Adapt Learning: Adapt Framework Concept and Vision

# Document control

|  |  |  |  |
| --- | --- | --- | --- |
| **Abstract:** | Describes the concept of the Adapt Framework | | |
| **Author:** | Sven Laux, Daryl Hedley, Paul Welch | Version: 0.3 | Date: 27 / 11 / 2013 |

| **Summary of Changes:** | **Versions** | **Date** | **Description** |
| --- | --- | --- | --- |
| 0.1 | 12 / 11 / 2013 | Initial draft for review. |
| 0.2 | 25 / 11 / 2013 | Updated concept diagram and added descriptions up to Glossary node. |
|  | 0.3 | 27 / 11 / 2013 | Added Daryl’s technical and Paul’s component descriptions. |

Document control 1

Purpose of document 3

What is the Adapt Framework? 3

Concept overview diagram 4

Explanation 5

# Purpose of document

The purpose of this document is to outline the vision and concept of the Adapt Learning Framework. It contains a concept diagram, similar of a mind map of the key functionality, components and elements of the underlying code framework, which make up the programming logic of the e-learning course.

This document is not a specification document. It is intended to help the project team and wider community understand the full product we are aiming for. As such, the document will therefore set the context in discussions about requirements, system architecture, specification etc.

The document is also intended to help newcomers to the project to get an overview.

# What is the Adapt Framework?

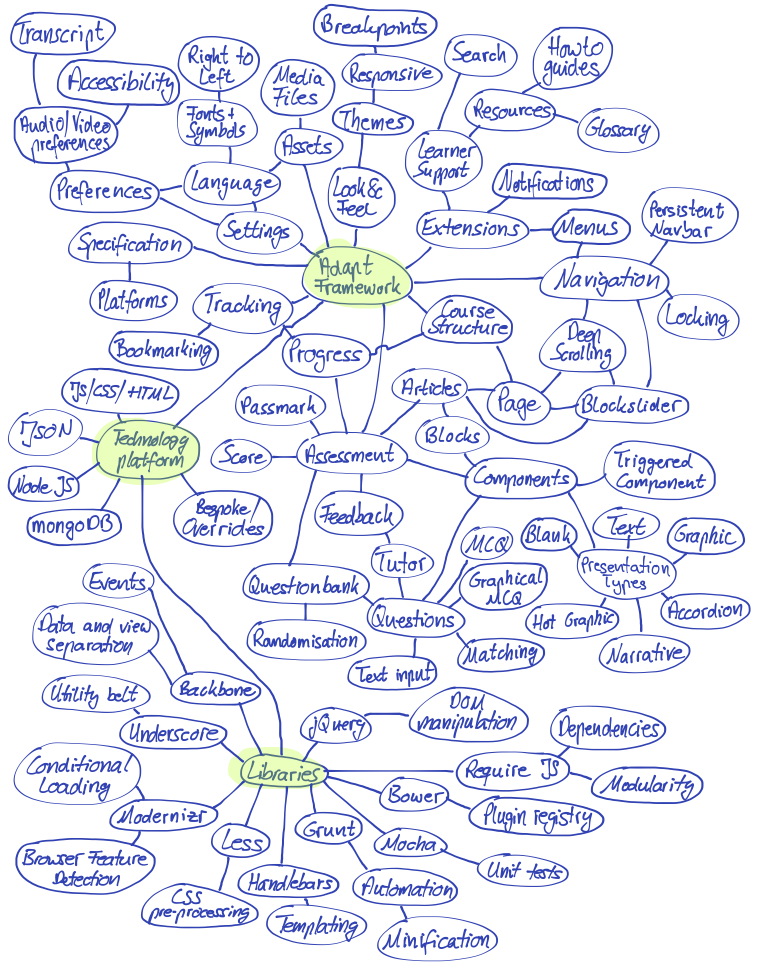
What is it?

How does it relate to the authoring tool?

Who uses it?

How can / should it be used?

# Concept overview diagram



# Explanation

|  |  |  |
| --- | --- | --- |
| **ID** | **Node title** | **Description** |
|  | Adapt Framework | The Adapt Framework is the program logic of a published e-learning course, which runs in the browser. It is the generic code, which makes up the output of the authoring tool. |
|  |  |  |
|  | Look & Feel | The appearance of the e-learning course, which includes branding, art direction themes etc. |
|  | Themes | A theme is the generic look and feel. It captures display settings, which apply throughout the course. This includes CSS styles, icons, base colours and background images. Components and extensions inherit the base colours and generic icons but have to define their own layouts. |
|  | Responsive | Responsive design is at the heart of the Adapt Framework and enables single version output, which adapts according to the device resolution and capabilities. We have implemented responsiveness is by defining pixel-width breakpoints (*see below*). The Adapt Framework is fully responsive, meaning that as the window width increases or decreases, the content adjusts automatically. If the width (in pixels) goes beyond a breakpoint, different styles and classes are applied to the content. |
|  | Breakpoints | Breakpoints are defined in numbers of pixels. They determine points in the width of the screen, at which different styles are applied to the on-screen content. There are three major breakpoints, which roughly reflect the three m main device types (smartphone, tablet and laptop/desktop). |
|  |  |  |
|  | Settings | Settings reflect the configuration options of the generic codebase. |
|  | Language | The language is one of the main configuration settings. Courses can be packaged to contain multiple languages within a single course package. The language setting determines the default language to be used and whether to display the option of choosing the language to the end learner. |
|  | Fonts & symbols | Fonts and symbols are a consideration as special characters or particular fonts are not ‘websafe’ and may have to be shipped with course. This node is in the concept diagram to remind us of particular requirements when working with languages other than English. |
|  | Right-to-left | Depending on the language, the reading order may be right to left, e.g. in the case of Hebrew. This node is in the concept diagram to remind us of particular requirements when working with languages other than English. |
|  |  |  |
|  | Assets | ‘Assets’ refers mainly to content images or animations used in the course. In most cases, the assets will be language independent. However, on occasion there may be differences in the imagery used for different languages, e.g. where a graphic contains text or where different graphics are used when localizing (as opposed to simply translating) a course. |
|  | Media files | Media files are also assets but refer more specifically to audio / video files, which are used in the course. Media files are more likely to be language dependent, especially when they contain an audio track. |
|  |  |  |
|  | Preferences | Preferences refer to settings the learner can change in order to adjust the display and behavior of the course. This includes audio/video and accessibility related preferences (such as high contrast look & feel) in particular. |
|  | Audio / video preferences | Audio/video preferences enable learners to change the default behavior of audio / video assets for a course. This includes enabling/disabling audio, setting the volume and deciding whether a written transcript for audio should be displayed alongside (or instead of) the media file. |
|  | Accessibility | Accessibility settings/preferences are intended to help users with visual or motor impairments access the course. |
|  | Transcript | A transcript is the content of the audio track (in particular) expressed as on-screen text. It enables delivery of the content to learners who are not able to play back or hear / understand the audio track. |
|  |  |  |
|  | Extensions | Extensions are a type of plug-in. The purpose of extensions is to enable developers to extend the functionality of the course without having to modify core code. Extensions contain functionality, which is not directly embedded in the article / block / component structure. For example, a course glossary and the ‘Tutor’ to deliver feedback. |
|  | Learner Support | Learner support is functionality that provides additional information, outside of the core content presented within the Adapt pages. Examples might be glossary, resources, page level progress and a search feature. |
|  | Search | The search functionality enables learners to search the course for specific content. |
|  | Resources | Resources are additional and related content items, such as reference materials and background reading. They may exist as part of the course or in downloadable format (e.g. as PDF documents). |
|  | How-to guides |  |
|  | Glossary | Reference section showing terms used in the course and their descriptions. |
|  |  |  |
|  | Notifications |  |
|  | Menus |  |
|  | Navigation |  |
|  | Persistent Nav bar |  |
|  | Locking |  |
|  |  |  |
|  | Deep scrolling |  |
|  | Blockslider |  |
|  |  |  |
|  | Course structure |  |
|  | Page |  |
|  | Articles |  |
|  | Blocks |  |
|  | Components |  |
|  | Triggered component |  |
|  |  |  |
|  | Presentation types |  |
|  | Blank | Creates single or spanned space on the page creating a window through to the background imagery. |
|  | Text | A single or spanned component. This is the only ‘proper’ component (see blank) that doesn’t have a widget consisting of a title and body. If spanned the same text will be ‘redrawn’ into a single sized component. |
|  | Graphic | A single or spanned component. If spanned the same image will scale down and be ‘redrawn’ into a single sized component (Adapt doesn’t load a smaller sized image). |
|  | Accordion | A single width component, which consists of a stack of clickable items.  Each item has a heading which, once selected, expands to reveal the accompanying items text. There is no limit on the number of items within an accordion but we’d recommend no more than six.  There are no graphics within each items display text area. |
|  | Narrative | A spanned component with a single width component fall-back. The learner can work through a sequence of images with an accompanying piece of display text via a forward icon (a back icon, post item one is also available). When the final item in the sequence is reached the default behaviour is to disable the forward arrow, not to take the learner back to the beginning of the sequence.  Each display graphic and text field is accompanied by a title.  The single spanned fall-back of narrative makes use of a clickable strapline which, when selected, triggers the display of the accompanying item text  On mobile phones the display text is presented in a full screen pop-up, which needs to be closed before the learner can continue. |
|  | Hot Graphic | A spanned component containing an interactive graphic (an image with clickable regions whose coordinates are set via the JSON).  When one of the items on the hot graphic is selected a window opens over the image containing some associated text and image that is relevant to the hot spot item selected.  The learner has the option of closing this window and choosing another item on the hot graphic or using discrete forward and backward navigation buttons within the display window to work through all the display items in order (the order will depend on the JSON structure).  For single spanned full-back, there is no mechanism for selecting items off the hot graphic. Instead the display window functions as a single span narrative. |
|  |  |  |
|  | Questions |  |
|  | MCQ | A single width multiple choice component. The body is used to display the question stem and the widget the options. One or more of these options can be marked as correct.  Feedback, delivered by tutor feature, can be either option specific or banded as correct, partially or incorrect. Any ‘number of attempts’ are possible but recommend two as standard. The tutor window must be closed to reattempt the question.  Indication of performance occurs via the placement of ticks and/or crosses. These icons are only located on the options the user has selected.  If multiple attempts are possible, then the ‘Submit’ button is replaced by ‘Reset’ once the answer is submitted. Selecting this button removes markings and resets the question, allowing for a further attempt.  Once the final attempt is made the reset button is replaced with the ‘View Model Answer’. Selecting this button allows the learner to toggle between their own answer and the perfect answer. In model answer ticks and crosses are present on every option.  There is no maximum number of question options but best practice would indicate a maximum of 5 or 6. |
|  | Graphical MCQ | A spanned or single width component. Very similar to MCQ in terms of functionality, the only difference being the learner selects one or more icons on an image rather than one or more options from a list. |
|  | Matching | A single width component that provides the learner with a stem presented via the component body, and then a series of statements. Each statement has an associated drop down which contains a series of options, one of which will be correct.  All option drop downs require the correct option to be selected to trigger the correct feedback. There is also partially correct and incorrect feedback available.  As with MCQ component the option marking, ‘Reset’ and ‘Model answer’ functionality is included.  There is no limit to number of statements/drop downs or the number of options within the list but best practice would dictate no more than 5 or 6 for each. |
|  | Text Input | A single spanned component.  As per matching, the question stem is presented via the component body but instead of statements with accompanying drop down lists, the learner is expected to enter their answers via a free text entry field.  Each free text entry fields entered answer is checked against a string and if present then it’s marked as correct. All free text entry fields need to be matched as correct to mark the component as correct. As with all questions, partial and incorrect feedback options are available. Number of attempts and number of question options are not limited other than by best practice.  The option marking, reset, model answer and number of attempts are also standard and as described above. |
|  |  |  |
|  | Assessment |  |
|  | Score |  |
|  | Feedback |  |
|  | Tutor |  |
|  | Questionbank |  |
|  | Randomisation |  |
|  |  |  |
|  | Tracking | The ability to track a users progress through a course. This is done through SCORM or Tin-Can/xAPI and enables LMS's to gain information about the user during a course. |
|  | Bookmarking | The ability to reload a course and take the user back to where they finished their last session. |
|  |  |  |
|  | Technology platform | The set of technologies (technology stack) that the Adapt Framework and the Adapt Authoring Tool are built upon. |
|  | JS/CSS/HTML | Adapt's core foundation technology. |
|  | JSON | All of Adapt's data is stored in JSON format. JSON is formatted in a key and value pairing. |
|  | Node JS | Node JS enables us to run an offline server on our development machines. Node powers our Grunt, Handlebars and Less compiling. Node JS is also the technology powering the Adapt Authoring Tool. |
|  | MongoDB | MongoDB is the NoSQL database that stores our document based JSON data for the Adapt Authoring Tool. |
|  | Bespoke / overrides | For developers we have a bespoke folder that enables us to do overrides / customisations of functionality without having to change the core code directly. |
|  |  |  |
|  | Libraries | We use a variety of open source code libraries at the heart of Adapts core. |
|  | Backbone | Backbone is a front-end library that enables us to separate our views and from our data. It comes with a built in events system and router to enable us to change pages. |
|  | Events | The Adapt framework is built upon an event system that enables our modules/classes to be separated and modular. One module should not talk to another module. Instead they trigger events that other modules can choose to listen to. |
|  | Data and view separation | By having a clear distinction between our models and views we're able to structure and maintain a growing/large framework. |
|  | Underscore | Underscore is the utility belt for Backbone. It enables Backbone to manipulate objects like Backbone.Model and Backbone.Collection. Underscore also comes with some handy methods that help deal with arrays/objects. |
|  | Utility belt | We use underscore when iterating over arrays or objects with \_.each or with Backbone.Collection.each() |
|  | Modernizr | Modernizr is a conditional loader and adds browser feature detection |
|  | Conditional loading | In the Adapt framework we conditionally load scripts based upon which browser the user is loading the course from. Modernizr comes with a built in yepNope conditional loader. |
|  | Browser feature detection | Modernizr attaches classes to the HTML tag based upon browser features. This can be used when styling or adding features and polyfills for different browsers. |
|  | Less | We use LESS as our CSS pre-processor. Less enables us to inherit and store variables that can be used across Adapt. |
|  | CSS pre-processing | CSS pre-processing enables developers to easily and quickly style courses, whilst inheriting styles, passing variables and nesting. |
|  | Handlebars | Handlebars is our core templating engine. We precompile our templates through a Grunt process to enable fast load times. |
|  | Templating | We use Handlebars to load our views HTML. This enables Adapt to be dynamic and act as a one page web app. |
|  | Grunt | Grunt is a node based JavaScript task runner. In Adapt, this is used for automation. |
|  | Automation | We use Grunt to automate our tasks. We use minification, handlebars and less compiling and requireJS optimisation dependency loading. |
|  | Minification | Running $ grunt build - a developer is able to minify all files including CSS and JS files. Minification means removing formatting, which is important for code-readability but not for machine processing, and thus making the code more efficient. |
|  | Mocha | Mocha is a Behaviour Driven Development framework that enables us to run unit tests across our code. |
|  | Unit tests | Unit tests enable us to test single methods or functions in our code to make sure that produce the correct output. This is an important measure in order to achieve high quality code. |
|  | Karma (*not on diagram yet*) | We use Karma as a spec runner that runs our unit tests in a variety of browsers. This also integrates with Travis-ci and runs test through a headless browser known as PhathomJS. |
|  | Bower | Bower is a front-end package manager. We wrap Bower in an Adapt-cli (command line interface) that enables developers to quickly download and install Adapt and plugins. |
|  | Plugin-registry | We have our own plugin registry that enables developers to register their plugins (components, extensions, menus or themes). Once registered these plugins are available through the adapt-cli by running commands like $ adapt install adapt-contrib-text |
|  | Require JS | RequireJS is an AMD (asynchronous module definition) module loader. It enables us a greater sense of modularity and dependency loading. |
|  | Modularity | Adapt is built upon a modular approach where modules shouldn't talk to each other and instead trigger events. This enables us to have a plugin architecture and smaller modular files. |
|  | Dependencies | The RequireJS optimizer loads all of the Adapt dependencies in the correct order so we don't have dependency collisions or unnecessary script tags in the index.html file. |
|  | jQuery | jQuery is a cross-browser compatible DOM manipulation library. It enables us to write less code whilst working across all the currently supported browsers |
|  | DOM manipulation | DOM manipulation allows us to move or affect DOM elements in the browser. |