Spirited: A Web Application for Structured Debate

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:33797385">http://nrs.harvard.edu/urn-3:HUL.InstRepos:33797385</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dashboard.current.terms-of-use#LAA">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dashboard.current.terms-of-use#LAA</a></td>
</tr>
</tbody>
</table>
Spirited: A Web Application for Structured Debate

Joshua McElfresh
jmcelfresh@fas.harvard.edu

A Thesis in the Field of Information Technology
for the Degree of Master of Liberal Arts in Extension Studies

Harvard University Extension School
October 2016
Abstract

Discussion forums have been used since the beginning of the mainstream internet and changed relatively little since then. The open-ended design of discussion forums allows them to be used for online debate. Separate from discussion forums, many structured argument tools have been built in the last 20 years. Unlike traditional forums, the vast majority of these tools seek to break the text of an argument into parts, such as statements and claims, which can be tracked and linked by the software.

Forums provide a space for debate because of their flexibility, but are easily disrupted. Argument software only allows arguments to be entered, but can be difficult to use and lack readability. There is currently no way to have an argument on the internet that is both approachable to outsiders, and structured in a way that keeps debate coherent.

Designing and building a web application to enable structured online debate was the focus of this project. An application was built that provides a blog-style interface for posting articles of any length, as well as additional features, like citations, to support the argument. Each sentence in the article becomes a point of potential debate. Sentences can be commented on, supported, or contradicted with additional citations and flagged for containing logical fallacies. Alternative or replacement sentences can be proposed. Voting on the statements that make up an article and responses to those statements shows users what parts of the argument are most controversial.

The application uses a web browser client built in the Ember JavaScript framework. The client is responsible for displaying the application, responding to events such as clicks, and maintaining a client-side copy of relevant application data. The client retrieves and saves data via an API server running the Ruby on Rails web application framework. The client communicates
with the API server using a RESTful protocol, where HTTP requests are used to specify requested data and the response is returned in the JSON format.

The application is designed to run on the public internet and support any number of users. The API server runs on Amazon’s EC2 platform utilizing a load balancer to direct requests and auto scaling to increase the number of servers if the load becomes high. Requests are restricted to HTTPS, with most requests requiring a token to authenticate the user.

The completed application improved upon current forum technology while keeping the core features intact. Just like typical forums, what users can write is not restricted in any way. The sentence level annotation interface is automatically added to their writing, providing an additional layer on the article without losing any of the flexibility that makes forums popular. By categorizing reader responses, the application allows other users to know the intention of each response, requiring users to read less. Finally, the application allows for an interaction that has been difficult if not impossible until now – having a productive argument on the internet.
## Table of Contents

Abstract .............................................................................................................................. iii

List of Tables .................................................................................................................... ix

List of Figures .................................................................................................................. x

Chapter 1: Debate & Technology ....................................................................................... 1

Introduction ..................................................................................................................... 1

Background ...................................................................................................................... 2

A Brief History of Online Discussion Forums ................................................................. 2

The Forum Format .......................................................................................................... 3

Embedding Forums in Other Media ................................................................................ 4

Motivation for the Use of Discussion Forums ................................................................. 5

Malicious Users .............................................................................................................. 7

Software Designed for Structured Debate ..................................................................... 8

Forums Built for a Specific Purpose ............................................................................. 10

Collaborative Annotation in Web Browsers .................................................................... 11

Chapter 2: Spirited, a Web Application for Structured Debate ......................................... 13

Addressing Problems with Debate Online .................................................................... 13

Meaningful Conflict Requires Structure ...................................................................... 13

Attention Currency ....................................................................................................... 14
Encouraging User Investment ....................................................................................... 15
Early User Testing ........................................................................................................ 15
The User Experience ..................................................................................................... 16
Linking to Articles from Outside the Application ...................................................... 17
Administrator Level Features ...................................................................................... 17
Content Policy ............................................................................................................. 18
Basic Application Flow .............................................................................................. 18
Articles as the Center of Debate ............................................................................... 19
Interaction with Articles ........................................................................................... 20
Responding to Articles with Annotations ................................................................... 22
Groups ......................................................................................................................... 25

Chapter 3: Software Components .............................................................................. 26
Use Cases .................................................................................................................... 26
Landing Page ............................................................................................................... 27
Login ............................................................................................................................ 27
User Registration ........................................................................................................ 28
Dashboard .................................................................................................................. 28
My Articles .................................................................................................................. 29
Create Article ............................................................................................................. 30
Article View ............................................................................................................... 31
Article View with Responses ........................................................................................ 32
Settings .......................................................................................................................... 34
Feedback Forum ............................................................................................................ 35
Article & Annotation Hiding, User Banning ................................................................. 35
Password Recovery ....................................................................................................... 36
Administrator Control Panel ......................................................................................... 36
Chapter 4: Architecture ............................................................................................... 38
Technology .................................................................................................................... 38
Frameworks ................................................................................................................... 38
Overall Component Structure ....................................................................................... 41
API Servers .................................................................................................................... 42
Spirited API .................................................................................................................... 44
Spirited Client ............................................................................................................... 52
Chapter 5: Usage and User Feedback ........................................................................... 63
The User Test Group & Feedback Collection ............................................................... 63
Difficulty Getting People to Use the Site ................................................................. 63
The Chicken & Egg Problem ....................................................................................... 63
Commenting vs All Other Annotation Types ............................................................... 64
Editorial vs Other Types of Writing ....................................................................... 65
How Minimal is a Minimum Viable Product? ............................................................ 65
Chapter 6: Conclusion ....................................................................................................... 67

Future Work .................................................................................................................. 67

References ....................................................................................................................... 69
List of Tables

Table 1 RESTful request patterns ................................................................................................. 46
Table 2 Spirited API resource types ............................................................................................. 48
Table 3 Spirited client URL patterns and resource descriptions....................................................... 58
List of Figures

Figure 1. On NYTimes.com comments must be toggled On to be viewed and are displayed to the right of the article. Users can view posts by the categories All, Readers’ Picks (highest votes) or NYT Picks (moderator selections)................................................................. 5

Figure 2 Pie chart of user motivation for using online forums...................................................... 6

Figure 3. The World of Warcraft forums contain several ways to flag abuse............................... 7

Figure 4. Reddit orders both its lists of posts and replies to a thread in order of votes by default. 8

Figure 5. Rationale maps responses to a statement into supporting and opposing nodes, each of which can have a rebuttal..................................................................................................... 9

Figure 6 Continuum of debate platform imposed structure. ....................................................... 10

Figure 7 Author and responder interaction with articles. ............................................................. 20

Figure 8 Use case diagram by user type ..................................................................................... 26

Figure 9 The landing page contains a demo article with most of the features of a full article.... 27

Figure 10 The login page............................................................................................................. 28

Figure 11 The new user form...................................................................................................... 28

Figure 12 The dashboard displays articles and allows users to create new ones......................... 29

Figure 13 The My Arguments page............................................................................................ 30

Figure 14 The article creation page............................................................................................ 31

Figure 15 An article view page without a sentence selected....................................................... 32

Figure 16 An article with a sentence selected and multiple responses........................................ 33

Figure 17 The response menu collapsed and with citations expanded........................................ 34

Figure 18 the settings page ......................................................................................................... 35

Figure 19 The forum post listing ............................................................................................... 35
Figure 20 The administrator control panel

Figure 20 Component structure

Figure 22 Spirited data structure

Figure 22 Requests are server independent. This request would use the same process on Server 2 as shown here for Server 1

Figure 23 Polymorphic associations allow sub-types to be treated like unique resources, or groups them with their parent type depending on context

Figure 25 An annotated sentence before it has been selected

Figure 26 A selected sentence displaying an attached citation and alteration

Figure 27 A selected sentence with the Respond menu open

Figure 28 Ember application component relationships
Chapter 1: Debate & Technology

Introduction

The unhindered sharing of ideas enabled by internet technology is almost always presented as a good thing. The boundaries to publish and share one’s writing have never been lower. There are no censors to tell us what we can and can’t say, no gatekeepers to restrict who can and can’t speak. Anyone may say anything about any topic they choose, for the whole world to see.

While this openness has brought many benefits – crowdsourcing an entire encyclopedia for example – it can become a hindrance in some situations. If you had been searching for answers to programming questions on internet forums 10 years ago you would have found one such situation. Many users would post different answers to a single question, and it was unclear which the correct one was. People posted follow-up questions in the same thread, and answers to those questions appeared as well. Some users just posted a comment about the question itself, others posted completely unrelated comments. Finding the correct answer with no indication of which posts were answers, and which answers were correct required a huge amount of effort. The Stack Exchange network of question and answer websites was created to address this problem.

A similar issue exists with online debate. Like question and answer interactions (Q&A), debate is an activity internet users have been trying to engage in since the internet became available. Like Q&A it is hindered by allowing anyone to write anything wherever they choose. Humans were aware that resolving conflicts requires extra structure well before the internet, as shown by the existence of complex, well-developed legal systems all over the world.
The goal of this project was to build an application that would allow internet users to engage in productive debates. This was done by imposing additional structure on the existing, popular model of an article with a comments section. Similar to how a Q&A application must recognize that questions and answers are distinct types of communication, and that only those types of communication are relevant to the task, this application recognizes that there are certain types of communication that are relevant in a debate, and discourages less relevant types. Responses relating to the logic of an argument and whether it is factually true are useful when resolving a debated issue, while opinions, comments, and other subjective responses are not.

The Ember JavaScript framework was used to build the application’s browser client. The client downloads and uploads data from an API server built using the Ruby on Rails framework. All application data is kept in a single database that stores all data necessary for the application.

Background

Debate on the internet is not a new idea - internet users have been trying to have debates for decades on internet forums, with mixed results. Many software developers (mostly in academia) have also tried to build applications to facilitate debate. This section explores the technologies most widely used for debates online, debate specific applications, and other tools that can be used to facilitate debate.

A Brief History of Online Discussion Forums

Publicly available discussion forums were initially based on email mailing lists. The discussion system Usenet gained early popularity in part by copying mailing lists directly into the system (Emerson, 1983). What distinguished forums from these lists was the public space they provide for user generated messages that resided outside of list members’ email inboxes.
The format of an email conversation - i.e. an original message followed by a series of replies - continues to be widely used as a structure of discussion forums. However, there are several key differences between email lists and forums. On an email list when a new user joins they have missed all the conversations on the list up until that moment. New users can join a forum and come up to speed on the discussions taking place, and browse historical discussions. This provides an easier path for outsiders to participate in the discussions currently taking place on the forum. With an email list one must both know about the list’s existence (it’s not a webpage that can be found using a search engine), request to be added to the list, and get an invitation from someone on the list.

The user base of early forum systems was greatly limited by access to any kind of computer network. When the World Wide Web became available to the public in the mid-90s forum usage grew with internet usage. Usenet continued to grow in popularity and many new web based forums emerged as well. These communities were organized by topics, locations or even the particular ISP of the users. Forum communities continue to attract large numbers of users, with Reddit recording over 140 million active users per month (traffic stats, n.d.).

The Forum Format

A forum is defined here as a web application with the following core features. (1) Users can create documents called posts which must have a title and optionally allow for a body, not unlike an email message. (2) Other forum users and the original post creator can then reply to this message with new messages which are untitled. (3) The responses will be attached to the original post and displayed in an order dependent on the forum’s design. Many forums display replies chronologically, while some modern forums use voting. (4) Users can browse a list of
posts by their title, and selecting a post displays it and all its replies. A post and its replies taken together are sometimes called a **thread**.

Forum behavior can be regulated by **moderators**, users with special privileges to lock or destroy posts or threads, and block misbehaving users’ access to the forum. Misbehavior is highly subjective; for forums that are dedicated to a topic, posts that are not on that topic may be moved to another area of the forum or removed completely. In some cases, asking a question that has already been answered in another post is grounds for deletion.

**Embedding Forums in Other Media**

Many modern web applications use a variation on the forum format. One common usage of the forum model can be seen on any website that publishes articles followed by a comments section. The article functions as the original post, and users can post replies, generally expected to be directed toward the article in some way. Unlike a forum, the author’s involvement in the subsequent discussion is generally not assumed.
Adding comments to some focal point is not limited to articles. Facebook allows users to comment on photographs or automated messages (status updates) on another user’s profile such as a change in relationship status or a new job. Sites containing media of almost any kind can add an optional forum at the end of the page to support open discussion about the content.

Motivation for the Use of Discussion Forums

An internet forum is a framework for discussion but provides no content itself. Use of the comment section in articles is optional. Therefore, users must volunteer their time and energy to create messages using these systems, and the system’s usefulness is totally dependent on a user’s desire to create posts.

A study that examined personality test scores of internet commenters (Buckels, Trapnell, & Paulhus, 2014) divided them into groups by their stated motivation for posting on forums:
debating issues (23.8%), chatting (21.3%), making friends (2.1%) and trolling (5.6%). Non-commenters accounted for the remaining 41.3%. This finding indicates a substantial percentage of forum posters are actively seeking the chance to engage in a debate.

The addition of a comments section to the articles on many major news websites has allowed further study of how comment volume is affected by article topic (Weber, 2014). Weber found the number of comments an article received was highly skewed depending on the subject of the article. The three news topics that prompt the most comments are power, controversy and damage. Power was defined as “the sphere of political influence of the persons or institutions associated with the event.” Damage referred to both property damage and individuals getting injured or killed. Controversy was defined as a high level of conflict taking place in communication only, as opposed to a physical conflict.

Pew found that a significant percentage (25%) of news readers also participate in comments (Kristen Purcell, 2010), sometimes first sharing the item of interest themselves on a social networking site and commenting on that site instead of the article website. Lee showed not
only that comments influence reader’s opinion on an issue, but that comments are used by readers to infer both the popular opinion of that topic and the impartiality of the author (Lee, 2012).

Taken together this research shows that articles can function as a prompt that can attract forum posters, and that some forum posters seek to debate issues.

Malicious Users

The term **trolling** is described as “the practice of behaving in a deceptive, destructive, or disruptive manner in a social setting on the Internet with no apparent instrumental purpose” (Buckels, Trapnell, & Paulhus, 2014). Users with the intention of disrupting discussion are a reality of running an online discussion forum (Atwood, 2015). Therefore, finding ways to quickly remove offending posts, while leaving the rest of the discussion intact is in the interest of both the forum operator and users. As discussed above, active moderation can provide one method for removing these types of posts.

To cut down the amount of effort required to moderate a forum a flagging system can be used. In this system users have the option to flag posts and replies as abusive. This way moderators can focus on flagged items, instead of checking every thread on the site.

![Figure 3. The World of Warcraft forums contain several ways to flag abuse.](image-url)
Instead of displaying replies in chronological order, some forums use a voting system where users may “up-vote” replies they like. Replies can then be displayed in order of votes. This has the potential downside of removing the continuity of a forum thread, but lessens the need to read the entire thread. This positive approach highlights posts of the highest quality, rather than removing posts of lower quality.

Figure 4. Reddit orders both its lists of posts and replies to a thread in order of votes by default.

Down-voting may accompany up-voting as a form of quality control. Replies that have more than some number of down-votes are automatically removed from the discussion or moved to the bottom of the thread. The potential downside of this approach is that posts that are simply unpopular, as opposed to truly inappropriate, may be down-voted by the community. When the post is on a controversial topic this issue can be amplified.

Software Designed for Structured Debate

While internet forums provide a platform flexible enough to facilitate online arguments, extensive work has been done in the field of computer supported argumentation (Oliver Scheuer, 2010). These systems perform many diverse functions including providing a structure (such as a tree or graph) relating positions in an argument, categorizing the parts of an argument (claim, data), providing pre-formulated sentence structure that can be filled in (“I assert _”) and much more.
Figure 5. Rationale maps responses to a statement into supporting and opposing nodes, each of which can have a rebuttal.

These systems require varying degrees of categorization for user input. This can be as simple as connecting more than one node with an arrow signifying one point follows from another, or as complex as tagging a statement with one of 30 categories, which can only be linked to other categories in prescribed ways (i.e. evidence can only be linked to claims).

A common feature of the systems surveyed by Scheuer et al is linking statements to other statements in a graph. Some systems using graphs only allow linking in a tree structure, while others allow any node to be linked to any other node, implicitly allowing loops in arguments. A formal structure may be added, such as making a legal argument or a scientific hypothesis, to a sequential threaded discussion, where the discussion still appears as it would in a forum, but the responses must fit in the imposed structure.

Internet forums provide a space for almost any text-based interaction from question and answer, to casual conversation, to debate. Structured debate programs, on the other hand, are designed specifically for argumentation and do not support dialogue of any other kind.
With the two types of debate platforms described above in mind it is possible to create a spectrum of debate platforms with argumentation software at one end and forums at the other. This application occupies a middle ground where some structure is imposed, but openness is also allowed.

<table>
<thead>
<tr>
<th>Forums</th>
<th>Spirited</th>
<th>Argumentation software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any types of content permitted</td>
<td>Main article permits any content, responses must fit structure</td>
<td>All content must fit the structure</td>
</tr>
<tr>
<td>Sequential posts regardless of content</td>
<td>Links responses to ideas</td>
<td>Visually maps argument</td>
</tr>
<tr>
<td>Users must read each post to understand relevance</td>
<td>Easy to read</td>
<td>Explicitly links ideas</td>
</tr>
<tr>
<td>Easy to read</td>
<td>Unstructured</td>
<td>Difficult to read</td>
</tr>
</tbody>
</table>

Figure 6 Continuum of debate platform imposed structure.

**Forums Built for a Specific Purpose**

One scenario where the flexibility of the standard forum can become a burden is a question and answer interaction. A typical forum lacks a prominent way to display at least two key pieces of information: (1) was the question already answered elsewhere and (2) which answer is the correct one. This means other users with the same question must read through the entire thread and find answers to these key questions themselves.

Stack Exchange, a network of sites built for question and answer interactions attempts to address the limitations of standard forums while retaining the benefits. On Stack Exchange all posts are assumed to be questions, and the author selects which answer was the correct one. The site allows users to vote on other answers, potentially letting the community select a different answer than the original asker. Comments are still allowed, but are limited in length and displayed less prominently. The benefits of the standard forum, like being able to browse topics,
come up to speed on discussions quickly and most importantly write anything a user wishes, are kept intact.

Stack Exchange also demonstrates a novel example of harnessing users' desire to contribute knowledge and directs that desire toward a specific purpose with a clear goal. This project applies a similar approach to debate, which also requires more structure than typical forum interaction to be productive.

Collaborative Annotation in Web Browsers

Collaborative annotation is defined here as adding localized comments to a specific part of a web page, be it a word, sentence, picture or some other component. A browser based collaborative annotation feature was being considered as early as the development of Netscape Navigator (in 1996) by Mark Andreessen (B Horowitz, 2012). Since then the idea has resurfaced several times. The most high profile projects were Third Voice, a browser plugin that launched in 1999 and was closed by 2001; Google Sidewiki, another plugin which launched in 2009 and was cancelled by Google in 2011; and most recently Genius, a web based annotation platform that launched in 2015 (Simonite, 2015). A more complete list maintained on Wikipedia shows more than 20 currently existing or defunct annotation startup companies dating back to 1997 (Web annotation, n.d.).

Currently an issue central to the success, or more often failure, of these systems is motivating users to create annotations. Several large news organizations, including Wikileaks, have created online annotation projects that have garnered minimal response (Simonite, 2015). One attendee of the I Annotate 2015 convention (devoted to online annotation in all forms) noted a key takeaway was “we need to develop more specific use cases and frameworks that inspire
people to participate.” (An Xiao Mina, 2015) This was in response to the recognition that simply creating the opportunity for users to annotate web pages does not generally inspire them to do so.

The only undisputed success in the annotation space is Rap Genius, the precursor to Genius, which is devoted to the annotation of rap lyrics. This is notable because unlike other annotation systems Rap Genius uses annotation for a specific purpose and with a clear goal (interpreting the meaning of lyrics). The other systems discussed act like sentence level commenting systems where users may write whatever they wish. It may be that giving users a clear goal when using annotation tools improves participation. Another possible reason for Rap Genius’s success could be that it gathers together enthusiasts who have a special interest in the material being annotated. When annotation is a means to an end rather than an end in itself user participation may not be an issue.

If the original definition of collaborative annotation is expanded, Facebook can be seen as another success in web annotation with the popularization of the Like button. Facebook Likes are similar to collaborative annotations in that they can be attached to parts of or an entire website, and users can see all other users’ Liked items. Likes can be created for almost anything on the internet and are widely used by Facebook users. However, the Like does not require users to create anything – they simply click a button. In February 2016 the menu of responses was expanded to include five other sentiments besides Like, but still does not allow users to enter text. The success of this simple mechanism may offer another clue to understanding what circumstances encourage users to participate in online annotation.
Chapter 2: Spirited, a Web Application for Structured Debate

The goal of this project was to create a web application built to facilitate debate online. The application builds on the popular model of a published article that allows comments, with additional features and restrictions designed to focus debate on the content of the article.

Addressing Problems with Debate Online

Besides the issues with open-endedness discussed above, a broader issue with any debate, online or otherwise, is that all participants must work from the same information for discourse to be coherent. For example, if one participant believes the national debt is the highest it’s ever been, and a second participant believes the government has a surplus of cash, a debate about the state of our country’s finances cannot make much progress.

Another issue that exists in all types of debate is that participants can make arguments that sound true, but don’t stand up to logical scrutiny, loosely referred to as logical fallacies. The US legal system prohibits these types of arguments from being made before a jury in a trial, categorizing them as “improper arguments.” One improper argument defined as “Comments attacking the opposing party, counsel or the opponent’s theory of the case” (Justice Peggy A. Quince, 2013) corresponds to the logical fallacy “ad hominem attack,” i.e. attacking the arguer instead of the argument. Using an improper argument in court can cause lawyers to be censured and in some cases lead to a case being declared a mistrial.

Meaningful Conflict Requires Structure

An application for online debate can address all the above issues by imposing more structure on the debate. Many systems designed to facilitate human conflict utilize a set of rules that govern how the conflict should take place. This can be seen in the legal system as mentioned above, but also sports – such as boxing – and even in war with treaties like the Geneva
conventions. The structure restricts how the conflict proceeds, limiting participants’ actions and the times of those actions. The outcome of a conflict that has taken place in such a structure becomes meaningful. Verdicts rendered by a jury hold legal weight. The winner of a boxing match is declared the superior boxer. This cannot be said of the winner of a barroom brawl, where anything can happen and brawlers are not bound by any rules. Similarly, an argument at a cocktail party does not hold the same weight as an argument in a debate competition.

In Spirited debates authors state their position in an article, and readers directly engage with another’s ideas and claims. User responses are focused on statements in a way that is designed to direct attention to only a few ideas at a time. A citation tool is provided to debaters to present evidence so that a set of facts may be agreed upon. This and other tools are designed with the goal of using debate about a statement to move toward a better alternative, by creating new statements that are both logically sound and factually true. When users want to support a statement they may also provide feedback by up-voting sentences and adding supporting responses to indicate agreement.

Attention Currency

The application uses a voting system where only up-voting is possible while negative attention is handled through contradicting annotations or flags in the case of abuse. This approach creates an environment where anything that prompts engagement is considered favorable. In an attention currency system, it is assumed that people ignore what they don’t like or are not interested in. Without down-votes it is not possible for users to simply hide or destroy what they do not agree with. Users must instead create something new to contradict a statement.
Encouraging User Investment

In his book *Hooked: How to Build Habit Forming Products* (Eyal, 2014) Nir Eyal discusses several techniques to encourage user engagement with software products. He explains that getting users to make some investment in a product will increase the chances that they will use the product again – investing more energy with each use and eventually forming a habit.

With this technique in mind, the application has many different ways for users to make small investments. Every level of content – articles, sentences and annotations – has a voting mechanism attached to it. Users may also create annotations attached to sentences, and are encouraged to write articles as it is the only way to express a new point of view. Creating many ways to interact, each with varying levels of investment, will ideally get users to invest in the application and thus encourage them to continue using it.

Early User Testing

In *Lean Startup* (Ries, 2011) Eric Ries suggests a process for designing, building and testing new software products with maximum efficiency. One of core principles is the Build-Measure-Learn cycle in which developers build a product, test it with real users and then apply what they have learned to the next build cycle. It is ideal to complete each cycle as quickly as possible to speed up development of the product.

This approach leads to the concept of the minimum viable product (MVP). This is an incomplete product that has the absolute minimum investment needed to implement the core idea so it can be tested by users. Ries claims it is advantageous to first test concepts with MVPs because often developers are incorrect about exactly what users want, and will end up changing the product once it is exposed to users anyway. If there was a large investment of time and effort in the software, it is ultimately wasted when these changes are made.
To implement this idea user testing began very early in development where only a bare-bones, un-styled version of the application was working. As the application changed, users were asked to return and test it again, providing new feedback each time.

Another principle that goes hand-in-hand with the early testing is enabling data collection of application usage. Because the development of the application was iterative, with each iteration based on the results of the previous version, it is necessary to collect data on how users are using the application as it changes. The Google Analytics platform was used to track how users navigated through the site and which features they used.

The User Experience

There are two primary reasons a theoretical internet user initially visits the site. One group has the same reasons they might visit a website like Huffington Post, Reddit or even NYTimes.com: to read and respond to opinion based articles. The other group, again comparable to Reddit and also Twitter or Facebook users, writes articles and watches how others respond.

When a user with an existing Spirited account logs in, they are shown the dashboard. Different sections display potential points of interest, some specific to the user:

- The weekly prompt, a question for the community to debate.

- Articles in the following groups or tabs:
  - Most liked articles
  - Newest articles

- A button to create a new article

From this point the user finds articles they want to read. Once the user has selected an article there are still many potential points of interest beyond the text of the article. Responses by other users can contain additional information, and the user can add a response if they find that
an issue they are thinking of has not yet been addressed, or to support the author’s argument. While reading responses can be interesting in itself, the option to respond or simply up-vote responses makes it an interactive experience. There may also be response articles, shown at the bottom, which the user can jump to.

In a typical session with the application a user would ideally find an article of interest on the dashboard, navigate to the article and continue to find statements they want to respond to and links to different articles. They also may be inspired to write a new article based on what they’ve read. Even after an article has been written the author continues to engage with the text through responses added by other users.

Linking to Articles from Outside the Application

Users will not always go to the home page and log in, they may be linked to an article from some other website. In this case the user will be able to view the article and annotations, as well as browse to other articles on the site. All buttons that allow the creation of annotations or votes are removed. If the user creates an account, their browsing experience continues like the one described above.

Administrator Level Features

Any website that allows anonymous users to create posts can become the target of abuse. It is important to have a fail-safe mechanism in place in the event that the abuse flagging system is either too slow or objectionable material is missed. Administrator features must allow site administrators to quickly remove articles or annotations that have been created by other users.

Administrators will have access to an administrator-only menu when viewing articles and annotations. This includes the ability to hide an article or response, making these items viewable only by administrators and adding a “Hidden” label to the item. A control panel area of the site is
also provided for administrators to search for and ban users. A ban logs out the user and prevents them from logging in again.

Content Policy

A key part of preventing objectionable material is defining what exactly the rules are for posting on the application. The application will have a clearly explained code of conduct that tells users what they may and may not post. Initially, abuse flags will only contain a single type of flag ("abuse"), but at later stages of development abuse flags will reference rules in the code of conduct directly. This will provide greater clarity to posters about why their posts are being flagged so they can correct their posts in the future.

Basic Application Flow

Usage of the application takes place mainly on the Article View page. The user may return to the dashboard to search for more articles if they don’t find an interesting link on an article page. The chart below shows the possible paths the user can take through the application.
Articles as the Center of Debate

When a formalized debate is held in the physical world participants come prepared to clearly make a case for their position on an issue. This opening statement, whether it is by a high school debate team or a prosecutor in court, anchors the rest of the discussion. The application allows authors to state a position in the form of an article and present evidence in favor of their position with citations, footnotes and definitions. Keeping the argument open ended, unlike the strict structures used in the debate programs discussed in Chapter 1, allows space for rhetoric and entertaining writing. Arguments that are more than a presentation of facts and logical conclusions may be more fun to read and garner more engagement from users.
Interaction with Articles

The interplay between article authors and other users with the text of the article is one of two primary ways a debate-like interaction takes place in the application. The other is through responding to articles with new articles. There are two types of actors on the text of an article:

- **Authors** create articles that can include annotations. Once the article is published they can use the Alteration annotation to replace sentences in their article with new versions. They can also create new annotations to the article. Authors can vote on any annotation they did not create themselves.

- **Readers** refers to any non-author user who reads an article.

- **Responders** are readers who are interacting with an author’s article. They can vote on all parts of the article (article, sentences and annotations), create new annotations and vote on other responder’s annotations.

The following use case diagram summarizes how both groups can interact with an article.

![Figure 7 Author and responder interaction with articles.](image)
Article Creation

Articles in this application serve a specific purpose: to make a clear and compelling case for a point of view. A text editor is provided to authors to enter their article, and they have access to a set of annotations that provide them with additional tools to make their argument. This gives authors a comparable suite of tools to responders who are given several types of annotations to respond to articles.

Author Citations

Citations are a standard feature of academic writing that attempts to make an evidence-based argument. The application allows authors to provide a link to an online document and a short description of why this document is relevant to their statement.

Author Footnotes

A footnote allows authors to make a comment or add clarification to a statement without interrupting the flow of the article. Unlike a standard footnote these are shown to the right of the document, mixed with other author annotations. Responders do not have access to a response similar to footnotes, so this annotation creates an advantage for authors, and an additional tool to make a clear case to readers.

Author Definitions

Authors may enter a definition of a word or phrase. Definitions allow the author to set the terms of the debate upfront and prevent confusion by indicating what exactly they mean when using certain words. This is especially important for charged terms, like socialism or feminism, which can mean any number of things depending on the context.
Editing Restriction

The application is based on a model where individual sentences anchor the subsequent debate. Therefore, it is very important for the original sentence remain available to view. Even a small edit could potentially change the meaning of a statement and make some of the responses invalid or irrelevant, creating confusion. For these reasons, once an article is published the original sentences can only be altered using the Alteration tool described below.

Responding to Articles with Annotations

In articles on news websites like CNN or the New York Times, an article is shown in its entirety and comments appear in a separate group, usually below the article. This creates a situation where commenters may respond to the article as a whole, or quote specific lines in the article when they want to address only that portion. But because comments allow any response at all, commenters can post messages that are completely unrelated to the article or any debate taking place. They can post different types of comments – those criticizing specific parts, overall criticism, conflicting information, and even unrelated material – which are all grouped together and listed chronologically or by votes.

This application splits articles into their component sentences and only allows users to respond to one sentence at a time. This is intended to both focus the debate on the content of the article and keep responses short and to the point. New articles may also be created as a response to another article, similar to a regular discussion forum. Users may respond to sentences with one of several annotation types described in the following sections.

Supporting and Contradicting Citations

Responders may provide a link to an online document, the title and a short description explaining why the document was cited. They may optionally mark their citation as supporting,
contradicting or both. Indicating the intention of the citation is meant to make browsing cited documents easier for readers.

Logical Fallacies

Logical fallacies are a flag-style response that exists only as a contradiction to the statement. Once the user selects logical fallacy from the annotation menu, he is shown a list of fallacies with descriptions to choose from. When a logical fallacy is applied it cannot be flagged again, but it can be up-voted. Flags do not allow text input.

Alterations

Alterations serve two purposes. For one they can accomplish the same function that answers do in the Stack Exchange model. They provide a potential end point for the debate surrounding the statement, where the author and responders can agree on a new statement. Responders can use alterations to provide a ‘better’ statement than the original. The definition of better is defined by the author with input from responders. Deletion of the statement entirely is a possible alteration.

Alterations can also be used by the author as a way to edit their article, even slightly changing the argument being made. Authors can create an alteration for a sentence, then vote for their own alteration. The application automatically replaces the original sentence with the author’s accepted alteration, while making the original available to view. It is also possible for the author to accept another user’s alteration and have that sentence replace their original in the article.

There is no way for responders to replace sentences in an article, only authors may use this function. While it is tempting to allow the community to agree upon changes to an article,
this could act as a form of graffiti where responders replace sentences with completely unrelated ones, change the meaning of arguments, or make articles incoherent.

Comments

Comments are free form responses where a user may write anything they wish, associated with a sentence. Due to their open ended nature, comments are relegated to a lower tier of response. These appear separately from other types of responses and are shown in a different format. Comments are meant to be used primarily to communicate about the statement for the purpose of clarification or inquiry.

Likes, Votes & Accepts

Every form of information in an article contains a voting mechanism, and total votes are displayed on each votable object. This includes the article itself, each sentence of the article, author annotations and responder annotations. Vote types vary based on what is being voted upon. Articles use Likes, a term that is already familiar to many internet users. Sentences and alterations use ‘Accept’ to differentiate this vote type from votes for other types of annotations which simply use Votes. The Accept type is unique in that only one Accept may be created per sentence or alteration of that sentence. An author’s accept has the additional effect of replacing the original sentence with the alteration.

Flags

Flags are an abuse checking mechanism. If enough users create more than some threshold number of flags for an article or annotation that item is automatically hidden. Administrators can override the effect of flags and show flagged items.
Responses to Articles with Articles

Responding to an argument sentence-by-sentence is a granular way to approach the positions taken by the author. If a user would rather respond to the article as a whole they may write a new article in response. A response article is like any other article, except that it is labeled as responding to the original article. Response articles will be shown prominently on the original article page so that users can follow the debate outside of the initial article.

Groups

The application allows administrators to create groups to separate articles from those that appear on the dashboard. Administrators can then publish articles to the group, causing it to appear in one of the special sections of the site, the blog or the content policy. These sections are set up to only load articles that are associated with the appropriate group.
Chapter 3: Software Components

This section details each component of the Spirited web application, the function of that component and the reasons for its implementation.

Use Cases

Before a user has created an account they are able to use a subset of features in the application. Anyone can use the demo on the landing page as well as being able to view articles (without being able to vote on them). The following use case diagram depicts how two types of users can interact with the Spirited application. The Existing Author already has an account on the application and has posted an article. The Enrolled User recently found the site, made an account, and started interacting with articles.

Figure 8 Use case diagram by user type
Landing Page

The landing page is the first page users see if they found a link to the site using a search engine or if they entered the URL in their browser. The goal is to provide an easy way to get to the login page if the user is returning, or to show what the site is about if the user is new.

The top of the page shows the name, a short explanation of the site and buttons to log in or create an account. The lower part of the page shows a small demo article with a few annotations. This article has the same features as an article on the site, except that any user may annotate it without logging in. The annotations are not saved. If a user annotates the demo a message appears suggesting they create an account.

Login

The login page only serves two functions for existing users: to log in or reset their password. An error message is displayed if a user enters the wrong email/password combination.
If a user has entered their email address they can click Forgot Password to have a password reset link sent to their email address.

User Registration

User registration allows new users to register on the application. The user receives notifications if (1) the username exists already, (2) their passwords don’t match or are too short, (3) the email address is already in use. The terms of service are displayed on the right. The Google reCAPTCHA service is used which require the user to check a box confirming they are a human (i.e. not a bot).

Dashboard

The dashboard provides users with a starting point for finding new articles of interest and creating articles. The prompt on the right suggests a topic for the user to create an article about. The area on the left is used to browse articles. Tabs allow users to sort by different groupings. It
is expected that users will return here to look for more articles once they have finished interacting with an article. A “Write Article” button is shown at the top right.

![Dashboard Displaying Articles](image)

**Figure 12** The dashboard displays articles and allows users to create new ones.

**My Articles**

The My Arguments page contains two simple lists of all the articles a user has created, including published articles and unpublished drafts. This page is only accessible from the dashboard.
Create Article

Articles are the central focus of the application, so the article editor plays a key role in the user experience. Users must be able to write an article and easily create and insert annotations. The editor uses a What-You-See-Is-What-You-Get (WYSIWYG) editor to format text, with annotations alongside the entry box. Users can discard the article or save it as a draft if they don’t want to publish it. The draft is saved automatically every 30 secs.

Draft annotations are created in the text editor, which opens a short form that prompts the user for the information required for that annotation. Draft annotations have buttons to insert, edit and delete the annotation. The arrow button inserts a reference to the annotation at the cursor in the article entry box. References are numbered according to the order they appear in the text, and renumbered if they are reinserted in a different order.
The article view page is meant to facilitate reading without distraction. The option to view all non-author responses to a sentence is always available, but these annotations are not shown initially by default. Sentences or author accepted alterations of that sentence are always shown, while previous sentences and other responses are only viewable by selecting sentences.

By default the author’s annotations are shown at the right of the article horizontally aligned with their references in the text. Sentences are highlighted with a progressively darker color depending on how many annotations are attached to that sentence. This highlight maybe be removed by clicking a Show Activity / Hide Activity button on the page.

Annotations have a reference number in the text that also appears next to the annotation on the right. When hovering the mouse over the number in the text the annotation on the right changes color. This makes it easier to find the referenced annotation.

At the bottom of the article a Like button and a Create Response Article button are shown. If there are response articles they are listed under a Responses heading below this button.
Article View with Responses

The article view with sentence’s responses toggled displays a few items to the right of the article: an accept button for the sentence, a reply button to create annotations, a filter button to select which annotation types are shown and any annotations that exist for that sentence. Selecting a different sentence shows only that sentence’s annotations. All the current annotations are shown with the number of votes and a vote button next to them. Users may select a certain type of response to sort the shown responses. A Respond button appears at the top to add new responses. Comments are shown on a separate tab, and in a simpler format than annotations.

In the article body the selected sentence changes to a color other than the default black, and all remaining sentences turn grey, making it easy to see which sentence is selected.
How Broken is This

Working on a thesis draft but I feel I must test

by josh8790

Having a method for calculating actual volatility allows us to calibrate the VIX against the market and track the ratio of the VIX to true volatility (VIX/true) over long periods of time, which is of course quite grand. This experiment is also supported by decades of freely available historical data that the CBOE has posted on its website. All of the information can be downloaded in spreadsheet form with the click of a mouse. The comparison yields phenomenal results.

A spreadsheet was constructed with daily closing prices for both the VIX and the S&P 500 beginning in January 1999, right before the new millennium. This time frame was chosen because it predates the NASDAQ market crash by more than one year. Across the entire time frame of the study, approximately 3,000 trading days, the ratio averaged 1.3—that is, the VIX averaged 1.3x the underlying volatility of the S&P 500 calculated in a 20-day window. It climbed sharply only four times between the beginning of 1999 and 2010. During each of these events the ratio rose above 2.3.

Figure 16 An article with a sentence selected and multiple responses.

If the Respond button is clicked a menu appears allowing the user to select one of several annotation types. Citations show a form which requires a user to add a link URL, name and description. Logical fallacies list possible types of fallacy and allow a user to select one.
Figure 17 The response menu collapsed and with citations expanded.

Alterations bring up a separate menu where the user can edit the current sentence and insert existing annotations and any new citations that have been created. Comments can also be created from this menu.

Users may vote for citations, logical fallacies and comments without restriction. A single vote is shared between each sentence and all its alterations. Users must decide which the ‘best’ version of the sentence is. Once a fallacy has been created, if another user tries to create this fallacy it acts as an up-vote for the original.

Settings

A bare-bones settings page is implemented solely for users to change their passwords. This page was designed to be expanded with additional settings as they become necessary.
Feedback Forum

A simple feedback forum is available to all users so that developers can communicate proposed changes and bug statuses, and for users to provide feedback about their experience. This forum is as minimal as possible, simply allowing posts and responses, and displaying them in chronological order. No voting mechanism will be used.

Article & Annotation Hiding, User Banning

Administrators have an option to hide articles and annotations using buttons that appear on articles and annotations. Individual sentences cannot be hidden. Hiding removes an item from the view of all users except administrators and the original author of that item. Users will also have the ability to hide their own annotations after they create them.
Password Recovery

A standard feature of virtually every modern website with a login system is password recovery. This is a system where a user may click a link indicating they have forgotten their password. They are then sent an email containing a link to enter a new password. This link corresponds to a unique page created for that specific recovery request, where the user can create a new password.

Administrator Control Panel

The control panel is accessible only by administrators and allows for tracking and banning of users. Users can be found by entering their email or username. The ban button logs the user out and prevents them from logging in again. The Show Locations button show all the IP addresses each user has logged in from.
Chapter 4: Architecture

This section describes the application architecture for the project. This includes the technology used to implement the application.

Technology

The user interface described in the previous section has several requirements that narrow the type of web based technologies that can be used for implementation. The application is based on internet forums and news websites, so the most basic requirement is that the technology be web based.

The user facing portion of the application implements several features that resemble desktop software – a full featured text editor and a text and response viewing system that uses extensive user interactions and alters the contents of the page dynamically. Static HTML webpages cannot provide these features, so heavy use of JavaScript was a forgone conclusion.

The application is collaborative in nature – it is most useful when users are seeing other user’s contributions. This meant that the browser client had to communicate with a centralized system that aggregates all user responses and provides each client with an overall picture of what is happening on a given article or sentence. An API server that provided real-time data to the application as the user interacts with it provided a simple solution to this problem.

Frameworks

Frameworks are generalized software packages that provide basic functions that can be changed and expanded by other programmers to create new applications. Each framework is designed to build some general class of application in a certain programming language. For example the Django framework allows webservers to be built using Python. The Bootstrap CSS framework provides a grid system for styling web pages.
Frameworks create a huge advantage for developers by providing code for common operations, leaving only new and novel features to build. Frameworks were used to build both the JavaScript client and the API server for this project because they greatly sped up development allowing work to focus on the unique features of this application.

Ember

The Ember JavaScript framework was used for the browser based client of the application. The client handles UI implementation and downloads data from a remote server. This framework uses strong conventions to simplify JavaScript code, as well as a templating framework that greatly reduces the amount of HTML markup that must be written. Ember uses its own object model that extends JavaScript objects.

This project also uses the optional Ember Data library, which allows interaction with the API server through a REST data service. This library greatly simplifies data access by importing data into Ember objects, as well as handling saving and creation of new records on the server.

Rails API

One way to utilize the conveniences provided by Ember Data is using a RESTful data service. REST imposes constraints on how queries are formulated so that they may be contained in HTTP requests. The Ruby on Rails (referred to simply as Rails) framework is used on the server to serve JSON data and connect to a MySQL database. Database structuring and modification is done through Rails; very little work will be done directly in the database. The framework has the advantage of being compatible with Ember out of the box. Very little additional configuration is required.
MySQL

The MySQL database is chosen because it works with Ruby on Rails and is inexpensive. If the application scales to a larger user base the database will be migrated to a system more appropriate for large datasets, such as Amazon Aurora.

Development Servers

Both of the core frameworks used in the project, Ember and Ruby on Rails, provide development webservers as part of their respective installations. Unlike production webservers, these small programs are easy to start and stop, automatically reload when a file is changed, and print detailed output every time a command is executed. These servers were used exclusively for development and debugging, with new versions of the project being deployed to the production servers only when the latest changed were working.

Both Rails and Ember allow for separate development and production configurations. With Rails this made it possible to use a local SQLite database during development, while using a MySQL database served remotely by Amazon RDS in production. Similarly Ember used the Rails API served on a local machine in development, while it pointed to the API server running on an Amazon Elastic Compute Cloud (EC2) server in production.

Version Control – Git

The Git version control system was used to track and save all application code. The repository was stored privately on GitHub servers. The use of Git simplified deployment from development to production because the Rails framework is designed to use Git repositories. A Rails project can be downloaded from a Git server then installed on a new machine with a few commands.
Amazon Simple Storage Service (S3) provides fast, reliable and inexpensive cloud storage. The Ember client is a static file, making S3 an ideal candidate to serve the application, rather than maintaining a webserver to server a single static file. The service organizes data by named groups called buckets. Amazon’s Route 53 DNS management service is used in conjunction with S3 to allow the domain name beta.spirited.net to refer to an S3 bucket containing the Ember application. An Ember add-on (ember-cli-deploy) deploys new versions of the application directly to the S3 bucket.

Overall Component Structure

Two key components communicate with one another to run the application: an Ember client that runs in the user’s browser and the Rails API server that serves data to and saves data from the client. The Rails API servers run on Amazon’s EC2 platform using the Ubuntu 14.04 LTS Linux distribution.

The API servers are part of an “Auto Scaling Group” that allows EC2 to create and destroy copies of the server based on CPU load. Figure 21 Component structures shows 3 servers, but there may be only one or as many as 10 servers running depending on demand. When a request is made to beta.spirited.net it is redirected to a load balancer, which routes traffic to one
of the available EC2 servers. Each Rails API server uses a shared MySQL database running on Amazon’s RDS platform. Other than initial creation, the database is managed exclusively by Rails.

API Servers

Amazon Elastic Compute Cloud (EC2) provides an array of services for managing cloud based servers. EC2 offers services are used by the application to control access to the servers, manage server load, log server errors and deploy new versions of the application.

The Server Image

A single EC2 instance running Ubuntu 14.04 LTS was created to host the Rails API server. Besides a fresh installation of the operating system this server was loaded with Ruby 2.3 (required for Rails) and the Node.js JavaScript library. The Ruby on Rails project was then installed by downloading it from the Github project. Finally, the Rails webserver package Passenger was installed and set up to run the instance as a production webserver.

Auto Scaling and Load Balancing

While the instance described above was capable of running as a standalone server for the API, allowing the application to scale indefinitely was a desirable quality. This was accomplished using EC2’s Auto Scaling service. The server was shut down and a copy of its hard drive, called an image, was created. This copy is used by the auto scaling service to create as many copies of the server as necessary, based on the CPU load of the existing servers.

A second feature used in conjunction with auto scaling is the Load Balancer. This service accepts requests meant for the web server and routes them to one of the servers in the auto scaling group. As the name implies this balances the load on the application between the available servers.
Because the application uses token based authentication it is necessary for server requests to be transmitted securely using the SSL protocol, which transmits data using HTTPS. This is enforced by setting the load balancer to only accept HTTPS requests. The load balancer decrypts the requests and sends them to each server using HTTP. Amazon provides a service for creating SSL certificates, making the load balancer a much simpler way to handle SSL requests and decryption, compared to configuring the servers to handle SSL.

Security Groups & Virtual Private Cloud

EC2 allows one or more servers to be hosted in a restricted group called a Virtual Private Cloud (VPC). The load balancer, all the web servers and the database are all contained within a single VPC, which acts as a virtual data center containing all the components that serve the Spirited API. Each instance in the group has a configuration file called a security group. Security groups limit access to the server to certain protocols, ports and IP address ranges.

The security groups for this project are set up to prevent any communication that is not absolutely necessary for the application to function. As such the load balancer is the only component in the VPC that is exposed to the public internet. Web servers only accept requests from the load balancer, and the database only accepts requests from the web servers.

Error Logging

Amazon provides software that allows Ubuntu to stream messages generated by each server to Amazon’s CloudWatch monitoring service. As described above the production servers are inaccessible by traditional means, and this service is used to allow server activity to be monitored without direct access to the server. The log files that are generated by Rails can be read, searched and filtered in the AWS console.
Spirited API

The Spirited API server runs on Ruby on Rails, a web application framework written in the Ruby programming language. This framework can be used exclusively to build a complete web application, including the client portion. In this project the Rails API variant was used, which removes the front end functionality of Ruby on Rails and only provides a RESTful API.

RESTful API

REST refers to “representation state transfer”. Rather than a protocol, REST is described by its creator Roy Fielding as “a core set of principles, properties, and constraints.” The Rails implementation of REST creates a web server that responds to HTTPS requests with certain data contingent on the type of request and the address requested.

For example, when the webserver receives a GET request to beta.spirited.net/articles it would select the all the articles in the database and return them in JSON format. If it received a GET to beta.spirited.net/articles/1 it would select the article with the id 1 and return it as JSON. If it received a POST request to beta.spirited.net/users it would expect a JSON payload containing the information required to create a new user, and insert the new user data into the database. These simple conventions can be implemented any number of ways, the only requirement is that each message be handled in the prescribed fashion by the receiving server.

The full extent of the Rails API functionality is elaborated in the Spirited Project Structure section of this document.

Rails Project Structure

Once the Ruby programming language and the Node.js JavaScript Library (two key dependencies) are installed on a server a Ruby on Rails web application can be created by
running a single command. After the project is set up additional features are added by executing specialized Rails commands.

In Rails a **scaffold** refers to a collection of files that together facilitate the RESTful interaction described above. If we wish for our server to be able to store and serve Post records with title and content attributes, the following command generates the necessary file collection:

```
rails generate scaffold Post name:string title:string content:text
```

The following sections describe each of the files created by the scaffold command:

**Database Migration & Schema Update**

Rails handles all interaction with its associated database; developers should never have to modify the database directly. A migration file is a set of instructions that allow Rails to add the necessary table and columns to the database. Rails also maintains a separate schema file that describes the structure of the database which is updated to match the current state of the database.

**Model**

A model contains information about the relationship between a resource and other resources in the system. For example, users and articles have a has-many/belongs-to relationship, so the Users model would declare that Users has-many Articles, while the articles model would declare that Articles belongs-to Users.

Models may also contain **scopes**, which are constraints on what data in a model should be selected, similar to the WHERE clause in an SQL statement. For example an Alert model might contain a scope that tells Rails to only select Alerts that have not been acknowledged.
Controller

The controller contains the primary logic for handling the various types of HTTP requests and responding accordingly. The following table lists the default function and response for each type of HTTP request.

<table>
<thead>
<tr>
<th>HTTP Request</th>
<th>URL Pattern</th>
<th>Action Description</th>
<th>Requires JSON Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/resource</td>
<td>Show all resources</td>
<td>No</td>
</tr>
<tr>
<td>GET</td>
<td>/resource/1</td>
<td>Show resource 1</td>
<td>No</td>
</tr>
<tr>
<td>POST</td>
<td>/resources</td>
<td>Create a new resource</td>
<td>Yes</td>
</tr>
<tr>
<td>PUT or PATCH</td>
<td>/resource/1</td>
<td>Update resource 1</td>
<td>Yes</td>
</tr>
<tr>
<td>DELETE</td>
<td>/resource/1</td>
<td>Delete resource 1</td>
<td>No</td>
</tr>
</tbody>
</table>

By default, these functions respond to any request. Modification was required to many of the default functions for security purposes and when only a subset of records was required.

Rails can parse parameters passed as part of the URL. For example, a GET to the following URL

```
http://beta.spirited.net/articles?sort=ASC&page=2
```

would allow the articles controller to make the parameters `sort` and `page` available with their corresponding values. These values can then be used in database queries, which are also specified in the controller.

Serializer

The serializer instructs Rails on what attributes are to be returned when serving data from a resource. This defaults to all the attributes in the model, but may need to be altered in cases
where private information is in the model (birthdays) or when information is only used by the server and is not used by the client.

Spirited Project Structure

There are only two key models in the Spirited application that all other models are organized around: Users and Articles. All other models belong to one or both of these, or in some cases belong to a child of one of these. Articles also belong to Users.
## Table 2 Spirited API resource types

<table>
<thead>
<tr>
<th>Name</th>
<th>Belongs To</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Group (optional)</td>
<td>A user account.</td>
</tr>
<tr>
<td>Article</td>
<td>Group (optional) User</td>
<td>Article around which debates are centered.</td>
</tr>
<tr>
<td>Sentence</td>
<td>Article</td>
<td>A single sentence of an article.</td>
</tr>
<tr>
<td>Fragment</td>
<td>Sentence</td>
<td>The text part of a sentence, or a reference number associated with an annotation.</td>
</tr>
<tr>
<td>Annotation</td>
<td>Sentence User Fragment (optional)</td>
<td>Content attached to sentences by authors or other users. Some fragments represent references to annotations.</td>
</tr>
<tr>
<td>ArticleVote</td>
<td>Article User Fragment (optional)</td>
<td>These are called ‘Likes’ on the site.</td>
</tr>
<tr>
<td>SentenceVote</td>
<td>Sentence User</td>
<td>Called ‘Accept’ on the site.</td>
</tr>
<tr>
<td>AnnotationVote</td>
<td>Annotation User</td>
<td>Votes for an annotation.</td>
</tr>
<tr>
<td>Group</td>
<td>User</td>
<td>Administrators can create groups to separate articles from the rest of the site.</td>
</tr>
<tr>
<td>Draft</td>
<td>User</td>
<td>Precursor to articles.</td>
</tr>
<tr>
<td>DraftAnnotation</td>
<td>Draft User</td>
<td>Annotations created in the Draft editor by authors.</td>
</tr>
<tr>
<td>Post</td>
<td>Post (optional) User</td>
<td>New or reply posts on the feedback forum.</td>
</tr>
<tr>
<td>Notice</td>
<td>User</td>
<td>Notices a created by administrators to disseminate information to users. They appear on the dashboard.</td>
</tr>
<tr>
<td>Reset</td>
<td>User</td>
<td>A record containing the status of a password reset.</td>
</tr>
<tr>
<td>Location</td>
<td>User</td>
<td>A location a user has logged in from.</td>
</tr>
</tbody>
</table>

### Additions and Changes to Generated Code

The Rails framework provides code for a general default use case where a web server provides any information requested, and stores any properly formatted information it receives.

The Spirited application is based on a model where users must log in to use most of the website’s features, and much of the user experience is tied to their identity. For example users need to see what articles they have liked, what annotations they have voted for, their own drafts, etc. These features are enabled by modifying the code generated by Rails in a number of ways.
Auth Tokens & User Identity

An auth token is a randomly generated string of characters that is passed in the HTTPS header of every request from the client to the server. The token is first transferred to the client in exchange for a correct username-password combination submitted by the client. The server-to-client transfer of the token only happens this one time. The client stores the token in the browser’s local storage and passes it back to the server with each request.

Spirited uses tokens to establish the user identity of the requestor for most requests by searching the database for a passed token. If no token is present the user is treated as anonymous, limited the data they can retrieve. User identity is important for several parts of the site to function properly:

- Drafts can only be viewed and edited by the author
- Only vote objects (likes, accepts and votes) created by that user should be downloaded and shown. Vote counts are a separate attribute.
- Only the notices a user has not acknowledged should be downloaded.
- Use of the Accept button on alterations modifies the text of the article only if the user is the article’s author

Administrator accounts have special privileges that alter or add features, making establishing identity crucial. For example the ability to set articles as hidden, removing them from public view is something only administrators can do. When a request is sent that attempts to modify an article’s ‘hidden’ attribute the controller must first check if the requesting user is an administrator before making this change.
Auth tokens use the autoscaling and load balancing features of the infrastructure to maximum effect. Because user identity is established at the beginning of every request there is no requirement that a client’s requests go to the same server during a session. This way any server can service any request. No matter how many servers are being used by the load balancer, all servers use the same database. The database handles transactions from the servers in the order received preventing conflicts or race conditions.

**Polymorphic Annotations**

Rails allows for polymorphic associations between models. In a polymorphic association a model exists as a sub-type of another model. Any association that parent type has will apply to all sub-types. This pattern is ideal for annotations in Spirited because while each type of annotation has unique fields (for example only citations contain links) all annotations are often grouped together in practice.
Polymorphic associations are executed in a simple manner in Rails. A ‘type’ field is added to the Annotations table that is used to store the sub-type of the annotation. The Annotations table also contains columns for every possible field in every type of annotation. Unused fields are set to null. A comment annotation would have null values for links, logical fallacy, and any other fields not used by comments. However, Rails allows querying and creating of all the sub-types as if they were unique resources.

It’s possible to write controllers for each annotation type separately, but that was not necessary for this project. The controllers for each sub-type of annotation simply inherited their logic from the parent Annotation controller.

Including Associated Data

The ability to mark resources with belongs-to or has-many allows for a useful feature where child associations can be included when sending data for the parent. For Spirited this was ideal because the key models in the application – articles, sentences and annotations – are meant to be shown as a group. Rather than making 3 separate calls to the server, one request can be
made for an article and the articles controller can be programmed to always include sentences and annotations with every article.

Password Reset

The password resetting creates a vulnerability in the application because anyone who knows a user’s email can request a password reset. It is therefore important to restrict the usage of this feature. The Reset object creates tokens similar to how they are created for logged in users. These tokens are then embedded in a URL that is sent to the user’s email. Visiting this URL is optional, and users are instructed to only use the URL if they did indeed request the password reset.

Finally, Resets are unique and have an expiration date. Once a Reset is created a new one cannot be created until the existing one expires. This prevents an attacker from ‘spamming’ password reset emails to another user.

Passenger Server

Rails contains all the code to handle requests on the server itself, but not the ability to actually receive those requests over the internet – this is done using the Passenger library. The library includes an integrated web server which receives requests from the internet as well as all necessary components to deliver those requests to the Rails application.

Spirited Client

The Ember JavaScript framework enables the same style of programming in the client that Rails provides on the server: a base application that provides core services which was extended to fit the requirements for the Spirited application.
Ember Project Structure & Features

The structure of an Ember application is similar to a Rails application in several respects. Ember uses URLs in a way that is similar to REST requests, loads data for that URL, and has customizable controllers that can change how that data is processed. Additionally, Ember uses a templating system that integrates programming features like variables and conditionals to HTML markup.

Routing by URL

When an Ember application initially loads all the code necessary to display every page in the application is loaded, with the exception of server data. The application takes control of the browser URL so that navigating to different parts of the application appear as though they are different web pages. In fact these are all the same application and URLs are instead being used to encode application state.

For example, pointing the browser to beta.spirited.net/articles/1 initially loads the same core application as beta.spirited.net/dashboard. The application then reads the URL and loads a different template, downloads appropriate data, and displays that data in the template. This powerful features allows links to pages on the application to be shared just like a standard web page, even though the pages exists only as an application state.

In Ember a URL is associated with a route. Routes at minimum include a template, a file that instructs Ember what to display at a route. They can optionally load data from the webserver and implement a controller for additional custom behavior.

Models & Ember Data

Ember Data is a separate library for communicating with a webserver and storing data locally, which can be included and integrated with Ember. A model file is created for each type
of resource the application needs to represent, then Ember Data requests this data from an application server using a RESTful request, and stores the response in the browser’s local storage.

Models must match the returned data for this process to work properly. In an Ember-Rails project like this one requiring the creation of models involves mirroring in the client what already exists on the server.

Models can be associated with routes so that when a route is visited Ember loads the associated data from the webserver. For example, the Ember route beta.spirited.net/articles could be associated with the webserver URL api.spirited.net/articles, which Rails associates with all the articles in the database. When the Ember application loads the articles route it makes a request to the API URL that matches its own route name. Both Ember and Rails have defaults that encourage routes and RESTful URLs to match, but this behavior can be altered if necessary.

Templates

The minimum requirements to display a page in Ember are a route and a template. The template can include at minimum HTML markup, and also supports displaying data from a model and/or controller associated with the route. If a model contained a name attribute, displaying the name in bold within the HTML markup would be accomplished by the following code:

```html
Hi my name is <b>{{model.name}}</b>!
```

Similarly controllers can provide computed values based on either the model or generated from the controller itself.
Routes can be created with a nested structure, for example /articles and /articles/index or /articles/new. This nested structure would be facilitated in these page’s templates using an **outlet**. An outlet can be inserted into a template and as the name suggests it allows another template to be displayed in a parent template. This feature is used extensively in Spirited to allow the main article to remain displayed at the left of the page while the right panel can contain annotations or the annotation creation interface.

A spreadsheet was constructed with daily closing prices for both the VIX and the S&P 500 beginning in January 1999, right before the new millennium. This time frame was chosen because it predates the

Figure 25 An annotated sentence before it has been selected.

Figure 26 A selected sentence displaying an attached citation and alteration.

Figure 27 A selected sentence with the Respond menu open.
Controllers

Unlike Rails, controllers are optional in Ember. A template can display model information without a controller. However, when more than just displaying information is required, such as responding to user interaction with the template, a controller may be used.

Controllers allow the creation of computed properties. These properties can incorporate information from the model as well as data entered into a field in the template. Model information is readily accessible to the controller just as it is in the template.

```javascript
// A computed property that combines first and last name
fullName: Ember.computed('model.firstname', 'model.lastname', function() {
    let first = this.get('model.firstname');
    let last = this.get('model.lastname');
    return first + ' ' + last;
});
```

Controllers support interaction with the template through **actions**. Actions are the label given to controller functions that run in response to an event on the template. To facilitate the action, code must exist to support it on both the template and in the controller.

```html
<!-- template -->
<button {{action 'popupWindow'}}>Show Pop Up</button>
```

```javascript
// controller.js
actions: {
    function popupWindow() {
        this.set('popup_showing',true);
    }
}
```
Using the parts of the Ember application described so far, loading an ‘articles’ route in an Ember application can be summarized by the following diagram.

Figure 28 Ember application component relationships.
Components

In Ember a component refers to a reusable sub-template with accompanying logic. A component uses the same structure as a template and controller, with one file providing the markup and a separate file containing executable logic. Components can be inserted into any number of template as many times as needed. The calling template must provide data for the component, unlike templates which receive data from their route.

Services

Services provide application-wide callable functions. Services work very similarly to controllers, except that instead of serving requests from a specific route they respond to requests from all controllers and components in the application. Services are useful for storing and loading persistent information like user identity.

Spirited Project Structure

The Spirited Ember application is structured similarly to the Rails application that supports it. All of the models represented by Rails are also reproduced in Ember, with the Ember application requiring a few more models to handle client specific tasks like logging in and out and creating new users.

The following table shows the routes the application supports and the associated data downloaded from the Rails server.

<table>
<thead>
<tr>
<th>Route</th>
<th>Rails Requested URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>/articles?frontpage=true/logical-fallacies</td>
<td>The index page implements a demo version of the article interface using a short article.</td>
</tr>
<tr>
<td>/login</td>
<td></td>
<td>The login page where users enter their email and password</td>
</tr>
</tbody>
</table>
Implementing the article editor presented a unique challenge. Allowing the editor to behave like a traditional desktop text editor with features such as text formatting, undo and copying and pasting was desirable. The novel feature of allowing citations to be created and inserted into the text was not implemented by any known third party libraries and had to be developed from the ground up.

HTML 5 included the new content-editable field attribute. This allows users to edit the actual text of the HTML on a webpage. A content-editable field seemed to be the ideal method of implementing the editor because it would allow for insertion of HTML elements (superscript numbers or images) directly into the text while a user was editing.

Unfortunately by itself a content-editable field has several issues due to uneven implementation between browsers. Below are a few of the differences (Joyce, 2015):

- Return can insert a &lt;br&gt; or wrap a paragraph in &lt;p&gt; or &lt;div&gt; tags, or both
Spaces may appear as a standard space or an HTML space character.

Backspace can sometimes insert span tags around the character before the backspace.

In addition to these discrepancies, the text of superscript annotation references becomes editable, which essentially breaks the editor. Allowing users to edit the annotation numbers of their annotations causes excessive complication.

Instead of an unmodified content-editable field, the Summernote JavaScript library was included in the project. This library occupies a middle ground between a content-editable element and a text editor. While content-editable is used it is heavily altered to produce the same behavior between browsers. This library also supports the insertion of images which provided a much simpler way to represent unalterable annotation reference numbers.

**Annotation Component**

Annotations are one of the core mechanics of the Spirited application. At minimum they will be displayed both on articles as well as on the front page of the site. The desire to reproduce this feature multiple places made Ember’s component structure ideal for building annotations.

**Auth Tokens**

The ember-simple-auth add-on was used to manage the auth token required to track user identity. This add-on implements several key features surrounding auth-tokens.

- Storing the auth token in the browser’s local storage
- Authenticating the email password/combination with the server
- Inserting the token into every HTTPS request to the server
- Protecting routes in the application from unauthorized access

The add-on provides these features, which could have been a project in itself, and can be installed via command line and easily integrated with the existing Ember application.
User Service

Many features in the application utilize the user’s identity when creating or altering objects. For example showing which annotations belong to a user requires knowledge of that user’s id when displaying annotations. Maintaining a persistent record of the user’s identity was done using an Ember service.

The user service provides global access to basic information about the user – their name, id and admin status– which are the only pieces of information that are generally relevant to both the user and the application itself. The id is reused any time the user creates a new object: articles, votes, annotations and posts all require this field. The id is also used to check when annotations have been created by the user and labeling them differently in response.

The user service is also key to implementing administrator privileges. Administrators are the last line of defense when it comes to removing objectionable content from the site. Because all content on the site is user generated it is important to be able to quickly and easily remove anything that has not been flagged.

Articles and annotations can display an administrator-only Hide button dependent on whether the user is an administrator or not. There is also the Control Panel section of the site, which is only accessible to administrators, that allows them to search user names and ban users. Banning resets that user’s token and prevents them from getting the new token via their login credentials.

Google Analytics

Google offers a full featured analytics platform free of charge. On a site with static web pages this tool can be enabled by pasting a block of JavaScript code on to every page. The
Ember application only loads one page, but dynamically changes the content of that page, so more advanced usage of the Google Analytics library was required.

Google Analytics allows pages to send different types of signals to its platform. The most course signal is a pageview, which like the name implies simply indicates a page was loaded in the user’s browser. It is also possible to send events which track user interactions with the page. Spirited makes extensive use of events to track interactions like up-votes, sentence selection and annotation creation.
Chapter 5: Usage and User Feedback

This section discusses the results of user interaction with the Spirited application.

The User Test Group & Feedback Collection

A group of the author’s friends were recruited to be the test users for the application. Requests to create an account and experiment with the application features were sent to each person on an individual basis, with more people being added over the course of the project. Although a feedback forum was provided, only one user choose to provide feedback using this forum. User impressions and experiences were gathered over emails and phone calls.

Difficulty Getting People to Use the Site

The author’s friends complied with requests to create an account, but getting them to use the features, or to return to the site even a second time, proved to be difficult. Two main issues were cited in subsequent discussions: they didn’t understand what they were supposed to do, and there were not articles they were interested in.

The first issue prompted the development of notices on each page for new users which explain how the site works. Most users did not understand that the application required them to select a sentence, or that they could create new annotations.

The Chicken & Egg Problem

The second issue – users not finding articles they liked – was a classic startup problem sometimes called the “chicken and egg problem.” Sites like Reddit, Stack Overflow and Spirited are based around user generated content. Users come to these sites to read things other people have written and perhaps write something themselves, with the understanding that it will be read by others. When there is very little to read there is no reason to stay, and there is no reason to write because there is no one to read it.
The articles initially posted on Spirited dealt with the US Senate’s refusal to hold hearing for a Supreme Court nominee, a topic the author thought was quite controversial. However, after several people had been invited to the site a pattern became clear – only the users with a personal interest in politics actually interacted with the article. This led to the conclusion that many articles need to be created to engage many different kinds of users. But this also creates a bigger chicken and egg problem. If users are spread throughout the application, all looking at different articles, it is as if they are the only person using the application, making many of the features pointless.

Commenting vs All Other Annotation Types

Users overwhelmingly choose to create comments versus any other type of annotation. For much of the development time alterations were not fully implemented. Only citations, fallacy flags and comments were useable. Several users mentioned the list of logical fallacies was too long and they didn’t want to scroll through them. The preference for comments can be seen as a reversion to the familiar – comments are the only type of response currently available on news sites and forums.

A few users said they wanted traditional comments in addition to sentence level comments. One user wrote a response article with the opening line “This is not an article but more of an observation.” This presented an interesting problem because the site was designed to minimize the impact of comments. It may make sense to add a smaller article creation tool at the end of article, and to show response articles in a way that is similar to traditional comments to meet user expectations.
Editorial vs Other Types of Writing

When user interest in article topics appeared to be an issue, a simple solution was to duplicate articles found on the internet covering variety of topics – news, speeches, etc. This exposed the need for a more clearly defined article type on the site. News articles in their most basic form deliver factual information. The tools implemented by Spirited are for debates, which are in some way opinionated. This showed that the application does not just require articles, it requires articles of a specific type, generally what are called editorials when published in a newspaper. This requirement presents a serious barrier to overcoming the chicken and egg problem, because unlike a site like Reddit where users can write anything about any topic, for a debate to make sense the user must write an opinion piece in favor of a position.

How Minimal is a Minimum Viable Product?

The entire development of the Spirited application was carried out with concept of a MVP in mind. Users were shown very early versions of the site before it was complete. The responses from the users and overall lack of interaction drew into question the very concept of a MVP. How little was too little?

The author sat behind an early user while she tried a very basic, un-styled version of the site. Her focus kept coming back to how ugly and unprofessional she thought the site looked. “I would never use a site that looks like this” was a more memorable comment. The lack of explanations, mentioned above, was another point where giving users access to the application too early seemed futile because the implementation lacked something so fundamental (like professional appearance) that the idea never really got tested.

In *The Lean Startup* Eric Ries share an anecdote about the development of his startup where each month he ‘buys’ users using advertising. A key part of the story is that he must
continually replace users because many people use his software only once and never return. I believe this tendency for users to try a product only once and decide if it is something they are interested in is important to consider when determining how minimal a MVP can be. If it is so minimal that users refuse to even try it this defeats the purpose of the MVP.
Chapter 6: Conclusion

While the desire to debate others on the internet clearly exists, there are a number of problems with current technologies that make this goal difficult to achieve. Online debate appears to be an enterprise that requires specialized technologies to be conducted effectively. This project developed and implemented one such technology.

An application was developed that attempted to recreate in software some of the format used in real life debates. These include the presentation of an argument with evidence, specific ways to refute arguments and evidence, and restriction of avenues for response to an argument.

Getting people to use the application proved to be difficult. It became clear that additional work on the user experience as well as many more articles to debate were required for users to get any value from the application. The low number of users also caused difficulty because of the chicken and egg problem.

Future Work

Many features never made it into Spirited, and the need for several more became apparent during development. An achievement system that rewarded user with titles for completing certain sets of actions in the application (vote for 10 responses, create an article that gets 100 Likes) was abandoned for lack of time. Other annotation types including graphs were left out because they were non-trivial to implement.

Some sort of traditional commenting systems was a common feature request, but finding a way to do this while maintaining the structure required for debate was tricky. Related to comments, a mechanism for quoting was another need that only became obvious after watching how users interacted with the application. Generally, people wanted to write a comment about a
specific part of the article, creating something that was not a full article, but not short enough to be considered an annotation.

Another feature that was proposed but did not make it for lack of time was some way to group articles by topic. After finding that users only wanted to interact with articles on their topics of interest it became clear this must be integral to Spirited so that users can find debates they are interested in.

A fundamental limitation of this application is the requirement to post articles on the application itself. Users may want to debate authors on other websites, and authors may wish to add the option to debate to their own site. An embedded version of the Spirited sentence level annotation tools that website owners could add to their existing articles would provide a useful service and expand the reach of the Spirited application.

While many features have not yet been implemented, the core application provides a strong foundation to build on. The use of an API and the Ember frontend make an embedded version of the site relatively simple to implement. New annotation types can be built and integrated with this foundation, making new types of documents – like scientific papers with graphs - a possibility. Websites like Medium and Reddit have shown that with an active community and the right tools, users are more than willing to use private websites as publishing platforms.
References


https://www.technologyreview.com/s/534276/is-genius-smarter-than-past-attempts-to-annotate-the-web/

Stack Exchange. (n.d.). Retrieved August 5, 2016, from Wikipedia:

https://en.wikipedia.org/wiki/Stack_Exchange#Site_features

*The top 500 sites on the web.* (n.d.). Retrieved August 5, 2016, from Alexa:


Web annotation. (n.d.). Retrieved August 5, 2016, from Wikipedia:

https://en.wikipedia.org/wiki/Web_annotation