Argon

The Plug ‘n Play Backend

Documentation

Getting Started

Setup

Download Argon as a .zip archive here: http://link.com/argon.zip. Extract **argon.js** and **argon.php** from the archive.

Upload **argon.js** and **argon.php** into a **writable directory** on the web server.

Add **argon.js** to the <*head*> of any web page stored on the *same* domain.

> *Note: To use Argon across multiple domains, see documentation within argon.php.*

In a script loaded after **argon.js**, add the following code.

```
var myServer = new Argon('url.to/argon.php');
```

*The first parameter of Argon() should be set to the location of **argon.php**.

In a browser window, load the url of a web page containing **argon.js**. In the Javascript console of the browser, run the following code.

```
myServer.get(function(data) { console.log(data) });
```

After a moment, the following should be logged to the Javascript console.

```
Object {argonError: "user is empty"}
```

If the above was logged, **Argon is now set up on the web server**. If the above was *not* returned, please see section “Troubleshooting Setup”.

The Basics

Users

Users are accounts within the Argon installation that can be assigned to a person, or simply used to separate data.

To create a user, use the following code.

```
myServer.user.create('username','password');
```

*Optionally, a callback may be defined in the third parameter to receive information on the attempt. see section "What is a callback?" for information on callbacks.

Once a user is created, it may be signed into using the following code.

```
myServer.user.signin(‘username’,’password’);
```

*Optionally, a callback may be defined in the third parameter to receive information on the attempt. see section "What is a callback?" for information on callbacks.

*When a user is signed into, it will be remembered indefinitely, unless signed out of.

Once a user has been signed into, data can be written and received.

```
.user.signout();
```

*Optionally, a callback may be defined in the first parameter to receive information on
the attempt. see section "What is a callback?" for information on callbacks.

The above method signs out of the current user. Authentication is required to login again.

Writing and Receiving Data

To write to a user’s storage, use the following code.

```
myServer.update(data);
```

The first parameter of .update() should be data contained within an **object**. All data stored in Argon is stored as json objects.

To read a user’s storage, use the following code.

```
myServer.get(callback);
```

The first parameter of .get() should be a callback function. When this command is sent, Argon will return the user’s data to the callback. see section "What is a callback?" for information on callbacks.

Advanced

User Methods

In the “Basics” section, a few user methods were discussed. In this section, we’ll discuss some of the more advanced and powerful methods that can be used on users.

```
.user.password(‘newpassword’);
```

*Optionally, a callback may be defined in the second parameter to receive information on the attempt. see section "What is a callback?" for information on callbacks.

The above method changes the current user’s password to the one defined in the first parameter.

```
.user.remove();
```

*Optionally, a callback may be defined in the first parameter to receive information on the attempt. see section "What is a callback?" for information on callbacks.

The above method does what you would assume: it removes the current user. When a user is removed, it cannot be restored.

User Permissions

Sometimes you may need a way of sharing data between two or more users, allowing a certain user to edit the current user’s data, or some other combination. To accomplish this, user permissions may be used.

The basis of user permissions is the method .user.permissions(). This method can be used to allow, block, or filter many different actions. The following is a table of acceptable permissions.

Permission Property	**Value**	**Outcome**
“.get”	*true*	allows everyone to use .get() on current user
	false	blocks everyone from using .get() on current user

	[‘username’,’username’]	allows certain users to use .get() on current user
“.update”	*true*	allows everyone to use .update() on current user
	false	blocks everyone from using .update() on current user
	[‘username’,’username]	allows certain users to use .update() on current user

Example .user.permissions() usage:

```
mySerer.user.permissions({ “.get”: false, “.update”: [‘benhmoore’,’heatherfeather’] });
```

*Optionally, a callback may be defined in the second parameter to receive information on the attempt. see section "What is a callback?" for information on callbacks.

When the method .user.permissions() is run, the permissions are set for the *currently signed in user*. Permissions *cannot* be set for a user other than the current.

In order to run methods against another user which has given you certain permissions, the following variations of the .get() and .update() methods are used.

To use .get() on another user’s (who has given the current user permission) data, use the following code:

```
myServer.get(callback,’username’);
```

To use .update() on another user’s (who has given the current user permission) data, use the following code:

```
myServer.update(data,’username’);
```

*In the above example, a callback (if used) should be *pushed to the third parameter*, instead of the second parameter.

Configuring the Backend

Config.json

Sometimes it may be necessary to prevent certain actions from being carried out on a server. For instance, you may not want to allow new users to be created, or you might not want certain users to use the .update() method. To define these rules, config.json can be used.

To get started, create a json file called *"config.json"* in the *"backend"* directory. *This directory is a subdirectory created by* ***argon.php***, it is located within the directory you uploaded **argon.php**.

Inside the config.json file, begin with this basic syntax:

```
{

    "*" : {
        "*" : {

        }
    }

}
```

As you can see, it is simply a basic json file. Inside the object that is formatted in red above, you can allow or block actions. **Here's an example:**

```
{

    "*" : {
        "*" : {
            "update": false,
            "removeUser": false
        }
    }

}
```

The above example blocks the use of .update() on the server, it also blocks any attempts of removing a user.

Here is a list of actions you can block:

“set”
“get”
“generateAuthToken”
“changePass”
“updateOther”
“update”
“setPermissions”
“getPermissions”
“removeUser”
“createuser”
“getOther”

The “*” sign tells Argon that anything inside is applied to all users. You may add properties with the usernames of specific users to apply user specific rules. **Here’s an example:**

```
{

    “*” : {
        “*” : {
            “update”: false,
            “removeUser”: false
        },
        “user_ben” : {
            “update”: true,
            “removeUser”: false,
            “getOther”: false
        }
    }

}
```

More

Troubleshooting Setup

Setup is the single most error prone point in time for Argon. If "*Object* {*argonError: "user is empty"*}" was not logged to console when testing your Argon installation, try the following:

1. Make sure the permissions for the directory you uploaded **argon.php** and **argon.js** are set to "***0777***" to grant **argon.php** write permissions.
2. Make sure the url to **argon.php** is correctly typed when defining a new instance of the Argon() on the client.
3. Check the console for network errors.
4. Make sure your server supports the latest version of PHP.

What is a callback?

A callback is a function or method. When data is returned from the server, it is passed to the **first parameter** of the callback. Here's an example:

```
var logIt = function(data) {
    console.log(data);
};
myServer.get(logIt);
```

The function logIt() is considered the callback.

If one or more of your questions was not answered in this document, don't hesitate to email us!
contact@loadfive.com.