Learnster.co.uk Documentation

Submitted in partial fulfilement of the B.Sc. in Information Systems Development

Anthony Troy (C10345703) | April 2014





# Declaration

I hereby declare that this project documentation is my own work and effort and that it has not been submitted anywhere for any award. Where other sources of information have been used, they have been acknowledged.

Signature: ………………………………………

Date: ………………………………………………

Table of Contents

Declaration 2

Chapter 1: Introduction 4

What is Learnster? 4

Learning Management Systems: What are they? 4

The Commodification of Learning Platforms: How is the market? 4

The Big Picture: Developing the Learnster business case 5

Chapter 2: Requirements Analysis 7

What is Learnster? Error! Bookmark not defined.

Bibliography 8

# Chapter 1: Introduction

# What is Learnster?

The software application named “Learnster”, the basis for this project report, is a web application that I have developed to support the administration, documentation, tracking, reporting and delivery of standard learning courses, e-learning courses or training programs. Similar to large scale learning management systems such as Moodle and Blackboard Learning, the Learnster platform is available in two forms: as a free service at [www.learnster.co.uk](http://www.learnster.co.uk); and through individual installation and hosting.

# Learning Management Systems: What are they?

Undoubtedly we have seen ICT have a significant influence on teaching and learning standards in the education sector and enterprise environment. It is argued that the development of ICT in education and training has been “a key priority in most EU and OECD countries in the last decade” (Aristovnik, 2012). This shifting educational paradigm is often referred to as e-learning, where ICT is intentionally used in teaching and learning (Naidu, 2006). Hence, we see that ICT tools, namely learning management systems (LMS), have become an indispensible part of higher education, as well as many K-12 and business environments.

An LMS, alternatively called a learning platform, refers to a wide range of systems that assist educators in accessing and administrating online learning services (Paulsen, 2002). The services provided by an LMS vary from one system to another. LMS studies forward that the high-level features of such systems are as follows: centralizes and automates educational administration; assembles and delivers learning content rapidly; centralizes collaborative learning; personalizes content; and enables knowledge reuse (Ellis, 2009).

# The Commodification of Learning Platforms: How is the market?

The overall LMS market, valued at $1.9 billion in 2013, is quite diverse. We must consider there exists two market sectors - the enterprise sector (HR-orientated training/learning platforms) and the academic sector (learning/administration platforms for higher education and K-12 organizations). In terms of LMS’s enterprise market sector, analysts have asserted that US companies increased their annual corporate training by 9.5% in 2012, bringing global corporate spending to over $130 billion. This LMS market growth has been possible through the shift away from formal enterprise training to a "continuous learning" environment (Baren, 2012).

In terms of LMS’s academic market sector we see a somewhat saturated picture. Recent studies reflect that over than 95% of universities and colleges in US have adopted one or more LMS and that the same adoption rate exists in institutions in the UK (Alharbi, 2014).

# The Big Picture: Developing the Learnster business case

Upon my initial investigations of the LMS market I began to consider a potential business case for this project. The business model similarities shared across the popular LMS providers forward two important considerations:

1. Dominant LMS providers model their offering around one market sector only, limiting their customer-base. For example, the Blackboard Learning platform simply targets “from K-12 and higher education to career colleges” (Blackboard.com, 2014). Considering this, along with the categories of features that Blackboard offer, one can assert that Blackboard’s LMS offering is directed at the LMS educational market sector exclusively
2. As a consequence of modeling a LMS around a specific market sector, these LMSs have typically limited the universality of their product’s features. For instance, products such as the Cornerstone Learning Cloud platform provide course management features that integrate with third-party enterprise e-learning course providers (Cornerstoneondemand.com, 2014). Such enterprise-centric features offer little value for educational organizations or small and medium enterprises.

Certainly these points can be considered to be competences, as modeling towards one market sector exclusively suggests that these providers are product specialists in their respective domains. For instance, Blackboard Learning leverages and builds upon their educational market specialty through successfully providing network, support and training services (Blackboard.com, 2014). Nevertheless, the predominance of product specialty in the LMS market poses entry opportunities.

There exists no “one-size-fits-all” LMS solution. Current LMS platforms are too specialized in either the higher education arena or the HRM training domain. Furthermore, there exists few lightweight LMS solutions. As a consequence of being such large solutions, established LMS products have heavy configuration and on-boarding costs. In fact we see that third-party cloud providers, Lambda Solutions, are now providing training, hosting, consulting and certification services for Moodle (Lambdasolutions.com, 2014). Such overheads are not suitable for independent educators/tutors or small and medium businesses who desire a more lightweight solution.

These market entry opportunities then inspired the three philosophies that I was to base the Learnster platform on:

* **Generic learning features over market specific learning features -** the product should favor providing features that cater to all LMS users rather that of either the education or enterprise market sectors
* **Learning-centered design over classic web-driven design –** the product should feel lightweight. The product’s overall design and architecture should provide this lightness/responsiveness through modern web techniques (single page architecture)
* **Ease of product configuration over consultancy –** installation and set-up costs should be painless. Suggestive web design should assist in on boarding. Data/user importing means should be comprehensive and standardized

These philosophies have served as my baseline goals for this overall project.

In turn, the vision behind these product philosophies has provided the narrative in how I developed my user stories. Thus, this vision has directly influenced my choice in, and design of, the product’s features. Similarly, these philosophies became the baseline for measuring the success of each minor release (v0.5, v0.6, etc.).

# Chapter 2: Initial Architecture before initial Requirements Analysis

# Why consider application architecture first?

Revisiting the Learnster philosophies, the product aims to emphasize on learning-centered design over classic web-driven design. Meaning Learnster must transcend over the traditional constraints faced when designing web applications and successfully implement a responsive and lightweight design to deliver a more effective learning experience. One reason traditional web applications are slow is because most MVC server frameworks emphasize on “serving page after page of static content to an essentially dumb client”. For example, consider when we click a link in a traditional web application slideshow, “the screen flashes white and everything reloads over several seconds: the navigation, ads, headlines, text, and footer are all rendered again” (Mikowski and Powel, 2013). However, the only data that is exchanged is the slideshow’s image and description text.

Our business case requires that the Learnster software must surpass these traditional web application shortcomings. Therefore, before defining any feature-specific requirements, the defining of an initial architecture requirement takes precedence. Such is as follows:

|  |  |  |
| --- | --- | --- |
| **Functional Requirement** | **Technical Requirement** | **Business Need** |
| All user experiences should be lightweight, dynamic and responsive | The application architecture should adhere to pure Single Page Application standards | All users must share a learning-centered user experience rather than classic web-driven experiences that our competitors provide |

# Single Page Applications: The “what” and “why”

A Single Page Application (SPA) is an application that is “delivered to the browser and does not reload the page during use", aiming to provide a more fluid user experience akin to a desktop application (Mikowski and Powel, 2013). SPAs have existed for some time. Until recently Flash and Java Applets have been the most widely used SPA platforms because their capability, speed, and consistency exceeded those of JavaScript and browser rendering. However, in recent times browser and JavaScript rendering have “reached a tipping point where they have overcome their most troublesome deficiencies while providing significant advantages over other platforms” (Mikowski and Powel, 2013). Certainly, SPAs offer the best of both worlds: the immediacy of a desktop application; and the accessibility and portability of a web application.

# The SPA Thin Sever Architecture: How does it differ?

SPAs see the application logic moved from the server to the client. Such results in the role of the web server evolving into a pure data API or web service, also known as the ”thin server architecture”. Meaning the client is now represented as “thick” and the server is “thin”. Such involves web-based clients consuming the API through AJAX calls rather than the traditional full-page post-back operations.

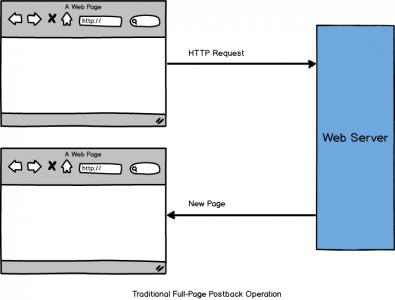


Figure 2.1

SPAs move away from page post-backs (see Fig 2.1) and instead generate HTML and execute business logic on the web client.

# Initial Architecture: Selecting an client MV\* candidate

This shifting of business logic towards the client warrants translating the traditional server application architecture patterns to the client. It is argued that SPAs require JavaScript coding at a scale an order of magnitude greater than of a traditional web application. For instance, SPAs being “over 100,000 lines of code are not uncommon” (Powell and Mikowski, 2012). One can assert that the extensive boilerplate coding involved in SPAs may be a consequence of both the newness and diversity of the client-side libraries and frameworks available. Consider Fig 2.2, a Google Trends measurement, which represents the worldwide searches over the last three years made for AngularJs, BackboneJs and EmberJs. These trends reflect the increasing in popularity of MV\* client-side libraries and frameworks, particularly over the last year. I denote these projects as libraries “and” frameworks intentionally, often utility libraries such as BackboneJs, and even JQuery, are used almost exclusively in developing SPAs. It is only in recent years we have seen the rise and acceptance of more “complete” client-side MV\* frameworks such as AngularJs and EmberJs. Due to its established and extensive enterprise success, my second commit, dated 8/8/2013, saw the introduction of the Backbone library.

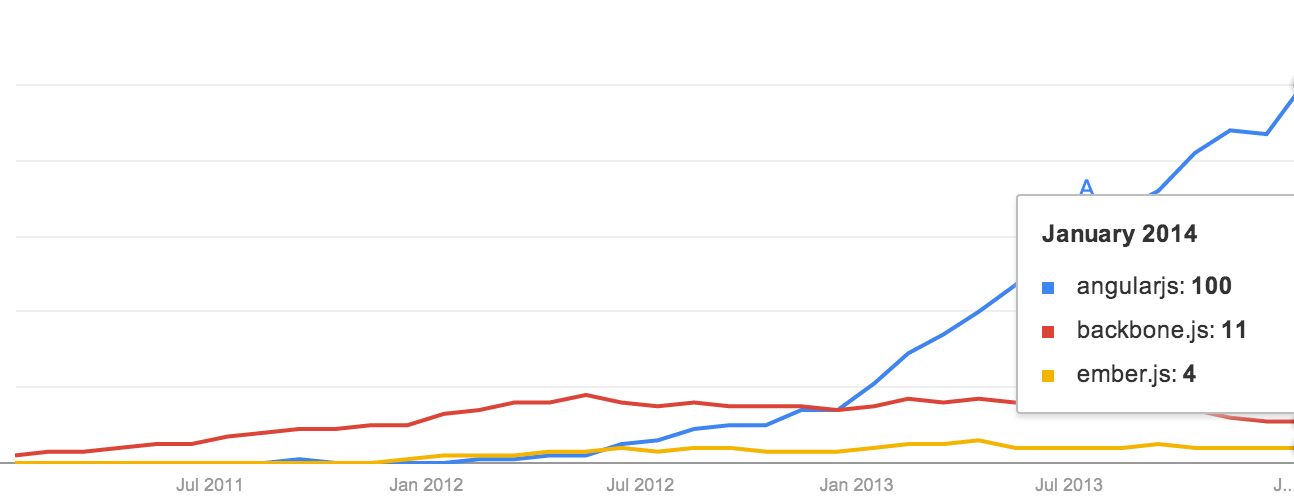


Figure 2.2

# Initial Architecture Implementation: Base Implementations for achieving “pure SPA standards”

Meeting “pure Single Page Application standards” simply means that a user can use every feature in your web application and the page never needs to refresh; the pure desktop app experience. The BackboneJs library, 60kb in size, is quite unopinionated in terms of application structure, workflow and so on. It simply offers the “minimal set of data-structuring and UI primitives that are generally useful when building web applications” (Backbonejs.org, 2014). Consequently, Backbone is missing some essential client-side infrastructure facilities. Such are as follows:

* Handling page transitions and state
* A Module/sub-application facility
* Garbage collection of “zombie” objects
* Nested collections and resources
* Message facility for cross-component communication (between models, views and controllers)

To ensure that application architecture “adheres to pure SPA standards” I needed to extend the Backbone library with these infrastructure facilities. In doing so I first reached for the MarionetteJs library, a simple composite application library to simplify constructing large-scale Backbone applications. It immediately provided most of these missing infrastructure facilities (Fig 2.3). However, two other critical SPA infrastructure facilities still needed to be implemented.

I need to develop library facilities for handling page/routing transitions and garbage collection. Routing transitions in SPAs describe the views response between sync requests and responses, where “sync” events are any GET, PUT, POST or DELETE of a resource. For example, such a transition may be a spinner covering a form while it requests a PUT or POST on a resource. Secondly, garbage collection for JavaScript is necessary in pure SPAs, as one page session could potential never be refreshed.

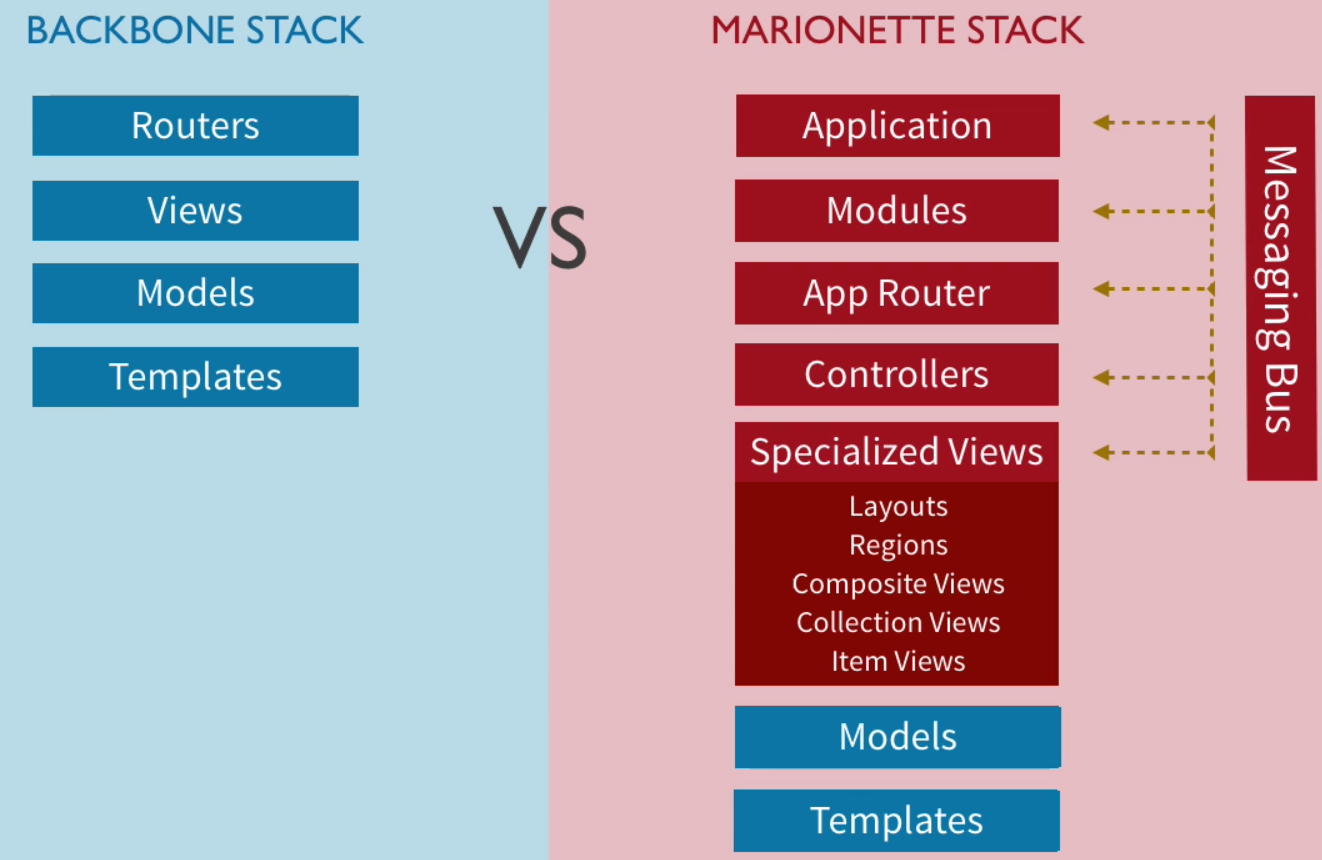


Figure 2.3

When binding events between Backbone/Marionette components, such as views, models and controllers, one must ensure that the component unregisters all events it listens to when it dies. As long as the involved objects are bound together, and there is a reference in the application code to at least one of them, “they won’t be cleaned up or garbage collected and the resulting memory leaks are like ‘zombies’” (Bailey, 2011). In my first minor release (v0.5) my first five feature tickets (Fig 2.4) informally involved implementing these facilities.

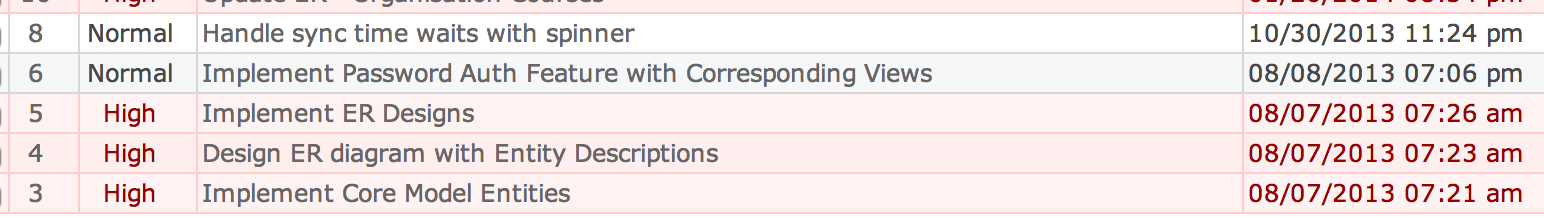


Figure 2.4

My initial architecture requirement was too ensure that all users share a learning-centered user experience rather than classic web-driven experiences that other LMS competitors provide. The technical implementation involved implementing a ‘pure’ SPA architecture to later build upon. We see that through having first researched, and experimented with, SPA technologies and I uncovered some interesting difficulties that one encounters when architecting pure SPAs. The solutions in overcoming these difficulties, and in turn, providing this base architecture, were to extend upon the components of Backbone/Marionette and define facilities for handling garbage collection and page transitions.

# Chapter 3: Initial Project Management and Requirements Analysis

**Implicit Requirements: What is already planned?**

Having informally implemented my base SPA client-side architecture the need for a more formal development process became apparent. Firstly, let us reflect that in the case of many Greenfield software projects, some high-level requirements are already understood. In terms of this project, the following were assumed:

* The web application should be available for consumption on the web (as a service and as a download)
* The project should have a standardized deployment process
* The project should be tracked and managed using an appropriate project management tool
* The QA process should also be tracked with this tool
* The overall software mission and initial actors should be defined first
* Then, high-level requirements and user-stories should be developed as a basis to work off

**Learning How to Crawl: Mission Statement, Actors, User Stories and Project Management Tools**

I first deployed an instance of Redmine, an open-source project management tool, to Heroku and configured an epic project named “Learnster-Webapp” (Fig 3.1). This Redmine instance is available for read-only use at <http://redmine-lms.herokuapp.com/>.



Figure 3.1

I then, in the “Documents” section of this epic project, created a documentation/wiki entry titled “Learnster Overview/Stories”. Mindful of the product’s basing philosophies, I submitted the following content:

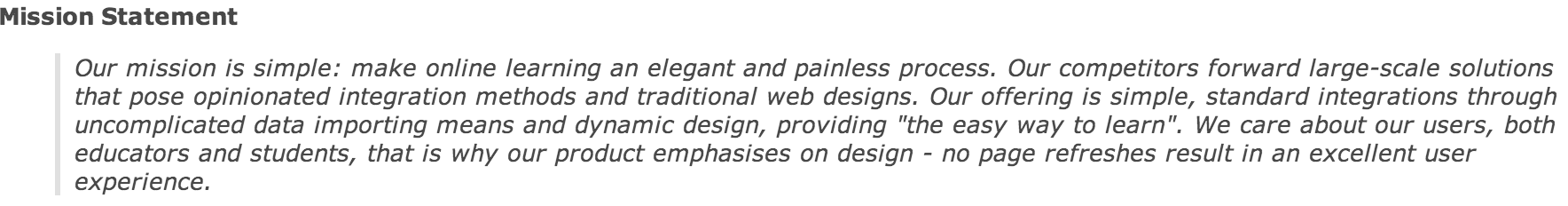
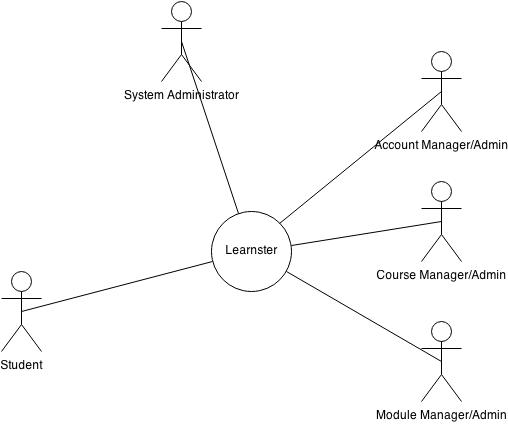


Figure 3.2

Furthering from this, I then began to consider the possible actors that would interact with the Learnster system and the according perspectives the system would offer. Such are represented in the following high-level use-case model and perspective descriptions/requirements:



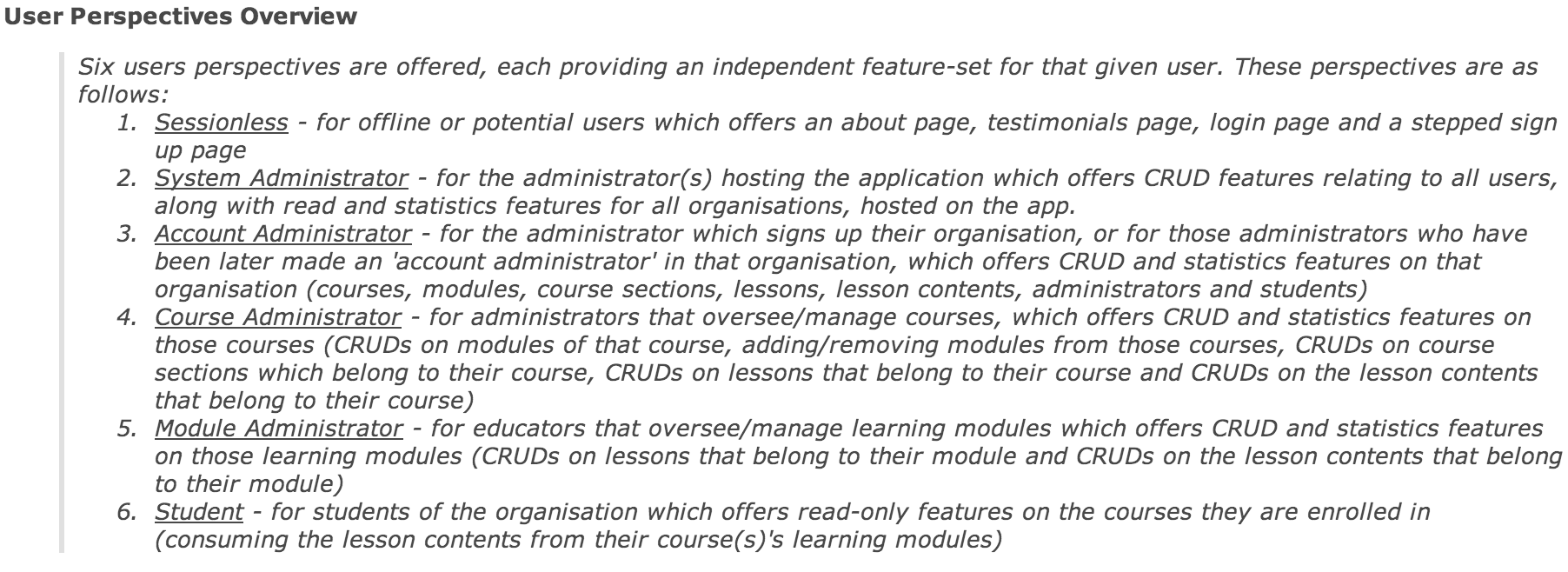


Figure 3.3

These descriptions (Fig 3.3) importantly served as the baseline requirements for the Learnster platform. As a minimal viable product, the platform was to meet at least these requirements. With such, I adopted these requirements to the following epics (big user stories)



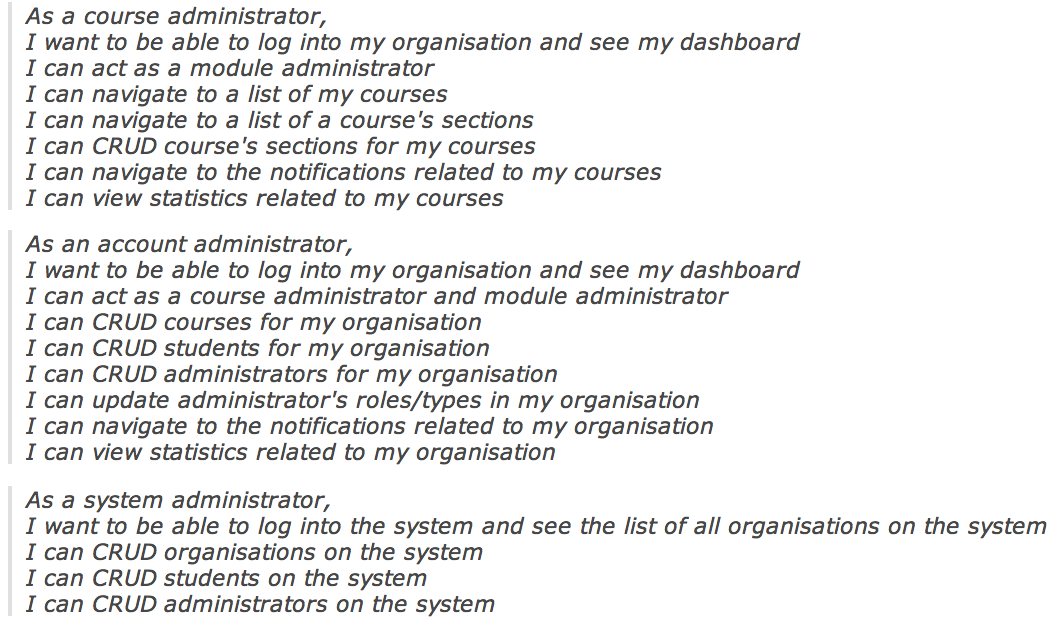


Figure 3.4

# Chapter 4: Process, Process, Process…

# Wearing all The Hats and Selecting a Methodology

Now in possession of a set of well-defined epics, I began to contemplate my development process. I argue that a ‘gray’ area of assessing non-group based software projects is the criterion of ‘process’. Software development methodologies forward practices to structure, plan and control the “process of developing large software projects developed in teams” (Schwaber, 2004). Being the only person working on this project I found myself serving the roles of product manager, project manager, devops engineer, software engineer, QA engineer and so on.

# Bibliography

* Alharbi, S. (2014). Using the Technology Acceptance Model in Understanding Academics’ Behavioral Intention to Use Learning Management Systems. International Journal of Advanced Computer Science and Applications, 5 (1).
* Aristovnik, A. (2012). The impact of ICT on educational performance and its efficiency in selected EU and OECD countries: a non-parametric analysis. University of Ljubljana, Faculty of Administration. Munich Personal RePEc Archive.
* Backbonejs.org. (2014). Backbone.js. Retrieved 22 April 2014, from <http://backbonejs.org/#introduction>
* Backbonerails.com. (2013). BackboneRails.com - Engineering Rich, Single Page Apps, [diagram - screenshot]. Retrieved 22 April 2014, from <http://www.backbonerails.com/series/engineering_single_page_apps>
* Bailey, D. (2011). Zombies! RUN! (Managing Page Transitions In Backbone Apps) | ThoughtStream.new :derick\_bailey. Lostechies.com. Retrieved 22 April 2014, from <http://lostechies.com/derickbailey/2011/09/15/zombies-run-managing-page-transitions-in-backbone-apps/>
* Baren, A. (2012). LMS 2013: The $1.9 Billion Market for Learning Management Systems.Bersin.com. Retrieved 21 April 2014, from <http://www.bersin.com/blog/post.aspx?id=5c1f98de-e439-4e56-879d-b7741bde216c>
* Blackboard.com (2014). Blackboard Learn LMS. Retrieved 21 April 2014, from <https://www.blackboard.com/Platforms/Learn/Products/Blackboard-Learn.aspx>
* Cornerstoneondemand.com. (2014). Learning Management System Software for Global Business - E-Learning Software. Retrieved 21 April 2014, from <http://www.cornerstoneondemand.com/global-business/talent-management/learning-management-cloud>
* Ellis, R. (2009). [Field Guide to Learning Management Systems](http://www.astd.org/~/media/Files/Publications/LMS_fieldguide_20091). ASTD Learning Circuits
* Fig 2.1 Blog.4psa.com. (2014). An Intro Into Single Page Applications (SPA) | 4PSA Blog, [diagram]. Retrieved 22 April 2014, from <http://blog.4psa.com/an-intro-into-single-page-applications-spa/>
* Fig 2.2 Google.com. (2014). Google Trends - Web Search interest - Worldwide, 2004 – present, [graph]. Retrieved 22 April 2014, from <http://www.google.com/trends/explore?hl=en-US#q=Angularjs%2C%20Backbone.js%2C%20Ember.js&date=1%2F2011%2037m&cmpt=q>
* Fig 2.3 Backbonerails.com. (2013). BackboneRails.com - Engineering Rich, Single Page Apps, [screenshot]. Retrieved 22 April 2014, from http://www.backbonerails.com/series/engineering\_single\_page\_apps
* Fig 2.4 Redmine-lms.herokuapp.com, (2014). Issues - v0.5 - Learnster: Redmine, [screenshot]. Retrieved 22 April 2014, from <http://redmine-lms.herokuapp.com/projects/v0_5/issues?utf8=%E2%9C%93&set_filter=1&f%5B%5D=status_id&op%5Bstatus_id%5D=c&f%5B%5D=tracker_id&op%5Btracker_id%5D=%3D&v%5Btracker_id%5D%5B%5D=2&f%5B%5D=&c%5B%5D=priority&c%5B%5D=subject&c%5B%5D=created_on&c%5B%5D=updated_on&group_by>=
* Fig 3.1 Redmine-lms.herokuapp.com. (2014). Overview - Leanrnster-Webapp - Learnster: Redmine, [screenshot]. Retrieved 23 April 2014, from <http://redmine-lms.herokuapp.com/projects/lrnstr-web>
* Fig 3.2, Fig 3.3 and Fig 3.4 Redmine-lms.herokuapp.com. (2014). Learnster Overview/Stories - Leanrnster-Webapp - Learnster: Redmine, [screenshot]. Retrieved 23 April 2014, from <http://redmine-lms.herokuapp.com/documents/6>
* Lambdasolutions.com. (2014). Moodle Implementation. Retrieved 22 April 2014, from <http://www.lambdasolutions.net/moodle-services/moodle-implementation>
* Naidu, S. (2006). E-learning: A guidebook of principles, procedures and practices. Commonwealth Educational Media Center for Asia. Sanjaya Mishra.
* Mikowski, M. & Powell, J. (2013). Single page web applications: JavaScript end-to-end. Shelter Island, NY: Manning.
* Paulsen, M. F. (2002). Online Education Systems: Discussion and definition of terms. NKI Distance Education.
* Powell, J. & Mikowski, M. (2012, October). Single Page Web Applications: JavaScript End-to-End (The Hard Stuff). Presented at Html5devconf, San Francisco, CA. From <https://www.youtube.com/watch?v=OrIFaWJ9Glo>
* Schwaber, K. (2004). Agile project management with Scrum (1st ed.). Redmond, Wash.: Microsoft Press.