

Adapt Authoring Tool

Server Refactor Proposal

06.02.2018

prepared by Tom Taylor

# **Overview**

The aim of this document is to outline an approach to improving the structure of the ~~node-based server component of the Adapt authoring tool.~~ node-based server component of the Adapt authoring tool.

# **Goals**

1. To lower the barrier to entry/reduce the overhead of working with core code for new and existing developers.
2. To facilitate a plug-in based architecture, and therefore reduce the need for major core changes in the future.
3. To improve the stability of the code, and allow more effective automated testing.
4. To expose a consistent and reliable public API for both internal use by the application and to third-party code.

# **Proposed changes**

## **Folder structure**

Something

## **Routing**

The entire routing strategy needs to be overhauled to allow both better readability and more complex function.

With regards to readability, I propose we separate the public API from the ‘controller’ code that uses it. One of the biggest problems with regards to the routing/API is that it’s hidden away in huge manager files.

In terms of functionality, there are various enhancements we can make to extend the existing server, and provide a more stable application. Most of these involve utilising existing Express 4 features such as sub-routers and middleware.

### **Middleware**

Resource permissions

## **‘Action’ hooks**

Implement a consistent, system-wide hook-based approach to core code to allow easy integration with system actions. These actions are completely arbitrary, but will likely be predominantly CRUD. A similar system is already available for content plugins. The interface should be easily added to existing functions and objects (possibly using Node.js’ utils.inherit function. The interface should allow both a type and a SOMETHING to be specified.

Potential use-cases:

* + User: CRUD
  + Output: publish, preview …

# **Other areas for consideration**

## **Lorem ipsum**

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

## **Dolor sit amet**

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

# **Action hooks**

* Decorator pattern
* Pre, post
* Generic hook for adding, removing hooks
* CRUD wrapper - content, users
* Output

# **root**

Everything contained directly in this folder should be relevant to the entire application. Anything specific to the front-end/back-end should now be found in their respective folders.

Files remaining in this folder are:

* various config files (.editorconfig, .travis.yml etc.)
* install.js
* upgrade.js
* server.js
* package.json
* README.md
* LICENSE
* ... you get the idea

Note: I'd like to look into moving the install/upgrade/server commands to the task-runner we choose.

# **backend**

This folder now contains everything related to the back-end server application.

### **api**

This folder will contain the public-facing API of the application. The majority of the code in here will be concerned with routing requests, the business logic will still be handled by the 'manager'/controller files in lib (to make sure we separate concerns properly, it may make sense to separate out the 'controller' code from lib into individual controller files, which could go into here too). I intend to use express 4 sub-routers to do this.

This folder will replace the existing routes folder. I think it's also important that we make the application API flexible enough to allow these API 'plugins' to self-register/initialise, as the current solution of handling the loading/preloading/initialisation/whatever in the main application itself (e.g. application.js) is difficult to follow, and not in any way self-documenting (which is something it always helps to aim for).

Depending on how much stuff we have related to individual APIs, the folder may contain sub-folders, or just a .js file for each api router.

e.g.  
// just files  
assetRoutes.js  
userRoutes.js  
// folders  
asset/  
  assetRoutes.js  
  assetModel.js  
  assetSchema.js  
  assetController.js  
  ... blah  
user/  
  userRoutes.js  
...etc.

### **conf**

I'd like to do more with this folder, as it's not all that useful at the moment. I think it would be useful to make a bit more of this, and create multiple levels of config (at least for each type of environment: dev, prod, test), and store all options in here. As it is we already have a testConfig.json in test/ which should probably be in here.

We could also hook into process.env.NODE\_ENV or something to set/determine this, and load the suitable file.

I.e. require('./' + env);

### **lib**

This folder contains reusable (and unit-testable!) libraries that are shared and used throughout the app. This folder will largely remain the same is it is now. The main change here will be that all REST/API related code will be moved out into the api folder.

### **models**

*Big question mark on this one, as not sure it's needed.*

Contains all model data.

### **plugins**

I'm still pondering what to do with this folder. I'm not too fond of the plugin architecture we use, as I found it pretty impenetrable coming on to the project, and still feel that it's more complex than it needs to be without being flexible enough. I'm also aware of the potential avalanche of extra work/regression issues that will come with completely rearchitecting this.

Particular things I don't like:

* **The name:** For me, being called 'plugins' confuses things for two reasons:
  1. The framework uses plugins; some things in here relate to framework plugins, most things don't.
  2. It gives the sense that this folder contains add-on/additional content/functionality. Everything in here is in fact core and required. With the front-end, I went with the name 'modules' -- this may work here.

1. **The API endpoints being hidden away deep in the folder:** Although I understand the need for this, it makes working with the API very cumbersome and difficult to follow. Considering everything in here is 'core', this seems slightly unnecessary.

### **test**

This folder will contain all of the tests for the back-end application, and will likely be the same as now.

# **tasks**

This folder will contain tasks related to the building/running of the application. Grunt is the chosen task runner currently, so these may all be grunt tasks.

# **temp**

This folder needs to be looked at. Current issues I see:

* Name: temp is a complete misnomer; delete this folder at your peril. Needs to be renamed, or rearchitected so that this is in fact temporary.