Many Voices Publishing Platform

Technology Review

D. Kevin McGrath & Dr. Kirsten Winters - CS461 Fall 2016

Commix

Steven Powers, Josh Matteson, Evan Tschuy

Abstract

The Many Voices Publishing Platform uses a variety of technologies to handle different aspects of the project, from the user interface to the backend database operations. These technologies enable to the Many Voices Publishing Platform to succeed in delivering a working platform for textbook collaboration.

CONTENTS

1 Technology Review 1

1.1 Introduction . . . 1

1.2 Steven . . . 1

1.2.1 User Interface Tools . . . 1

1.2.2 User Login & Authentication . . . 5

1.2.3 Interface Design . . . 8

1.3 Josh . . . 11

1.3.1 Testing . . . 11

1.3.2 Version Control . . . 15

1.3.3 Database . . . 19

1.4 Evan . . . 23

1.4.1 Server Back-end . . . 23

1.4.2 Text Formatting . . . 27

1.4.3 Password Storage . . . 31

1.5 Conclusion . . . 35

References 36

1 TECHNOLOGY REVIEW

1.1 Introduction

The Many Voices Publishing Platform is being developed for the purpose of fixing the problems currently associated with the textbook market. We will accomplish this by giving the MVP Platform an easy to use interface, a search bar with a built in results pane, source control, and many other features. Authorship is divided by subsection header.

1.2 Steven

1.2.1 User Interface Tools

Option 1 - React [1]

React is a JavaScript rendering engine that is developed by Facebook. Originally used with Instagram, React is often paired with Redux for added functionality. React is a popular JavaScript library meant for building user interfaces that is component based.

Option 2 - Aurelia [2]

Aurelia is a newer JavaScript client framework for mobile, desktop, and the web, by using simplistic integration.

Option 3 - Ember [3]

Ember uses web components and templates to increase productivity.

Option 4 - Angular2 [4]

Angular2 is a project started by Google for their internal Green Tea project. Angular2 is a widely documented JavaScript cross-platform library that is used to create native mobile and desktop web applications.

Goals

The use of this technology will aid in the development of the user interface. Having a beautiful and scalable user interface will help users interact with the platform more easily, on whatever device they choose to use it on.

Evaluation Criteria

The options are evaluated on

- Ease of Use
- File Size
- Features
- Performance
- Standards Compliance
- Non-Compliance
- Release
- License

Option Comparison

[5]	React	Aurelia	Ember	Angular 2
Ease of Use	Substantial setup required for working system, lots of documentation and tutorials.	Simple setup using NPM and installation	Simple setup using NPM and installation	Substantial setup required for working system, lots of documentation and tutorials.
File Size	156kb to ???kb, due to added frameworks	323kb	435kb	1023kb
Features	View rendering engine with plugin frameworks	Router, Animation, HTTP Client	Router, HTTP Client	Router, HTTP Client
Performance (Paints per Second)	45-50	90-150 (Higher end with additional plugins)	60-100	80-130 (Higher end with additional plugins)
Standards Compliance	ES 2015	HTML, ES 2016, Web Components	HTML, ES 2015	ES 2016
Non-Compliance	JSX	N/A	N/A	NG2 Markup, Dart
Release	15.x	Beta	2.x	Release Candidate
License	BSD	MIT	MIT	MIT

Discussion

All of the chosen options have their pros and cons for our web application. All of them however would be a learning and research experience. Angular2 and React have the benefit of being created by large software companies, Google and Facebook respectively. This means that there will be large adoption and documentation / tutorials available. Aurelia and Ember seem to be easier to implement however, they are much newer products and they have a smaller adoption population. This could prove troublesome if we run into problems. If our implementation ends up being a fork of Ward Cunningham's Federated Wiki, then this decision will be null most likely.

Selection

Initially we were set on using Angular2 as part of the team has experience using this JavaScript library, before meeting with our client. Angular2 has a wide adoption and is used by Google for internal projects so the longevity of the framework is expected to last. With this in mind, we plan to use Angular2 if we need to use a JavaScript framework for our user interface.

1.2.2 User Login & Authentication

Option 1 - OpenID Connect

OpenID Connect allows for clients of all types, including browser-based JavaScript and native mobile apps, to launch sign-in flows and receive verifiable assertions about the identity of signed-in users [6].

Option 2 - Facebook

Facebook Login for Apps is a fast and convenient way for people to create accounts and log into your app across multiple platforms [7].

Option 3 - PHP & SQL

Using PHP and SQL to compare submitted usernames and passwords against stored data on a database.

Goals

An efficient and secure method for allowing for users to login and continue editing their documents from any computer or device they choose.

Evaluation Criteria

The options are evaluated on

- Ease of Use
- Features
- Security

Option Comparison

	OpenID	Facebook Login	PHP & SQL
Ease of Use	Requires Credentials with Corresponding Login Providers, Lots of available libraries	Requires Credentials with Facebook and an App ID with Facebook	Easy to implement, but if setup incorrectly can lead to problems
Security	Relies on credential host and user security	Relies on Facebook and user security	Relies on password protection implementation and user security
Features	Easily log in with OpenID partner credential hosts (Google, Microsoft, Yahoo, etc)	Easily log in with Facebook credentials	Easily log in with user created account and password

Discussion

The ideal user authentication system would be a combination of all three of the above implementations. While logging in with Facebook would make it easier to determine who is using the service, preventing unauthorized users from accessing unreleased copyrighted material, not everyone has a Facebook. Additionally using an OpenID login system would be reliant on other platform holders that use OAuth 2.0. Finally, using a self created account is often the easiest and can allow users to not be tied to a given account and also prevent private information from being retrieved from user accounts.

Selection

For our implementation, we plan on using initially a PHP and SQL system to validate user account information on our database. Additionally, we will look into adding both OpenID and Facebook Login down the road.

1.2.3 Interface Design

Option 1 - User Centered Design

A deep understanding of the target audience is able to provide insights into how to design and develop your application to suit your intended users [8].

Option 2 - Activity-Centered Design

Instead of focusing on research about intended users, the design and development are focused around making a given activity logically designed [9].

Option 3 - Self Design

The designer is responsible for representing the target audience. Though this can be a poor representation of the intended audience [9].

Goals

A design principle that allows for user interfaces that lead to user interfaces that are accepted by users and are easy to understand.

Evaluation Criteria

The options are evaluated on

- Ease of Use
- Strengths
- Weaknesses

Option Comparison

	User Centered Design	Activity-Centered Design	Self Design
Ease of Use	Long process, that takes a lot of data gathering to provide insights into a target audience.	Easier to design an activity when not trying to cater a specific audience.	Very easy to design what works well for you as a designer.
Strengths	Allows for the designer to understand what makes a user think the way they do. This allows for an interface design to be molded to an expected user.	Allows the designer to design a user interface based around an activity that a user will be performing instead of designing to a users wants and desires.	Allows for easy creation of user interface of how the designers see fit. Perfect for a target audience that is just like the designer.
Weaknesses	Takes a long time to gather enough information to be able to design a good solution that feels natural to a target audience user.	Designed interface might work well for an intended activity, but could be antagonistic to a target audience.	Interface could be intended for an entirely different audience, leaving a confusing experience.

Discussion

The MVP Platform is highly user focused, which initially led the team to decided on User Centered Design. Activity-Centered Design or Self Design would greatly reduce the burden of research and discovery into what our target audience would like to see or be comfortable with naturally. Activity-Centered Design, if performed properly would result in interfaces that clearly work as intended, though might be off putting to our users. Self Design would allow for one of the team members to decide how a certain element shall look, but again can fall into an interface that does not satisfy our users.

Selection

For our implementation, we plan on using User Centered Design. This is because users are our very important for our project. If our users do not like our user interface, then they will be less likely to use our software.

1.3 Josh

1.3.1 Testing

Option 1 - Mocha

Mocha is a JavaScript testing framework, loaded with features. It runs on Node.js and also in the browser, making asynchronous testing simple and easy to use. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. [10]

Option 2 - QUnit

QUnit is a powerful, easy-to-use JavaScript unit testing framework. It's used by the jQuery, jQuery UI and jQuery Mobile projects and is capable of testing any generic JavaScript code. [11]

Option 3 - Jasmine

Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests. [12]

Goals

Using this technology will aid in proper functionality and minimize errors. Without properly testing code, a number of problems can occur that can disrupt and slow down progress in a team. In extreme cases, not properly testing could lead to failure of the application.

Evaluation Criteria

The options are evaluated on

- Ease of Use
- Features
- Documentation
- Integration
- Other

Option Comparison

	Mocha	QUnit	Jasmine
Ease of Use	Language is like spoken language, which makes for easier to understand	Not as friendly, language is similiar to other basic Unit testing frameworks	Language is like spoken English, fairly easy to understand
Features	Spys, mocks, stubs, callbacks, etc. Most features	Most testing is done through assertions only, not as many features	Spys, mocks, stubs, callbacks, etc. Lacks Assertions, but can be implemented with library
Documentation	Fairly common, moderate amount of documentation	Not well documented	Most popular, abundant documentation
Integration	Moderate difficulty to set up	Easiest to set up	Moderate difficulty to set up
Other			Familiar

Discussion

It is well known that an application that ensures proper functionality supersedes other applications and, more importantly, competitors. Quality assurance is acknowledged as highly distinguished, and therefore, an attribute deserving of notable consideration. Having this in mind, we will be considering these mentioned testing frameworks: Mocha, QUnit, and Jasmine.

Selection

Jasmine, even though more difficult to integrate and set up, would be most feasible. Jasmine has many features that win out over Mocha and QUnit, as well as thorough documenation. The test are easily grouped through describe blocks, which makes it easy to identify where certain problems are. There is familiarity with Jasmine over the other testing frameworks, which means less time learning for the team.

1.3.2 Version Control

Further information is needed to begin planning built in source control.

Option 1 - Git

Option 2 -

Option 3 -

Goals

Evaluation Criteria

The options are evaluated on

- API
- License
- Integration
- Other

Option Comparison

	Option 1	Option 2	Option 3

Discussion

Selection

1.3.3 Database

Option 1 - SQL Server

SQL Server is owned by Microsoft, and is known for its scalable, hybrid database platform. Being owned by Microsoft, this database is well known. [**?**]

Option 2 - MongoDB

MongoDB is a document based database, with an Expressive Query Language and Secondary Indexes. [**?**]

Option 3 - MySQL

Many of the worlds largest companies rely on MySQL, such as Facebook, Google, and others. It is a relational database. [**?**]

Goals

Having an understandable, well built database can aid in the flow of building an application as well as delivery. This in mind, we want to balance cost with effectiveness.

Evaluation Criteria

The options are evaluated on

- Cost
- Type
- Ease of Use

Option Comparison

	SQL Server	MongoDB	MySQL	
	Cost	Enterprise: $12,256, Standard: $3717, Developer: Free	OpenSource, but essential periphery software can range from $45 a month to as much as $5225 a month	Enterprise: $5000, Standard: $2000, Community Edition: Free
	Type	Structured Query Language	Expressive Query Language	Structured Query Language
	Ease of Use	Uses Queries to gather information	EQL data access and manipulation in sophisticated ways, operational and analytical applications [**?**]	Uses Queries to gather information

Discussion

The most obvious and transparent qualities for a database are the cost and type, however, databases that come with a free edition allow for us to test and experiment while developing. This is imperative to the development process, and shouldn't be overlooked. Allowing for an easy transition from the free edition to a standard or express edition is important as well. A non relational database like MongoDB could be beneficial when not knowing the type of data that will be stored.'

Selection

All things considered, SQL Server would be considered our top pick. Experience using an SQL based database before aids in the difficulty of using a query. Having a free edition contributes a moderate amount into the decision as well.

1.4 Evan

1.4.1 Server Back-end

Option 1 - NodeJS

NodeJS is a modern web back-end framework developed by the Node Foundation, primarily led by Joyent. By using JavaScript its language of choice, Node allows developers to use the language's unique concurrency paradigms to quickly develop scalable applications.

Option 2 - Django

The Django framework is a massive web framework developed in Python that comes "batteries included". The Framework includes everything from geo-libraries to support for four different kinds of databases, meaning a large initial learning curve but a large payoff.

Option 3 - Flask

Flask is a micro-framework. It comes with the bare minimum needed to do HTTP handling, leaving what other frameworks come with to an array of choices from third party developers. This means the core framework is quick to learn, but can quickly leave a developer feeling constrained.

Option 4 - Ruby on Rails

Ruby on Rails is the old standard of web frameworks. It was the original batteries included framework, and has over the years been known for its ease of use. However, the framework is quite old and shows some signs of age, using sometimes outdated paradigms and generally being less friendly to beginners than more modern frameworks.

Evaluation Criteria

The options are evaluated on

- Ease of use
- Language
- Features
- Ecosystem
- License

Option Comparison

	NodeJS	Django	Flask	Ruby on Rails
Ease of Use	Javascript backend shares language with frontend; super quick iteration	Large framework containing all needs within, with a large learning curve	Microframework containing only minimal needs; requires finding external packages	Old framework with massive, but aging, ecosystem
Language	Javascript	Python	Python	Ruby
Features	Minimal HTTP interaction, massive external ecosystem for extras	Massive built-ins	Minimal with external ecosystem for extras	Massive built-ins
Ecosystem	Massively popular today with expansive and growing ecosystem	Decently large ecosystem with built-ins for most tasks	Decently large ecosystem	Large ecosystem but fairly old
Release	7.1.0	1.10.3	0.11	5.0.0.1
License	MIT	BSD	BSD	MIT

Discussion

All backends listed are popular within their respective communities. However, more new projects are being created using NodeJS, as its modern paradigms and sharing of a language with frontend development allow developers familiar with it to iterate quicker and write more expressive code. Django's built-ins allow for a quicker initial development time but mean being isolated from the rapidly expanding ecosystem around NodeJS. Ruby on Rails is a rather old backend, and has not shown to have the modern flexibility of Node.

Selection

As NodeJS has the most expansive ecosystem, and allows us to share a common language between the front and backennds of the project, we will be using it over the other options considered. Additionally, Ward Cunningham's Federated Wiki uses it as one of its backends, and if we fork it, we can continue to use its NodeJS backend.

1.4.2 Text Formatting

Option 1 - Markdown

Markdown is a highly lightweight markup language that allows easy, human-readable markup of text to include headings, bold/italic/underline/etc, bullets, and numbered lists. The original markdown does not support things like images or videos; Markdown has various ”flavors”, or implementations, that sometimes allow for such things.

Option 2 - Restructured Text

Restructured Text is a markup language written in Python for writing documentation, simple websites, etc. It allows for highly varied but still restricted markup; it allows for image embeds, fancy linking, titles, etc. It does not allow users to embed arbitrary elements.

Option 3 - Raw HTML

Storing simply raw HTML allows the greatest flexibility, as it is literally the same elements rendered in browser. Raw HTML allows for things like scripting, video embeds, etc., and as such must be filtered to a restricted subset to be suitable for use in a public-facing scenario.

Evaluation Criteria

The options are evaluated on

- Ease of use for end-users
- Markup options
- Compile language
- Security

Option Comparison

	Markdown	ReStructured Text	HTML
Ease of Use	Easy to use, with minimal options and human-readable markup; different implementations have slight differences leading to confusion	Relatively human-readable markup but with massive number of options	Essentially infinite options but not very human-readable/human-writeable

Markup	Options readable in single-page document, not allowing for high flexibility	Highly featureful with well-defined language	Allows for infinite options along; can use CSS to fine-tune display
Compile Language	Dozens of different libraries for different languages, each with slightly different interpretation of the markup	Python	No compilation needed to display but some backend processing needed for security
Security	Small language leads to minimal exploits	Well-defined with real-world-tested libraries	Needs careful processing to stop end-users from inputting harmful raw input (including scripts)

Discussion

All languages listed allow users to do simple things like bold text and link to other pages. However, HTML offers the most flexibility and allows users to be able to do anything they want. Allowing this while maintaining ease-of-use would require a frontend library that can allow users to interact with the document in ”what you see is what you get”, or WYSIWYG, mode.

Selection

The Federated Wiki uses HTML with minimal security processing. If we fork the Federated Wiki, we will use HTML with some added processing to increase its security. Otherwise, for ease of implementation, we will use Markdown for its backend language support, but implemented in such a way as to allow easy replacement of the code with some other markup language as wanted.

1.4.3 Password Storage

Option 1 - Bcrypt

Bcrypt is a password hashing function that takes a very large amount of time to crack an individual password – it is designed to be slow. This means a hacker cannot simply crack a database worth of passwords in one sitting, as with older hashes like MD5.

Option 2 - Scrypt

Scrypt is designed to take up large amounts of time, and large amounts of RAM, when hashing. This ensures that a hacker cannot simply buy a powerful CPU and crack passwords with pure power. However, scrypt, being designed more-so for computer hard disk passwords, can take multiple seconds and hundreds of megabytes of RAM to process.

Option 3 - pbkdf2

PBKDF2 is a function that repeatedly hashes a password using the HMAC, or ”keyed-hash message authentication code”, function. For a CPU, cracking a large number of passwords using pbkdf2 is difficult, as it takes a large amount of time to crack an individual password. Using a GPU, however, a large number of hashes can be run in parallel, making it quick to crack with high end hardware.

Option 4 - raw storage

Another option for password storage is to store the passwords in plain text. This allows users to recover their passwords directly through a password reminder email. However, this comes with the major downside that compromising the database allows a hacker to be able to access any accounts on unrelated services where users use the same username and password (a common pattern in non-technical and technical users alike).

Evaluation Criteria

The options are evaluated on

- Cracking
- Storage
- Resistance to Hacking

Option Comparison

	Bcrypt	Scrypt	pbkdf2	plain text
Cracking	Bcrypt is highly resistant to cracking on CPUs and GPUs, but can be cracked quickly using specialized FPGAs.	Scrypt is highly resistant to cracking on CPUs, GPUs, etc but takes a large amount of time to verify a valid password.	pbkdf2 is highly resistant to cracking on CPUs but can be easily cracked on a GPU.	A plaintext password does not need to be cracked as it is already stored as a raw password.
Storage	can be stored in a database without issue.	can be stored in a database without issue.	can be stored in a database without issue.	Plain passwords must be stored in a way that ensures they can never be hacked, which is impossible.
Resistance to Hacking	A bcrypt password hash must be cracked before it can be used elsewhere.	Scrypt hashes must be cracked before use elsewhere.	pbkdf2 hashes must be cracked before use elsewhere.	A plain password, once retrieved from a database, can be used along with the username/e-mail associated to hack other sites.

Discussion

Password storage is a tradeoff between ease of use and difficulty of reversing. Scrypt is too slow for use on a website with many users, whereas plain text passwords are too insecure as a hacker can reuse the password immediately without cracking. pbkdf2 and bcrypt do a good job defending against CPU cracking, but as pbkdf2 can be cracked using a GPU, bcrypt is left remaining as the best tradeoff between speed and cracking.

Selection

As mentioned above, bcrypt-hashed passwords present a good tradeoff between cracking ability and verification speed. As such, the Many Voices Platform will use bcrypt to securely verify any password used with the system.

1.5 Conclusion

The Many Voices Publishing Platform is a combination of User Interfaces, Documentation, User Centered Design, Testing, User Authentication, Databases, Server Back-end, Text Formatting, Password Storage, and the users themselves. Determining the technologies behind these parts and pieces is a difficult task to accomplish, as many choices can satisfy the requirements of the project. Finding the best solution however is the goal of this document, to provide a clear path forward for the platform as a whole.

REFERENCES

[1] Facebook, "A javascript library for building user interfaces - react," https://facebook.github.io/react/index.html.

[2] Aurelia, "Aurelia," http://aurelia.io/.

[3] Ember, "A framework for creating ambitious web applications," http://emberjs.com/.

[4] Google, "Our framework," http://angular.io/.

[5] R. Eisenberg, "Choosing a javascript framework," https://www.youtube.com/watch?v=6I_GwgoGm1w.

[6] OpenID, "Openid the internet identity layer," http://openid.net/connect/faq/.

[7] Facebook, "Facebook login for apps," https://developers.facebook.com/docs/facebook-login/overview.

[8] U. D. of Health & Human Services, "User-centered design basics," https://www.usability.gov/what-and-why/user-centered-design.html.

[9] C. Bowles, "Looking beyond user-centered design," http://alistapart.com/column/looking-beyond-user-centered-design.

[10] MochaJs, "Mocha, simple, flexible, fun," https://mochajs.org/.

[11] QUnit, "Qunit: A javascript unit testing framework," https://qunitjs.com/.

[12] Jasmine, "Jasmine," https://jasmine.github.io/2.0/introduction.html.