
Airmad

Marc Anton Dahmen

Nov 29, 2020

CONTENTS

1	Get Started	3
1.1	Installation	3
1.2	Configuration	3
2	Usage	5
2.1	Options	6
2.2	Runtime Variables	6
2.3	Multiple Instances	7
3	Templates	9
3.1	Image Sliders	9
3.2	Linked Tables	9
3.3	Active Filters	10
4	Pagination	11
4.1	Example	11
5	Filters	13
5.1	Autocompletion	13
6	Example	15
6.1	Filters	15
6.2	Listing Records	16
6.3	Pagination	16

Airtable in Automad — a flexible yet simple to use [Automad](#) extension that let's you easily integrate [Airtable](#) bases into your site by using [Handlebars](#) templates.

[Get Started](#)

GET STARTED

Airtable is a great tool to quickly create your own database using a intuitive UI. While the possibilities of structuring data go far beyond the capabilities of **Automad** as a blogging platform, you might find out that Airtable lacks of flexibility and design options when it comes to sharing tables publicly. This is where Airmad comes in. The concept is rather simple. Airmad pulls a table — and optionally also its **linked** tables — using Airtable's REST API. To speed things up and align them with the user experience of a small and lightweight Automad site, all retrieved records are cached on your server. Updated data is pulled from time to time.

Attention: Airmad requires your webserver to run **PHP 7+** in order to work properly!

1.1 Installation

Airmad can be installed by using the Automad dashboard. However in case you would like to install the package by using Composer, just run the following command on your command line:

```
$ composer require airmad/airmad
```

1.2 Configuration

Airtable requires an [API](#) token to authenticate when accessing bases using their REST API. In case you don't have one, you can easily create one on your Airtable profile page. After successfully creating such token, it has to be added to Automad's `config/config.php` file. That can be done by navigating to **System Settings > More > Edit Configuration File** in the Automad dashboard as demonstrated below. Aside from the authentication, there you can also configure the Airtable cache lifetime in seconds.

```
{
  "AIRMAD_TOKEN": "keyXXXXXXXXXXXXXXX",
  "AIRMAD_CACHE_LIFETIME": 43200,
  ...
}
```


USAGE

Airmad can either be used in template files as part of a theme or, as recommended, in a snippet block. The latter one allows for integrating Airmad into any existing theme that supports Automad's block editor. The markup looks as follows:

Attention: You can simply paste an Airmad snippet directly into a code field of the new **Template Snippet** block on any page in the Automad dashboard.

```
<@ Airmad/Airmad {  
  base: 'appXXXXXXXXXXXXXXX',  
  table: 'Products',  
  view: 'Grid view',  
  linked: 'Type',  
  template: '  
    <div class="card">  
      <div class="card-content uk-panel uk-panel-box">  
        <div class="uk-panel-title">  
          {{ fields.Name }}  
        </div>  
        <p>  
          {{# fields.@.Type }}  
            <i>{{ Name }}</i>  
          {{/ fields.@.Type }}  
        </p>  
      </div>  
    </div>  
  ',  
  filters: 'Name, Type',  
  limit: 20,  
  page: @ { ?page | 1 }  
} @>
```

The code above doesn't produce any output. Instead it populates some Runtime *variables* that can be used in the Automad template to at any point after the Airmad instance above. To display the generated output, the `:airmadOutput` variable can be used in a template for example as follows.

```
<div class="cards grid am-stretched">  
  @ { :airmadOutput }  
</div>
```

Attention: In case you want to use multiple Airmad instances on one page, you will have to define unique prefixes for each one in order to avoid conflicts between them. Read more about using *multiple* instances below.

2.1 Options

The example above shows a typical use case of an Airtable integration. Find below a list of all available options.

Name	Description
<code>base</code>	The Airtable base ID
<code>table</code>	The main table to be used to pull records from
<code>view</code>	The view of the main <i>table</i> to be used
<code>linked</code>	A comma separated list of tables that are linked to a field of the main table records — note that is only required to list linked tables here that include information that you want to display. In case the field name differs from the actual table name to be linked, it is also possible to pass a list of strings like <code>fieldName1 => tableName1, fieldName2 => tableName2</code> to the parameter to link such fields to any table.
<code>template</code>	The Handlebar template to be used to render a record — can be either a string, a variable containing a string or a file path
<code>filter</code>	A comma separated list of fields that can be used to filter the records by — check out the examples below for more information about <i>filtering</i>
<code>limit</code>	The maximum number of records to be displayed on a page
<code>page</code>	The current page of records (pagination)
<code>prefix</code>	An optional prefix for the generated runtime variables instead of the default <code>:airmad</code> — it is required to define unique prefixes, in case <i>more than one</i> Airmad instance is used on a page

2.2 Runtime Variables

Aside from the output, Airmad provides more variables as shown in the table below.

Name	Description
<code>:airmadOutput</code>	The rendered output of the table records
<code>:airmadCount</code>	The number of found records
<code>:airmadPage</code>	The current page number — this has to be seen in context to the <code>limit</code> of items displayed on a page
<code>:airmadPages</code>	The amount of pages the records are spread over, also related to the <code>limit</code> option
<code>:airmadMemory</code>	The max memory used by Automad in bytes

Attention: Note that you can define an unique prefix to be used instead of `:airmad*` in the Airmad *options* when creating a new instance.

2.3 Multiple Instances

As soon as you want to use filters and select dropdowns to let a user control the displayed set of records on a page, you will have to use multiple instances of Airmad on one page. For example one instance request all records of a fictional table called `Type` to generate a list of all existing product types in your database, while another one gets the actual products for example from a table called `Products`. To avoid overwriting the output the first table with the output of the second one, the generated runtime variables need to have a unique prefix that can be defined in the options by using the `prefix` parameter.

```
<@ Airmad/Airmad {  
  base: 'appXXXXXXXXXXXXXXX',  
  table: 'Type',  
  view: 'Grid view',  
  template: '<option value="{{ fields.Name }}">',  
  prefix: ':type'  
} @>  
@{ :typeOutput }  
  
<@ Airmad/Airmad {  
  base: 'appXXXXXXXXXXXXXXX',  
  table: 'Products',  
  view: 'Grid view',  
  template: '<option value="{{ fields.Name }}">',  
  prefix: ':products'  
} @>  
@{ :productsOutput }
```


TEMPLATES

As mentioned earlier, Airmad uses [Handlebars](#) templates to render record data. While iterating table records, all record data is exposed to the engine and can be accessed by using the normal variable tags. The main items here are the `id`, the `fields` and the `createdTime`. The `fields` item actually contains all table fields entered by you. For example to get the `Name` of a record, you can simply use `{{ fields.Name }}` in a template. Aside from the default tags, Airmad provides some other useful helpers to let you easily use fields in linked tables or build slideshow.

3.1 Image Sliders

In case your table has an attachment field, you can use the `{{#slider fields.images}}` helper function to create an image slider containing all provided images as that are listed in a field called `fields.images`. By default the slide will have an aspect ratio of 1:1 — in other words a height of 100% relative to the width. You can pass an optional second argument to the helper to define a custom height as follows:

```
{{#slider fields.images 75%}}
```

3.2 Linked Tables

In case you have fields that actually link to other tables in your base, the content of such a field is just a bunch of record IDs. In most cases you would want to be able to actually get the values of the one or more fields of that record. Therefore Airmad adds a dedicated fields to your data model at runtime called `fields.@`. The `@` field contains all referenced records in linked tables. The example below demonstrates the usage of such fields.

To simply get the IDs of the records in a linked table, you can just loop over the list of IDs as usual.

```
{{# fields.Type }}  
  {{ . }}  
{{/ fields.Type }}
```

Instead of just getting the ID, you can directly loop over a list of the linked records by inserting a `@` into `{{# fields.Type }}` like `{{# fields.@.Type }}`. It is important to understand that the name after the `@` is here again the name of the **field** and might differ from the actual table name.

```
{{# fields.@.Type }}  
  <i>{{ Name }}</i>  
{{/ fields.@.Type }}
```

3.3 Active Filters

When building dropdown menus or similar to filter the set of elements, it is important to know what filter is currently active. Therefore Airmad the `active` field to any record that appears as value for a table filter in the query string. The field can be used as follows:

```
<option value="{{ id }}" {{#if active}}selected{{/if}}>
    {{ fields.Name }}
</option>
```

PAGINATION

In many cases, the amount of records in a table is simple to much for a single page. You will probably break down the list of records into multiple pages by setting the `limit` [option](#) to a fixed number. To help you building a simple pagination navigation, Airmad provides the `:airmadPage` and `:airmadPages` [runtime](#) variables.

4.1 Example

A very simple example for a pagination within an Automad snippet could look as follows:

```
<ul class="uk-pagination">

  <@ if @{ ?Page } > 1 @>
    <li><a href="?<@ queryStringMerge { Page: @{ ?Page | -1 } } @>">←</a></li>
  <@ end @>

  <@ for @{ :airmadPage | -3 } to @{ :airmadPage | +3 } @>
    <@ if @{ :i } > 0 and @{ :i } <= @{ :airmadPages } @>
      <li>
        <a
          href="?<@ queryStringMerge { Page: @{ :i } } @>"
          <@ if @{ ?Page | def(1) } = @{ :i } @>class="uk-active"<@ end @>
        >
          @{:i}
        </a>
      </li>
    <@ end @>
  <@ end @>

  <@ if @{ ?Page } < @{ :airmadPages } @>
    <li><a href="?<@ queryStringMerge { Page: @{ ?Page | +1 } } @>">→</a></li>
  <@ end @>

</ul>
```

You can simply copy and paste this into a snippet block after creating an Airmad instance. The classes in use will work out of the box with the **Standard** and **Adam** themes.

FILTERS

Searching and filtering are essential functions for displaying database content. In Airmad filtering records is pretty straight forward. The following example demonstrated the basic idea:

```
<form action="">
  <input type="text" name="Name" value="@{ ?Name }">
  <input type="text" name="Type" value="@{ ?Type }">
</form>
<ul>
<@ Airmad/Airmad {
  base: 'appXXXXXXXXXXXXXXX',
  table: 'Products',
  view: 'Grid view',
  linked: 'Type',
  template: '<li>{{ fields.Name }}</li>',
  filters: 'Name, Type',
  limit: 20,
  page: @{ ?Page | def(1) }
} @>
</ul>
```

In the snippet above, we have a simple form at the top including two input fields with the names `Name` and `Type`. The Airmad instance below that form has those names defined as `filters` as you can see in the highlighted line. Note that since in this example **Type** is a linked table, defining the `linked` parameter allows for searching in linked records as well.

5.1 Autocompletion

To enhance the user experience for your visitors, you might want to provide an autocompletion list of **Type** names for the second input field. You can simply use a second Airmad instance to pull all type names from the **Type** table and populate such a list with the `Name` field of each record. In the following example we use a datalist for such purpose.

Warning: Note in the snippet below that this time the **prefix** parameter must be set to a unique value to avoid conflict between both Airmad instances.

```
<form action="">
  <input type="text" name="Name" value="@{ ?Name }">
  <input type="text" list="types" name="Type" value="@{ ?Type }">
  <@ Airmad/Airmad {
    base: 'appXXXXXXXXXXXXXXX',
```

(continues on next page)

(continued from previous page)

```
      table: 'Types',
      view: 'Grid view',
      template: '<option value="{{ fields.Name }}">',
      prefix: ':type'
    } @>
    <datalist id="types">
      @{ :typeOutput }
    </datalist>
  </form>
  <ul>
    <@ Airmad/Airmad {
      base: 'appXXXXXXXXXXXXXXX',
      table: 'Products',
      view: 'Grid view',
      linked: 'Type => Types',
      template: '<li>{{ fields.Name }}</li>',
      filters: 'Name, Type',
      limit: 20,
      page: @{ ?Page | def(1) }
    } @>
  </ul>
```

EXAMPLE

The following example is supposed to wrap all features of Airmad like getting records, filtering and building a pagination. To allow for quick testing, the base for this example is the **Project tracker** database that serves as sample content when creating a new account on Airtable. Therefore it should be easy to just copy and paste the code — by replacing the app ID of course — after setting up authentication.

Attention: Make sure that you already have added the **AIRMAD_TOKEN** to your configuration as described in the **Get Stared** guide. And don't forget to replace the base ID in the snippet below with the one in your API documentation!

To be easily understandable, this example code is broken down into three sections. You can simply paste all sections together into a **Template Snippet** block.

6.1 Filters

The first part is creating the filter menu. Note that the naming of the input fields is essential here! In our example we want to filter the records by the **Client** field that is linked to the **Clients** table (note the “s” in the table name).

```
<form action="">
  <input type="text" name="Name" value="@{ ?Name }">
  <input type="text" list="clients" name="Client" value="@{ ?Client }">
  <@ Airmad/Airmad {
    base: 'appXXXXXXXXXXXX',
    table: 'Clients',
    view: 'All clients',
    template: '<option value="{{ fields.Name }}">',
    prefix: ':clients'
  } @>
  <datalist id="clients">
    @{ :clientsOutput }
  </datalist>
  <button type="submit">Filter</button>
</form>
```

6.2 Listing Records

The second part is building the actual list of record. Again, the mapping of linked tables is important here! The code below will generate a grid of record including a little slider to be used with with the **Standard** theme.

```
<@ Airmad/Airmad {
  base: 'appXXXXXXXXXXXXXXX',
  table: 'Design projects',
  view: 'All projects',
  linked: 'Client => Clients',
  template: '
    <div class="card">
      <div class="card-content uk-panel uk-panel-box">
        <div class="uk-panel-teaser">
          {{#slider fields.Project images 75%}}
        </div>
        <div class="uk-panel-title">
          {{ fields.Name }}
        </div>
        <p>
          {{# fields.@.Client }}
            <b>{{ Name }}</b>
          {{/ fields.@.Client }}
        </p>
      </div>
    </div>
  ',
  filters: 'Name, Client',
  limit: 8,
  page: @ { ?Page | def(1) }
} @>

<div class="cards grid am-stretched">
  @ { :airmadOutput }
</div>
```

6.3 Pagination

The last part will create the pagination navigation. Again the generated markup will work out of the box with the **Standard** theme.

```
<ul class="uk-pagination">
  <@ if @ { ?Page } > 1 @>
    <li><a href="?<@ queryStringMerge { Page: @ { ?Page | -1 } } @>">←</a></li>
  <@ end @>
  <@ for @ { :airmadPage | -3 } to @ { :airmadPage | +3 } @>
    <@ if @ { :i } > 0 and @ { :i } <= @ { :airmadPages } @>
      <li><a href="?<@ queryStringMerge { Page: @ { :i } } @>" <@ if @ { ?Page |
→def(1) } = @ { :i } @>
        class="uk-active"
      <@ end @>>@ { :i } </a></li>
    <@ end @>
  <@ end @>
  <@ if @ { ?Page } < @ { :airmadPages } @>
```

(continues on next page)

(continued from previous page)

```
<li><a href="?<@ queryStringMerge { Page: @{ ?Page | +1 } } @>"></a></li>
<@ end @>
</ul>
```