The Wikirate Project

Annual Development Review 1
D6.6.1

Ethan McCutchen ▸ Grass Commons ▸ 10/16/2014

Dissemination level	Public
Contractual date of delivery	Month 12 | 30 September 2014
Actual date of delivery	Month 13 | 20 October 2014
Work package	WP6 | Wikirate Architecture and Development
Deliverable number	D6.6.1 | Annual Development Review 1
Type	Report
Approval status	Approved
Version	1.0
Number of pages	21
File name	D6_6_1-20141016_V01_GrassCommons.doc

Abstract
This review of Year 1 technical progress surveys the current state of Wikirate.org modules, the Wagn framework, their technical support systems, and their API's.

SEVENTH FRAMEWORK PROGRAMME

Co-funded by the European Union

History

Version	Date	Reason	Revised by
1.0	2014-10-16	Full Draft	Ethan McCutchen

Author list

Organization	Name	Contact information
Grass Commons	Ethan McCutchen	ethan@grasscommons.org

Executive Summary

Year 1 involved major advances of the Wikirate codebase (T6.1), the Wagn platform (T6.2), the server technologies that support them (T6.3), and the mechanisms for connecting them all to the broader world (T6.4).

As part of Task 6.1 (*Implementation of Features of WP2 and integration of WP5*), all of Wikirate's current core content types (Companies, Topics, Analyses, Claims, Sources, and Users) have been made dramatically more powerful and approachable. The improvements include both extensive use of Wagn configuration (or "wagneering) and code customizations (or "mods"). These content types also benefit from many new patterned implementations, including tagging and voting. And we have now deployed a strategically important integration with CERTH in the form of suggested sources.

The many *Improvements to Wagn platform* (Task 6.2) can organized into three major systemic refactors: Wagn's re-architecting as a ruby gem, reconciling subsystems into the *everything is a card* principle, and comprehensive name tracking. Each of these three was driven by critical Wikirate needs, involved deep restructuring, and produced a broad array of benefits to Wikirate.org and other Wagn sites.

Our efforts on *Wikirate system administration* (Task 6.3) have had solid yields, including an appropriate server architecture, productive development environments, reliable deployment and backup tools, and quality testing environments. Our current system is now smoothly supporting both the community using Wikirate.org and the developer team behind it.

Finally, we have made great progress on our *Application Programming Interface (API) and Plug-ins* (Task 6.4). This year we clarified, harmonized, and fleshed out both our Mods API and our RESTful Web API. Each is already playing a central role in Wikirate.org, and together they are opening up the possibility for many more system integrations in the future.

Table of Contents

1 Introduction

This document provides an overview of the current state of affairs of the four WP6 tasks:

- Task 6.1 *Implementation of Features of WP2 and integration of WP5*
- Task 6.2 *Improvements to Wagn platform*
- Task 6.3 *Wikirate system administration*
- Task 6.4 *Application Programming Interface (API) and Plug-ins*

Here we focus on *product* not *process*. This review is *not* intended as a complete record of the efforts of the development team – that job is delegated to the technical review component of annual reporting.

Instead we here outline the technical distance traveled in the first year of the grant. Therefore in the case of functionality that has undergone multiple rounds of improvements, this document will describe only the final state, not the incremental changes explored en route.

While the review does piece apart the deep integration of Wikirate.org and the Wagn framework, it does *not* make comparable efforts to piece apart the deep collaborations of the Grass Commons and Cambridge development teams.

In general, the University of Cambridge focuses most heavily on Task 6.1 while Grass Commons focuses more heavily on the other three tasks. But both partners have contributed to most of the functionality that we've built, and pair coding and pair design sessions with representatives of both consortium partners present have become increasingly routine.

2 Task 6.1: Implementation of Features of WP2 and integration of WP5

Wikirate.org is an instance of the Wagn wiki framework. This chapter investigates all the *technical* ways in which the Wikirate instance differs a standard Wagn instance. These enhancements fall into two camps:

1. Code customization through the Wagn Mod API (see section 5.2)
2. Data configuration via the Wikirate.org site itself

As we survey the Wikirate features implemented this year, we will regularly refer to this distinction as "mod code" vs. "wagneering" respectively.

It should be noted that very many changes (like, for example, the JavaScript API) began as Wikirate customizations and have then later been worked into the Wagn core. Because the present review focuses on product not process, such changes will not be mentioned here. (Although Wagn core changes will be covered in Chapter 3).

The changes that are currently managed in the Wikirate mod are those that are judged not (yet) sufficiently generalized in concept or implementation to be added to the Wagn core.

The code customizations for Wikirate.org are publicly stored, managed, and discussed at **https://github.com/wagn/wikirate**.

2.1 WP2 Features

At present, Wikirate.org features six main content types: Companies, Topics, Analyses, Claims, Sources, and Users. Here we will first explore what has been implemented for each of these *cardtypes,* both in terms of code customization and data configuration. Then we will explore some enhanced functionality shared across these types.

2.1.1 Companies and Topics

As Wikirate's most central subject matter, Company and Topic pages are among our most important. At present their core materials are lists of Analyses. An Analysis is an exploration of the connection between a Company and a Topic, and the core aim of Company and Topic pages is to entice users to click to an Analysis page (See the *Adder's Churn* concept described in D2.2.2).

These pages are currently accomplished almost entirely through wagneering; very little code customization is involved, other than a custom view of card editors. The Analysis lists ("with Articles", "needing Articles", etc.) are achieved with WQL (the Wagn Query Language, which is the mechanism for creating searches on Wagn). Because WQL queries can refer to each other, we are able to build fairly complex out of simple parts. There is also minimal custom JavaScript for making the "about" sections expandable and managing Analysis tabs.

2.1.2 Analyses

Considerably more code customization is required for Analyses, which are formed by connecting a Company to a Topic. For example, if "Apple" is a Company, and "Climate Change" is a Topic, then "Apple+Climate Change" will automatically be treated as an Analysis, even if no such card is in the database. This "type plus type" *set pattern* that is used to set the Analysis type is not currently part of the Wagn core; it's defined in the mod code.

The primary content of an Analysis page is an *Article*, which is in many ways a standard rich text card, but the key Wikirate enhancement was to support for Articles citing Claims (rather than citing sources directly). For this, we created a custom cite view and the capacity to handle citation numbers in the list of cited Claims beneath the Article.

Analyses also support a special "citation-ready" view when linked to from a Claim. This view is intended to teach users how citation works and lower the social barrier to editing Articles.

In addition to directly editing Articles, which may be seen by many users as a difficult first step, Analysis pages also invite Users to explore potential source material. The right sidebar of an Analysis is predominantly focused on this functionality, including both a simple WQL query for existing relevant sources and a more involved integration with CERTH for suggested sources (see section 2.2 *WP5 Integration*).

2.1.3 Claims

Claims have quite a few customized views and behaviors, including:

- Server- and client-side enforcement of a 100 character limit
- Sample citation views for aided article edits
- Clipboard views for automatically adding claim citations to a user's clipboard for quick pasting
- Source validation (all claims must cite at least one source)
- Special source editor / listing, which automatically adds sources to Wikirate.org when a url is pasted.
- Cached counts for optimizing Company and Topic browsing pages
- "Next step" tips

- Voting (in development; see below)

We've also designed a new view of claims that will appear throughout the site where Claims are listed. It includes a citation count, and clicking on it will invite citation in all Articles suggested by the claim's +company and +topic tags.

2.1.4 Sources

Conceptual design is underway for dramatically extending the kinds of sources that Wikirate will handle (see D2.2.2 *Development Priorities and related user/system requirements and architecture*). At present, sources are limited to webpages, which is why the Cardtype for sources is currently *Page*.

Sources require significant code customization on Wikirate.org, including:

- The special "source preview" page, which allows users to perform core source-related actions (adding claims and tagging) while viewing the source in a full-width iframe.
- Automated parsing of Pages and extraction of title, image, and description.
- Automatic generation of Website card and association therewith. Eg, if I add a Page for http://mynewssite.com/myarticle, then Wikirate will also create a Website card for mynewssite.com and associate the Page with that Website
- Exact duplicate prevention

As described, each Page is automatically associated with a *Website* card. Website cards have been wagneered to include a list of associated pages, a description, and an *owner*, which can be used to reconnect Sources to the Companies behind the websites. While Website pages are currently de-emphasized and minimalist, they are part of a larger long-term plan for Sources and therefore merit mention here.

2.1.5 Users

There is currently no mod code for User cards on Wikirate.org; though we've dramatically enhanced these cards this year, all these improvements to date have been made in the Wagn core.

That said, the User profile itself has received (and will continue to receive) design attention and is currently serving as the epicenter of a user's activity and reputation. The tabbed activity component involves some special JavaScript, but it is otherwise implemented purely through wagneering.

2.1.6 Tags

Tags on Wikirate.org are implemented as simple *Pointers* (a built-in Wagn cardtype). However their behavior has been extended in two key ways:

- We have created a Tag cardtype for displaying items linked to by +tag cards
- Tag cards are automatically created when items are added to these cards

2.1.7 Voting

We have not yet deployed our new voting functionality, but it is under development and only about a week away from deployment-readiness.

This implementation involves using cards both for canonical vote representations (associated with the voting user) and for cached vote totals (associated with the subject voted upon).

2.1.8 Home Page

At the time of writing, we have not yet implemented the newly designed Home Page, so we won’t go into great detail here, but it’s merits mentioning that:

- The old home page is entirely wagneered
- The coming home page has received broader and deeper consortium input than any prior undertaking and benefited the shared wisdom of our entire team
- The new home page introduces new visualizations for Companies, Topics, Analyses, Claims, and Sources, and *all of these* visualizations will be reused on the deeper browse pages for those types.

2.1.9 Styling

While styling (CSS) plays a central role in all of our implementation, it receives relatively little attention here, because describing what we have done in this realm cannot compete in clarity with simply browsing the website. *However*, it’s worth bringing attention to a few particular styling decisions:

- We have foregone a site-wide sidebar in order to preserve that real estate for Wikirate’s many content-intensive activities. This need is currently most evident on Claim and Analysis pages but as Measurements are introduced to Company pages, the value of this decision will become even more evident.
- We are increasingly using color and other simple visual indicators to differentiate core content types and help users make key conceptual distinctions. This is currently notable in the orange (company) and blue (topic) tags seen throughout the site, but is made

more prominent in the new homepage designs, which much better differentiate Claims and Sources (our most often confused types)
- Our color palette was chosen after careful deliberation and was made with an effort to balance a sense of professionalism / impartiality on the one hand and welcoming/community on the other.
- Wikirate replaced Wagn's default "gear" icon with a pencil-on-paper icon in order to shift the emphasis from configuration to editing.

2.1.10 Other

A few other Wikirate customizations:

- Comparison pages, generated via mod_code, allow users to compare two companies' performance on a given topic. We expect this kind of comparison to play a much deeper strategic role in the future when we have implemented measurements, which allow for richer direct comparison.
- The search result page has been wagneered to provide type-specific results
- The top menu, which appears throughout the site, involves some custom JavaScript
- The Community Dashboard page relies on several wagneered searches but no custom code

2.2 WP5 Integration

Analysis pages on Wikirate.org prominently feature a list of potential sources "from the web". More precisely, these source suggestions are "from CERTH's servers via webservice". This is the centerpiece of our current WP5 integration, which has served a key strategic goal of completing the circle of the "Adder's Churn" (see D2.2.2) by connecting Analyses to new Sources.

When a user loads an Analysis page in a browser, it initializes a request to CERTH's servers, sending them JSON including the relevant Company, Topic, and (for prevention of duplicate feedback) user_id. In return CERTH sends basic details of a potential source (url, title, description, image, etc). JavaScript on Wikirate then converts these into clickable source listings.

If a user chooses to click on a listing, he/she is taken to a variant of the source preview page, which has a small Wikirate banner at the top and a large full-width iframe of the source page below. The banner provides the user with a simple choice: Is this source *relevant or irrelevant* to the company and topic at hand.

If irrelevant, the user can indicate whether the article is irrelevant to the company, the topic, or both. Wikirate then passes this feedback to CERTH (again with a JSON-based web request). CERTH can then use this feedback to improve their suggestions.

If relevant, several things happen:

1. Positive feedback is sent to CERTH
2. The source is added as a Page on Wikirate.org and tagged with the relevant Company and Topic
3. The top banner is updated to the interface for an existing claim
4. The user is invited to add a Claim based on this source

3 Task 6.2: Improvements to Wagn platform

3.1 Gem Distribution

A gem is a standard ruby library. We now release Wagn as a gem and distribute it through the leading gem host service, RubyGems, at **http://rubygems.org/gems/wagn**.

Prior to this grant, Wagn was distributed as a Ruby-on-Rails application. Its main distribution mechanism was GitHub, which meant that installing Wagn required using advanced developer tools. While the code was already quite modular, this distribution mechanism also blurred the lines between platform and instance and made it challenging to host multiple Wagn instances on a single site.

By contrast, Wagn can now be deployed by a few simple command lines by anyone with a standard ruby setup. Platform code is cleanly separated into a gem and can be easily shared by multiple instances. While Wagn still makes use of many core Ruby-on-Rails libraries, these dependencies are all managed with standard gem dependency tracking mechanisms and never need to be considered by site administrators.

3.2 Everything is a Card

Wagn's foremost organizing principle is "everything is a card". The value of this *deep atomism* is explored in greater depth in Interim Technical Evaluation I (D7.7.4).

Since the start of the grant we have deepened our exploration of this principle by reconciling three different subsystems previously tracked using non-card structures: accounts, JavaScript, and emails. All are now handled in cards. Each of these changes was driven by specific Wikirate needs but was implemented in a generalized way in order to insure its value was assured both to other Wagn users and to future Wikirate use cases.

3.2.1 Card-based Accounts

Traditionally, Wagn instances have relied on a separate users table to manage user accounts. Having obviated separate tables for permissions, cardtypes, and roles prior to the grant's start, the users table was the last remaining database table outside of the cards system. Due to the additional measures needed to make this data secure, the users table was saved for last, but the conversion was carried off successfully, and the upgrade includes support for automatically migrating all older Wagns into the current system.

The driving Wikirate need for the effort was upgrading the signup process to use token-based authentication. However there are many other benefits:

- All account changes, such as email updates, now have a history and can be monitored by administrators using standard card history views.
- Account data, like emails, are exposed to the Wagn Query Language, making possible many new kinds of account reports
- Account cards are now fully exposed to both the mods API and the REST API, making all kinds of new modifications possible, including automated role assignment, external signins via Google, Twitter, Facebook, etc, and all sorts of service-based account integrations.
- Account card permissions are as flexible as all card permissions, meaning a host of new administrative configurations are now possible.

3.2.2 Card-based JavaScript

Prior to Q3, one major challenge with developing Wikirate.org on the Wagn platform was the tension between the increasing need for custom dynamic (JavaScript-based) interactions with Wagn's lack of a clear framework for introducing custom JavaScript. The platform's primary javascript was hard-coded, and adding additional code was a challenge.

This challenge was resolved by creating a new card-based JavaScript pipeline in Wagn. **A user can now paste JavaScript into a card and automatically have it concatenated, and compressed into a single fingerprinted JavaScript file**. The fingerprinting optimizes site performance, because it allows us to instruct browsers to cache the file permanently; any updates will trigger a new fingerprinting.

We additionally added support for CoffeeScript, which is automatically compiled and then treated like any other JavaScript card. The new system now gives site administrators complete control over JavaScript and a simple mechanism for adding or editing code.

3.2.2 Card-based Email

Prior to the grant's start, almost all Wagn emails were hard coded with a few variable substitutions for site names, user names, etc. Though there was a little-used subsystem we called "flexmail" that could be used to configure custom emails to be created when certain cards were created, it had little supporting interface and was implemented independently of all other emails. Customization ranged from very difficult to impossible.

In our current code (deployed to Wikirate in late September and publicly available on GitHub but not yet formally released as a gem), **templates for system emails are now handled in cards and can easily be customized.** The email content itself is now rendered as a card view and is thus fully exposed to the mods API.

3.3 Comprehensive Change Tracking

Wagn has long followed the standard wiki pattern of tracking content revisions. However, **we have traditionally not tracked the two other core data attributes of cards: names and types. Now we do**. Because both name and type can determine Set membership, they can dramatically impact a card's appearance and behavior, so this data is quite significant. Moreover, this information can be very important for understanding systemic changes; Wagn supports automatically updating all references to a card when it is renamed, and to interpret these patterns well, we need to track more than just the content changes.

This tracking need is even more obvious from a Wikirate perspective, because names and types contain very important data. The core statement of a Claim name, for example, is represented as a card name. This means **renaming a Claim card can change the very nature of a Claim, so these data are clearly vital**.

3.3.1 Data Representation

Previously, changes were stored in a single table (*card_revisions*). This has now been expanded to **three tables to provide a much fuller and more coherent history**:

- *card_changes* – A *change* tracks which fields (name, type, content) where changed to what value. Each change is associated with an *action.*
- *card_actions* – An *action* represents a group of changes to a specific card. Each action is associated with a card and an *act*.
- *card_acts* – An act represents a group of actions to one or more cards initiated by a single user behavior. An act is associated with an *actor* (user) and also stores metadata la timestamp and IP address.

Additionally, the current card *content* is now stored directly in the cards table, just like *name* and *type*. Previously the cards table contained only a *current_revision_id* reference to the *card_revisions* table. The benefit of this new structure is that **the cards table is now a complete and independent representation of the current state of the deck**, while the other tables are necessary only to reconstruct its history.

3.3.2 History Views / Rollbacks

Wagn's **history viewing got a much-needed upgrade** in support of the new change tracking. Because these views previously showed only the history of a single card, and a structured card (like a Claim or Company) generally involves many different cards, it was very difficult to get an integrated view of related changes.

The new history views introduce:

- rows of expandable change views
- name and type changes
- changes to nested cards
- optimized diffs with improved visualization
- changes grouped by *act*
- multi-change rollbacks

While rollbacks are currently limited to changes that occur in a single act, it's worth mentioning that the new data tracking also opens up the possibility of implementing much more extensive systemic rollback actions. Since all changes are tracked, and standard deletions are handled as marking cards as "in the trash" in the database, we now have the data (but not yet the functionality) for large-scale date-based rollbacks.

3.3.3 Following / Notifications

Wagn's "following" behavior is of central strategic significance to Wikirate, and the need to enhance this functionality was the strongest impetus for moving up the timetable for implementation of the new change tracking.

The essence of the functionality is that users choose to "follow" a given card or set of cards and then receive notifications when changes to those cards are made. These notifications are critical for enticing users to return to Wikirate.org (and other Wagn sites) and deepen their engagement. Not surprisingly, change notifications are deeply intertwined with change representation.

While we have many more notification improvements in progress (as discussed in D2.2.2 *Development Priorities and related user/system requirements and architecture*), the current code includes many significant improvements, including:

- notification emails now contain the actual changes themselves (including name, type, and content changes) rather than just the news that a changes has been made
- permissions are much improved to allow both great flexibility and security
- a "follow" action will now use our standard RESTful Web API (see chapter 5)

- lists of followed cards are now represented as standard *Pointer* cards, which already feature a refined interface and are easy to edit.
- developers can override functionality via the mods API

The final point is particularly valuable for Wikirate, because it provides an implementation path for all the remaining specifications for this subsystem.

3.3.4 Research Benefits

Finally, the new change tracking provides much more data for researching user behavior on Wikirate.org. The “act” groupings are much more indicative of user behavior than the prior “revision” groupings were, and

In combination with the new structured request log, the research team can now generate a *much* more complete picture of site activity.

4 Task 6.3: Wikirate system administration

4.1 Server Architecture

Wikirate.org is currently provided by a single Hetzner EX40 dedicated root server as follows:

- high-performance Intel® Core i7-4770 Quad-Core processor
- 32 GB DDR3 RAM.
- two 2 TB SATA 6 Gb/s

The same server is used for our internal documentation site (also using Wagn) and our public project site (wikirate.eu).

The development server is identical. It is used for testing and staging the production site and for continuous integration testing (in progress).

Both servers are running:

- Ubuntu 12.04.4 LTS
- Ruby 1.9.3p484
- Mysql 5.5.35
- Apache 2.2.22
- Passenger 4.0.29

As discussed in D7.7.4 *Interim Technical Evaluation 1*, this architecture is currently appropriate, but we plan to upgrade to a multi-server architecture when our traffic and data demands increase.

4.2 Development Environments

Developers at Grass Commons and Cambridge develop primarily on personal laptops and then coordinate code changes with the rest of the development team through GitHub pull requests.

Each development team members also maintains an open HipChat window for Wikirate, which integrates:

- GitHub updates and pull request notifications
- Updates to PivotalTracker stories

- Server Error notifications from Airbrake
- Instant messages from team members

All ruby developers are using TextMate development environments and frequently share tips and tricks with each other, both during pair coding sessions (usually remote) and whole team "show and tell" sessions.

4.2 Deployment and Backup Tools

We are currently using Capistrano to manage all our deployments to both the development and the production server. With Capistrano we can run full updates with a single command that manages:

- Wikirate code updates and data migrations
- Wagn code updates and data migrations
- Updates to dependent libraries
- Clearing tmp directories
- Restarting Passenger

We have also created Capistrano commands to:

- generate backups
- refresh the development copy of Wikirate.org from a backup of the production site
- copy and install a backup to a local development environment

Additional daily backups are generated on the production server and copied to the development server using cron jobs.

While authentication is naturally private, all Capistrano commands are made public in Wikirate's github repository for reuse/modification in comparable environments.

4.4 Testing Environments

Both Wagn and Wikirate are tested using Rspec, and Wagn is additionally tested via Cucumber.

Wagn's test coverage is currently 90%. While our target is 99%, the vast majority of the remaining uncovered code is either unused by Wikirate (as in the case of utilities and edge-case error handling) or is old and not recently edited. Both should be covered regardless, but urgency is low.

By contrast, Wikirate's test coverage is at 70%, and it is a high priority to elevate this number rapidly. It should be noted that support for instance testing was only introduced this summer, and Wikirate has moved from 0% to 70% quite rapidly. Because of the significant use of JavaScript, we also plan to introduce Cucumber testing for Wikirate before the end of the year, and we will also investigate JavaScript unit testing options.

We're currently working on continuous integration testing and expect to have all our current tests running automatically with every code update by the end of October.

For cross-browser testing, we have been using BrowserStack, which provides convenient screenshots of pages renderings in many browsers / environments.

5 Task 6.4: Application Programming Interface (API) and Plug-ins

Wagn currently features two primary API's: the Mods API and the RESTful Web API. The former is the means by which Wagn's functionality can be altered or extended, and the latter is the means of interacting with Wagn data over the web. Both have made dramatic strides this year.

5.1 MoFoS Architecture

In the *Interim Technical Evaluation I* (D7.7.4), we described Wagn's innovative *MoFoS* architectural pattern, which has received external inquiries for academic investigation. The improved articulation of this new pattern has given great guidance to the refinement of both API's. While the pattern's merits are presented briefly there, here we will focus on a brief technological overview here to help contextualize the API advances. The basic structures are:

- one Model (*Card*, in Wagn's case)
- many Formats (html, txt, json, etc). All format objects are associated with an instance of the model.
- both Models and Formats can be extended by Sets

5.2 Mods API

Wagn's mods API is the primary means by which functionality not distributed in default Wagn instances is created or customized for Wikirate.org.

Because *everything is a card*, essentially every Wagn mod involves changing how cards look or behave. This principle makes the API *simple*; what makes it *powerful* is its organization into Sets. Wagn's *Atomic Sets* are described in some detail in *Interim Technical Evaluation I* (D7.7.4), but the basic idea is that a Set is a configurable group of cards. A Set can be *all cards*, *a single card*, or something in between. The basic file in a mod is a *set module*, which, in essence, allows developers to customize how cards in a given Set are transformed (via Model methods) or displayed (via Format methods).

In the past year, the Mods API has matured rapidly and harmonized in exciting ways. Key improvements include:

- Everything that happens when creating, updating, or deleting a card was translated into the *events* pattern, which cleanly exposes all transformations to the set modules API (note: one exception, attachment saving, remains unconverted. This will soon be addressed).
- Format instances now follow the set module loading pattern of Card instances. This exposes *all* Format methods to the set module API, where previously only view methods could be assigned to Sets. This synthesis makes the set_module both more comprehensive and more comprehensible.
- Much richer handling of transactions involving multiple cards. The API now supports finer control over early exits of a transaction as a either a success or failure.
- Everything mentioned in section 3.2 above (*Everything is a Card*) was previously unexposed to the mods API but is now fully available for programmer manipulation.

5.3 RESTful Web API

A year ago, Wagn had four controllers and dozens of controller actions. Now, after a great deal of work, we have managed to consolidate everything into one controller and just four actions on cards: create, read, update, and delete. This makes our RESTful Web API extremely simple; we merely map these four actions onto HTTP *post, get, put,* and *delete* respectively.

Even account actions are completely reconciled to this API. To sign up, for example, you *create* a “Sign up” card. When the account is verified, it can *updated* with a type change and converted into a User card. Of course, these transactions generally involve many cards; but all these transformations will be initiated by an *act* upon a single card.

The Mods API outlined above allows developers to manipulate *events* can be used to add all sorts of additional meaning to these actions, but these customizations at most require custom parameters; the core RESTful API does not need to change in order for site administrators to make use of mod developers’ creations.

Here, then, is the entire API (including GET variants of transactional requests):

HTTP Request	Description
GET /	index
GET /name	read if exists, "new" form if not
GET /name?view=edit	read in specified view if exists, "new" form if not
POST / **POST /name** **GET /create/name**	create new card
PUT / **PUT /name** **GET /update/name**	update card
DELETE / **DELETE /name** **GET /delete/name**	delete card (move to trash)

6 Looking Forward

Despite early hiring difficulties, the development team had a highly productive year. Our efforts on Wikirate and Wagn have cohered into compelling array of functionality around the quantitative (or “Wiki”) aspects of Wikirate. As detailed in D2.2.2 *Development Priorities and related user/system requirements and architecture*), the qualitative (or “Rate”) aspects of Wikirate lie ahead.

This quantitative work will be the most significant new functionality of Task 6.1 (*Implementation of Features of WP2 and integration of* WP5) in months ahead, though we expect to spend a lot of time on research-driven refinements of existing features.

Similarly, Task 6.2 (*Improvements to Wagn platform*) will face a grand challenge in multilingual support while also tackling steady research-driven refinements towards Wagn’s major rebranding release, Decko 1.0.

Our testing environments and coverage will be much enhanced in coming months, but we don’t expect other aspects of Task 6.3 (*Wikirate system administration*) to receive great attention unless traffic begins increasing at rates far above those projected.

Finally, Task 6.4 (*Application Programming Interface (API) and Plug-ins*), will continue to respond to the demands of T6.1.

Given our strong momentum and established infrastructure, we anticipate an even more productive second year.