



# WORKING WITH REST APIS

Pieter Frenssen

# ABOUT ME

Pieter Frenssen / @pfrenssen



**REST?**

**REPRESENTATIONAL**

**STATE**

**TRANSFER**

# WHAT IS REST?

- Software architecture
- Resource based (using URIs)
- Communication over HTTP
- Introduced by Roy Fielding in 2000

[http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)

# WHY USE REST?

- Standard CRUD implementation
- Easy to use
- Widely supported
- Built in to popular JS frameworks

# CRUD OVER HTTP

- Create: POST
- Read: GET
- Update: PATCH (or PUT)
- Delete: DELETE

# HTTP REST CLIENT

- Desktop application
- Browser plugin
- IDE plugin
- Command line client
- Curl

# BROWSER PLUGINS

## POSTMAN FOR CHROMIUM

<https://www.getpostman.com/>



RunnerImport

BuilderTeam Library

OFFLINESign In

Search

HistoryCollections

Today

GET

http://drupal.local/node/2?format=json

GET

http://drupal.local/node/2?format=hal\_json

GET

http://drupal.local/rest/session/token

GET

http://drupal.local/node/2

GET

http://drupal.local

http://drupal.local/node/2

No environment

GET

http://drupal.local/node/2?format=json

Params

Send

Save

Authorization

Headers

Body

Pre-request Script

Tests

Generate Code

key

value

Bulk Edit

Presets

Body

Cookies

Headers (16)

Tests

Status: 200 OK

Time: 270 ms

Pretty

Raw

Preview

JSON

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

{

"nid": [

{

"value": "2"

}

],

"uuid": [

{

"value": "ee1f52f8-cf1c-4f93-bf2a-f99804878b79"

}

],

"vid": [

{

"value": "2"

}

]

}

# HTTPREQUESTER FOR FIREFOX

<https://addons.mozilla.org/firefox/addon/httprequester>

The screenshot displays the HTTPREQUESTER FOR FIREFOX interface, which is divided into several sections:

- REQUEST**: Contains a URL field with `http://drupal.local/node/2?_format=json`, a dropdown menu set to `GET`, and buttons for `Submit`, `GET`, `POST`, and `PUT`. Below these are buttons for `New request`, `Paste Request`, and `Authentication...`. A tabbed interface shows `Content to Send`, `Headers`, and `Parameters`. The `Parameters` tab is active, showing a table with one parameter: `_format` with value `json`. Buttons for `Add`, `Delete`, `Move Up`, and `Move Down` are present.
- RESPONSE**: Shows the result of the GET request: `GET on http://drupal.local/node/2?_format=json`. The status is `200 OK`. There are radio buttons for `Browser` and `Text` (selected), and a checked checkbox for `Pretty format`. A link `View raw transaction` is available. The response body is a JSON object: 

```
{  "nid": [    {      "value": "2"    }  ],  "uuid": [    {      "value": "ee1f52f8-cf1c-4f93-bf2a-f99804878b79"    }  ],  "vid": [    {      "value": "2"    }  ]}
```
- HEADERS**: Displays response headers in a table:

Header	Value
Date	Sun, 03 Apr 2016 11:05:11 GMT
Server	Apache/2.4.18 (Unix) PHP/7.0.4
X-Content-Type-Options	nosniff, nosniff
- HISTORY**: A table listing previous requests:

Request	Response	Date	Size	Time	
GET http://drupal.local/node/2?_format=json	200 OK	Apr 3 2016 - 2:05:11 PM	1070 B	73 ms	

Buttons for `Clear history`, `Copy to clipboard`, `Delete request`, and `Edit raw request...` are on the right.

# VIEWING JSON IN-BROWSER

- Chromium: [JSON Formatter](#)
- Firefox: [JSONView](#)

# REST API FIRST LOOK: DRUPAL.ORG

- <https://www.drupal.org/drupalorg/docs/api>
- Exposes projects, issues, users, nodes, comments, ...
- Read-only
- Anonymous access (fair use)
- Returns data in JSON and XML formats
- Supports pagination and sorting
- Built in Drupal 7 with the RESTful Web Services module

# AN EXAMPLE REQUEST

[https://www.drupal.org/api-d7/user.json?  
field\\_country=Belgium&sort=uid&limit=10](https://www.drupal.org/api-d7/user.json?field_country=Belgium&sort=uid&limit=10)

- A JSON object is returned
- Contains pagination data and a list of users
- User objects contain many fields
- Can be filtered and sorted by field
- Do all fields make sense?

# PROVIDING A REST API IN DRUPAL

- Why use Drupal?
- Designing a REST API
- REST modules
- Programmatically creating endpoints
- ~~Headless Drupal~~

# WHY USE DRUPAL?

- No need to write custom code
- Powerful data modeling UI
- Versioning
- Caching
- Pagination
- Flood control
- Sorting
- Filtering
- Awesome content authoring tools

# DESIGNING A REST API

- **GET /api/v1/books** - Lists books
- **GET /api/v1/books/123** - Retrieve a book
- **POST /api/v1/books** - Create a book
- **PATCH /api/v1/books/123** - Update a book
- **DELETE /api/v1/books/123** - Delete a book



# DESIGNING A REST API

- **GET /api/v1/books** - Lists books
- **GET /api/v1/books/123** - Retrieve a book
- **POST /api/v1/books** - Create a book
- **PATCH /api/v1/books/123** - Update a book
- **DELETE /api/v1/books/123** - Delete a book

## CHILD ENTITIES

- **GET /api/v1/books/123/chapters** - Lists chapters
- **GET /api/v1/books/123/chapters/456** - Retrieve a chapter
- **POST /api/v1/books/123/chapters** - Create a chapter
- **PATCH /api/v1/books/123/chapters/456** - Update a chapter
- **DELETE /api/v1/books/123/chapters/456** - Delete a chapter

# RETURN CORRECT HTTP RESPONSES

- **200 OK** - Successful GET or PATCH
- **201 Created** - Successful POST
- **204 No Content** - Successful DELETE
- **400 Bad Request** - Invalid data received
- **401 Unauthorized** - Not authenticated
- **403 Forbidden** - Successfully authenticated, no access
- **404 Not Found** - Resource does not exist
- **405 Method Not Allowed** - Successfully authenticated, wrong HTTP method
- **410 Gone** - API version is deprecated
- **422 Unprocessable Entity** - Validation error
- **429 Too Many Requests** - Exceeded rate limiting

# JSON OR XML?

## JSON

- Compact
- Easy to read
- Still reasonably hipster

# XML

- Totally enterprise

# REST RELATED MODULES

- RESTful Web Services
- REST UI
- HAL
- HTTP Basic Authentication
- Views
- Services
- JSON API

# RESTFUL WEB SERVICES

- Part of Drupal 8 core
- Depends on Serialization
- No user interface
- Pluggable formats (JSON, XML, ...)
- Pluggable authentication methods

# REST UI MODULE

- User interface for the REST module
- Supports all content entities
- Optional alternative to raw config



# USING REST UI

/admin/config/services/rest

## Enabled

RESOURCE NAME	PATH	DESCRIPTION	OPERATIONS
Content	/node/{node}	GET authentication: cookie formats: json  POST authentication: cookie formats: json	Edit ▾

## Disabled

RESOURCE NAME	PATH	DESCRIPTION	OPERATIONS
Action	/entity/action/{action}		Enable
Base field override	/entity/base_field_override/{base_field_override}		Enable
Block	/entity/block/{block}		Enable

# PERMISSIONS

- Available for every enabled entity
- Separate permission per HTTP method

PERMISSION	ANONYMOUS USER	AUTHENTICATED USER	ADMINISTRATOR
<b>RESTful Web Services</b>			
Access DELETE on <i>Content</i> resource	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Access GET on <i>Content</i> resource	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Access PATCH on <i>Content</i> resource	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Access POST on <i>Content</i> resource	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

# HAL

- Adds support for [Hypertext Application Language](#)
- Adds formats "application/hal+json" & "application/hal+xml"
- Provides location of itself and relations (\_links)
- Describes embedded resources (\_embedded)

# HAL LINKS

Always provide a link to yourself with **self**

```
"_links": {  
  "self": {  
    "href": "http://my.domain/users?page=3"  
  }  
}
```

# HAL LINKS

## Paging

```
"_links": {  
  "first": {  
    "href": "http://my.domain/users"  
  },  
  "prev": {  
    "href": "http://my.domain/users?page=2"  
  },  
  "next": {  
    "href": "http://my.domain/users?page=4"  
  },  
  "last": {  
    "href": "http://my.domain/users?page=133"  
  }  
}
```

# HAL EMBEDDED RESOURCES

```
"_embedded": {  
  "profile_pictures": [  
    {  
      "_links": {  
        "self": {  
          "href": "http://my.domain/files/123"  
        }  
      },  
    },  
  ],  
}
```

# HTTP BASIC AUTHENTICATION

- Small core module
- Allows to authenticate over HTTP Basic Auth
- **Authorization: Basic dXNlcjpwYXNz**
- Credentials are base 64 encoded: only use over HTTPS!

# VIEWS

## Create REST endpoint through UI

**Add view** ☆

[Home](#) » [Administration](#) » [Structure](#) » [Views](#)

**VIEW BASIC INFORMATION**  
**View name \***  
 Machine name: newest\_members [\[Edit\]](#)

**VIEW SETTINGS**  
**Show:**  **sorted by:**

**REST EXPORT SETTINGS**  
☒ Provide a REST export  
REST export path

**Save and edit** **Cancel**



# SELECT DATA TO EXPOSE

Display 'fields'

REST export: How should each row in this view be styled

For  
This rest\_export (override)

**Row**  
☐ Entity  
☒ Fields  
☐ Search results

You may also adjust the [settings](#) for the currently selected row style.

Apply (this display)

Cancel

# OUTPUT RAW DATA

Edit field settings

REST export: Row style options

FIELD	ALIAS	RAW OUTPUT
name	<input type="text" value="username"/>	<input checked="" type="checkbox"/>

Apply

Cancel

# FLESHING OUT THE DATA

Add more fields

**FIELDS**

Add ▼

User: Name
User: Created
User: Last access
User: Last login
User: Picture
User: Roles
User: User ID
User: UUID

# RESULT

/api/v1/members/newest

```
▼ [
  ▼ {
    "name": "drupal",
    "created": "1473495363",
    "access": "1473498708",
    "login": "1473495994",
    "user_picture": false,
    ▼ "roles": [
      "administrator"
    ],
    "uid": "1",
    "uuid": "8a5a823b-1292-4679-9164-9118e2f3cf07"
  }
]
```

# SERVICES

- Port of the D7 modlie
- Define end points
- Expose data

▼ <u>USER</u>		
DEFINITION	ENDPOINT	OPERATIONS
User: Retrieve	user/{user}	Configure ▼
User: Delete	user/{user}	Enable
User login	user/login	Enable
User logout	user/logout	Enable
User: Create	user	Configure ▼
User: View	user/{user}/view	Configure ▼
User: Update	user/{user}	Enable
User: Index	user	Configure ▼

# JSON API

- JSON formatter
- Follows best practices
- Provides resources and links

# PROGRAMMATICALLY CREATING AN ENDPOINT

- Create a Controller
- Return a JsonResponse object

# RETURNING A JSON RESPONSE

```
class RestApiController extends ControllerBase {  
  
    use Symfony\Component\HttpFoundation\JsonResponse;  
  
    public function getStats($date_range) {  
        $data = [  
            'unique_visitors' => 100,  
            'page_views' => 3440,  
            'date_range' => ['2016-03-15T08:14:45Z', '2016-04-15T08:14:45Z'],  
        ];  
        return new JsonResponse($data);  
    }  
}
```



# CONSUMING A REST SERVICE WITH DRUPAL

- No modules yet
- Easy to do programmatically

## EXAMPLE IMPLEMENTATION

- Use the Drupal.org REST API
- Retrieve 5 change records with Guzzle
- Display results in a block

# INJECTING GUZZLE

## Example implementation in a block

```
class ChangeRecordsBlock extends BlockBase implements ContainerFactoryPluginInterface {

    public function __construct(array $configuration, $plugin_id, $plugin_definition, Client $httpClient) {
        parent::__construct($configuration, $plugin_id, $plugin_definition);
        $this->httpClient = $httpClient;
    }

    public static function create(ContainerInterface $container, array $configuration, $plugin_id,
        return new static(
            $configuration,
            $plugin_id,
            $plugin_definition,
            $container->get('http_client')
        );
    }
}
```

# REQUESTING DATA FROM THE REST API

```
public function build() {
    $options = [
        'query' => [
            'type' => 'changenotice',
            'limit' => 5,
            'status' => 1,
            'sort' => 'created',
            'direction' => 'DESC',
        ],
    ];
    $result = $this->httpClient->get( 'https://www.drupal.org/api-d7/node.json' , $options);
    $change_records = json_decode($result->getBody())->list;
    // ...
}
```

# RETURNING THE DATA

```
public function build() {  
    // ...  
    $items = [];  
    foreach ($change_records as $change_record) {  
        $items[] = [  
            '#type' => 'link',  
            '#title' => $change_record->title,  
            '#url' => Url::fromUri($change_record->url),  
        ];  
    }  
    return [  
        '#theme' => 'item_list',  
        '#items' => $items,  
        '#list_type' => 'ol',  
    ];  
}
```

# THE RESULT

## Change records block

---

1. [New templates introduced for views listing, old ones deprecated](#)
2. [Menu links that point to nodes now reference the node UUID in normalized representation](#)
3. [Simplified Views listing page so it is consistent with other admin listings](#)
4. [Add security coverage indicators to project pages](#)
5. [Search box being moved to an icon in the top nav](#)

# MODELING REST DATA AS ENTITIES

- `json_decode()` returns a plain (meh) object
- Entities are the ideal match in Drupal
- Integrates well with everything
- Local caching
- Importing / exporting
- Requires the Serialization module

# HOW TO MODEL REST DATA

- Create custom entity representing REST data
- Inject the 'serializer' service
- Deserialize the REST data into an entity

```
$data = $this->httpClient->get( 'https://www.drupal.org/api-d7/node/2692565.json' );  
$serializer = \Drupal::service( 'serializer' );  
$entity = $serializer->deserialize($data, 'Drupal\change_records\Entity\ChangeRecord', 'json');
```



# RESOURCES

- [RESTful Web Services module documentation](#)
- [REST: top priorities for Drupal 8.4.x](#)