Adapt Authoring Tool

Server Rewrite MVP Definition

14.05.2018

prepared by Tom Taylor

Contents

Preamble

Definitions

Application: unless otherwise specified, the use of the term 'application' signifies the Node.js server component of the Adapt authoring tool.

Core: this term will be used to describe the set of code/features which define the open-source application. In the existing implementation, this applies to all code/features found in the *adapt_authoring* repository, but going forward will apply to all modules officially supported by the core development team.

Exclusions

This work will not introduce any new features to the system, but instead focus on recreating the existing feature-set in a more workable form.

This work will not include any changes to the front-end application (with the exception of any changes required as a result of modifications made to the server API.).

Overview

The aim of this work is to improve the architecture of the Node.js server application of the Adapt authoring tool to better accommodate the future goals of the Adapt project.

Goals

- To reduce the overhead of working with the core codebase for existing developers.
- To lower the barrier to entry for new developers thus encouraging greater collaboration.
- To allow easier customisation and extension of the core application by third-parties.
- To make the core codebase more maintainable.
- To reduce the need for major changes to the core codebase in the future.
- To improve the stability of the core codebase.
- **To recreate the existing feature-set**

Essential requirements

SRW-01: To split the application into modular units

SRW-02: To allow areas of the application to be replaced or enhanced by third-parties

SRW-03: The system should allow an arbitrary number of modules to be loaded into the application

SRW-04: To reduce the overall complexity of the codebase

SRW-05: To document code such that docs can be generated automatically

SRW-06: To produce a more 'readable' codebase (i.e. self-documentation)

SRW-07: To expose the public API in a transparent way

SRW-08: To treat security as a primary objective

SRW-09: To introduce project standards and conventions for the core team and third-parties

SRW-10: To remove the tight-coupling of particular technologies/libraries (in particular the database and file-storage mechanisms)

SRW-11: To provide more comprehensive test-coverage of the application

SRW-12: To adopt modern ES6 and Node.js language features and conventions

To facilitate the future switch to ES6 in the front-end application

Additional requirements

SRW-13: To coordinate with the Adapt framework technology stack (where appropriate)

SRW-14: To integrate with the Adapt CLI tool

Design & architecture

This section will discuss in more detail the proposed architecture of the system, including any changes to the technology stack.

Technology

Node.js

In order to strike a balance between having access to the latest JS/Node.js language features and overall stability, we have decided to support the latest LTS release of Node.js (at time of writing v8). This will give us access to:

1. Major performance improvements (both to Node.js and NPM)
2. Enhanced debugging toolset
3. Improved security/stability through the updating of existing dependencies which have been deprecated by their developers

The latest ES6+ language features (e.g. async/await, spread, destructuring parameter defaults….)NPM

Adopting a truly modular architectural approach brings with it extra logistical issues around installation, updating and dependency resolution. To combat this, we will use NPM to manage the system dependencies as well as the standard Node package dependencies. We can take advantage on NPM's **orgs** feature to group all Adapt packages, and allow multiple users to publish under the Adapt name..

System architecture

We have proposed a completely modular architecture, with all of the application functionality being contained in separate repositories, and managed by NPM.

Taking a fully modular approach like this will allow us to:

- Enable a true separation of concerns by decoupling all separate pieces of functionality, resulting in more focussed code, documentation and unit tests
- Promote third-party code, as there will be no distinction between what is considered 'first' or 'third' party
- Allow parts of the application to be easily 'swapped' for other modules should the need arise
- Allow third-party developers to override/replace entire chunks of the application without needing to alter the core code and risk creating an incompatible 'fork'
- Keep the application 'lean' by removing the need to add code to the 'core'
- Allow more frequent updates to 'core' functionality without the need for formal application-level releases (i.e. using SEMVER ranges in package.json to allow some updates via NPM[1])
- Allow developers to specialise in certain areas of the application, and assign 'product owners' to allow for better maintenance

[1] https://docs.npmjs.com/getting-started/semantic-versioning

Structure breakdown

With the above in mind, The existing **adapt_authoring** repository will become a 'container' for the metadata required by the application, the most important of which being the list of dependencies required by the application.

In addition to the dependency definition, this repository will also contain some basic documentation and the scripts required to install, update and start the application.

Core

The core module will be a very lightweight collection of abstract classes and utilities to be shared throughout the application. This module in itself will not provide any 'function', but will be consumed by other modules..

Modules

All functionality of the application will be found in individual 'modules' managed by NPM, **require**d and initialised by the top-level application.

Module requirements:

- The system should allow each module to run arbitrary 'initialisation' code where required
- Modules should be able to define an external API
- Modules should contain all necessary documentation
- Modules should define their own unit tests

It is important to note at this point that there will be no distinction between modules supported by the core development team (i.e. first-party) and third-party modules; all modules will be treated as first-class citizens, and have access to the core module. This allows us to better enable the community to develop their own functionality.

Module definition

To keep the system as flexible as possible, there will be few hard restrictions on what defines a module. Instead we will add a core interface which exposes the needed functionality, and can be extended by the developer if required.

Core modules

The existing application covers the following main areas which will need to be replicated.

- Express server
- Frontend application
- Users
- Logger
- Configuration
- Authentication
- Mailer
- Tenants
- Assets

- Database/Mongo
- Filestorage/localfs
- Adapt/output
- Content:
    - Theme
    - Courseasset
    - Course
    - Contentobject
    - Config
    - Component
    - Clipboard
    - Bower
    - Article
    - Extension
    - Menu
    - Tag

Documentation

To further drive adoption and reduce the barrier to entry for new developers, we are looking to introduce documentation-generating tools to allow us to automate the creation of API documentation without a big overhead on the part of the development team.

The main requirements for the documentation tool are as follows:

- Can automatically generate documentation in HTML format when given ES6 code input
- Produces easily readable/navigable output
- Produces searchable output
- Can be configurable to selectively include/ignore files
- Has an good quality inference engine, so as not to require excessive comment annotations

More information on possible solutions can be found in the resources section,

Resources

Prototype

You can find a working prototype for the new application architecture here:

Main application:

- adapt_authoring

Modules:

- adapt-authoring-core
- adapt-authoring-server
- adapt-authoring-helloworld
- adapt-authoring-logger

Documentation generators

See below for tested examples of JS/ES6 documentation generators.

JSDOC

Website | Example

ESDOC

Website | Example

Documentation.js

Website | Example