Airmad

Marc Anton Dahmen

Jun 30, 2022

CONTENTS

Airtable in Automad — a fexible yet simple to use Automad extension that let's you easily integrate Airtable bases into your site by using Handlebars templates.

Docs GitHub

CHAPTER

ONE

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 recordes 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 authentictaion, there you can also configure the Airtable cache lifetime and the model cache lifetime in seconds.

```
{
  "AIRMAD_TOKEN": "keyXXXXXXXXXXXXXX",
  "AIRMAD_CACHE_LIFETIME": 7200,
  ...
}
```

1.3 Disabling the Cache

In some cases you want to be able to trigger an update of the cached tables and models instantly. A fresh sync of the Airtable data with your Automad site can be forced by appending the `airmad_force_sync` parameter the URL of a page including an Airmad instance as follows:

```
https://domain.com/page?airmad_force_sync=1
```

That also comes in handy in case you want to automate the database sync using **cron** or **bash**:

```
curl -s 'https://domain.com/page?airmad_force_sync=1' > /dev/null
```

CHAPTER

TWO

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: 'appXXXXXXXXXXXXXX',
    table: 'Design projects',
    view: 'All projects',
    filters: 'Client, Category',
    formula: 'SEARCH(LOWER("@{ ?search }"), LOWER({Name}))',
    linked: 'Client => Clients',
    prefix: ':example',
    template: '
        {{#records}}
            {{#fields}}
                <div class="card">
                    <h3>{{Name}}</h3>
                    <p>
                        {{#Client}}
                            {{Name}}
                        {{/Client}}
                    </p>
                </div>
            {{/fields}}
        {{/records}}
    ',
    partials: '/packages/some/partials',
    limit: 20,
    page: @{ ?Page | def(1) }
} @>
```

The code above doesn't produce any output. Instead it populates some Runtime *variables* that can be used in the Automad template at any point after the Airmad instance above. Note the `prefix` parameter. The prefix is **required** to make sure that all runtime variables have **unique** names. To display the generated output, the `:exampleOutput` variable can be used in a template for example as follows.

```
<div class="cards">
    @{ :exampleOutput }
</div>
```

Attention: In case you want to use multiple Airmad instances on your **site**, you will have to define **unique prefixes** for each one in order to avoid conflicts between them.

2.1 Options

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

Name	Description
`base`	The Airtable base ID
`table`	The main table to be used to pull records from
`view`	The view of the main *table* to be used
`prefix`	A **required** prefix for the generated runtime variables — prefixes have to be unique, in case more than one Airmad instance is used on the site
`linked`	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 `fieldName1 => tableName1, fieldName2 => tableName2` to the parameter to link such fields to any table.
`template`	The Handlebar template to be used to render the model (the collection of records) — can be either a string, a variable containing a string or a file path
`partials`	You can optionally specify a path to a directory containing partials. That path must be an absolute path in relation to your Automad installation directory, like for example `/packages/extension/partials`. Note that partial files **must** have the `.handlebars` extension in order to be loaded.
`filters`	A comma separated list of fields that can be used to filter the records by — check out the examples below for more information about *filtering*
`formula`	A formula to be used to for the `filterByFormula` parameter when making API requests.
`limit`	The maximum number of records to be displayed on a page
`page`	The current page of records (pagination)

2.2 Runtime Variables

Aside from the output, Airmad provides more variables as shown in the table below. Note that `:prefix` can be replaced with any other valid string and is just used for demonstration here.

Name	Description
`:prefixOutput`	The rendered output of the table records
`:prefixCount`	The number of found records
`:prefixPage`	The current page number — this has to be seen in context to the `limit` of items displayed on a page
`:prefixPages`	The amount of pages the records are spread over, also related to the `limit` option
`:prefixMemory`	The max memory used by Automad in bytes

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

2.3 Filters and Formula

Airmad offers two ways of searching an Airtable base — filters and formulas. While filters are very easy to use and allow for automatic filtering of records whenever there is a query string parameter with a column name present, formulas are way more flexible and powerful. In contrast to filters, formulas allow for searching across multiple fields by a custom formula. Take a look at the official formula documentation provided by Airtable for a full list of available options and examples.

```
<@ Airmad/Airmad {
    base: 'appXXXXXXXXXXXXXX',
    table: 'Design projects',
    view: 'All projects',
    formula: 'SEARCH(LOWER("@{ ?search }"), LOWER(CONCATENATE({Name}, {Client})))',
    prefix: ':example',
    template: '
        {{#records}}
            {{#fields}}
                <h3>{{Name}}</h3>
            {{/fields}}
        {{/records}}
    '
} @>
```

The example above will demonstrates how you can implement searching the `Name` and the `Client` fields of records at the same time by only a single search parameter in the query string.

CHAPTER

THREE

TEMPLATES

As mentioned earlier, Airmad uses Handlebars to render table data. Basically all **model** data is structured in a multi-dimensional array and can be accessed in a template. You can see the actual data while developing your templates by enabling *debugging*.

3.1 Model

The model contains seven main elements — `records`, `filters`, `filteredFilters`, `query`, `count`, `pages` and `automad`. A typical structure looks as follows:

```
records
    record
        id
        fields
            column
            column
            ...
    record
        id
        fields
            column
            column
            ...
filters
    ...
filteredFilters
    ...
query
    ...
count
pages
automad
    title
    date
    ...
```

Name	Description
`records`	The records element basically contains all rows in the given table
`filters`	A collection of items of columns specified in the filters option
`filteredFilters`	The `filteredFilters` reprensent a relevant and unique collection of items of columns specified in the filters option that match the actual set of records
`query`	The query element contains all parameters of the query string
`count`	The count of records in filtered the model
`pages`	The number of pages needed to display all records in the model
`automad`	A bridge to a merged array of shared and page data.

In a template you can therefore iterate all records using the **Handlebars** syntax as demonstarted below. Note that `column` just represents any column name in your table.

```
{{#records}}
    {{#fields}}
        {{column}}
    {{/fields}}
{{/records}}
```

Attention: Airtable provides some example bases when setting up an account. The Airmad repository includes some example snipptes that are made to work with the **Project tracker** example base of Airtable. Take a look at the example template on GitHub.

3.1.1 Query String Parameters

To access a parameter in a **query string** like for example `https://domain.com?parameter=value` you can simply use:

```
{{query.parameter}}
```

3.1.2 Automad Data

You can also access Automad's page and shared data as follows as shown below. For example to get the title of the current page within a template you can simply use:

```
{{automad.title}}
```

3.1.3 Record ID

Since the actual record ID is by default not a field, Airmad provides the dedicated `_ID` field that contains the actual record ID.

```
{{#records}}
    {{ _ID }}
{{/records}}
```

3.1.4 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. Fortunately Airmad automatically looks up the linked fields for you and replaces the ID string with an array of the actual fields. The replaced ID is then moved to the `_ID` field of the record's array. Let's assume you have a `Type` table and you want to access the `Name` of each type linked to your product. The data returned by the Airtable API looks for example as follows:

```
{
  "fields": {
    "Type": [
      "recmD5WiE2GeV3ZIW",
      "recuBUENcDgqnzSww",
      "recj0zpg9qo8M7SeM"
    ]
  }
}
```

Airmad will look up all contained fields automatically and expose the following data to the render engine:

```
{
  "fields": {
    "Type": [
      {
        "Name": "Chair",
        "Product": ["recUtSDeLJ4HQI0uD", "recJcjDC9IN8Vws16"],
        "_ID": "recmD5WiE2GeV3ZIW"
      },
      {
        "Name": "Table",
        "Product": ["recUtSDeLJ4HQI0uD"],
        "_ID": "recuBUENcDgqnzSww"
      },
      {
        "Name": "Carpet",
        "Product": ["recJcjDC9IN8Vws16"],
        "_ID": "recj0zpg9qo8M7SeM"
      }
    ]
  }
}
```

In a template you can therefore simple loop over the types and get the `Name` as follows:

```
{{#Type}}
    {{Name}}
{{/Type}}
```

3.2 Debugging

To quickly understand the actual structure of the model returned by the Airtable API, you can enable the Debug Mode in Automad and then take a look at the browser console. Since there will be a lot of output, you can then simply filter the console by `Airmad->Airmad`.

3.3 Helpers

While there are all official Handlebars helpers available in templates, Airmad also provides some additional useful helpers as listed here below.

3.3.1 Image Sliders

In case your table has an attachement field, you can use the `{{#slider images}}` or `{{#sliderLarge images}}` helper functions to create an image slider containing all provided images as that are listed in a field called `images` in the field context of a record. 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 images 75%}}
```

The normal slider uses resized thumbnails as source files. It is also possible to get the original image in a slider as follows:

```
{{#sliderLarge images 75%}}
```

3.3.2 Sanitize Values

In order to use values in a query string, it is good practice to sanitize those before as follows:

```
{{#sanitize field}}
```

3.3.3 Regex Search and Replace

The `replace` helper can be used to search and replace within a field value using Regex:

```
{{#replace "/regex/", "replace", field}}
```

3.3.4 JSON Output

To get the actual context data instead of a rendered HTML output, you can render JSON output instead. To get all data simply use this as your template:

```
{{#json this}}
```

To just get the records for example, you can use:

```
{{#json records}}
```

3.3.5 Markdown

Rich text content in Airtable fields is returned from the Airtable API in the Markdown format. In order to convert such content to actual HTML, the *markdown* helper can be used as follows:

```
{{#markdown field}}
```

3.3.6 If Equals

In case you quickly want to compare a field value with any other value or string you can use the `if==` helper:

```
{{#if== field, "value"}} ... {{/if==}}
{{#if== field, otherField}} ... {{/if==}}
```

Note that it might sometimes be required to compare **sanitized** values. This can be done as follows as well:

```
{{#ifsan== field, "value"}} ... {{/ifsan==}}
{{#ifsan== field, query.field}} ... {{/ifsan==}}
```

3.3.7 If Not Equals

The counterpart to `if==` helper is the `if!=` helper that lets you check for inequality:

```
{{#if!= field, "value"}} ... {{/if!=}}
{{#if!= field, otherField}} ... {{/if!=}}
```

To compare sanitized values, please use:

```
{{#ifsan!= field, "value"}} ... {{/ifsan!=}}
{{#ifsan!= field, query.field}} ... {{/ifsan!=}}
```

3.3.8 Each Loops

Handlebars provides a great feature to enhance the use of lists. While it is possible to simply loop over items like:

```
{{#Type}}
    {{Name}}
{{/Type}}
```

You can alternatively use the `{{#each Type}} ... {{/each}}` helper to get more access to built-in data variables like `@first`, `@last` and `@index`. This is for example very useful in case you need to concatenate a list of items with a comma:

```
{{#each Type}}
    <i>{{Name}}</i>{{#unless @last}},{{/unless}}
{{/each}}
```

You can find more about the use of data variables in here.

3.3.9 Sorted Loops

The `sorted` loop works exactly like a normal `each` loop except the fact that all items are sorted.

```
{{#sorted Category}}
    {{this}}
{{/sorted}}
```

As a second argument, a key can be specified in order to sort an array of objects by the value for the given key.

```
{{#sorted Type, "Name"}}
    <i>{{Name}}</i>{{#unless @last}},{{/unless}}
{{/sorted}}
```

3.3.10 Unique Loops

The `unique` loop works exactly like a normal `each` loop except the fact that duplicate items are ignored.

```
{{#unique Type}}
    <i>{{Name}}</i>{{#unless @last}},{{/unless}}
{{/unique}}
```

3.3.11 More Handlebars Helpers

Aside from the examples above, Handlebars offers even more helpers that can be used in templates such as `with`, `if`, `unless` and others. You can find the documentation of those features as well on GitHub.

3.4 Partials

In order to use Handlebars partials, you have to define an absolute path in relation to your Automad directory as the value for the `partials` parameter when creating a new Airmad instance. All `*.handlbars` files in that directory can be used as partials as follows:

```
{{> myPartial }}
```

CHAPTER

FOUR

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 `:[prefix]Page` and `:[prefix]Pages` runtime variables. As mentioned before, please replace *:[prefix]* with a unique string in the Airmad options.

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

    <@ if @{ ?Page } < @{ :prefixPages } @>
        <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 and change the prefix. The classes in use will work out of the box with the **Standard** and **Adam** themes.

CHAPTER

FIVE

FILTERS

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

```
<form action="">
    <input type="text" name="Category">
    <input type="text" name="Client">
</form>
<@ Airmad/Airmad {
    base: 'appXXXXXXXXXXXXXX',
    table: 'Design projects',
    view: 'All projects',
    filters: 'Client, Category',
    linked: 'Client => Clients',
    template: '
        {{#records}}
            {{#fields}}
                <div class="card">
                    <h3>{{Name}}</h3>
                    <p>
                        {{#Client}}
                            {{Name}}
                        {{/Client}}
                    </p>
                </div>
            {{/fields}}
        {{/records}}
    ',
    limit: 20,
    page: @{ ?Page | def(1) }
} @>
```

In the snippet above, we have a simple form at the top including two input fields with the names `Category` and `Client`. 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 **Client** is a linked table, defining the `linked` parameter allows for searching in linked records as well.

5.1 Exact Matches

By default, all items with a field that contain the filter string are included in the list of matched records. However in case you prefer to only include exact matches where the filter string equals the actual field value or one of the field values, the filter value has to be wrapped in double quotes.

So basically a query string like

```
https://domain.com?name=value
```

has to be changed to

```
https://domain.com?name="value"
```

to only match records with a `name` that is equals `value`.

5.2 Autocompletion

To enhance the user experience for your visitors, you might want to provide an autocompletion list of categories for the **Category** input and **Client** names for the second input field. The Airmad data model contains a **filters** element at the top level for such purpose. It contains lists of records that are contained in one ore more items in **records** for each name defined in the `filters` option. In the following example, such **filters** element contains all **Client** elements that match any record in the **records** list. Note that it is also possible to just have a reduced list of filters that actually match an already filtered set of records. Such a reduced list can be accessed by using the **filteredFilters** element. You can use filters as follows:

```
{{#with filteredFilters}}
    <form action="">
        <input type="text" list="Categories" name="Category">
        <datalist id="Categories">
            {{#each Category}}
                <option value="{{this}}">
            {{/each}}
        </datalist>
        <input type="text" list="Clients" name="Client">
        <datalist id="Clients">
            {{#each Client}}
                <option value="{{Name}}">
            {{/each}}
        </datalist>
        <button type="submit">Apply</button>
    </form>
{{/with}}
<@ Airmad/Airmad {
    base: 'appXXXXXXXXXXXXXX',
    table: 'Design projects',
    view: 'All projects',
    linked: 'Client => Clients',
    filters: 'Client, Category',
    template: '
        {{#records}}
            {{#fields}}
```

(continues on next page)

(continued from previous page)

```
                <div class="card">
                    <h3>{{Name}}</h3>
                    <p>
                        {{#Client}}
                            {{Name}}
                        {{/Client}}
                    </p>
                </div>
            {{/fields}}
        {{/records}}
    ',
    limit: 20,
    page: @{ ?Page | def(1) }
} @>
```

CHAPTER

SIX

EXAMPLE

The following example is supposed to wrap all features of **Airmad** like getting records, filtering and building a pagination navigation. 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!

6.1 Using a Block

You can simply paste the following snippets to a snippet block — don't forget to change the base ID. The first part is the template string:

```
<@ set {
    :base: 'appXXXXXXXXXXXXXX',
    :tmplt: '
        <form id="example" action="" class="uk-flex uk-flex-space-between">
            {{#with filteredFilters}}
                <input
                class="uk-button uk-width-medium-3-10"
                type="text" list="Categories"
                name="Category"
                placeholder="Category"
                value="{{../query.Category}}"
                >
                <datalist id="Categories">
                    {{#each Category}}<option value="{{this}}">{{/each}}
                </datalist>
                <input
                class="uk-button uk-width-medium-3-10"
                type="text" list="Clients"
                name="Client"
                placeholder="Client"
                value="{{../query.Client}}"
                >
                <datalist id="Clients">
```

(continues on next page)

(continued from previous page)

```
                    {{#each Client}}<option value="{{Name}}">{{/each}}
                </datalist>
            {{/with}}
            <button class="uk-button reset uk-width-medium-3-10"> Reset</button>
        </form>
        <div class="am-stretched grid cards">
            {{#records}}
                {{#fields}}
                    <div class="card">
                        <div class="card-content uk-panel uk-panel-box">
                            <div class="uk-panel-teaser">
                                {{#slider Project images}}
                            </div>
                            <div class="uk-panel-title">
                                {{Name}}
                            </div>
                            <p>
                                {{#Client}}
                                    {{Name}}
                                {{/Client}}
                            </p>
                        </div>
                    </div>
                {{/fields}}
            {{/records}}
        </div>
    '
 } @>
```

The next snippet contains the actual plugin initialization of Airmad:

```
<@ Airmad/Airmad {
    base: @{ :base },
    table: 'Design projects',
    view: 'All projects',
    linked: 'Client => Clients',
    filters: 'Client, Category',
    template: @{ :tmplt },
    limit: 8,
    prefix: ':design',
    page: @{ ?Page | 1 }
} @>

@{ :designOutput }
```

Now we can add the pagination:

```
<ul class="uk-pagination">
    <@ if @{ ?Page } > 1 @>
        <li><a href="?<@ queryStringMerge { Page: @{ ?Page | -1 } } @>">←</a></li>
    <@ end @>
    <@ for @{ :designPage | -4 } to @{ :designPage | +4 } @>
```

(continues on next page)

(continued from previous page)

```
        <@ if @{ :i } > 0 and @{ :i } <= @{ :designPages } @>
            <li><a href="?<@ queryStringMerge { Page: @{ :i } } @>" <@ if @{ ?Page |␣
↪def(1) } = @{ :i } @>
                class="uk-active"
            <@ end @>>@{:i}</a></li>
        <@ end @>
    <@ end @>
    <@ if @{ ?Page } < @{ :designPages } @>
        <li><a href="?<@ queryStringMerge { Page: @{ ?Page | +1 } } @>">→</a></li>
    <@ end @>
</ul>
```

And finally, we need a tiny bit of Javascript too:

```
<script>
    let form = document.getElementById('example'),
        inputs = form.querySelectorAll('input'),
        reset = form.querySelector('.reset');

    inputs.forEach((input) => {
        input.addEventListener('change', () => {
            form.submit();
        });
    });

    reset.addEventListener('click', () => {
        inputs.forEach((input) => {
            input.value = '';
        });
        form.submit();
    });
</script>
```

6.2 Loading a Snippet

The above example is also included as a snippet file in the Airmad repository. Like the example above, the snippet is tailored to work with Airtable's default **Project tracker** base. After adding the Airtable token as described before, you can load the `example.php` snippet in a block. To actually make it work with the base in your account, as a last step you have to define the base ID as follows right before the **Include Snippet File** dropdown in the block editor:

```
<@ set { :base: 'appXXXXXXXXXXXXXX' } @>
```