Adapt Authoring Tool

Server Refactor Meeting Agenda

25.04.19

24.04.2019

prepared by Tom Taylor

Agenda

Prototype overview

Feedback on the prototype, with particular consideration to the following:

- **Approach**: general modular approach, module lifecycle (instantiation, preload, boot), inter-module comms.
- **Code style**: use of classes, async/await, events etc. particularly areas which work/don't work/should be expanded on.
- **Documentation**: how well the docs read/work, whether we need to investigate alternatives.
    - ESDocs manuals.

Architecture

Discuss the future architecture with respect to the modular approach, how existing functionality will be 'modularised' and how these modules will interact.

- Cover use-cases explored by the prototype:
    - APIs
        - Middleware
    - UIs/Theming
    - Inter-module comms
- Discuss any additional use-cases
    - Multitenancy/alternatives
    - Multiple modules of a similar type (e.g. data storage)
- Discuss coupling between modules, and any necessary abstractions:
    - Auth (authentication + authorisation)
    - Logging
    - Configuration
    - File storage
    - Data storage

Conventions

Defining a consistent coding style across the project as a whole, and how this can be enforced/automated using CI tools.

- Coding style rules
- Coding best practice
    - Module definition
    - RESTfulness
    - Error handling
- Documentation
- Unit tests

Next steps

Any work outstanding on the prototype, and what will be required going forward.

- Further areas for research
    - Auth
    - Content
    - Multitenancy (or alternatives)
    - Testing/CI
    - Any other tech
- Preliminary work
    - Prototype finalisation
    - Standards definition
    - Define MVP
        - Stretch goals
- Module development
    - Delegation

Meeting Notes

Attendees: Dan Gray (**DG**), Tom Greenfield (**TG**), Paul Hilton (**PH**), Pete Smith (**PS**), Tom Taylor (**TT**), Nicola Willis (**NW**).

Prototype overview

Assess need for Hooks

An assumption has been made that the main application must be restarted for any new modules to be included.

Architecture

Aim to replicate the existing API verbatim for MVP (possibly versioned). Define second-phase API to tidy up inconsistencies.

Important: there's a need to customise how the app is structured, particularly with regards to scaling. Should be able to separate each component (e.g. front-end, RESTful API server, image/video manipulation?).

Database

Schema data currently has 3 different applications, and requires separation:

- Content data (i.e. required by the framework)
- App data (e.g. tags, hero image)
- API data (e.g. custom LESS, theme settings)

Concurrency

Revise approaches for concurrency control in MongoDB[1]. One option being **optimistic**[2], where any DB operations will fail if the state has changed since the request was made. This should be satisfactory on the API/backend side.

From the front-end, we should look to only save the data which has been updated using `PATCH`-style requests rather than full `PUT` requests. MongoDB has various features that will ensure the likelihood of any concurrency issues are very low, such as a `$pop/$push` for array types, `$inc` for incrementing, etc.

Abstraction/API layers

Need to weigh up the need for separation between front/back end module elements (i.e. are we looking to implement a 'pure' REST/API server and completely separate any front-end elements into separate modules? If so, there will be more overhead for the community. **Regardless of the solution, we will need build steps for each**.

Common app layers (use-cases for abstraction):

[1] Concurrency: MongoDB FAQ
[2] Optimistic concurrency control: Wikipedia

- Authentication
- Authorisation
- Asset services (image/video processing)
- Asset library

- Output (preview/publish)
- Front-end application

Multitenancy

No to multi-tenancy. Need to discuss requirements. Possible solutions:

- Single sign on
- Content sharing
- Shared assets

New tech

Generally a no to new tech for the back-end (besides ES6).

Conventions

Introduce CI tools to enforce consistent code style, and run unit tests.

Next steps

- List any needs for multi-tenancy, and alternative solutions
- Define API:
  - List existing API (this will become MVP)
  - Map existing API into logical modules
  - Define 2nd-gen API
    - Highlight new API functionality we want to include in 'core'
- Document second-generation API
  - Assess risks to front-end
- Research authentication (keep local?)
- Research authorisation
- CI tools:
  - Code style
  - Unit tests
  - Warnings for TODO comments
- Approaches to multi-lang (still a consideration?)
  - App translation
  - Content translation