Adapt Authoring Tool

Server Update 08/19

09.08.2019

prepared by Tom Taylor

Contents

File/class Renaming **2**

Utilities **2**

Discussion Points 2

Config 2

TODO 3

Lang 3

Auth 3

TODO 3

API creation **3**

Course Output **3**

Docs **4**

Resources **4**

File/class Renaming

Certain files and classes have been renamed for clarity (particularly in the documentation).

Module => AbstractModule

I've also renamed all AbstractModule subclasses to include the name 'module', such as UsersModule etc. for added clarity. This is particularly relevant given the introduction of utilities (see below), as a single dependency can contain both an AbstractModule subclass and an AbstractUtility subclass.

Utilities

- A specific type of App dependency
- **Not** an AbstractModule (duh!), no create/preload/boot cycle
- Bits of functionality used frequently throughout the app (a reusable library of sorts)
- Needs to be initialised prior to any modules being loaded (as are used by many modules)
- Examples of the functions of utilities:
    - Logging error messages
    - Translating language strings
    - Retrieving user configuration options
- Certain types of utility are required for the app to function. At present, these are lang, config, logger and auth. If the app is missing implementations of any of these, it fails with an error. This makes it safe to write modules which assume the required utilities are present and correct. It also allows for easy replacement of the core-supported utilities.

Discussion Points

- The API-side is a bit too complex at present; I'd like to make it easier/shorter to call the utility code. At current: this.app.logger.log(), this.app.lang.t().
- The required API for each of the required utils should be defined in abstract classes.

Config

Allows user-specified options to be stored and loaded by the app.

- Files *.config.js from /conf are loaded according to the NODE_ENV env value
- Process.env values stored in config on startup
- Each module can include a config.schema.js to define what config options are required by their module (name, type, default value, is required). The app then compares the loaded config file to the schema data to ensure that the user has specified all required options, and fills in any blanks with default values. Anything missing will cause the app to quit with a useful message, removing the need for defensive checks (i.e. if(config.server.port) etc.). I also intend to add custom validation functions to extend this further, to allow for conditional settings.

TODO

- I'd like to remove the hard-coded path of /conf for the location of the config files for added flexibility.

Lang

- Language string translation using Polyglot
- Ability to specify multiple language files
- Lang files can be defined both in a module folder and the root repo
- Different languages can be provided for the back/front end apps
- Strings split by filename (error, info)

Auth

Basic implementation of an auth module (note that so far I've only implemented the authorisation component).

- All routes blocked by default
- Any route can be completely unsecured (i.e. access without any authentication/authorisation)
- Any route can be exposed to specific authorisation scopes
- Scopes are arbitrary strings (e.g. 'read:users')

TODO

- Discuss/agree upon format of scope strings

API creation

New AbstractApiModule class (AbstractModule subclass) which takes most of the work out of defining new APIs, and ensures consistency across the board.

Automates the following (where wanted):

- Adding new schemas to the DB
- Creation of sub-routers
- Adding generic handlers for POST/GET/PUT/DELETE methods
- Adding middleware
- Securing routes with auth utility

All of the above is completely optional, and configured via a static class property attached to the AbstractApiModule subclass, known as an ApiDefinition. See the CoursesModule#def as a simple example of this.

Course Output

Started to implement course output to create an Adapt course build from the source.

Problems left to solve:

- Serving a preview
    - Should be viewable by anyone (i.e. unsecured)?

- Changes to adapt-cli needed to allow us to consume it via an API rather than building directly via grunt.

Docs

Some custom 'plugins' have been added to ESDoc to auto-generate various pages based on the source code. These include:

- Configuration
- List of core modules
- External type reference

Resources

- Application Boot Illustration
- Module Boot Illustration

Adapt content (course, config, contentobject, page, article, block, component)

- CRUD (needs hooks/events)
- Delete needs recursion

Adapt Output

- build
  - get json
  - apply plugins
  - grunt
- import
- export
- preview
- publish
- download

Adapt plugins (component, extension, menu, theme)

- CRUD

getPluginTypes

getInstalledPlugins

getPlugins

getPlugin

isUpgradeRequired

installPlugin

uninstallPlugin

isInstalled

testPluginState

dependenciesInstalled

Assets

- CRUD (DB + filesystem, thumbs)

Auth

- Authentication middleware

- Finish authorisation
- Tokens
  - Create
  - Revoke
- Password resets

FileStorage
- read (+ dir)
- write (+ dir)
- remove (+ dir)
- copy (+ dir)
- processFileUpload?

Mailer
- send

Roles
- CRUD?
- Assign
- Unassign

Users
- CRUD