Angular-Wakanda

Angular-Wakanda is a JavaScript connector that allows you to manage the data from your AngularJS application that is hosted on Wakanda Server via a set of Angular friendly APIs (the connector will manage all the REST requests).

Angular-Wakanda is also a yeoman generator that helps you to setup your Wakanda projects working with AngularJS, including many routines that can help you in your development workflow.

Connector

The Connector is an AngularJS service. The **$wakanda** service is available in the wakanda module once you include the "angular-wakanda.min.js" file.

You can inject the $wakanda service anywhere at anytime in your AngularJS application.

Most of the methods in this API will not be available until the Wakanda model of your application is loaded. To load your model, use **init( )**.

If you have a password-protected application, the $wakanda service provides a set of methods that allow you to log into it. You do not need to have the model loaded to use these methods. Refer to the **Authentication** chapter.

Generator

With the Connector, we provide a yeoman generator that lets you scaffold your AngularJS-based Wakanda applications. By following the yeoman workflow, you can improve your productivity when building an application using the stack's standard tools (yo, grunt, bower, npm) that are available for your Wakanda applications.

Resources

Here are a few of the Angular-Wakanda resources available to you:

- **Connector (GitHub bower repository)**: https://github.com/Wakanda/bower-angular-wakanda
- **Generator**: https://www.npmjs.org/package/generator-angular-wakanda
- **ngWakandaPack**: http://ng-wakanda-pack.us.wak-apps.com/
- **ngWakandaPack (GitHub)**: https://github.com/Wakanda/NG-Wakanda-Pack

Basic concepts

If you are new to Wakanda, below is a brief description of the basic conepts:

- **Dataclass**: A table in the datastore model. After calling **init( )**, you can access all the datastore classes as attributes in **$ds**. You create an entity (see below) by calling the **$create( )** function. For more information about datastore classes in Wakanda, refer to **Working with datastore classes**.
- **Entity (NgWakEntity)**: An entity is an instance of your dataclass encapsulated in a wrapper to simplify accessing it in AngularJS by way of APIs. An entity is a part of the entity collection, which is returned by **$find( )**. The **$findOne( )** function returns only one entity. You can also save a new or existing entity by calling **$save( )**. For more information about entities in Wakanda, refer to **Working with Entities**.
- **Entity collection (NgWakEntityCollection)**: An entity collection is an array of entities. Different methods are available to manipulate the entity collection as well as paging the entities in the entity collection. For more information about server-side entity collections in Wakanda, refer to **Working with Entity Collections on the Server**.

Authentication

To get the required permissions to use the Wakanda datastore class model, you must login for authentification by using either **$login( )** or **$loginByPassword( )**. You can use **$logout( )** method to log out.

Once logged in, you can use the **$currentUser( )** method to retrieve information regarding the current user. If you want to find out if the current user is in a specific group, you can pass the group name to **$currentUserBelongsTo( )**.

$currentUser( )

$q.promise **$currentUser**( )

Returns	$q.promise	$q.promise in which the result contains information about the current user

Description

$currentUser( ) allows you to return information about the current user. The returned object includes the following attributes:

Property	Description
ID	The user ID
fullName	User's full name
userName	User's username (corresponding to the *name* property server-side)

For more information about users and groups, refer to chapter **Users and Groups**.

$currentUserBelongsTo( )

$q.promise **$currentUserBelongsTo**(String *groupName*)

Parameter	Type	Description
groupName	String	Group name
Returns	$q.promise	The result property in the promise returns true if the user is in groupName

Description

$currentUserBelongsTo( ) allows you to know if the current user belongs to *groupName*.

For more information about users and groups, refer to chapter **Users and Groups**.

$login( )

$q.promise **$login**(String *user*, String *password*)

Parameter	Type	Description
user	String	User name
password	String	User name's password
Returns	$q.promise	$q.promise in which the result parameter is the result of the login

Important Note

$login( ) is an alias for **$loginByPassword( )**.

Description

$login( ) allows you to log into the Wakanda datastore model by specifying the *user* and *password*. In the *$q.promise* that is returned, you can retrieve the result to find out if the user was correctly logged in or not.

Example

The following example logs a user into the Wakanda datastore model:

```
$wakanda.$loginByPassword(employee.userName,employee.password).then(function(loginResult){
  if(loginResult.result === true){
    // logged in
  } else {
    // not logged in
}
```

$loginByPassword( )

$q.promise **$loginByPassword**(String *user*, String *password*)

Parameter	Type	Description
user	String	User name
password	String	User name's password
Returns	$q.promise	$q.promise in which the result parameter is the result of the login

Description

$loginByPassword( ) allows you to log into the Wakanda datastore model by specifying the *user* and *password*. In the *$q.promise* that is returned, you can retrieve the result to find out if the user was correctly logged in or not.

Example

The following example logs a user into the Wakanda datastore model:

```
$wakanda.$loginByPassword(employee.userName,employee.password).then(function(loginResult){
  if(loginResult.result === true){
    // logged in
  } else {
    // not logged in
}
```

$logout( )

$q.promise **$logout**( )

Returns	$q.promise	$q.promise in which the result parameter is the result of the logout

Description

$logout( ) allows you to log the user out of the application.

DataClass

A dataclass is a table in the datastore model. You can access all the datastore classes, or only the ones you specify, as attributes in **$ds** after you call **init( )**.
The example below allows you to retrieve information about the "ID" attribute in the Product datastore class:

```
var myID = $wakanda.$ds.Product.$attr('ID');
```

The methods in this category allow you to:

- obtain information regarding the datastore class,
- retrieve the datastore class's collection, entity, or datastore class methods,
- create an entity in the datastore class,
- or query the datastore class to return either one entity or an entity collection.

Information regarding the datastore class

The following properties and methods allow you to obtain information regarding the datastore class:

- **$collectionName**: The collection name defined for the datastore class.
- **$name**: The datastore class name.
- **$attr( )**: Information about a specific attribute or all the attributes in a datastore class.

Datastore class's methods

The following methods return the different methods defined for a datastore class:

- **$collectionMethods( )**: Collection methods.
- **$entityMethods( )**: Entity methods.
- **$dataClassMethods( )**: Datastore class methods.

***Note**: Only the methods whose scope is public are returned.*

Creating an entity

You can create an entity by using the **$create( )** method. This entity will not be saved in the datastore class until you call **$save( )**.

Creating an entity collection

The following methods create entity collections:

- **$find( )**: Query the datastore class and return the data found based on the criteria passed in an entity collection.
- **$findOne( )**: Find one entity based on its primary key and create an entity collection with it.

$name

Description

$name contains the class name defined for the datastore class in the Model.

$collectionName

Description

$collectionName contains thecollection name defined for the datastore class in the Model.

$attr( )

Object **$attr**([String *attributeName*])

Parameter	Type	Description
attributeName	String	Attribute name
Returns	Object	Attributes in the datastore class

Description

With **$attr( )**, you can return either the object for a specific attribute or an object of objects (one for each attribute). If the attribute's scope is not public, this method returns nothing.
For more information, refer to **DatastoreClassAttribute**.

Example

In this example, we retrieve the specific properties for an attribute:

```
var myID = $wakanda.$ds.Product.$attr('ID');
```

$collectionMethods( )

Array **$collectionMethods**( )

Returns	Array	Datastore class's collection methods

Description

$collectionMethods( ) allows you to retrieve the dataclass's collection methods whose scope is public. Refer to **Datastore Class Methods** for more information.

Example

This example retrieves the datastore class's collection methods:

```
var collectionMethods = $wakanda.$ds.Product.$collectionMethods();
```

$create( )

Entity **$create**([Object *attributes*])

Parameter	Type	Description
attributes	Object	Object defining the values for the attributes
Returns	Entity	Entity created, but not yet saved

Description

With **$create( )**, you create a new entity in the datastore class. The entity is not saved until you call **$save( )**.
The *attributes* parameter is an object in which you define the attributes and their values.

Example

In this example, you specify the values for the attributes in your datastore class:

```
$scope.newPerson = $wakanda.$ds.Person.$create( { firstName : "John", lastName : "Smith", salary: 90000 } );
```

Example

In the following example, we have an HTML file that contains an input field in which you enter a product name and an "Add" button.

```
<p>Product Name: <input type="text" ng-model="newProduct.name">
<input type="button" value="Add" ng-click="addOnClick(newProduct)"></p>
```

When you click on the "Add" button, the product is saved to the datastore class and the product list is updated in order to show all the products along with the newly created one.

```
$scope.newProduct = { };
$scope.addOnClick = function(newProduct){
    var newEntity;
    if($scope.newProduct.name){
        newEntity = $wakanda.$ds.Product.$create(newProduct);
        newEntity.$save().then(function(e){
            $scope.products = $wakanda.$ds.Product.$find({});
        });
    }
};
```

The list of products is displayed in our HTML:

```
<ul>
    <li ng-repeat="product in products">{{product.name}}</li>
</ul>
```

$dataClassMethods( )

Array **$dataClassMethods**( )

Returns	Array	Datastore class methods in the class

Description

$dataClassMethods( ) allows you to retrieve the dataclass's datastore class methods whose scope is public. Refer to **Datastore Class Methods** for more information.

$entityMethods( )

Array **$entityMethods**( )

Returns	Array	Datastore class's entity methods

Description

$entityMethods( ) allows you to retrieve the dataclass's entity methods whose scope is public. Refer to **Datastore Class Methods** for more information.

Example

The following example retrieves the datastore class's entity methods:

```
var entityMethods = $wakanda.$ds.Product.$entityMethods();
```

$find( )

EntityCollection **$find** (object)

Parameter	Type	Description
object	Object	Query parameters
Returns	EntityCollection	Entity collection of the records found

Description

$find( ) allows you to do a query on a datastore class. You define the information for your query in an object that is passed to this method.
In *object* you pass, you can specify the following properties:

Property	Type	Description
select	String	Specify one or more relation attributes so that Wakanda preloads the data in them after executing the query
filter	String	Query string in which you can include placeholders, e.g., "firstName = :1"
params	Array	The data to pass for each of the placeholders specified in the filter
orderBy	String	Attribute and sort order, e.g., "lastName desc"
pageSize	Number	Number of entities to return. By default, this number is 40.

For more information about these properties, refer to **Querying**.

Example

The following example searches employees whose last name begins with "A" and earn more than $60,000, orders them by salary in descending order, and returns the first 20:

```
$scope.listOfEmployees = $wakanda.$ds.Employee.$find({
    filter: 'lastName > :1 && salary > :2',
    params: ['a*', 60000],
    orderBy: 'firstName desc',
    pageSize: 20
});
```

$findOne( )

Entity **$findOne**(Number | String *primaryKey* [, Object *options*])

Parameter	Type	Description
primaryKey	Number, String	Value of the primary key
options	Object	Query parameter
Returns	Entity	Entity returned

Description

$findOne( ) allows you to return the entity whose primary key is specified in ***primaryKey***.
In *object* you pass the following property:

Property	Type	Description
select	String	Specify one or more relation attributes so that Wakanda preloads the data in them after executing the query

Example

This example returns one entity in the Product datastore class whose primary key is equal to 4:

```
$scope.product = $wakanda.$ds.Product.$findOne(4);
```

This example returns one entity in the Company datastore class whose primary key is equal to 9 and loads the related entities in the relation attribute, staff:

```
$scope.company = $wakanda.$ds.Company.$findOne(9, { select: 'staff' });
```

Entity

An entity is an instance of your dataclass encapsulated in a wrapper to simplify accessing it in AngularJS by way of APIs. An entity is a part of the entity collection, which is returned by **$find( )**. The **$findOne( )** function returns only one entity. You can also save a new or existing entity by calling **$save( )**.
In this section, the following methods allow you to:

- Retrieve the related data to an entity using **$fetch( )**,
- Find out if the entity is already loaded using **$isLoaded( )**,
- Delete the current entity from the datastore class using **$remove( )**, and
- Save an entity into the datastore class using **$save( )** by modifying the existing values or by having created it beforehand using **$create( )**.

$fetch( )

void **$fetch**( )

Description

$fetch( ) allows you to retrieve the related data related for an entity and return either one entity or an entity collection.

Example

The following example retrieves the employees for a specific company that is displayed in a list.
First, retrieve one company entity:

```
$scope.company = $wakanda.$ds.Company.$findOne(39431);
```

Then, execute a $fetch( ):

```
$scope.company.staff.$fetch();
```

To display the data, we are just showing the employees in a list:

```
<h4>Staff</h4>
<ul>
    <li ng-repeat="employee in company.staff">{{employee.firstName}} {{employee.lastName}}</li>
</ul>
```

$isLoaded( )

Boolean **$isLoaded**( )

Returns	Boolean	True = entity is loaded; False = entity is not loaded

Description

$isLoaded( ) allows you to find out if the entity has been loaded.

Example

If you want to know if a specific entity has already been loaded after doing a **$find( )**, you can write the following:

```
var isLoaded = products[2].$isLoaded();
```

$remove( )

void **$remove**

Description

$remove( ) allows you to delete the current entity from the datastore class.

Example

The following example allows you to remove the current entity and then update the entity collection to show that it has been removed:

```
 $scope.removeOnClick = function(product){
    product.$remove().then(function(e){
    $scope.products = $wakanda.$ds.Product.$find({});
    });
};
```

The corresponding HTML code on our page is the following:

```
<li ng-repeat="product in products">
    {{product.name}} <input type="button" value="Remove" ng-click="removeOnClick(product)">
</li>
```

$save( )

void **$save**( )

Description

$save( ) allows you to save an entity to the datastore class.

Example

In the following example, we have an HTML file that contains an input field in which you enter a product name and an "Add" button.

```
<p>Product Name: <input type="text" ng-model="newProduct.name">
<input type="button" value="Add" ng-click="addOnClick(newProduct)"></p>
```

When you click on the "Add" button, the product is saved to the datastore class and the product list is updated in order to show all the products along with the newly created one.

```
$scope.newProduct = { };
$scope.addOnClick = function(newProduct){
    var newEntity;
    if($scope.newProduct.name){
        newEntity = $wakanda.$ds.Product.$create(newProduct);
        newEntity.$save().then(function(e){
            $scope.products = $wakanda.$ds.Product.$find({});
        });
    }
};
```

The list of products is displayed in our HTML:

```
<ul>
    <li ng-repeat="product in products">{{product.name}}</li>
</ul>
```

EntityCollection

An entity collection is an array of entities. Different methods are available to manipulate the entity collection as well as paging the entities in the entity collection, which was created using the **$find( )** method.

The properties and methods in this category allow you to:

- retrieve information about the query used to create the entity collection,
- query data in the entity collection,
- find out the number of entities in the entity collection,
- display another page of data, and
- navigate the pages of data.

Retrieving query information

You can use the following properties and methods regarding the entity collection. The **$query** property allows you to know the *pageSize* and the *start* properties defined when the entity collection was created using the **$find( )** method or modified using the **$fetch( )** method.

The **$fetching** property indicates if the entity collection is currently being fetched using the **$fetch( )** method.

The **$totalCount** property indicates the number of entities in the entity collection.

Paging data

The following methods either display another page (based on the pageSize property defined in the options for the **$find( )** method) of data on the same page or navigate the pages of data:

- **$more( )**: Display another page of data without changing pages.
- **$prevPage( )**: Replace the data being displayed with the previous page of data.
- **$nextPage( )**: Replace the data being displayed with the next page of data.

Finding out if a collection is loaded

Using the **$isLoaded( )** method on a relation attribute, you can find out if the entity collection for it is loaded or not.

$totalCount

Description

$totalCount allows you to return number of entities in the collection.

$fetching

Description

$fetching allows you to return True if the entity collection is currently being fetched and False if it is not using the **$fetch( )** function.

$query

Description

$query allows you to retrieve the current values for *pageSize* and *start*. Here is a description of the properties:

Property	Description
pageSize	Number of entities to display per page of data
start	Entity number on which the paging starts

The *pageSize* property can be set by either defining it when calling **$find( )** or **$fetch( )**.

The *start* property can be modified by **$prevPage( )**, **$nextPage( )**, and **$more( )**.

You can calculate the current page number using two of the properties as shown below:

current page number = (*start* + *pageSize*) / *pageSize*

Example

You can display the page number based on the number of entities per page in the entity collection:

```
<p>Page {{(employees.$query.pageSize+employees.$query.start)/employees.$query.pageSize}}</p>
```

$fetch( )

EntityCollection **$fetch** (options)

Parameter	Type	Description
options	Object	Query parameters
Returns	EntityCollection	Entity collection of the records found in the current entity collection

Description

$fetch( ) allows you to query data in an entity collection and update the entity collection. You define the information for your query in an object that is passed to this method.

In *object*, you can specify the following properties:

Property	Type	Description
filter*	String	Query string in which you can include placeholders, e.g., "firstName = :1"
params*	Array	The data to pass for each of the placeholders specified in the filter
start	Number	Start at this entity in the collection.
pageSize	Number	Number of entities to return. By default, this number is 40.

***(*)** The filter and params properties are not currently available, but will be in the near future.*

For more information about these properties, refer to **Querying**.

Example

In our example, we first retrieve all the employees whose salary is greater than $90,000 and place them in an entity selection:

```
$scope.employees = [];
employees = $scope.employees = $wakanda.$ds.Employee.$find({
    select : 'employer',
    filter : 'salary >= 90000',
    start: 140,
    pageSize : 70,
    orderBy : 'salary asc'
});
```

Then, we go to the third page of data:

```
employees.$fetch({ start: 210 });
```

Example

If you want to go back to the entity collection before calling $fetch( ), you can set the *start* property to 0:

```
employees.$fetch({
    start: 0
});
```

$isLoaded( )

Boolean **$isLoaded**( )

Returns	Boolean	True = collection is loaded; False = collection is not loaded

Description

$isLoaded( ) allows you to find out if a collection based on a relation attribute has been loaded.

Example

If you want to know if you should execute a **$fetch( )** on an entity to retrieve related data, you can write the following to see if the data has already been loaded

```
var isLoaded = company.staff.$isLoaded();
```

$more( )

void **$more**

Description

$more( ) allows you to display another page of data appended to what is currently being displayed.
The page is defined by the *pageSize* property defined when calling the **$find( )** method.

Example

You can write the following in your template to retrieve more from the entity collection currently displayed:

```
<span ng-click="employees.$more()">Display more</span>
```

$nextPage( )

void **$nextPage**( )

Description

$nextPage( ) allows you to retrieve the next page of data in the entity collection that was created after calling **$find( )**.
The *pageSize* defined for **$find( )** is what will be used when retrieving the next page of data.

Example

In this example, we search the Employee datastore class and retrieve an entity collection:

```
$scope.employees = ds.Employee.$find({
    select : 'firstName, lastName, salary, employer',
    filter : 'firstName = "*"',
    pageSize : 20,
    orderBy : 'firstName asc'
});
```

On our page, we can call the following method from our template as shown below:

```
<span ng-click="employees.$nextPage()"> next </span>
```

$prevPage( )

void **$prevPage**( )

Description

$prevPage( ) allows you to retrieve the previous page of data in the entity collection that was created after calling **$find( )**.
The *pageSize* defined for **$find( )** is what will be used when retrieving the previous page of data.

Example

Refer to the example for **$nextPage( )**.

Initialization

The **init( )** method loads a proxy of the Wakanda datstore class model in the AngularJS application. Like all angular-wakanda async methods, **init( )** returns a $qPromise. You can either specify which datastore classes to load. Otherwise, all the datastore classes will be loaded.

```
// Load the entire model
// A promise is returned which callback receive the Wakanda Datastore proxy
$wakanda.init().then(function (ds) {
    // use the datastore
});

// Specify which dataclasses to load
$wakanda.init('Product, Person');

// Get the datastore object
$wakanda.$ds  // alias for $wakanda.getDatastore();
```

A router example

Here is an example of a simple router in our "app.js" file using the promise returned by **init( )**:

```
'use strict';

angular.module('myApp', [
  '...',         // your modules
  'wakanda' //the Wakanda module must also be loaded here
])

.config(['$routeProvider', function($routeProvider) {
    // Since $wakanda.init() returns a promise, you can use it inside of a resolve
    var routeResolver = {
      app: ['$wakanda', function($wakanda) {
          return $wakanda.init();
        }]
    };
    $routeProvider
            .when('/', {
              templateUrl: 'views/main.html',  //This page doesn't need a connection to Wakanda
              controller: 'MainCtrl'
            })
            .when('/example1', {
              templateUrl: 'views/example1.html',     //This page needs a connection to Wakanda and therefore will be
              controller: 'Example1Ctrl',   //launched when the promise returned by $wakanda.init is resolved
              resolve: routeResolver
            })
            .otherwise({
              redirectTo: '/'
            });
  }]);
```

$ds

Description

$ds allows you to return the datastore object. You can define which datastore class you'd like to load by specifying it. See example below.
$ds is the alias for **getDatastore( )**.

Example

The following example returns the datastore for the Product datastore class:

```
var prodDataClass = $wakanda.$ds.Product;
```

getDatastore( )

Datastore **getDatastore**( )

Returns	Datastore	Datastore (ds) to return

Description

getDatastore( ) allows you to return the datastore object. You can define which datastore class you'd like to load by specifying it. See the example below.
$ds is the alias for **getDatastore( )**.

Example

The following example returns the datastore for the Product datastore class:

```
var prodDataClass = $wakanda.getDatastore().Product;
```

You can also do the same using **$ds**:

```
var prodDataClass = $wakanda.$ds.Product;
```

init( )

$q.promise **init**([String *datastoreClasses*])

Parameter	Type	Description
datastoreClasses	String	Specify one or more datastore classes delimited by a comma
Returns	$q.promise	Promise

Description

init( ) allows you to initialize the Angular service and also specify which datastore classes to load. You can specify the dataclasses to load by passing them in a comma-delimited string.

This method, when successful, returns a *$q.promise*.

For more information on how to use this method, refer to **A router example**.

Example

The following example loads only two datastore classes:

```
return $wakanda.init('Product,Person');
```

Example

In this example, we retrieve all the employee entities:

```
$wakanda.init().then(function (ds) {
   $scope.employees = ds.Employee.$find({
        select : 'employer',
        filter : 'firstName = "*"',
        pageSize : 10,
        orderBy : 'firstName asc'
    });
});
```