Doctrine in Apigility Book Documentation

Release latest

Apr 03, 2018

Table of Contents

Welcome to the Doctrine in Apigility Book. This documentation will guide you from the zfcampus/zf-apigility-skeleton to a fully Richardson Maturity Model Level 3 HATEOAS API built with the Doctrine Object Relational Mapper.

CHAPTER 1

About

The author, Tom H Anderson, was the original author of zfcampus/zf-apigility-doctrine and has created many supporting libraries for Doctrine in Apigility. He contributed much to the early development of Apigility and continues to give to open source in PHP.

As soon as Apigility was announced developers were asking for Doctrine support though none of us really knew what that ment. What was created serves as a strong platform for an ORM based API.

This book details a strategy for implementing Doctrine in Apigility. It does not dictate the method for using Doctrine in Apigility. Instead, this book gives detailed instructions for a strategy for using Doctrine in Apigility. YMMV.

CHAPTER 2

Introduction

The landscape of API strategies grows every day. In the field of PHP there are strategies from simple REST-only resources to fully Richardson Maturity Model Level 3 API engines. Apigility falls into the Level 3 category. As a contributing author of Apigility and the primary author of Doctrine in Apigility I've designed a method to serve ORM data though an API in a white-gloved approach to handling the data.

What do I mean by a "white-gloved approach"? Thanks to the large number of open source libraries for Zend Framework and Apigility the direct manipulation of data is not usually necessary. Certainly there will be times you'll want to add a user to an entity before it is persisted but the majority of validation and filtering is handled inside Apigility. Doctrine is a powerful ORM and the integration with Apigility is very clean when implemented correctly. This book will instruct you how to do just that.

2.1 When should you use Doctrine in Apigility?

When you have a Doctrine project with properly mapped associations (metadata) between entities Doctrine in Apigility will give you a powerful head-start toward building an API. Correct metadata is absolutly core to building an API with this tool. To help design your ORM Skipper is strongly recommended. See Entity Relationship Diagramming with Skipper

You can use Doctrine in Apigility to serve pieces of your schema by filtering with hydrator strategies or you can serve your entire schema in an "Open-schema API" where relationships between entities are fully explored in the HAL `_embedded` data.

If you're familiar with the benefits of ORM and will use it in your project and you require a fully-fledged API engine then this API strategy may what you're looking for.

2.2 What makes Doctrine in Apigility different?

Because Doctrine is an object relational mapper the relationships between entities are part of the Doctrine metadata. So when a Doctrine entity is extracted with a hydrator the relationships are pulled from the entity too and included in

the HAL response as `_embedded` data and naturally create a HATEOAS response. By using hydration strategies you may include just a link to the canonical resource or embed the entire resource and in turn embed its resources.

Consider this response to a request for an Artist


```
{
    "id": 1,
    "name": "Soft Cell",
    "_embedded": {
        "album": [
            {
                "id": 1,
                "name": "Non-Stop Erotic Cabaret",
                "_embedded": {
                    "artist": {
                        "_links": {
                            "self": "https://api/artist/1"
                        }
                    },
                    "song": {
                        "_links": {
                            "self": "https://api/song?filter%5B0%5D%5Bfield%5D=album&
↪filter%5B0%5D%5Btype%5D=eq&filter%5B0%5D%5Bvalue%5D=1"
                        }
                    }
                },
                "_links": {
                    "self": "https://api/album/1"
                }
            }
        ],
    },
    "_links": {
        "self": "https://api/artist/1"
    }
}
```


The `album` collection for this artist is returned because a `CollectionExtract` hydration strategy was used. Inside the `album`, instead of including every `song`, just a link to the collection filtered by the `album` is included. This way a consumer of the API is directed how to fetch the data when it is not included.

Each album entry has an embedded `artist` but this would cause a cyclic reference so an `EntityLink` hydrator is used to just give a link to the `artist` from within an `album`. This is common when using the `CollectionExtract` hydrator.

Within each API response when using Doctrine in Apigility there will never be a dead-end. Any reference to an entity or collection will be handled by a hydrator thereby making the API fully implement HATEOAS.

CHAPTER 3

Design Summary

This summary gives an overview of a very good design pattern for using Doctrine in Apigility. Each section will have it's own page and is included here to give the developer a birds-eye view of the pattern.

3.1 Files

Doctrine in Apigility creates two files when a new resource is assigned to an API. These are

```
<?php
[Resource]Collection.php
[Resource]Resource.php
```

As an example for an Artist resource these files full path will be

```
<?php
module/DbApi/src/DbApi/V1/Rest/Artist/ArtistCollection.php
module/DbApi/src/DbApi/V1/Rest/Artist/ArtistResource.php
```

There should never be a need to modify these files. Let me repeat, these files are not intended to override the ancestor objects. They exist here as part of the depenency injection strategy Doctrine in Apigility uses. Again, **DO NOT** modify these files.

3.2 Security

The design for Doctrine in Apigility expects a two-layered security strategy. The first layer is ACL (or RBAC if you prefer and are dedicated) and the second layer is Query Providers. ACL Authorization is handled by Apigility and Query Providers are handled by Doctrine in Apigility.

3.3 ACL Security

Doctrine in Apigility expects you to implement the Authorization created with zfcampus/zf-mvc-auth for your project. This probably means implementing ACL in your application and assigning roles to the differnet HTTP verbs each role can access. For instance a DELETE verb may only be available to an administrator.

3.4 Query Provider Security

For any resource where the access to the resource is limited a Query Provider should be created. Query Providers are small classes which return a Doctrine QueryBuilder object. By default the QueryBuilder contains only the entity assigned to the resource the user is requesting. By extending the QueryBuilder with filters and joins the query will return filtered data based on a particular user or security permission of the user the QueryBuilder, when ran, will produce SQL that adds new security to the resource.

For instance, if a UserResource is secured by ACL to only USER roles but each user can only PATCH to their own entity the Query Provider may read

```
<?php
final class UserPatch extends AbstractQueryProvider
{
    public function createQuery(ResourceEvent $event, $entityClass, $parameters)
    {
        $queryBuilder = $this->getObjectManager()->createQueryBuilder();
        $queryBuilder
            ->select('row')
            ->from($entityClass, 'row')
            ->andWhere($queryBuilder->expr()->eq('row.user', ':user'))
            ->setParameter('user', $this->getAuthentication()->getIdentity()->
↪getUser())
            ;

       return $queryBuilder;
    }
}
```

Now when the QueryBuilder is ran inside the DoctrineResource the id for the user passed to the patch will be appended to the QueryBuilder. If the id does not belong to the current user then the QueryBuilder will return no results and a 404 will be thrown to the user trying to edit a record which is not theirs.

More complicated examples **rely on your metadata being complete**. If your metadata defines joins to and from every join (that is, to an inverse and to a owner entity for every relationship) you can add complicated joins to your Query Provider

```
<?php
$queryBuilder
    ->innerJoin('row.performance', 'performance')
    ->innerJoin('performance.artist', 'artist')
    ->innerJoin('artist.artistGroup', 'artistGroup')
    ->andWhere($queryBuilder->expr()->isMemberOf(':user', 'artistGroup.user'))
    ->setParameter('user', $this->getAuthentication()->getIdentity()->getUser())
    ;
```

3.5 Hydrators

If you're unfamiliar with hydrators read Zend Framework's manual on Hydrators then read Doctrine's manual on Hydrators then read phpro/zf-doctrine-hydration-module

Hydrators in Doctrine in Apigility are handled by phpro/zf-doctrine-hydration-module. Familiarity with this module is very important to understanding how to extend hydrators without creating special case hydrators. Doctrine in Apigility uses an Abstract Factory to create hydrators.

There should be no need to create your own hydrators. That bold statement is true because we're taking a white-gloved approach to data handling. By using Hydrator Strategies and Filters we can fine tune the configuration for each hydrator used for a Doctrine entity assigned to a resource.

phpro/zf-doctrine-hydration-module makes working with hydrators easy by moving each field which could be hydrated into Doctrine in Apigility's configuration file. The only configuration we need to concern ourselves with is `strategies` and `filters`

```
<?php
'doctrine-hydrator' => array(
    'DbApi\\V1\\Rest\\Artist\\ArtistHydrator' => array(
        'entity_class' => 'Db\\Entity\\Artist',
        'object_manager' => 'doctrine.entitymanager.orm_default',
        'by_value' => true,
        'filters' => array(
            'artist_default' => array(
                'condition' => 'and',
                'filter' => 'DbApi\\Hydrator\\Filter\\ArtistDefault',
            ),
        ),
        'strategies' => array(
            'performance' => 'ZF\\Doctrine\\Hydrator\\Strategy\\CollectionLink',
            'artistGroup' => 'ZF\\Doctrine\\Hydrator\\Strategy\\CollectionLink',
            'artistAlias' => 'ZF\\Doctrine\\Hydrator\\Strategy\\CollectionLink',
        ),
        'use_generated_hydrator' => true,
    ),
```

3.6 Hydrator Filters

Here is the ArtistDefault filter

```
<?php
namespace DbApi\Hydrator\Filter;

use Zend\Hydrator\Filter\FilterInterface;

class ArtistDefault implements
    FilterInterface
{
    public function filter($field)
    {
        $excludeFields = [
            'artistMergeKeep',
            'artistMergeMerge',
        ];
```

```
        if (in_array($field, $excludeFields)) {
            return false;
        }

        return true;
    }
}
```

This should be quite obvious; fields are excluded from being hydrated (or extracted) based on the filter.

3.7 Hydrator Strategies

The module API-Skeletons/zf-doctrine-hydrator provides all the hydrator strategies you will need. More information on these strategies in hydration.

3.8 max_depth

Because Doctrine hydrators can extract relationships the default response from a Doctrine in Apigility Resource will include an `_embedded` section with the extracted entities and their `_embedded` and so on. **For special cases only** does zfcampus/zf-hal have a max_depth parameter. This special case is not intended to correct issues with HATEOAS in Doctrine in Apigility. When you encounter a cyclic association in Doctrine in Apigility the correct way to handle it is using Hydrator Strategies and Filters.

3.9 HATEOAS

Hypertext as the engine of application state is the goal of serving data from Doctrine in Apigility. Creating a response with no dead ends. That is, anytime a reference is made to another entity or collection and that resource is not part of the response there will be an http self link to that resource. This way a requesting application can fetch all data associated with a resource even if it takes more than one request.

A very good example of a practical response of HATEOAS can be found in the README for API-Skeletons/zf-doctrine-hydrator

The data returned from each resource is the data for that resource' entity. You should not try to add data to a response which is not naturally hydrated. However, there may be times when computed data is required as part of a response. This is covered in detail in HATEOAS.

3.10 An Example

Finally here is an example created by applying the rules listed above and the details listed in this book. You'll see this performance has an embedded artist as well as links to every place in the API a client may wish to go to next. It is not the job of the API to decide where to go next. The job of the API is to serve data and give directions for where a client may go

```
{
  "performanceDate": "1995-02-21",
  "venue": "Delta Center",
  "city": "Salt Lake City",
  "state": "UT",
```

```
  "set1": "Salt Lake City\nFriend Of The Devil\nWang Dang Doodle\nTennessee␣
↪Jed\nBroken Arrow\nBlack Throated Wind*\nSo Many Roads\nThe Music Never Stopped",
  "set2": "Foolish Heart \u0026gt;\nSamba In The Rain\nTruckin\u0027 \u0026gt;\nI␣
↪Just Wanna Make Love To You \u0026gt;\nThat Would Be Something \u0026gt;\nDrums␣
↪\u0026gt;\nSpace \u0026gt;\nVisions Of Johanna \u0026gt;\nSugar Magnolia\n\nEncore:␣
↪\nLiberty",
  "set3": " ",
  "description": "* Weir on acoustic, First Salt Lake City. First Want To Make Love␣
↪To You since 10\/8\/84, First Visions 4\/22\/86.  Salt Lake City from Weir\u0027s␣
↪solo album Heaven Help the Fool\n\nThis show was originally entered with the year␣
↪1995 which does not match the year shown in the date above. Please submit a␣
↪correction or confirmation of the performance date if you are able.",
  "lastUpdateAt": {
    "date": "2016-08-01 12:41:18.000000",
    "timezone_type": 3,
    "timezone": "UTC"
  },
  "createdAt": {
    "date": "2001-07-10 22:15:08.000000",
    "timezone_type": 3,
    "timezone": "UTC"
  },
  "year": 1995,
  "title": "",
  "isApproved": true,
  "id": 2333,
  "performanceGroup": null,
  "_embedded": {
    "performanceCorrection": {
      "_links": {
        "self": {
          "href": "http:\/\/docker.api.etreedb.org\/performance-correction?filter%5B0
↪%5D%5Bfield%5D=performance\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D
↪%5Bvalue%5D=2333"
        }
      }
    },
    "performanceLink": {
      "_links": {
        "self": {
          "href": "http:\/\/docker.api.etreedb.org\/performance-link?filter%5B0%5D
↪%5Bfield%5D=performance\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue
↪%5D=2333"
        }
      }
    },
    "source": {
      "_links": {
        "self": {
          "href": "http:\/\/docker.api.etreedb.org\/source?filter%5B0%5D%5Bfield
↪%5D=performance\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=2333"
        }
      }
    },
    "userPerformance": {
      "_links": {
        "self": {
          "href": "http:\/\/docker.api.etreedb.org\/user-performance?filter%5B0%5D
↪%5Bfield%5D=performance\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue
↪%5D=2333"
```

```
        }
      }
    },
    "artist": {
      "name": "Grateful Dead",
      "icon": "\/images\/gdskullsmall.gif",
      "createdAt": {
        "date": "2001-07-10 22:15:08.000000",
        "timezone_type": 3,
        "timezone": "UTC"
      },
      "abbreviation": "gd",
      "isTradable": true,
      "description": "",
      "id": 2,
      "artistLink": {},
      "_embedded": {
        "artistAlias": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/artist-alias?filter%5B0%5D
↪%5Bfield%5D=artist\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=2"
            }
          }
        },
        "performance": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/performance\/2333?filter%5B0
↪%5D%5Bfield%5D=artist\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue
↪%5D=2"
            }
          }
        },
        "user": {
          "username": "toma",
          "email": "toma@etree.org",
          "name": "Tom Anderson",
          "createdAt": {
            "date": "1999-09-15 00:00:00.000000",
            "timezone_type": 3,
            "timezone": "UTC"
          },
          "rules": "\u003Cp\u003E\r\n\tWelcome to my site. I hope you find it useful.
↪\u003Cbr \/\u003E\r\n\t\u003Cbr \/\u003E\r\n\tYou can contact the db team at␣
↪etreedb@googlegroups.com\u003C\/p\u003E\r\n",
          "isActiveTrading": true,
          "city": "San Francisco",
          "state": "CA",
          "postalCode": null,
          "description": "",
          "lastUpdateAt": {
            "date": "2017-05-21 16:24:02.000000",
            "timezone_type": 3,
            "timezone": "UTC"
          },
          "id": 1,
          "_embedded": {
```

```
            "source": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/source?filter%5B0%5D
↪%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            },
            "sourceComment": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/source-comment?filter%5B0
↪%5D%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            },
            "userFamily": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/user-family?filter%5B0%5D
↪%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            },
            "userFamilyExtended": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/user-family-extended?
↪filter%5B0%5D%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D
↪%5Bvalue%5D=1"
                }
              }
            },
            "userFeedback": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/user-feedback?filter%5B0
↪%5D%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            },
            "userFeedbackPost": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/user-feedback?filter%5B0
↪%5D%5Bfield%5D=postUser\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue
↪%5D=1"
                }
              }
            },
            "userList": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/user-list?filter%5B0%5D
↪%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            },
```

```
            "userPerformance": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/user-performance?filter
↪%5B0%5D%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue
↪%5D=1"
                }
              }
            },
            "media": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/media?filter%5B0%5D
↪%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            },
            "userWantlist": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/performance\/2333?filter
↪%5B0%5D%5Bfield%5D=wantlistUser\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D
↪%5Bvalue%5D=1"
                }
              }
            },
            "role": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/role?filter%5B0%5D%5Bfield
↪%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            }
          },
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/user\/1"
            }
          }
        },
        "lastUser": {
          "username": "toma",
          "email": "toma@etree.org",
          "name": "Tom Anderson",
          "createdAt": {
            "date": "1999-09-15 00:00:00.000000",
            "timezone_type": 3,
            "timezone": "UTC"
          },
          "rules": "\u003Cp\u003E\r\n\tWelcome to my site. I hope you find it useful.
↪\u003Cbr \/\u003E\r\n\t\u003Cbr \/\u003E\r\n\tYou can contact the db team at␣
↪etreedb@googlegroups.com\u003C\/p\u003E\r\n",
          "isActiveTrading": true,
          "city": "San Francisco",
          "state": "CA",
          "postalCode": null,
          "description": "",
```

```
          "lastUpdateAt": {
            "date": "2017-05-21 16:24:02.000000",
            "timezone_type": 3,
            "timezone": "UTC"
          },
          "id": 1,
          "_embedded": {
            "source": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/source?filter%5B0%5D
↪%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            },
            "sourceComment": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/source-comment?filter%5B0
↪%5D%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            },
            "userFamily": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/user-family?filter%5B0%5D
↪%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            },
            "userFamilyExtended": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/user-family-extended?
↪filter%5B0%5D%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D
↪%5Bvalue%5D=1"
                }
              }
            },
            "userFeedback": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/user-feedback?filter%5B0
↪%5D%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            },
            "userFeedbackPost": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/user-feedback?filter%5B0
↪%5D%5Bfield%5D=postUser\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue
↪%5D=1"
                }
              }
            },
            "userList": {
```

```
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/user-list?filter%5B0%5D
↪%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            },
            "userPerformance": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/user-performance?filter
↪%5B0%5D%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue
↪%5D=1"
                }
              }
            },
            "media": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/media?filter%5B0%5D
↪%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            },
            "userWantlist": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/performance\/2333?filter
↪%5B0%5D%5Bfield%5D=wantlistUser\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D
↪%5Bvalue%5D=1"
                }
              }
            },
            "role": {
              "_links": {
                "self": {
                  "href": "http:\/\/docker.api.etreedb.org\/role?filter%5B0%5D%5Bfield
↪%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=1"
                }
              }
            }
          },
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/user\/1"
            }
          }
        },
        "artistGroup": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/artist-group?filter%5B0%5D
↪%5Bfield%5D=artist\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=2"
            }
          }
        }
      },
      "_links": {
```

```
        "self": {
          "href": "http:\/\/docker.api.etreedb.org\/artist\/2"
        }
      }
    },
    "user": {
      "username": "aikox2",
      "email": "aiko",
      "name": "aikox2",
      "createdAt": {
        "date": "2004-01-24 18:15:06.000000",
        "timezone_type": 3,
        "timezone": "UTC"
      },
      "rules": "\u003Cp\u003E\r\n\tHey Now,\u003C\/p\u003E\r\n\u003Cp\u003E\r\n\tThis␣
↪list is for my personal reference.\u0026nbsp; I do not \u0026nbsp;trade via postal␣
↪mail.\u0026nbsp;\u003C\/p\u003E\r\n\u003Cp\u003E\r\n\tThis is a work in progress; I␣
↪have hundreds of shows that have yet to be added to the list.\u003C\/
↪p\u003E\r\n\u003Cp\u003E\r\n\tDisclaimer:\u0026nbsp; This list\u0026nbsp;
↪contains\u0026nbsp;shows that are commercially available, as well as shows by␣
↪artists who do not allow trading.\u0026nbsp; These shows are included for␣
↪reference\u0026nbsp;only, and are not available for trade.\u0026nbsp; No shows are␣
↪available for sale.\u003C\/p\u003E\r\n\u003Cp\u003E\r\n\tThe \u0026quot;I Was␣
↪There\u0026quot; list are shows I attended.\u0026nbsp; There are shows on this list␣
↪that I do not have recordings of.\u003C\/p\u003E\r\n\u003Cp\u003E\r\n\tThe␣
↪\u0026quot;ALL\u0026quot; list is large and thus loads slowly; you may want to␣
↪select a sub-list from the drop-down menu (i.e.: DVD, GD, WSP, PHIL, ABB, CLAPTON,␣
↪JAZZ, etc.)\u003C\/p\u003E\r\n\u003Cp\u003E\r\n\tA zero disc count means that I␣
↪have not yet updated that info.\u0026nbsp; If it is on my list, I have the show.
↪\u0026nbsp; If there is no media type designated, it is audio CDR.\u0026nbsp; Audio␣
↪source may be aud (audience microphone), SBD, FM or RIP (commercial CD backup copy).
↪\u0026nbsp; If no source is listed, it predates my adding this info.\u0026nbsp; All␣
↪audio is lossless sourced except for a handful of shows that are MP3 sourced and so␣
↪indicated.\u003C\/p\u003E\r\n\u003Cp\u003E\r\n\tAll DVDs are videos.\u0026nbsp; All␣
↪DVDs are so designated.\u0026nbsp; I do not own any DVD audio.\u0026nbsp; If the␣
↪Media field does not specify DVD, it is an audio CDR that I have not added the␣
↪media type to yet.\u0026nbsp; These predate my collecting video.\u0026nbsp; Video␣
↪source may be aud (audience camera), PRO (multi-camera, not broadcast), TV (proshot␣
↪for broadcast), WEB (proshot for webstream) or RIP (commercial DVD backup copy).
↪\u0026nbsp;\u003C\/p\u003E\r\n\u003Cp\u003E\r\n\tIf a show is listed twice on␣
↪my\u0026nbsp;a list, that means I have an audio CDR version and a video DVD version,
↪ or multiple sources of the same show.\u0026nbsp;\u003C\/
↪p\u003E\r\n\u003Cp\u003E\r\n\tThough many of the DVDs indicate they are PAL, not␣
↪all PAL DVDs have been so designated.\u0026nbsp;\u003C\/
↪p\u003E\r\n\u003Cp\u003E\r\n\taikox2\u003C\/
↪p\u003E\r\n\u003Cp\u003E\r\n\t\u0026nbsp;\u003C\/p\u003E\r\n",
      "isActiveTrading": true,
      "city": "",
      "state": "NC",
      "postalCode": null,
      "description": null,
      "lastUpdateAt": {
        "date": "2017-11-11 21:01:42.000000",
        "timezone_type": 3,
        "timezone": "UTC"
      },
      "id": 78828,
      "_embedded": {
```

```
        "source": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/source?filter%5B0%5D%5Bfield
↪%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=78828"
            }
          }
        },
        "sourceComment": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/source-comment?filter%5B0%5D
↪%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=78828
↪"
            }
          }
        },
        "userFamily": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/user-family?filter%5B0%5D
↪%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=78828
↪"
            }
          }
        },
        "userFamilyExtended": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/user-family-extended?filter
↪%5B0%5D%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue
↪%5D=78828"
            }
          }
        },
        "userFeedback": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/user-feedback?filter%5B0%5D
↪%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=78828
↪"
            }
          }
        },
        "userFeedbackPost": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/user-feedback?filter%5B0%5D
↪%5Bfield%5D=postUser\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue
↪%5D=78828"
            }
          }
        },
        "userList": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/user-list?filter%5B0%5D
↪%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=78828
↪"
```

```
            }
          }
        },
        "userPerformance": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/user-performance?filter%5B0%5D
↪%5Bfield%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=78828
↪"
            }
          }
        },
        "media": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/media?filter%5B0%5D%5Bfield
↪%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=78828"
            }
          }
        },
        "userWantlist": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/performance\/2333?filter%5B0
↪%5D%5Bfield%5D=wantlistUser\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D
↪%5Bvalue%5D=78828"
            }
          }
        },
        "role": {
          "_links": {
            "self": {
              "href": "http:\/\/docker.api.etreedb.org\/role?filter%5B0%5D%5Bfield
↪%5D=user\u0026filter%5B0%5D%5Btype%5D=eq\u0026filter%5B0%5D%5Bvalue%5D=78828"
            }
          }
        }
      },
      "_links": {
        "self": {
          "href": "http:\/\/docker.api.etreedb.org\/user\/78828"
        }
      }
    },
    "wantlistUser": {
      "_links": {
        "self": {
          "href": "http:\/\/docker.api.etreedb.org\/user?filter%5B0%5D%5Bfield
↪%5D=userWantlist\u0026filter%5B0%5D%5Btype%5D=ismemberof\u0026filter%5B0%5D%5Bvalue
↪%5D=2333"
        }
      }
    }
  },
  "_links": {
    "self": {
      "href": "http:\/\/docker.api.etreedb.org\/performance\/2333"
    }
```

```
    }
}
```

CHAPTER 4

Entity Relationship Diagramming with Skipper

In the world of Object Relational Mapping one tool stands alone. Skipper is a visual diagramming tool which exports to Doctrine metadata.

Often when a project is new the metadata for the application is handled solely through annotations. This is not a good solution. Projects usually start this way because the full suite of tools available to ORM developer is not realized. Skipper is one of these overlooked tools.

Using Skipper every relationship in your ORM can be visualized and all the metadata can be handled without touching the code. Take for instance you want to add a field to an entity: Using Skipper you will visually add the new field, export the XML metadata (note: always us XML metadata), run `orm:generate-entities` then run `orm:schema-tool:update --dump-sql` to get the SQL you must add to your migrations.

The visual diagram Skipper provides you enables your entire team from CEO to JS Programmer to understand the data model of the application. To compare if you used annotations any user interested in the schema would need to read each file and interpret the annotions and memorize each annotation. So, when your CEO can speak the diction of your application because he can see the field names and relationships in the ORM your entire organization will benefit. And when a JS Programmer can see the relationships visually and map those to the HAL response they recieve from the API everyone will benefit.

It is strongly recommended to migrate your ORM metadata into Skipper before creating a Doctrine in Apigility API.

CHAPTER 5

Skeleton Application

This section describes how to build a Doctrine in Apigility application from the Apigility Skeleton.

First clone the Apigility Skeleton module:

```
git clone git@github.com:zfcampus/zf-apigility-skeleton
```

cd into the zf-apigility-skeleton and require these repositories:

```
composer require zfcampus/zf-apigility-doctrine
composer require doctrine/doctrine-orm-module
composer require zendframework/zend-mvc-console
```

When prompted select the `config/modules.config.php` file to add the new module with the exception of the `ZF\Apigility\Doctrine\Admin` which should be added to the `config/development.config.php.dist`.

The direction to include `zend-mvc-console` is debatable. This module will no longer be maintained in the future but the discussion of how to replace it is outside the scope of this book.

5.1 Db Module

For the entities in my application I create a module called simply 'Db'. Your mileage may vary as this is often given the namespace of the application or company name. At any rate this will be the module where your XML annotations and entity classes are stored.

You will be using the command line to create your entities. It should be rare to create a custom function inside an entity as these usually belong in the Repository for the entity instead. Following this rule your entities can be managed by the command line tool `orm:generate-entities`. This command takes an argument of the path to create the entities. If you give it the path `module/Db/src` it will create the entities in the ''module/Db/src/Db/Entity directory based on the namespace of the entities. This shows that we need the subdir *Db* under *src*. In recent Zend Framework work the autoloading of modules has been moved to the composer.json file and you can still use that but you must be sure to have the correct directory structure for the code generation.

Inside the Db module create a `config` directory and inside that directory create an `orm` directory. This is where you will export your entity metadata to from Skipper.

5.2 config/autoload/local.php

You'll need to create a configuration for Doctrine in your local.php. This file is used so each deployment can have independent configuration. Here is an example file

```
<?php
return array(
    'doctrine' => array(
        'connection' => array(
            'orm_default' => array(
                'driverClass' => 'Doctrine\\DBAL\\Driver\\PDOMySql\\Driver',
                'params' => array(
                    'user' => 'root',
                    'password' => '123',
                    'host' => 'mysql',
                    'dbname' => 'etreedb',
                    'port' => '3306',
                    'charset' => 'utf8',
                    'collate' => 'utf8_general_ci',
                ),
            ),
        ),
    ),
);
```

5.3 Export Metadata and Create Entities

For a brand new ERD export the XML to the `module/Db/config/orm` directory and add the driver config to your module.config.php file:

```
'doctrine' => [
    'driver' => [
        'db_driver' => [
            'class' => 'Doctrine\\ORM\\Mapping\\Driver\\XmlDriver',
            'paths' => [
                __DIR__ . '/orm',
            ],
        ],
        'orm_default' => [
            'class' => 'Doctrine\\ORM\\Mapping\\Driver\\DriverChain',
            'drivers' => [
                'Db\\Entity' => 'db_driver',
            ],
        ],
    ],
],
```

then run:

```
php public/index.php orm:generate-entities module/Db/src
```

and your PHP skeleton code will be done.

CHAPTER 6

Recommended Extension Repositories

6.1 Migrations and Fixtures

This is a new topic to many developers so I'm including this special note to you, reader, to become familiar with them. migrations and fixtures

6.2 Doctrine QueryBuilder

Implementing the repository zfcampus/zf-doctrine-querybuilder is the most important extension for Doctrine in Apigility. This repository allows your clients to create complex queries and sorting on individual resources. For instance if you give a user access to an `Performance` resource and that resources returns performances then `zf-doctrine-querybuilder` will allow a client to return only a subset of the data they have access to, for instance just performances from a given state. Implementation is covered in Doctrine QueryBuilder.

6.3 Doctrine Repository Plugins

API-Skeletons/zf-doctrine-repository provides a method to override the default `Repository` factory for Doctrine and implements a plugin architecture which can be used in lieu of dependency injection into repositories. This repository provides a clean method for interacting with external resources from within a repository and its use is strongly encouraged.

6.4 Doctrine Hydrators

Covered also in hydrators is API-Skeletons/zf-doctrine-hydrator. This repository includes three hydrator plugins which are used to create a fluent HATEOAS HAL API response.

6.5 OAuth2 for Doctrine in Apigility

OAuth2 is implemented with several repositories, each building on the last. The first is API-Skeletons/zf-oauth2-doctrine which provies the metadata to attach OAuth2 entities to your existing schema via a dynamic hook to your User entity.

API-Skeletons/zf-oauth2-doctrine-console provies console routes for managing `zf-oauth2-doctrine` resources.

API-Skeletons/zf-oauth2-doctrine-identity should have been a part of `zf-oauth2-doctrine` from the beginning. That being said, this repository replaces the `AuthenticatedIdentity` of `zfcampus/zf-mvc-auth` with an identity which contains access to the `AccessToken`, `User`, `Client`, and `AuthorizationService`. This allows you to inject the `AuthenticationService` into your classes then access the identity via `$authorizationService->getIdentity()` then get the User class via `->getUser()`. The result of all this is a cleaner way to work with ORM objects only throughout your application.

API-Skeletons/zf-oauth2-doctrine-permissions-acl uses the identity from `zf-oauth2-doctrine-identity` to create ACL permissions on your resources. This module cleanly provides integration with `zfcampus/zf-mvc-auth` and is covered in authorization.

Note: Authored by API Skeletons.

CHAPTER 7

Files

When a Doctrine resoruce is created through the Apigility UI there are two new files created. For a resource named Artist these files are stored in the API module the resource was created in:

```
V1\Rest\Artist\ArtistCollection.php
V1\Rest\Artist\ArtistResource.php
```

For the strategy this book defines it is important you do not try to extend these files or modify them in any way. They exist because they could be modified but allow me to repeat: do not modify these files. Any reason you may find to modify these files can be accomplished through other means such as a Query Provider or subscribing to one of the many events which Doctrine in Apigility throws.

7.1 New Files

You will create new files as you build your API. It is best to put these new files inside the `V1\` directory inside your API. Examples are `V1\Query\Provider`, `V1\Query\CreateFilter`, `V1\Hydrator\Strategy`, `V1\Hydrator\Filter`.

CHAPTER 8

Security

8.1 Authentication

Knowing which user is logging into Apigility is not as strait forward as you may guess. The Basic and Digest authorization mechanisms rely on a .htpasswd file to store user credentials. This is not how the majority of web sites work.

For this document we will assume a User table inside a database with username and password fields. This is not a viable data store for Basic and Digest authentication. This is viable for a Password Grant Type using OAuth2. However using the Password Grant Type is highly frowned upon because it gives the site which builds the login form access to the user credentials and that is not what OAuth2 is about.

We need to authenticate a user before proceeding with an Implicit or Authorization Code grant type in order to assign the generated Access Token to that authenticated user for future API calls beyond OAuth2. Here is where traditional Authentication is useful. We need to secure the `ZF\OAuth2` resource prefix to authenticate via an old-fashioned login page and here zfcampus/zf-mvc-auth comes in.

As mentioned in this Auth section `zf-mvc-auth` can force a configured Authorization adapter to a given API. The mechanism *zf-mvc-auth* uses to accomplish this is configuration of resource prefixes to Authentication Types. When you create a custom Authentication Adapter you define the Authentication Type it supports. Let's begin with the configuration

```
<?php
  'zf-mvc-auth' => [
      'authentication' => [
          'map' => [
              'DbApi\\V1' => 'oauth2',
              'ZF\\OAuth2' => 'session',
          ],

          'adapters' => [
              'oauth2' => [
                  'adapter' => 'ZF\MvcAuth\Authentication\OAuth2Adapter',
                  'storage' => [
                      'adapter' => 'pdo',
```

```
                    'dsn' => 'mysql:host=localhost;dbname=oauth2',
                    'username' => 'username',
                    'password' => 'password',
                    'options' => [
                        1002 => 'SET NAMES utf8', // PDO::MYSQL_ATTR_INIT_COMMAND
                    ],
                ],
            ],

            'session' => [
                'adapter' => 'Application\\Authentication\\Adapter\\SessionAdapter',
            ],
        ],
    ],
],
```

With this configuration we have two adapters and they are each mapped to the section of the application we want them to secure. The `oauth2` adapter will be ignored since we're dedicated to finding a user to assign an Access Token to.

8.2 Creating an Authentication Adapter

Adapters must implement ZFMvcAuthAuthenticationAdapterInterface This interface includes

- `public function provides()` - This function will return the Authentication Type(s) this adapter supports. For our example it will be *session*.
- `public function matches($type)` - <sic> (from code) Attempt to match a requested authentication type against what the adapter provides.
- `public function getTypeFromRequest(Request $request)` - Still looking for Authentication Types this allows more generic matching based on the request.
- `public function preAuth(Request $request, Response $response)` - A helper function ran before `authenticate`
- `public function authenticate(Request $request, Response $response, MvcAuthEvent $mvcAuthEvent)` - Do an authentication attempt

For our examples we will use a route `/login` where any unauthenticated user who does not have their credentials stored in the session and is trying to access a resource under `ZF\OAuth2` will be routed to. This route will show the login page, let the user post to it, and if successful it will set the userid into the session where our adapter will be looking for it. When a user successfully authenticates with this adapter they will be assigned an `Application\Identity\UserIdentity`

```
<?php
  namespace Application\Authentication\Adapter;

  use ZF\MvcAuth\Authentication\AdapterInterface;
  use Zend\Http\Request;
  use Zend\Http\Response;
  use ZF\MvcAuth\Identity\IdentityInterface;
  use ZF\MvcAuth\MvcAuthEvent;
  use Zend\Session\Container;
  use Application\Identity;

  final class SessionAdapter implements
      AdapterInterface,
```

```
{
    public function provides()
    {
        return [
            'session',
        ];
    }

    public function matches($type)
    {
        return $type == 'session';
    }

    public function getTypeFromRequest(Request $request)
    {
        return false;
    }

    public function preAuth(Request $request, Response $response)
    {
    }

    public function authenticate(Request $request, Response $response, MvcAuthEvent
↪$mvcAuthEvent)
    {
        $session = new Container('webauth');

        if ($session->auth) {
            $userIdentity = new Identity\UserIdentity($session->auth);
            $userIdentity->setName('user');

            return $userIdentity;
        }

        // Force login for all other routes
        $mvcAuthEvent->stopPropagation();
        $session->redirect = $request->getUriString();
        $response->getHeaders()->addHeaderLine('Location', '/login');
        $response->setStatusCode(302);
        $response->sendHeaders();

        return $response;
    }
}
```

To use this authentication adapter you must assign it to the DefaultAuthenticationListener

```
<?php
namespace Application;

use ZF\MvcAuth\Authentication\DefaultAuthenticationListener;
use Zend\ModuleManager\Feature\BootstrapListenerInterface;
use Zend\EventManager\EventInterface;

class Module implements
    BootstrapListenerInterface
{
    public function onBootstrap(EventInterface $e)
```

```
        {
            $app = $e->getApplication();
            $container = $app->getServiceManager();

            // Add Authentication Adapter for session
            $defaultAuthenticationListener = $container->
↪get(DefaultAuthenticationListener::class);
            $defaultAuthenticationListener->attach(new␣
↪Authentication\AuthenticationAdapter());
        }
    }
```

The `Application\Identity\UserIdentity` requires a `getId()` function or public id property to return the user id of the authenticated user. This will be used by `zfcampus/zf-oauth2` to assign the user to `AccessToken`, `AuthorizationCode`, and `RefreshToken` using the `ZF\OAuth2\Provider\UserId` server manager alias.

The Basic and Digest authentication can assign the user because they read the .htpasswd file. For OAuth2 the user must be fetched using the `ZF\OAuth2\Provider\UserId` alias. You may create your own provider for a custom method of fetching an id.

This is the default

```
<?php
    'service_manager' => [
        'aliases' => [
            'ZF\OAuth2\Provider\UserId' =>
↪'ZF\OAuth2\Provider\UserId\AuthenticationService',
        ],
    ],
```

With this alias in place the OAuth2 server will store the userid and assign it to the Identity during future requests. The `getId()` or `id` property of the provider of the identity will be used to assign to OAuth2. When an OAuth2 resource is requested with a Bearer token the user will be fetched from the database and assigned to the AuthenticatedIdentity.

Here is an example `UserIdentity`

```
<?php
    namespace Application\Identity;

    use ZF\MvcAuth\Identity\IdentityInterface;
    use Zend\Permissions\Rbac\AbstractRole as AbstractRbacRole;

    final class UserIdentity extends AbstractRbacRole implements IdentityInterface
    {
        protected $user;
        protected $name;

        public function __construct(array $user)
        {
            $this->user = $user;
        }

        public function getAuthenticationIdentity()
        {
            return $this->user;
        }

        public function getId()
```

```
        {
            return $this->user['id'];
        }

        public function getUser()
        {
            return $this->getAuthenticationIdentity();
        }

        public function getRoleId()
        {
            return $this->name;
        }

        // Alias for roleId
        public function setName($name)
        {
            $this->name = $name;
        }
    }
```

8.3 Authorization

With our adapter in place it will not secure the ZFOAuth2 routes because they are by default secured with the `ZF\MvcAuth\Identity\GuestIdentitiy`. So we need to add Authorization to the application:

First we'll extend the onBootstrap we just created

```
<?php
    public function onBootstrap(EventInterface $e)
    {
        $app = $e->getApplication();
        $container = $app->getServiceManager();

        // Add Authentication Adapter for session
        $defaultAuthenticationListener = $container->
↪get(DefaultAuthenticationListener::class);
        $defaultAuthenticationListener->attach(new␣
↪Authentication\AuthenticationAdapter());

        // Add Authorization
        $eventManager = $app->getEventManager();
        $eventManager->attach(
            MvcAuthEvent::EVENT_AUTHORIZATION,
            new Authorization\AuthorizationListener(),
            100
        );
    }
```

And we need to create the AuthorizationListener we just configured

```
<?php
    namespace Application\Authorization;

    use ZF\MvcAuth\MvcAuthEvent;
```

```
  final class AuthorizationListener
  {
      public function __invoke(MvcAuthEvent $mvcAuthEvent)
      {
          $authorization = $mvcAuthEvent->getAuthorizationService();

          // Deny from all
          $authorization->deny();

          $authorization->addResource('Application\Controller\IndexController::index
↪');
          $authorization->allow('guest',
↪'Application\Controller\IndexController::index');

          $authorization->addResource('ZF\OAuth2\Controller\Auth::authorize');
          $authorization->allow('user', 'ZF\OAuth2\Controller\Auth::authorize');
      }
  }
```

Now when a request is made for an implicit grant type through `ZF\OAuth2` our new Authentication Adapter will see the user is not authenticated and store the user's requested url and redirect them to login where, after successfully logging in they will be directed back to the oauth2 request. The user will be granted access to the `ZF\OAuth2\Controller\Auth::authorize` resource and they will be assigned an Access Token.

8.4 Query Providers

A query provider is a class which provides a Doctrine QueryBuilder to the DoctrineResource in `zfcampus\zf-apigility-doctrine`. This prepared QueryBuilder is then used to fetch the entity or collection through the Doctrine Object Manager. The same Query Provider may be used for querying an entity or collection because when querying an entity the id from the route is assigned to the QueryBuilder after it is fetched from the Query Provider. For every verb (GET, POST, PATCH, etc.) your API handles through a Doctrine resource a Query Provider may be assigned.

Query Providers are used for security and for extending the functionality of the QueryBuilder object they provide. For instance, given a User API resource for which only the user who owns a resource may PATCH the resource, a QueryBuilder object can assign an `andWhere` parameter to the QueryBuilder to specify that only the current user may fetch the resoruce

```
<?php
  final class UserPatch extends AbstractQueryProvider
  {
      public function createQuery(ResourceEvent $event, $entityClass, $parameters)
      {
          $queryBuilder = $this->getObjectManager()->createQueryBuilder();
          $queryBuilder
              ->select('row')
              ->from($entityClass, 'row')
              ->andWhere($queryBuilder->expr()->eq('row.user', ':user'))
              ->setParameter('user', $this->getAuthentication()->getIdentity()->
↪getUser())
              ;

         return $queryBuilder;
      }
  }
```

The entity class we are `select()` from in the QueryBuilder will always be aliased as `row`. This is the only data which should be returned from a QueryBuilder as a complete Doctrine object.

More complicated examples **rely on your metadata being complete**. If your metadata defines joins to and from every join (that is, to an inverse and to a owner entity for every relationship) you can add complicated joins to your Query Provider

```
<?php
  $queryBuilder
      ->innerJoin('row.performance', 'performance')
      ->innerJoin('performance.artist', 'artist')
      ->innerJoin('artist.artistGroup', 'artistGroup')
      ->andWhere($queryBuilder->expr()->isMemberOf(':user', 'artistGroup.user'))
      ->setParameter('user', $this->getAuthentication()->getIdentity()->getUser())
      ;
```

8.5 Query Create Filters

Query Create Filters are the homolog to Query Providers but for POST requests only. These are intended to inspect the data the user is POSTing and if anything is incorrect to return an `ApiProblem`. These are not intended to correct the data. **If an API receives data which is incorrect it should reject the data, not try to fix it.**

CHAPTER 9

Hydrators

If you're unfamiliar with hydrators read Zend Framework's manual on Hydrators then read Doctrine's manual on Hydrators then read phpro/zf-doctrine-hydration-module

Doctrine in Apigility uses an Abstract Factory to create hydrators. **There should be no need to create your own hydrators.** That bold statement is true because we're taking a white-gloved approach to data handling. By using Hydrator Strategies and Filters we can fine tune the configuration for each hydrator used for a Doctrine entity assigned to a resource.

phpro/zf-doctrine-hydration-module makes working with hydrators easy by moving each field which could be hydrated into Doctrine in Apigility's configuration file. The only configuration we need to concern ourselves with is `strategies` and `filters`

```
<?php
  'doctrine-hydrator' => array(
      'DbApi\\V1\\Rest\\Artist\\ArtistHydrator' => array(
          'entity_class' => 'Db\\Entity\\Artist',
          'object_manager' => 'doctrine.entitymanager.orm_default',
          'by_value' => true,
          'filters' => array(
              'artist_default' => array(
                  'condition' => 'and',
                  'filter' => 'DbApi\\V1\\Hydrator\\Filter\\ArtistFilter',
              ),
          ),
          'strategies' => array(
              'performance' => 'ZF\\Doctrine\\Hydrator\\Strategy\\CollectionLink',
              'artistGroup' => 'ZF\\Doctrine\\Hydrator\\Strategy\\CollectionLink',
              'artistAlias' => 'ZF\\Doctrine\\Hydrator\\Strategy\\CollectionLink',
          ),
          'use_generated_hydrator' => true,
      ),
```

9.1 Hydrator Filters

A hydrator filter returns a boolean for whether the field passed to the filter should be rendered or not. They are used for removing fields, associations, and collections from a hydrating entity where that data should not be passed through the API. For instance a `User` entity may contain a `password` field used for authentication. While this field is used inside the application is has no business being returned as part of a `User` resource of the API

```
<?php
  namespace DbApi\V1\Hydrator\Filter;

  use Zend\Hydrator\Filter\FilterInterface;

  class UserFilter implements
      FilterInterface
  {
      public function filter($field)
      {
          $excludeFields = [
              'password',
          ];

          return (! in_array($field, $excludeFields));
      }
  }
```

Hydrator filters are attched to hydrators through configuration which is part of the Doctrine in Apigility configuration

```
<?php
  'doctrine-hydrator' => array(
      'DbApi\\V1\\Rest\\User\\UserHydrator' => array(
          ...
          'filters' => array(
              'user_filter' => array(
                  'condition' => 'and',
                  'filter' => 'DbApi\\V1\\Hydrator\\Filter\\UserFilter',
              ),
          ),
      ),
```

It is recommended to only use one hydrator filter per hydrator.

9.2 Hydrator Strategies

A hydrator strategy may be attached to any field, association, or collection which is derived by hydrating an entity. API-Skeletons/zf-doctrine-hydrator has three hydration strategies and rather than create a long article about how to create your own strategies it is the recommendation of this book that you only use one of these three strategies for your hydrated data.

There is a pitfall to using strategies; especially when a strategy extracts a collection. An entity which is a member of a collection which is extracted as part of a strategy for a parent entity will (should) have a reference back to the parent entity. This creates a cyclic relationship. Often developers turn to the `max_depth` parameter of `zf-hal` to correct this but this approach is really hack and should be avoided. Instead of trying to limit the depth replace the reference to the parent entity in the collection with an `EntityLink`; that is, just provide a link to the canonical resource rather than the whole extracted entity.

Using hydrator strategies you can create an elegant response for your API. A good strategy for applying Hydrator Strategies is to create your API resource through the Apigility UI then fetch an entity through the API. You'll see every relationship for the entity often as an empty class `{}`. For each of these empty classes, often they are collections, assign a hydrator strategy. Don't try to over-do it; you don't need to return the entire database with each request; just make sure the requesting client can get to any data which is related to the resource. It's ok if a client makes 2 or 3 requests to get all thier data.

CHAPTER 10

HATEOAS and Hypertext Application Language

Hypertext As The Engine Of Application State. This is the lofty goal an API developer should aspire to. By embedding URLs in your response a client application doesn't need to know details about the API such as how to paginate or where to find a referenced resource. By default Doctrine in Apigility creates canonical self referential links for every entity in a response. This is a big step for an API and you get it for free with Doctrine in Apigility.

Hypertext Application Language (HAL) is the dialect of JSON which Apigility speaks. The notable properties of HAL are self referential `_links` and an `_embedded` sections in each entity response (`_embedded` included only when an entity has referenced data). The `_links` array can be modified by the programmer as the resource is composed thereby allowing custom links to be included with the response. For instance a link to the audit trail for a resource may be included along with the canonical self referential link.

10.1 Adding Additional Links

Links may be used for anything to link to anywhere. Some HATEOAS tutorials suggest using links to show other actions such as a POST. To add more links to an entity as it is rendered use the `renderEntity` event in your Module.php file

```
<?php
  use Zend\EventManager\Event;
  use Zend\EventManager\EventInterface;
  use ZF\Hal\Link\Link;

  public function onBootstrap(EventInterface $e)
  {
      $app = $e->getTarget();
      $services = $app->getServiceManager();
      $this->container = $services;
      $sharedEvents = $services->get('SharedEventManager');
      $sharedEvents->attach('ZF\Hal\Plugin\Hal', 'renderEntity', array($this,
↪'onRenderEntity'));
  }
```

```
  public function onRenderEntity(Event $e)
  {
      $entity = $e->getParam('entity');

      switch (get_class($entity->getEntity())) {
          case 'Db\Entity\Artist':
              $link = new Link('home');
              $link->setUrl('https://apiskeletons.com');
              $entity->getLinks()->add($link);

              break;
          default:
              break;
      }
  }
```

10.2 Computed Data

Often it's useful to include computed data with an entity response. You can do this by attaching to the `renderEntity.post` event in your Module.php file

```
<?php
  use Zend\EventManager\Event;
  use Zend\EventManager\EventInterface;

  public function onBootstrap(EventInterface $e)
  {
      $app = $e->getTarget();
      $this->container = $app->getServiceManager();
      $sharedEvents = $this->container->get('SharedEventManager');
      $sharedEvents->attach('ZF\Hal\Plugin\Hal', 'renderEntity.post', array($this,
↪'onRenderEntityPost'));
  }

  public function onRenderEntityPost(Event $e)
  {
      $objectManager = $this->container->get('doctrine.entitymanager.orm_default');
      $entity = $e->getParam('entity');

      switch (get_class($entity->getEntity())) {
          case 'Db\Entity\Artist':
              $queryBuilder = $objectManager->createQueryBuilder();
              $queryBuilder->select('count(p)')
                  ->from('Db\Entity\Performance', 'p')
                  ->innerJoin('p.artist', 'a')
                  ->andWhere('a.id = :id')
                  ->setParameter('id', $entity->getEntity()->getId())
              ;
              $payload = $e->getParam('payload');
              $payload['_computed'] = array(
                  'performance' => array(
                      'count' => $queryBuilder->getQuery()->getSingleScalarResult(),
                  ),
              );
              break;
```

```
            default:
                break;
        }
    }
```

Note: Authored by API Skeletons. All rights reserved.

CHAPTER 11

Doctrine Events

This purely Doctrine subject has a good reason to be here. Often when a new entity is POSTed to a resource there will be default values you want to save to the new entity. For this reason you should use Doctrine events to catch lifecycle events to an entity and modify the entity as needed.

If you were to create an API where you tried to use a Query Create Filter, for example, to add default data then if you ever needed to create the same entity within the application the same defaults would not be applied. Doctrine events act globally in an application. You should use Doctrine events as often as needed.

Within a Doctrine event you have access to the Object Manager. This in turn has access to the Repositories (see API-Skeletons/zf-doctrine-repository) which may have access to plugins if you use the referenced module, as you should.

11.1 Doctrine Event Subscriber Manager Factory pattern

Here I will give an example of a good pattern for creating and subscribing event subscribers.

First the configuration, done with a ConfigProvider

```
<?php
namespace Db;

use Zend\ServiceManager\Factory\InvokableFactory;

final class ConfigProvider
{
    /**
     * Loaded at the time the DoctrineEventSubscriber is created
     */
    public function getDoctrineEventSubscriberConfig()
    {
        return [
            'factories' => [
                EventSubscriber\Doctrine\Artist::class
```

```
                    => InvokableFactory::class,
            ],
        ];
    }

    public function getDependencyConfig()
    {
        return [
            'factories' => [
                EventSubscriber\Doctrine\DoctrineEventSubscriberManager::class =>
                    ␣
↪EventSubscriber\Doctrine\DoctrineEventSubscriberManagerFactory::class,
            ],
        ];
    }
    ...
}
```

This config is loaded by the factory into the manager

```
<?php
namespace Db\EventSubscriber\Doctrine;

use Interop\Container\ContainerInterface;
use Db\ConfigProvider;

final class DoctrineEventSubscriberManagerFactory
{
    public function __invoke(
        ContainerInterface $container,
        $requestedName,
        array $options = null
    ) {
        $objectManager = $container->get('doctrine.entitymanager.orm_default');
        $configProvider = new ConfigProvider();

        $instance = new $requestedName($configProvider->
↪getDoctrineEventSubscriberConfig());
        $instance->setServiceLocator($container);
        $instance->setObjectManager($objectManager);

        return $instance;
    }
}
```

This event subscriber manager factory will create an event subscriber manager for your event subscribers.

```
<?php
namespace Db\EventSubscriber\Doctrine;

use Interop\Container\ContainerInterface;
use Zend\ServiceManager\ServiceManager as ZendServiceManager;
use DoctrineModule\Persistence\ObjectManagerAwareInterface;
use DoctrineModule\Persistence\ProvidesObjectManager;
use Db\ConfigProvider;

final class DoctrineEventSubscriberManager extends ZendServiceManager implements
    ObjectManagerAwareInterface
```

```
{
    use ProvidesObjectManager;

    private $serviceLocator;

    public function getServiceLocator()
    {
        return $this->serviceLocator;
    }

    public function setServiceLocator(ContainerInterface $serviceLocator)
    {
        $this->serviceLocator = $serviceLocator;

        return $this;
    }

    public function subscribe()
    {
        foreach ((array) $this->factories as $name => $squishedname) {
            $instance = $this->get($name);
            $instance->setAuthentication($this->getServiceLocator()->get(
↪'authentication'));

            $this->getObjectManager()->getEventManager()->addEventSubscriber(
↪$instance);
        }
    }
}
```

The subscribe() function, as it creates each event subscriber, injects the zf-mvc-auth authentication so an abstract is used.

```
<?php
namespace Db\EventSubscriber\Doctrine;

use Zend\Authentication\AuthenticationService;

abstract class AbstractEventSubscriber
{
    private $authentication;

    public function getAuthentication()
    {
        return $this->authentication;
    }

    public function setAuthentication(AuthenticationService $authentication)
    {
        $this->authentication = $authentication;
    }
}
```

Next you'll create your event subscribers with only one subcriber per entity.

```
<?php
namespace Db\EventSubscriber\Doctrine;
```

```
use Datetime;
use Doctrine\Common\Persistence\Event\LifecycleEventArgs;
use Doctrine\Common\EventSubscriber;
use Doctrine\ORM\Events;
use Db\Entity;

final class Artist extends AbstractEventSubscriber implements
    EventSubscriber
{
    public function getSubscribedEvents()
    {
        return [
            Events::prePersist,
            Events::preUpdate,
            Events::postUpdate,
        ];
    }

    public function prePersist(LifecycleEventArgs $args)
    {
        if (! $args->getObject() instanceof Entity\Artist) {
            return;
        }

        $args->getObject()->setUser($this->getAuthentication()->getIdentity()->
↪getUser());
        $args->getObject()->setLastUser($this->getAuthentication()->getIdentity()->
↪getUser());
        $args->getObject()->setCreatedAt(new Datetime());
    }

    public function preUpdate(LifecycleEventArgs $args)
    {
        if (! $args->getObject() instanceof Entity\Artist) {
            return;
        }

        $args->getObject()->setLastUser($this->getAuthentication()->getIdentity()->
↪getUser());
    }

    public function postUpdate(LifecycleEventArgs $args)
    {
        if (! $args->getObject() instanceof Entity\Artist) {
            return;
        }

        $args->getObjectManager()
            ->getRepository(Entity\Artist::class)
            ->enqueueIndexArtist($args->getObject());
    }
}
```

Notice the last call to the *enqueueIndexArtist*. Running domain code from event subscribers if you follow the guidelines in How repositories in Doctrine replace the "model" layer

Finally bootstrap the module to load all the event subscribers and subscribe them:

```php
<?php
public function onBootstrap(EventInterface $e)
{
    $sm = $e->getApplication()->getServiceManager();

    $doctrineEventSubscriberManager =
        $sm->get(EventSubscriber\Doctrine\DoctrineEventSubscriberManager::class);
    $doctrineEventSubscriberManager->subscribe();
}
```

CHAPTER 12

QueryBuilder

There is a library just for turning $_GET requests into QueryBuilder parameters for Doctrine. This library was written about the same time as Doctrine in Apigility and it is intended to be used together.

You can find zfcampus/zf-doctrine-querybuilder here. Please read the Philosopy first to understand why this library was written and to ensure you're comfortable with the excellent level of access the library provides to clients of your API. The rest of the documentation for that repository details well the capabilities so here I'll show a complete example for implementing zf-doctrine-querybuilder with Query Providers.

12.1 Abstract Factory

We'll need a common factory for all Query Providers to implement zf-doctrine-querybuilder

```
<?php
  namespace DbApi\Query\Provider;

  use Interop\Container\ContainerInterface;
  use Zend\ServiceManager\Factory\AbstractFactoryInterface;
  use Doctrine\Instantiator\Instantiator;
  use ZF\Doctrine\QueryBuilder\Filter\Service\ORMFilterManager;
  use ZF\Doctrine\QueryBuilder\OrderBy\Service\ORMOrderByManager;

  class QueryProviderAbstractFactory implements
      AbstractFactoryInterface
  {
      public function canCreate(ContainerInterface $container, $requestedName)
      {
          $instantiator = new Instantiator();
          $instance = $instantiator->instantiate($requestedName);

          return ($instance instanceof AbstractQueryProvider);
      }

      public function __invoke(ContainerInterface $container, $requestedName, array
↪$options = null)
```

```
        {
            $instance = new $requestedName();
            $instance->setFilterManager($container->get(ORMFilterManager::class));
            $instance->setOrderByManager($container->get(ORMOrderByManager::class));
            $instance->setObjectManager($container->get('doctrine.entitymanager.orm_
↪default'));
            $instance->setAuthentication($container->get('authentication'));

            return $instance;
        }
    }
```

We're including the filter and order by manager from zf-doctrine-querybuilder in our Query Providers. Additionally the Object Manager is set (note for this class the object manager alias is static) and the authentication from zf-mvc-auth is set. This gives access to the currently authenticated user.

12.2 Abstract Query Provider

Each Query Provider created through this abstract factory must be a AbstractQueryProvider and here is that code

```
<?php
  namespace DbApi\Query\Provider;

  use Zend\Authentication\AuthenticationService;
  use ZF\Apigility\Doctrine\Server\Query\Provider\AbstractQueryProvider as␣
↪ZFCampusAbstractQueryProvider;
  use ZF\Doctrine\QueryBuilder\Filter\Service\ORMFilterManager;
  use ZF\Doctrine\QueryBuilder\OrderBy\Service\ORMOrderByManager;
  use ZF\Rest\ResourceEvent;
  use DoctrineModule\Persistence\ProvidesObjectManager;

  abstract class AbstractQueryProvider extends ZFCampusAbstractQueryProvider
  {
      use ProvidesObjectManager;

      private $filterManager;
      private $orderByManager;
      private $authentication;

      public function getAuthentication()
      {
          return $this->authentication;
      }

      public function setAuthentication(AuthenticationService $authentication)
      {
          $this->authentication = $authentication;
      }

      public function getFilterManager()
      {
          return $this->filterManager;
      }

      public function setFilterManager(ORMFilterManager $filterManager)
      {
```

```
        $this->filterManager = $filterManager;

        return $this;
    }

    public function getOrderByManager()
    {
        return $this->orderByManager;
    }

    public function setOrderByManager(ORMOrderByManager $orderByManager)
    {
        $this->orderByManager = $orderByManager;

        return $this;
    }

    public function createQuery(ResourceEvent $event, $entityClass, $parameters)
    {
        $request = $event->getRequest()->getQuery()->toArray();
        $queryBuilder = $this->getObjectManager()->createQueryBuilder();
        $queryBuilder->select('row')
            ->from($entityClass, 'row');

        if (isset($request['filter'])) {
            $metadata = $this->getObjectManager()->getClassMetadata($entityClass);
            $this->getFilterManager()->filter(
                $queryBuilder,
                $metadata,
                $request['filter']
            );
        }

        if (isset($request['order-by'])) {
            $metadata = $this->getObjectManager()->getClassMetadata($entityClass);
            $this->getOrderByManager()->orderBy(
                $queryBuilder,
                $metadata,
                $request['order-by']
            );
        }

        return $queryBuilder;
    }
}
```

The interesting function here is `createQuery`. This function is part of the ZFCampusAbstractQueryProvider's interface. With this we parse the Request's query() data and send it through the filter manager and order by manager. These managers apply the filters from the query to the QueryBuilder.

12.3 Configuration

Enable the abstract factory for zf-apigility-doctrine-query-provider

```
<?php
  'zf-apigility-doctrine-query-provider' => array(
```

```
        'abstract_factories' => array(
            'DbApi\\Query\\Provider\\QueryProviderAbstractFactory',
        ),
    ),
```

12.4 Query Provider Example

To create a query provider extend it from the new AbstractQueryProvider and call the parent createQuery as the first line of the `createQuery` function

```
<?php
  namespace DbApi\Query\Provider;

  use ZF\Rest\ResourceEvent;
  use DbApi\Query\Provider\AbstractQueryProvider;
  use Db\Fixture\RoleFixture;

  final class PerformanceCorrectionPatch extends AbstractQueryProvider
  {
      public function createQuery(ResourceEvent $event, $entityClass, $parameters)
      {
          $queryBuilder = parent::createQuery($event, $entityClass, $parameters);

          if ($this->getAuthentication()->getIdentity()->getUser()->
↪hasRole(RoleFixture::$ADMIN)) {
              return $queryBuilder;
          }

          // The creating user can edit this
          $queryBuilder
              ->andWhere($queryBuilder->expr()->eq('row.user', ':user'))
              ->setParameter('user', $this->getAuthentication()->getIdentity()->
↪getUser())
              ;

          return $queryBuilder;
      }
  }
```

CHAPTER 13

External Resources

Tom H Anderson's Blog contains many articles specific to Doctrine in Apigilty. If you run across an question not answered in this book this blog should be your next stop.

API Skeletons official website contains specific information about Doctrine in Apigility.

Apigility Documentation is the official home of Apigility documentation.

Note: Authored by API Skeletons. All rights reserved.