of values
- guillotina.schema.Date
- guillotina.schema.Datetime
- guillotina.schema.Decimal
- guillotina.schema.Dict
- guillotina.schema.Float
- guillotina.schema.Int

- guillotina.schema.JSONField
- guillotina.schema.List
- guillotina.schema.Set
- guillotina.schema.Text
- guillotina.schema.TextLine
- guillotina.schema.Time
- guillotina.fields.PatchField: allow updating value without patching entire value
- guillotina.fields.BucketListField: optimized storage for very large lists of data
- guillotina.files.CloudFileField: file field for storing in db or cloud storage

Patch field

Guillotina provides a `PatchField` which allows you to patch values of `List` and `Dict` fields without having the original value.

Patch list field

```
from zope.interface import Interface
from guillotina.fields import PatchField
from guillotina import schema

class IMySchema(Interface):
    values = PatchField(schema.List(
        value_type=schema.Text()
    ))
```

Then, payload for patching to append to this list would look like:

```
{
    "values": {
        "op": "append",
        "value": "foobar"
    }
}
```

Extend:

```
{
    "values": {
        "op": "extend",
        "value": ["foo", "bar"]
    }
}
```

Delete:

```
{
    "values": {
        "op": "del",
        "value": 0
```

```
    }
}
```

Update:

```
{
    "values": {
        "op": "update",
        "value": {
            "index": 0,
            "value": "Something new"
        }
    }
}
```

Patch dict field

```
from zope.interface import Interface
from guillotina.fields import PatchField
from guillotina import schema

class IMySchema(Interface):
    values = PatchField(schema.Dict(
        key_type=schema.Text()
        value_type=schema.Text()
    ))
```

Then, payload for patching to add to this dict would look like:

```
{
    "values": {
        "op": "assign",
        "value": {
            "key": "foo",
            "value": "bar"
        }
    }
}
```

Delete:

```
{
    "values": {
        "op": "del",
        "value": "foo"
    }
}
```

Bucket list field

```
from zope.interface import Interface
from guillotina.fields import PatchField
from guillotina import schema
```

```
class IMySchema(Interface):
    values = BucketListField(
        value_type=schema.Text(),
        bucket_len=5000
    )
```

Then, payload for patching to append to this list would look like:

```
{
    "values": {
        "op": "append",
        "value": "foobar"
    }
}
```

Extend:

```
{
    "values": {
        "op": "extend",
        "value": ["foo", "bar"]
    }
}
```

Delete:

```
{
    "values": {
        "op": "del",
        "value": {
            "bucket_index": 0,
            "item_index": 0
        }
    }
}
```

1.6.19 API Reference

Contents:

guillotina.content

guillotina.utils

1.7 Training

Prerequisites:

- Python >= 3.6
- Docker
- Postman

Contents:

1.7.1 Introduction

The Guillotina training is designed to give a complete experience of using and extending Guillotina.

The training can be useful for consumers of the Guillotina API as well as developers extending the framework with customizations/addons.

Please read the about chapter for details about what Guillotina is and why you should use it.

Using the training materials

The training materials make use of Python 3.6, Docker and Postman so please have up-to-date versions of all these ready.

References

- About Guillotina

1.7.2 Installing Guillotina

Guillotina is a simple Python package so it can be installed with any of the number of installation methods available to Python.

In the traing here, we will focus on using pip and docker. You can use, for example, buildout as well.

with pip

Note: It is recommended you install along with a virtualenv:

```
virtualenv-3.6 genv
cd genv
source ./bin/activate
```

It's as simple as...

```
pip install guillotina
```

For the purpose of this training, you'll also need to install `cookiecutter`.

```
pip install cookiecutter
```

Guillotina also provides docker images.

References

- Quickstart
- Installation
- About Guillotina

1.7.3 Starting Guillotina

Once you have guillotina installed, you can easily run it with the `g` executable that it installs.

However, before we begin, we'll need to run a postgresql server for Guillotina to use.

```
docker run -e POSTGRES_DB=guillotina -e POSTGRES_USER=guillotina -p 127.0.0.
↪1:5432:5432 postgres:9.6
```

Note: This particular docker run command produces a volatile database. Stopping and starting it again will cause you to lose any data you pushed into it.

Command

Then, simply run the default Guillotina command `g`.

```
g
```

Which should give you output like:

```
$ g
Could not find the configuration file config.yaml. Using default settings.
======== Running on http://0.0.0.0:8080 ========
(Press CTRL+C to quit)
```

The `g` executable allows you to potentially run a number of commands with Guillotina. The default command is `serve` if none provided; however, you can explicitly run it with the serve command name as well.

```
g serve
```

The serve command also takes `--host` and `--port` options to quickly change without touching configuration.

In future sections, we'll explore other commands available.

Check installation

Open up Postman and do a basic `GET` against `http://localhost:8080` with basic auth credentials for `root` user and `root` password.

Also, do a `GET` on `http://localhost:8080/db`.

Congratulations! You have Guillotina running!

Useful run options

- `--reload`: auto reload on code changes. `requires aiohttp_autoreload`
- `--profile`: profile Guillotina while it's running
- `--profile-output`: where to save profiling output
- `--monitor`: run with aiomonitor. `requires aiomonitor`

References

- Quickstart
- Installation
- Configuraion
- Command Options

1.7.4 Configuration

You may have wondered how running `g` command without any configuration and options knew to connect and configure the database. Well, it's only because we provide default settings in our application and documentation to make that step easy.

In this section, we'll talk about working with the Guillotina configuration system.

Getting started

Guillotina provides a command to bootstrap a configuration file for you.

```
g create --template=configuration
```

This will produce a `config.yaml` file in your current path. Inspect the file to see what some of the default configuration options are.

Modifying configuration

A detailed list of configuration options and explanations can be found in the configuration section of the docs.

Note: Guillotina also supports JSON configuration files

Configuration file

To specify a configuration file other than the name `config.yaml`, you can use the `-c` or `--config` command line option.

```
g -c config-foobar.yaml
```

Installing applications

Guillotina applications are python packages that you install and then configure in your application settings.

For an example, we'll go through installing swagger support.

```
pip install guillotina_swagger
```

Then, add this to your `config.yaml` file.

```
applications:
- guillotina_swagger
```

Finally, start Guillotina again and visit `http://localhost:8080/@docs`.

References

- Configuration Options

1.7.5 Using the Guillotina API

Before we start using the Guillotina API, let's get us some test data to play with.

Using the `testdata` command, we'll populate our database with some data from wikipedia.

```
g testdata --per-node=5 --depth=2 --container=container
```

Interacting with the API

You can use whatever you'd like but this training will mention use of Postman.

Open up Postman and do a `GET` on `http://localhost:8080/db/container` with the username `root` and password `root` for basic auth.

We can not necessarily go over every single API but will touch on a few and give a general understanding of how to explore and use the API.

Creating content

To create content, do a `POST` request on a container or folder object.

`POST /db/container`
Create Item

Example request

```
POST /db/container HTTP/1.1
Accept: application/json
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
  "@type": "Item",
  "id": "foobar"
}
```

Example response

```
HTTP/1.1 201 OK
Content-Type: application/json

{
  "@id": "http://localhost:8080/db/container/foobar",
  "@type": "Item",
  "parent": {
      "@id": "http://localhost:8080/db/container",
```

```
        "@type": "Container"
    },
    "creation_date": "2017-10-13T23:34:18.879391-05:00",
    "modification_date": "2017-10-13T23:34:18.879391-05:00",
    "UID": "f4ab591f22824404b55b66569f6a7502",
    "type_name": "Item",
    "title": null,
    "__behaviors__": [],
    "__name__": "foobar",
    "guillotina.behaviors.dublincore.IDublinCore": {
        "title": null,
        "description": null,
        "creation_date": "2017-10-13T23:34:18.879391-05:00",
        "modification_date": "2017-10-13T23:34:18.879391-05:00",
        "effective_date": null,
        "expiration_date": null,
        "creators": [
            "root"
        ],
        "tags": null,
        "publisher": null,
        "contributors": [
            "root"
        ]
    }
}
```

Request Headers

- Authorization – Required token to authenticate

Status Codes

- 201 Created – no error
- 401 Unauthorized – Invalid Auth code
- 500 Internal Server Error – Error processing request

Adding behaviors

To add a dynamic behavior, we use the `@behavior` endpoint.

PATCH /db/container/foobar/@behaviors

Add behavior

Example request

```
PATCH /db/container/foobar/@behaviors HTTP/1.1
Accept: application/json
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
  "behavior": "guillotina.behaviors.attachment.IAttachment"
}
```

Example response

```
HTTP/1.1 201 OK
Content-Type: application/json
```

Request Headers

- Authorization – Required token to authenticate

Status Codes

- 201 Created – no error
- 401 Unauthorized – Invalid Auth code
- 500 Internal Server Error – Error processing request

Uploading files

Simple file uploads can be done with the `@upload` endpoint.

PATCH /db/container/foobar/@upload/file
Upload file

Example request

```
PATCH /db/container/foobar/@upload/file HTTP/1.1
Authorization: Basic cm9vdDpyb290

<binary data>
```

Example response

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

Request Headers

- Authorization – Required token to authenticate

Status Codes

- 200 OK – no error
- 401 Unauthorized – Invalid Auth code
- 500 Internal Server Error – Error processing request

Then, to download the file, use the `@download` endpoint.

GET /db/container/foobar/@download/file
Download file

Example request

```
GET /db/container/foobar/@downlaod/file HTTP/1.1
Authorization: Basic cm9vdDpyb290
```

Example response

```
HTTP/1.1 200 OK
<binary data>
```

Request Headers

- Authorization – Required token to authenticate

Status Codes

- 200 OK – no error
- 401 Unauthorized – Invalid Auth code
- 500 Internal Server Error – Error processing request

Uploading files with TUS

Guillotina also supports the TUS protocol using the `@tusupload` endpoint. The TUS protocol allows you to upload large files in chunks and allows you to have resumable uploads.

First, initialize the TUS upload with a POST

POST /db/container/foobar/@tusupload/file

Upload file

Example request

```
POST /db/container/foobar/@tusupload/file HTTP/1.1
Authorization: Basic cm9vdDpyb290
UPLOAD-LENGTH: 2097152
TUS-RESUMABLE: 1
```

Example response

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

Request Headers

- Authorization – Required token to authenticate

Status Codes

- 200 OK – no error
- 401 Unauthorized – Invalid Auth code
- 500 Internal Server Error – Error processing request

Next, upload the chunks(here we're doing chunks of 1MB):

PATCH /db/container/foobar/@tusupload/file

Upload file

Example request

```
PATCH /db/container/foobar/@tusupload/file HTTP/1.1
Authorization: Basic cm9vdDpyb290
Upload-Offset: 0
TUS-RESUMABLE: 1
CONTENT-LENGTH: 1048576

< binary data >
```

Example response

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

Request Headers

- Authorization – Required token to authenticate

Status Codes

- 200 OK – no error
- 401 Unauthorized – Invalid Auth code
- 500 Internal Server Error – Error processing request

And final chunk of 1MB:

PATCH /db/container/foobar/@tusupload/file

Upload file

Example request

```
PATCH /db/container/foobar/@tusupload/file HTTP/1.1
Authorization: Basic cm9vdDpyb290
Upload-Offset: 1048576
TUS-RESUMABLE: 1
CONTENT-LENGTH: 1048576

< binary data >
```

Example response

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

Request Headers

- Authorization – Required token to authenticate

Status Codes

- 200 OK – no error
- 401 Unauthorized – Invalid Auth code
- 500 Internal Server Error – Error processing request

Unknown upload size

Guillotina's TUS implementation has support for the `Upload-Defer-Length` header. This means you can upload files with an unknown final upload size.

In order to implement this correctly, you will need to provide the `Upload-Defer-Length: 1` header and value on the initial POST to start the TUS upload. You are then not required to provide the `UPLOAD-LENGTH` header.

Then, before or on your last chunk, provide a `UPLOAD-LENGTH` value to let TUS know the upload can not finish.

Simultaneous TUS uploads

Guillotina's TUS implementation also attempts to prevent simultaneous uploaders.

If two users attempt to start an upload on the same object + field at the same time, a 412 error will be thrown. Guillotina tracks upload activity to detect this. If there is no activity detected for 15 seconds with an unfinished TUS upload, no error is thrown.

To override this, send the `TUS-OVERRIDE-UPLOAD: 1` header.

Modifying permissions

The `@sharing` endpoint is available to inspect and modify permissions on an object.

`GET /db/container/foobar/@sharing`
Get sharing information

Example request

```
GET /db/container/foobar/@sharing HTTP/1.1
Authorization: Basic cm9vdDpyb290
```

Example response

```
HTTP/1.1 201 OK
Content-Type: application/json

{
  "local": {
      "roleperm": {},
      "prinperm": {},
      "prinrole": {
          "root": {
              "guillotina.Owner": "Allow"
          }
      }
  },
  "inherit": [
      {
          "@id": "http://localhost:8080/db/container",
          "roleperm": {},
          "prinperm": {},
          "prinrole": {
              "root": {
                  "guillotina.ContainerAdmin": "Allow",
                  "guillotina.Owner": "Allow"
              }
          }
      }
  ]
}
```

Request Headers

- Authorization – Required token to authenticate

Status Codes

- 200 OK – no error

- 401 Unauthorized – Invalid Auth code
- 500 Internal Server Error – Error processing request

To modify, we use the same endpoint but with a `POST`.

`POST /db/container/foobar/@sharing`

Add local permissions

Example request

```
POST /db/container/foobar/@sharing HTTP/1.1
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
  "prinperm": [
    {
      "principal": "foobar",
      "permission": "guillotina.ModifyContent",
      "setting": "Allow"
    }
  ]
}
```

Example response

```
HTTP/1.1 201 OK
Content-Type: application/json

{}
```

Request Headers

- Authorization – Required token to authenticate

Status Codes

- 200 OK – no error
- 401 Unauthorized – Invalid Auth code
- 500 Internal Server Error – Error processing request

There are three types of permission settings you can modify:

- prinperm: principal + permission
- prinrole: principal + role
- roleperm: role + permission

Each change can use the following settings:

- Allow : you set it on the resource and the children will inherit
- Deny : you set in on the resource and the children will inherit
- AllowSingle : you set in on the resource and the children will not inherit
- Unset : you remove the setting

Exploring the API with Swagger

In the previous step, we installed `guillotina_swagger`. With Swagger, we can inspect any context and explore the API.

Visit `http://localhost:8080/@docs`

click the `Authorize` button

The `Base API Endpoint` setting is what the current context is that you're exploring on. If you create content at `/db/container/foobar` and want to explore that content's API, you should change the URL. Different content types will have different services available.

References

- REST API
- Behaviors
- Security

1.7.6 AsyncIO

Python's asyncio library allows you to run single threaded "concurrent" code using coroutines inside an event loop.

The event loop is designed for I/O over sockets and other resources, it is especially good for working with client/server network connections.

Python >= 3.4(best features and performance in 3.6)

Explanation

Benefits

The event loop allows you to handle a larger number of network connections at once.

No network blocks, so you can have long running connections with very little performance impact (HTML5 sockets for example).

How web servers are typically designed

- (Pyramid, Flash, Plone, etc)
- Processes X Threads = Total number of concurrent connections that can be handled at once.
- Client makes a request to web server, request is assigned thread, thread handle request and sends response
- If no threads available, request is blocked, waiting for an open thread
- Threads are expensive (CPU), Processes are expensive on RAM

How it works with AsyncIO

- All requests are thrown on thread loop
- Since we don't block on network traffic, we can juggle many requests at the same time
- Modern web application servers connect with many different services that can potentially block on network traffic — BAD
- Limiting factor is maxed out CPU, not costly thread switching between requests — GOOD

Where is network traffic used?

- Web Client/App Server
- App Server/Database
- App Server/Caching(redis)
- App Server/OAUTH
- App Server/Cloud storage
- App Server/APIs(gdrive, m$, slack, etc)

Implementation details

In order to benefit, the whole stack needs to be asyncio-aware.

Anywhere in your application server that is not and does network traffic WILL BLOCK all other connections while it is doing its network traffic (example: using the `requests` library instead of `aiohttp`)

Basics

Get active event loop or create new one

Run coroutine inside event loop with `asyncio.run_until_complete`

```
import asyncio


async def hello():
    print('hi')
```

```
event_loop = asyncio.get_event_loop()
event_loop.run_until_complete(hello())
```

Basics(2)

`asyncio.run_until_complete` automatically wraps your coroutine into a Future object and waits for it to finish.

`asyncio.ensure_future` will wrap a coroutine in a future and return it to you

So you can schedule multiple coroutines that can run at the same time

```
import asyncio


async def hello1():
    await asyncio.sleep(0.5)
    print('hi 1')


async def hello2():
    print('hi 2')


event_loop = asyncio.get_event_loop()
future1 = asyncio.ensure_future(hello1(), loop=event_loop)
future2 = asyncio.ensure_future(hello2(), loop=event_loop)
event_loop.run_until_complete(future2)
event_loop.run_until_complete(future1)
```

Long running tasks

You can also schedule long running tasks on the event loop.

The tasks can run forever. . .

"Task" objects are the same as "Future" objects(well, close)

```
import asyncio
import random


async def hello_many():
    while True:
        number = random.randint(0, 3)
        await asyncio.sleep(number)
        print('Hello {}'.format(number))


event_loop = asyncio.get_event_loop()
task = asyncio.Task(hello_many())
print('task running now...')
event_loop.run_until_complete(asyncio.sleep(10))
print('we waited 10 seconds')
```

```
task.cancel()
print('task cancelled')
```

ALL YOUR ASYNC BELONGS TO US

gotcha

If you want part of your code to be async(say a function), the complete stack of the caller must be async and running on the event loop.

```
import asyncio


async def print_foobar1():
    print('foobar1')


async def print_foobar2():
    print('foobar2')


async def foobar():
    await print_foobar1()
    print_foobar2()  # won't work, never awaited


event_loop = asyncio.get_event_loop()
event_loop.run_until_complete(foobar())
print_foobar1()  # won't work, never awaited
# await print_foobar1()  # error, not running in event loop
```

"multi" processing

AsyncIO isn't really multiprocessing but it gives you the illusion of it.

A simple example can be shown with the `asyncio.gather` function.

```
import asyncio
import aiohttp


async def download_url(url):
    async with aiohttp.ClientSession() as session:
        resp = await session.get(url)
        text = await resp.text()
        print(f'Downloaded {url}, size {len(text)}')


event_loop = asyncio.get_event_loop()
event_loop.run_until_complete(asyncio.gather(
    download_url('https://www.google.com'),
    download_url('https://www.facebook.com'),
    download_url('https://www.twitter.com'),
    download_url('https://www.stackoverflow.com')
))
```

asyncio loops

Using `yield` with loops allows you to "give up" execution on every iteration of the loop.

```
import asyncio


async def yielding():
    for idx in range(5):
        print(f'Before yield {idx}')
        yield


async def foobar2():
    async for idx in yielding():
        print(f"Yay, I've been yield'd {idx}")


event_loop = asyncio.get_event_loop()
event_loop.run_until_complete(foobar2())
```

Scheduling

`loop.call_later`: arrange to call on a delay `loop.call_at`: arrange function to be called at specified time

Executors

An executor is available to use when you have non-async code that needs to be made async.

A typical executor is a thread executor. This means, anything you run in an executor is being thrown in a thread to run.

It's worse to have non-async code than to use thread executors.

Executors are also good for CPU bound code.

```
import asyncio
import requests
import concurrent.futures


def download_url(url):
    resp = requests.get(url)
    text = resp.content
    print(f'Downloaded {url}, size {len(text)}')


async def foobar():
    print('foobar')


executor = concurrent.futures.ThreadPoolExecutor(max_workers=5)

event_loop = asyncio.get_event_loop()
event_loop.run_until_complete(asyncio.gather(
    event_loop.run_in_executor(executor, download_url, 'https://www.google.
↪com'),
```

```
    event_loop.run_in_executor(executor, download_url, 'https://www.facebook.
↪com'),
    event_loop.run_in_executor(executor, download_url, 'https://www.twitter.
↪com'),
    event_loop.run_in_executor(executor, download_url, 'https://www.
↪stackoverflow.com'),
    foobar()
))
```

Subprocess

Python also provides a very neat asyncio subprocess module.

```
import asyncio


async def run_cmd(cmd):
    print(f'Executing: {" ".join(cmd)}')
    process = await asyncio.create_subprocess_exec(*cmd, stdout=asyncio.
↪subprocess.PIPE)
    out, error = await process.communicate()
    print(out.decode('utf8'))


event_loop = asyncio.get_event_loop()
event_loop.run_until_complete(asyncio.gather(
    run_cmd(['sleep', '1']),
    run_cmd(['echo', 'hello'])
))
```

1.7.7 Extending

In our training, we'll be working on creating a simple chat application.

To extend Guillotina, we need to write a Python package.

Let's start by using the cookiecutter to bootstrap an application for us.

```
g create --template=application
```

Follow the prompts and name your application `guillotina_chat`.

Then,

```
cd guillotina_chat
python setup.py develop
```

Configuration

All application extension configuration is defined with Guillotina's `configure` module and the `app_settings` object.

Defining content types, behaviors, services, etc all require the use of the `configure` module. Guillotina reads all the registered configuration in code for each install application and loads it.

app_settings

Guillotina also provides a global `app_settings` object::

```
from guillotina import app_settings
```

This object contains all the settings from your `config.yaml` file as well as any additional configuration settings defined in addons.

`app_settings` has an order of precedence it will use pick settings from:

- guillotina's default settings
- each application in order it is defined can override default guillotina settings
- config.yaml takes final precedence over all configuration

`app_settings` has an extra key '**file**' that contains the path of the configuration file, allowing relative paths to be used in an application settings.

Content types

For chatting, we'll need a content type for conversations and messages.

Create a `content.py` file in your application and create the content types.

```
from guillotina import configure, content, Interface, schema


class IConversation(Interface):

    users = schema.List(
        value_type=schema.TextLine()
    )


@configure.contenttype(
    type_name="Conversation",
    schema=IConversation,
    behaviors=["guillotina.behaviors.dublincore.IDublinCore"],
    allowed_types=['Message'])
class Conversation(content.Folder):
    pass


class IMessage(Interface):
    text = schema.Text(required=True)


@configure.contenttype(
    type_name="Message",
    schema=IMessage,
    behaviors=[
        "guillotina.behaviors.dublincore.IDublinCore",
        "guillotina.behaviors.attachment.IAttachment"
    ])
class Message(content.Item):
    pass
```

In order for Guillotina to detect your configuration, you'll need to add a scan call inside your `includeme` function in the `__init__.py` file.

```
configure.scan('guillotina_chat.content')
```

Test it out

Open up Postman and test creating a conversation and message instead of it.

Install an addons

Guillotina differentiates `applications` from `addons`.

An application is a python package you install into your environment and add to your list of applications in the configuration file.

Addons on the otherhand are when you want to perform installation logic into a container.

Define addon

To define an addon for Guillotina, we use the `@configure.addon` decorator in the `install.py` file.

For our case, we want to create a Folder with all our conversations with some default permissions.

```
from guillotina import configure
from guillotina.addons import Addon
from guillotina.content import create_content_in_container
from guillotina.interfaces import IRolePermissionManager


@configure.addon(
    name="guillotina_chat",
    title="Guillotina server application python project")
class ManageAddon(Addon):

    @classmethod
    async def install(cls, container, request):
        if not await container.async_contains('conversations'):
            conversations = await create_content_in_container(
                container, 'Folder', 'conversations',
                id='conversations', creators=('root',),
                contributors=('root',))
            roleperm = IRolePermissionManager(conversations)
            roleperm.grant_permission_to_role(
                'guillotina.AddContent', 'guillotina.Member')
            roleperm.grant_permission_to_role(
                'guillotina.AccessContent', 'guillotina.Member')

    @classmethod
    async def uninstall(cls, container, request):
        registry = request.container_settings  # noqa
        # uninstall logic here...
```

Testing

Then, using Postman, do a `POST` request to the `@addons` endpoint:

```
{"id": "guillotina_chat"}
```

Permissions/Role

Permissions are defined in your application code.

For our app, we'll create roles that users are granted inside a conversation.

Add the following inside your `__init__.py` file.

```
configure.role("guillotina_chat.ConversationParticipant",
               "Conversation Participant",
               "Users that are part of a conversation", False)
configure.grant(
    permission="guillotina.ViewContent",
    role="guillotina_chat.ConversationParticipant")
configure.grant(
    permission="guillotina.AccessContent",
    role="guillotina_chat.ConversationParticipant")
configure.grant(
    permission="guillotina.AddContent",
    role="guillotina_chat.ConversationParticipant")
```

Event subscribers

Events in Guillotina are heavily influenced from zope events with the caveat in that we support async event handlers.

For our chat application, we want to make sure every user that is part of a conversation has permission to add new messages and view other messages.

A simple way to do this is with an event handler that modifies permissions.

A an `subscribers.py` file inside your application.

```
from guillotina import configure
from guillotina.interfaces import IObjectAddedEvent, IPrincipalRoleManager
from guillotina.utils import get_authenticated_user_id, get_current_request
from guillotina_chat.content import IConversation


@configure.subscriber(for_=(IConversation, IObjectAddedEvent))
async def container_added(conversation, event):
    user_id = get_authenticated_user_id(get_current_request())
    if user_id not in conversation.users:
        conversation.users.append(user_id)

    manager = IPrincipalRoleManager(conversation)
    for user in conversation.users or []:
        manager.assign_role_to_principal(
            'guillotina_chat.ConversationParticipant', user)
```

In order for Guillotina to detect your configuration, you'll need to add a scan call inside your `includeme` function in the `__init__.py` file.

```
configure.scan('guillotina_chat.subscribers')
```

Test it out

Using Postman, add a Conversation and then a Message to that conversation and then use the `@sharing` endpoint to inspect the assigned permissions.

Users

Guillotina does not come with any user provider OOTB and is designed to be plugged in with other services.

However, there is a simple provider that stores user data in the database called `guillotina_dbusers` that we will use for the purpose of our training.

Install guillotina_dbusers

Just use pip

```
pip install guillotina_dbusers
```

And add the `guillotina_dbusers` to the list of applications in your `config.yaml`. Also make sure you are not overriding the `auth_user_identifiers` configuration value in your `config.yaml` as `guillotina_dbusers` uses that to work.

After you restart guillotina, you can also install `dbusers` into your container using the `@addons` endpoint:

```
POST /db/container/@addons
{
  "id": "dbusers"
}
```

Add users

Creating users is just creating a user object.

```
POST /db/container/users
{
  "@type": "User", "username": "foobar", "email": "foo@bar.com", "password":
↪"foobar"
}
```

Logging in can be done with the `@login` endpoint which returns a jwt token.

```
POST /db/container/@login
{
  "username": "foobar", "password": "foobar"
}
```

Then, future requests are done with a `Bearer` token with the jwt token. For example, to create a conversation with your user:

```
POST /db/container/conversations
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
↪eyJleHAiOjE1MDgwMTU0OTcsImlkIjoiZm9vYmFyIn0.
↪vC6HHuLmcf8d1I7RpOTxAeHQDfMRjsOoBS-xH4Q1sdw
{
  "@type": "Conversation",
  "title": "New convo with foobar2",
  "users": ["foobar", "foobar2"]
}
```

Serialize content

Guillotina provides default serializations for content. It provides mechanisms for giving full content serialization of interfaces and behaviors as well as summary serializations that show in listings.

For customize a serialization for a type, you need to provide a multi adapter for the `IResourceSerializeToJsonSummary` or `IResourceSerializeToJson` interfaces.

For our use-case, we want to make sure to include the `creation_date` and some other data in the summary serialization of conversations and messages so we can get all the info we need for our application without doing full objet serialization.

Defining a custom serialization

Let's define these serializers in a in a file named `serialize.py`.

```
from guillotina import configure
from guillotina.interfaces import IResourceSerializeToJsonSummary
from guillotina.json.serialize_content import DefaultJSONSummarySerializer
from guillotina.utils import get_owners
from guillotina_chat.content import IConversation, IMessage
from zope.interface import Interface


@configure.adapter(
    for_=(IConversation, Interface),
    provides=IResourceSerializeToJsonSummary)
class ConversationJSONSummarySerializer(DefaultJSONSummarySerializer):
    async def __call__(self):
        data = await super().__call__()
        data.update({
            'creation_date': self.context.creation_date,
            'title': self.context.title,
            'users': self.context.users
        })
        return data


@configure.adapter(
    for_=(IMessage, Interface),
    provides=IResourceSerializeToJsonSummary)
class MessageJSONSummarySerializer(DefaultJSONSummarySerializer):
    async def __call__(self):
```

```
        data = await super().__call__()
        data.update({
            'creation_date': self.context.creation_date,
            'text': self.context.text,
            'author': get_owners(self.context)[0]
        })
        return data
```

And make sure to add the scan.

```
configure.scan('guillotina_chat.serialize')
```

Services

Services are synonymous with what other frameworks might call `endpoints` or `views`.

For the sake of our application, let's use services for getting a user's most recent conversations and messages for a conversation.

Creating the services

We'll name our endpoints `@get-conversations` and `@get-messages` and put them in a file named `services.py`.

```
from guillotina import configure
from guillotina.component import get_multi_adapter
from guillotina.interfaces import IContainer, IResourceSerializeToJsonSummary
from guillotina.utils import get_authenticated_user_id
from guillotina_chat.content import IConversation


@configure.service(for_=IContainer, name='@get-conversations',
                   permission='guillotina.Authenticated')
async def get_conversations(context, request):
    results = []
    conversations = await context.async_get('conversations')
    user_id = get_authenticated_user_id(request)
    async for conversation in conversations.async_values():
        if user_id in getattr(conversation, 'users', []):
            summary = await get_multi_adapter(
                (conversation, request),
                IResourceSerializeToJsonSummary)()
            results.append(summary)
    results = sorted(results, key=lambda conv: conv['creation_date'])
    return results


@configure.service(for_=IConversation, name='@get-messages',
                   permission='guillotina.AccessContent')
async def get_messages(context, request):
    results = []
    async for message in context.async_values():
        summary = await get_multi_adapter(
            (message, request),
            IResourceSerializeToJsonSummary)()
```

```
        results.append(summary)
    results = sorted(results, key=lambda mes: mes['creation_date'])
    return results
```

And make sure to add the scan.

```
configure.scan('guillotina_chat.services')
```

Async Utilities

An async utility is a utility that run persistently on the asyncio event loop. It is useful for long running tasks.

For our training, we're going to use an async utility with a queue to send messages to logged in users.

Create a `utility.py` file and put the following code in it.

```
from guillotina import configure
from guillotina.async_util import IAsyncUtility
from guillotina.component import get_multi_adapter
from guillotina.interfaces import IResourceSerializeToJsonSummary
from guillotina.renderers import GuillotinaJSONEncoder
from guillotina.utils import get_authenticated_user_id, get_current_request

import asyncio
import json
import logging

logger = logging.getLogger('guillotina_chat')


class IMessageSender(IAsyncUtility):
    pass


@configure.utility(provides=IMessageSender)
class MessageSenderUtility:

    def __init__(self, settings=None, loop=None):
        self._loop = loop
        self._settings = {}
        self._webservices = []

    def register_ws(self, ws, request):
        ws.user_id = get_authenticated_user_id(request)
        self._webservices.append(ws)

    def unregister_ws(self, ws):
        self._webservices.remove(ws)

    async def send_message(self, message):
        summary = await get_multi_adapter(
            (message, get_current_request()),
            IResourceSerializeToJsonSummary)()
        await self._queue.put((message, summary))

    async def initialize(self, app=None):
```

```
        self._queue = asyncio.Queue()

        while True:
            try:
                message, summary = await self._queue.get()
                for user_id in message.__parent__.users:
                    for ws in self._webservices:
                        if ws.user_id == user_id:
                            await ws.send_str(json.dumps(
                                summary, cls=GuillotinaJSONEncoder))
            except Exception:
                logger.warn(
                    'Error sending message',
                    exc_info=True)
                await asyncio.sleep(1)
```

Async utilities must implement a `initialize` method and performs the async task. In our case, it is creating a queue and waiting to process messages in the queue.

For us, we will send messages to registered websockets.

Make sure, like all other configured moduels, to ensure this file is scanned by the packages `__init__.py` file.

Sending messages

We'll need to add another event subscriber to the `subscribers.py` file in order for the utility to know to send out new messages to registered web serveices. So your `utility.py` file will now look like:

```
from guillotina import configure
from guillotina.component import get_utility
from guillotina.interfaces import IObjectAddedEvent, IPrincipalRoleManager
from guillotina.utils import get_authenticated_user_id, get_current_request
from guillotina_chat.content import IConversation, IMessage
from guillotina_chat.utility import IMessageSender


@configure.subscriber(for_=(IConversation, IObjectAddedEvent))
async def container_added(conversation, event):
    user_id = get_authenticated_user_id(get_current_request())
    if user_id not in conversation.users:
        conversation.users.append(user_id)

    manager = IPrincipalRoleManager(conversation)
    for user in conversation.users or []:
        manager.assign_role_to_principal(
            'guillotina_chat.ConversationParticipant', user)


@configure.subscriber(for_=(IMessage, IObjectAddedEvent))
async def message_added(message, event):
    utility = get_utility(IMessageSender)
    await utility.send_message(message)
```

Websockets

Websocket support is built-in to Guillotina.

It's as simple as using an `aiohttp` websocket in a service.

Create a `ws.py` file and put the following code in:

```
from aiohttp import web
from guillotina import configure
from guillotina.component import get_utility
from guillotina.interfaces import IContainer
from guillotina.transactions import get_tm
from guillotina_chat.utility import IMessageSender

import aiohttp
import logging

logger = logging.getLogger('guillotina_chat')


@configure.service(
    context=IContainer, method='GET',
    permission='guillotina.AccessContent', name='@conversate')
async def ws_conversate(context, request):
    ws = web.WebSocketResponse()
    utility = get_utility(IMessageSender)
    utility.register_ws(ws, request)

    tm = get_tm(request)
    await tm.abort(request)
    await ws.prepare(request)

    async for msg in ws:
        if msg.tp == aiohttp.WSMsgType.text:
            # ws does not receive any messages, just sends
            pass
        elif msg.tp == aiohttp.WSMsgType.error:
            logger.debug('ws connection closed with exception {0:s}'
                         .format(ws.exception()))

    logger.debug('websocket connection closed')
    utility.unregister_ws(ws)

    return {}
```

Here, we use the `utility = get_utility(IMessageSender)` to get our async utility we defined previously. Then we register our webservice with `utility.register_ws(ws, request)`.

Our web service is simple because we do not need to receive any messages and the async utility sends out the messages.

Using websockets

In order to use websockets, you need to request a websocket token first.

```
GET /db/container/@wstoken
Authentication Bearer <jwt token>
```

Then, use this token to generate a webservice URL(JavaScript example here):

```
var url = 'ws://localhost:8080/db/container/@conversate?ws_token=' + ws_
↪token;
SOCKET = new WebSocket(url);
SOCKET.onopen = function(e){
};
SOCKET.onmessage = function(msg){
  var data = JSON.parse(msg.data);
};
SOCKET.onclose = function(e){
};

SOCKET.onerror = function(e){
};
```

Static files

To pull this all together, we'll create our web application that uses the api to provide a very simple chat experience.

Copy the following files into a new folder `static` in your application:

- `chat.js`.
- `index.html`.
- `main.css`.

Configure

Then, we'll setup Guillotina to serve the folder.

Modify your `config.yaml` file to add:

```
static:
  status: ./static
```

JS Applications

You can also serve the static files in a way where it works with JavaScript applications that need to be able to translate URLs from something other than root.

```
jsapps:
  static: ./static
```

With this configuration any request to a url like `http://localhost:8080/static/foo/bar` will serve files from `http://localhost:8080/static`.

1.7.8 Commands

Guillotina comes with a great set of commands you can use to help debug and inspect your install.

We've already gone through the `serve`, `create` and `testdata` commands so we'll now cover `shell` and `run`.

Make sure to also read the commands reference in the docs to learn how to create your own commands.

Shell

The `shell` command allows you to get an interactive prompt into guillotina.

From here, you can connect to the database, accees objects and commit new data.

```
g -c config.yml shell
```

Then, to connect to the database and get your container object.

```
txn = await use_db('db')
container = await use_container('container')
```

From here, you can access objects:

```
conversations = await container.async_get('conversations')
await conversations.async_keys()
```

Run

The `run` command allows you to run a python script directly.

```
g -c config.yaml run --script=path/to/script.py
```

In order for you to utilize this, the script must have an async function named `run` inside it.

```
async def run(container):
    pass
```

1.7.9 Kitchen Sink

This part of the training material is going to talk about the guillotina_kitchensink repository.

This repository gives you a working configuration and install of:

- guillotina_dbusers: Store and manage users on the database
- guillotina_elasticsearch: Index on content in elasticsearch
- guillotina_swagger: Access site swagger definition at http://localhost:8080/@docs
- guillotina_rediscache: Cache db objects in redis

The components it runs as part of the docker compose file are:

- postgresql
- elasticsearch
- redis

First off, start by cloning the repository and starting it.

```
git clone https://github.com/guillotinaweb/guillotina_kitchensink.git
cd guillotina_kitchensink
docker-compose -f docker-compose.yaml run --rm --service-ports guillotina
```

Add some content using Postman and then let's do an elasticsearch query:

```
POST /db/container/@search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "title": "foobar"
          }
        }
      ]
    }
  }
}
```

CHAPTER 2

What is Guillotina like?

Example configuration:

```
---
applications:
- guillotina_dbusers
- myapp
databases:
- db:
    storage: postgresql
    dsn:
      scheme: postgres
      dbname: guillotina
      user: postgres
      host: localhost
      password: ''
      port: 5432
port: 8080
root_user:
  password: root
```

Example service:

```
from guillotina import configure

@configure.service(name='@foobar')
async def foobar(context, request):
    return {"foo": "bar"}
```

Example content type:

```
from guillotina import configure
from guillotina import content
from guillotina import Interface
from guillotina import schema
```

```
class IMyType(Interface):
    foobar = schema.TextLine()

@configure.contenttype(
    type_name="MyType",
    schema=IMyType,
    behaviors=["guillotina.behaviors.dublincore.IDublinCore"])
class Foobar(content.Item):
    pass
```

Example usage:

POST /db/container

Create MyType

Example request

```
POST /db/container HTTP/1.1
Accept: application/json
Content-Type: application/json
Authorization: Basic cm9vdDpyb290

{
  "@type": "MyType",
  "id": "foobar",
  "foobar": "foobar"
}
```

Example response

```
HTTP/1.1 201 OK
Content-Type: application/json
```

Request Headers

- Authorization – Required token to authenticate

Status Codes

- 201 Created – no error
- 401 Unauthorized – Invalid Auth code
- 500 Internal Server Error – Error processing request

GET /db/container/foobar/@foobar

Get MyType

Example request

```
GET /db/container/foobar HTTP/1.1
Accept: application/json
Authorization: Basic cm9vdDpyb290
```

Example response

```
HTTP/1.1 201 OK
Content-Type: application/json

{"foo": "bar"}
```

Request Headers

- Authorization – Required token to authenticate

Status Codes

- 200 OK – no error
- 401 Unauthorized – Invalid Auth code
- 500 Internal Server Error – Error processing request

/db