A KNOWLEDGE DATABASE

Lukas Nabergall

August 5, 2016

1 Principles

This knowledge database application is designed with the following principles in mind:

1. Should contain content which rapidly converges in time towards validated knowledge.
2. Should be relevant and useful to advanced research, development, and academia.
3. Should optimize searchability and quick content recognition and absorption.
4. Should be based upon open content submission, editing, and validation.

2 Users

The knowledge database will be completely open to all for viewing and contributing, but users will have the option to create an account and, for transparency, will be encouraged to provide their full name. Logged in users will be able to create new content, have the option to be denoted as authors of pieces of content that they have created or edited, and will be able to validate new edits on all pieces of content of which they are authors. Users who are not logged into an account, or who have opted to remain anonymous as editors of a piece of content, will not have the ability to submit new pieces of content or validate new edits.

3 Content

The knowledge database will be populated, primarily via open user submission, with brief entries explaining or documenting some single idea related to or relevant to advanced research and development fields, including mathematics, physics, chemistry, biology, statistics, computer science, programming, advanced technology and engineering, and other areas. Although the emphasis will likely at first be on content relevant to research and development in areas related to the hard sciences, the scope of the database could expand to include the social sciences, the humanities, the arts, and other disciplines, as well as more elementary content which is typically encountered at the secondary school and undergraduate levels, as appropriate.

All pieces of content will be of a single form containing the following items:

- Name — the primary name of the idea to be displayed in searches and lists, may contain LaTeX and other special formatting; e.g. *abc* conjecture, the W-trick, Equipartition theorem, Finite Field, etc.

- Alternative Names [Aliases] (optional) — alternate names (aliases) of the idea which may also be used for identification and search.

- Content Type — the type of the idea, e.g. idea, theorem, conjecture, lemma, equation, formula, inequality, code, algorithm, visualization, technique, proof, etc.

- Text — the actual explication/description/explanation of the idea, ~50 characters minimum[1], ~2000 characters maximum,[2] can contain LaTeX, TeX, code, rich text, HTML, and other special formatting;[3] e.g. "Given double occurrence words w_1, w_2, define the word distance δ by...".

- Images (optional) — related images, including all common file types (JPEG, PNG, TIFF, BMP, GIF, etc.).

- Keywords / Subject Designations — keywords and subject designations contained in or related to the idea, approximately 2 or 3 required; e.g. for a piece of content on finite fields, could include field, group, field theory, abstract algebra, associativity, operation, etc.

- Dependencies (optional) — references contained in the text to other pieces of content via their names or aliases, can be manually specified via special formatting and automatically generated[4] by matching words in the text with other pieces of content in the database, will be given hyperlinks on display; e.g. "Given double occurrence words w_1, w_2, define the word distance δ by...".

- Citations (optional) — Including both implicitly and explicitly (in-text) cited works, input with some standard academic citation format[5] (although including less information, e.g. only a url, is acceptable but discouraged), in-text citations are referenced in the text via special formatting.[6]

To prevent mass fraudulent content submissions, an authentication scheme (e.g. CAPTCHA[7]) will be used to verify that a human is indeed making the submission.

1. To be finalized.
2. To be finalized.
3. The input field will be a rich text editor containing buttons which automate much of the formatting process.
4. Possibly with user validation.
5. To be specified.
6. To be specified.
7. In the sequel, it is assumed that CAPTCHA will be used for authentication in the knowledge database.

3.1 Editing and Validation

All aspects of all pieces of content will be fully editable by any user of the knowledge database. To combat malicious, inaccurate, or inappropriate content submissions and edits, an essentially democratic validation scheme will be implemented on most aspects of the content.

- Name — All modifications will be validated using the system specified below.[8]
- Aliases — Validated modifications, open (not validated) additions with CAPTCHA authorization.
- Content Type — Validated modifications.
- Text — Validation modifications, including insertions and deletions.
- Images — Validated deletions and additions.
- Keywords / Subject Designations — Validated modifications, open additions with CAPTCHA authorization.
- Dependencies — Validation of automatically (system) and manually (user) specified dependencies.
- Citations — Validated modifications, open additions with CAPTCHA authorization.

In order to facilitate rapid growth and large throughput, content submissions will not be validated.[9] Suppose a user modifies some aspect of a piece of content which requires validation. The user's modification will not be visible until it is validated, and accepted, by the authors of the piece of content.

The validation process will proceed as follows. After a user submits an edit, all authors of the corresponding piece of content will be signalled to the existence of the modification and will have the option of either voting for or against accepting the modification.[10] An author may change their vote at any time prior to the closing of the vote. The edit is accepted and the voting is closed if any of the following conditions are satisfied:[11]

1. A majority of the authors[12] have voted for acceptance of the edit, or
2. if there are $N \in \mathbb{N}$ authors, at least $\lceil N/2 \rceil$ authors have voted and at least $3/4$ of the votes are for acceptance of the edit,[13] or

8. Which will henceforth be assumed when discussing validation, unless otherwise specified.
9. Except via later edits.
10. Or abstaining from (/ignoring) the vote.
11. Note that these conditions are evaluated with respect to the time at which they are satisfied, not with respect to the time of the edit. In particular, users who become authors after the validation process begins for some modification may still participate.
12. Note that each author can vote at most once.
13. Note that these details are not finalized, although currently $N/2$ and $3/4$ are chosen because at least $3/4$ of the remaining $N/2$ authors would have to vote against acceptance of the edit for there to be less than a majority for acceptance. This is in general very unlikely, assuming benevolent authors.

3. at 5 days past modification, at least 2 votes have been submitted and at least 2/3 of the votes are for acceptance of the edit, or

4. at 10 days past modification, at least 1 vote has been submitted and a majority of the votes are for acceptance of the edit, or

5. there are no authors,[14] or

6. no votes are submitted within 10 days.

Otherwise, after 10 days, the edit is rejected and the voting process is closed. Upon acceptance of a logged in user's edit, the user[15] is denoted an author of the piece of content and given the ability to participate in the edit validation process of that piece of content. In this way, there is an essentially *viral authorship* system which ensures that those who validate edits have contributed useful, accurate, and appropriate edits in the past and therefore can be assumed to be relatively benevolent and have some knowledge of the content.

The only other caveat to the editing process is that we will assign a probability of ~1/7[16] to the possibility that CAPTCHA authorization will be required in order for any given user to submit an edit.[17] This will reduce the likelihood of a malicious user successfully using a series of computers to perform mass fraudulent modification of content in the database, while also minimizing the fixed costs of editing content for benevolent human users.

The above procedure covers all cases except when a user has submitted a piece of content that is entirely inappropriate or unsuitable for inclusion in the knowledge database or a group of authors are maliciously or inappropriately rigging the validation process in order to advance an agenda which runs counter to the principles of the knowledge database. In this case, any user will have the option to report a piece of content or a group of authors to the administrators[18] of the database, one or more of whom will investigate and make an appropriate action.

4 Application Map

The following is a list of each of the (functional) pages[19] in the knowledge database and, for each page, a list of linking pages, that is, the pages that a user can navigate to from that page. Note that these pages are only defined functionally and may not correspond to separate HTML pages or any page-like interface elements. Furthermore, linked pages may not correspond to a unique button (i.e. they may be dependent on previous navigation history). This generates a corresponding "application graph" or map, although it is too intricate to be effectively visualized. Each page (if applicable) contains a list of contained content.

14. Although it will likely be very rare, this could occur if all the authors delete their accounts.
15. If they were logged in when they submitted the edit and are not already an author.
16. To be finalized.
17. Where we assume that a user is defined by a unique IP address (or some similar metrics).
18. Likely individuals from among the organization which will maintain and develop the knowledge database.
19. Or areas/features.

1. Home — 2*, 3, 4, 6, 7, 12
2. Search Results — 3, 4, 5*, 6, 7, 12, 1

   10 content pieces matching the search criteria arranged in descending order.

   For each content piece: Best matching name, content type, snippet(s) of text with matches highlighted, and all keywords.
3. Sign-Up — 1*, 5
4. Login — 1*, 5
5. Content Page — 8, 6, 4, 3, 2, 1, 7, 9, 11, 12, 13, 14

   Name, alternate names, content type, text, citation, keywords, author user names, links to edits page and, if applicable, content author pages.
6. User Page — 1, 2, 7, 11

   Four separate "tabbed" subpages: recent activity, authored content pieces, edit history, and settings.

   Recent activity: All activity on authored edits and content pieces, including acceptance/rejection of authored edits, acceptance/rejection of edits of authored content pieces, submission of validating edits, and admin report notifications. If there is a vote, a notification will urge the user to vote and link to the authored content pieces tab or directly to the edit depending on whether there is a single or multiple validating edits.

   Authored content pieces: Name, content type, last modified date, notification icon indicating whether a validating edit exists or not and, if so, a "vote" button.

   Edit history: List of authored validating/accepted/rejected edits with the content piece name, content part, submission timestamp, acceptance/rejection timestamp (if applicable), and a quantitative summary of net modifications.[20]
7. Content Submission Page — 1, 5*, 4
8. Content Editing Page — 5*, 1, 2, 3, 4, 6, 9, 12
9. Report Content Page — 5*
10. Report Authors Page — 11*
11. Content Authors Page[21] — 5*, 6, 1, 2, 9, 10, 12, 14

20. e.g. number of characters/words added/removed, total formatting modifications made, and images added/removed.
21. Or content validation page.

A list of author user names, a list of (up to 10) validating proposed edits and corresponding current vote results,[22], and a list of (up to 10) recently accepted/rejected edits with corresponding vote results.

For each validating edit: a "vote" button or "vote submitted" icon will be displayed depending on whether the user has voted or not.

12. Admin Account Page — 5*, 13, 1, 2

13. Admin Action Page — 5, 12

14. Edits Page — 5*, 11

    Two lists:[23] the first contains currently validating proposed edits and the second contains accepted/rejected edits.

    For each validating edit: content part, creation timestamp, user name or IP address, a quantitative summary of net modifications,[24] and, if applicable, the edit rationale.

    For each accepted/rejected edit: content part, acceptance/rejection timestamp, user name or IP address, a quantitative summary of net modifications,[25] and, if applicable, the edit rationale.

An asterisk indicates that this is the primary page to which a user will next navigate.

5 Use Cases

Users

- A user must register with a username composed of at least two words separated by a space (with each word containing at least 2 characters),[26] a password containing at least 5 characters, and a unique valid email address.
- A user logs into their account by entering their email and password[27] and are then redirected to their account page.[28]

22. Still need to decide the level of information released prior to closure of the vote.

23. Paginated with 20 edits displayed per page.

24. e.g. number of characters/words added/removed, total formatting modifications made, and images added/removed.

25. See previous footnote.

26. That is, with their full name. These will be stored with underscores replacing the spaces.

27. Since their username and password might not be unique; i.e. the username is functionally only a display name.

28. This behavior could be more customized—a user could be redirected to their account page only if there is a pending vote on a piece of content they have authored and, otherwise, they are redirected to the home page.

- Upon registration, a user is sent an email containing a unique[29] link which they must follow in order to confirm their email address. Confirmation must be completed in order to submit a piece of content or be denoted an author after having an edit accepted.[30]
- A user must be logged in to submit a piece of content or be denoted an author after having made an accepted edit; if they attempt to submit a piece of content anonymously, they will be directed to login or sign up.
- When an edit is made to a piece of content which a user has authored, the user can see the existence of the edit, and be redirected to the author page of that piece of content to inspect and vote on the edit, from their account page.

Content Retrieval

- Each piece of content will have its own unique url which a user can follow to view content.
- A user can search for content via a search bar located on all pages of the application, although the home page will be the primary entry site. Ten entries will be displayed per page in the search results.

Content Submission

- After submitting a piece of content, the user will be directed to a preview page to confirm the content of their submission. Upon final submission, the user will be directed to the page of the posted piece of content.
- Each user will be limited to submitting at most one piece of content every minute.

Content Editing

- Each user will be limited to at most one edit every 10 seconds.
- Upon editing, a user will have the option of including an explanation of the rationale behind the edit which will be displayed to all authors when voting.
- After submission of an edit, all users can see the existence of that pending edit on the edits page.

Content Validation

- When an edit is made to a piece of content which a user has authored, the user is sent an email notifying them of the existence of the edit and urging them to vote. Until the voting ends or they have voted, such an email is sent to the user every 4 days.
- When an edit is accepted or rejected, a log of it is displayed on the edits page of the associated piece of content. From the edits page, users can view the changes made or proposed by all past edits.

29. And unguessable.

30. Primarily because an email is sent each time an edit is made to a piece of content the user has authored.

- When an edit is accepted, the piece of content is updated to display the edit.
- While an edit is being validated, that is, a vote is open, all authors can see how many authors have voted thus far. No other information will be available until after the vote is complete, when a complete summary of the results of the vote are displayed in the authors page (including the votes of all authors).

Admins and Violation Reports

- All users may submit a violation report against a piece of content, possibly calling for the deletion of the piece of content. They will only be required to give a nontrivially detailed report of the alleged violation.
- The authors of a piece of content may also submit a violation report against another author or group of authors. They will be required to list exactly which authors are committing the alleged violation and give a nontrivially detailed report of the alleged violation.
- When a violation report is submitted, an administrator is selected to investigate and resolve the situation. The admin may elect to take no action, send messages of guidance or warning against authors or other editing users, temporarily or permanently suspend user accounts, revoke authorship, or grant authorship[31] in the case[32] where there are no authors.[33][34] In the event that the reporter has requested the piece of content be deleted (e.g. if the piece of content was a duplicate of another piece of content or it contained inappropriate text), then an admin may elect to immediately delete the piece of content or give authors five[35] days notice of upcoming deletion.[36] An admin may also elect to take action against a user who is determined to be abusing the violation reporting feature.

6 Technology Stack

The following is a basic outline of the main technologies which will likely be used to implement the knowledge database. The emphasis at first will be on using technologies that are simple,

31. To registered users that have made some number of "good" edits to the piece of content.
32. Likely rare.
33. Or the current authors have become essentially unactive.
34. This may not be an exhaustive list of all possible admin actions.
35. Approximately.
36. Primarily if the piece of content was a duplicate or was only partially unsuitable for inclusion in the knowledge database. This gives the authors time to transport some of its information over to other pieces of content if appropriate. The contents of the piece of content will also be emailed to all authors.

proven, minimal, easy to work with, and ideally familiar.

Database — PostgreSQL, for general storage of content and metadata.
ORM — SQLAlchemy, for interfacing with the database.
Search — Elasticsearch, for searching and matching text content.
Processing — Celery and RabbitMQ, for asynchronous and scheduled task processing.
———————— Custom Python API Layer ————————

Framework — Pyramid, for routing requests and serving webpages.
———————— REST Framework API Layer ————————

Front-End — Bootstrap / Foundation / etc. (HTML/CSS).
Polymer / React / jQuery / etc.
Various JavaScript libraries (LaTeX, rich text editing, etc.).

7 Backend Design

The server-side backend of the knowledge database application will be composed of 3 layers, themselves containing 6 sublayers, each with varying levels of access to others. The system will essentially have the following architecture (lower on the list roughly implies closer to the client):

1. Database Interface Layer
   (a) Storage API — SQLAlchemy-based interface to the Postgres database.
   (b) Search API — ElasticSearch-based interface to the Postgres database.
2. Main Logic Layer
   (a) Content API — Submission, editing, validation, retrieval, etc. Interfaces with Storage API.
   (b) User API — Creation, authentication, sessions, permissions, etc. Interfaces with Storage API.
   (c) Admin API — Inherits from User API, creation, authentication, etc. Interfaces with Storage API.
3. RESTful Pyramid API — Pyramid-based, handles all communication with web clients. Interfaces with Search API, Content API, User API, and Admin API.

7.1 Storage

A Postgres database will be the main store of all persistent data in the knowledge database application. The database will contain 11 (non-linking[37]) tables: Text, Content_Piece,

37. There will be some additional linking tables used to implement many-to-many relationships.

Content_Type, Keyword, Name, Accepted_Edit, Citation, User, Rejected_Edit, Vote, and User_Report.

Text			
Attribute Name	Type	Indexed	Required
text_id	Integer	Yes, Primary Key	Yes
text	Text	No	Yes
timestamp	Timestamp	No	Yes
last_edited_timestamp	Timestamp	No	Yes
Foreign Relationships			
Type	Table	Attribute	
One-to-Many	Accepted_Edit	edit_id	
One-to-Many	Rejected_Edit	edit_id	

Content_Piece			
Attribute Name	Type	Indexed	Required
content_id	Integer	Yes, Primary Key	Yes
timestamp	Timestamp	No	Yes
last_edited_timestamp	Timestamp	No	Yes
deleted_timestamp	Timestamp	No	No
Foreign Relationships			
Type	Table	Attribute	
One-to-One	Text	text_id	
One-to-One	Name	name_id	
One-to-Many	Name	name_id	
One-to-Many	Accepted_Edit	name_id	
One-to-Many	Rejected_Edit	name_id	
Many-to-One	Content_Type	content_type_id	
Many-to-One	User	user_id	
Many-to-Many	Keyword	keyword_id	
Many-to-Many	Citation	citation_id	
Many-to-Many	User	user_id	

Content_Type			
Attribute Name	Type	Indexed	Required
content_type_id	Integer	Yes, Primary Key	Yes
content_type	Text	No	Yes
Type	Table	Attribute	
One-to-Many	Accepted_Edit	edit_id	
One-to-Many	Rejected_Edit	edit_id	

Keyword			
Attribute Name	Type	Indexed	Required
keyword_id	Integer	Yes, Primary Key	Yes
keyword	Text	Yes	Yes
timestamp	Timestamp	No	Yes
Foreign Relationships			
One-to-Many	Accepted_Edit	edit_id	
One-to-Many	Rejected_Edit	edit_id	

Name			
Attribute Name	Type	Indexed	Required
name_id	Integer	Yes, Primary Key	Yes
name	Text	No	Yes
name_type	Text	No	Yes
timestamp	Timestamp	No	Yes
last_edited_timestamp	Timestamp	No	Yes
Foreign Relationships			
Type	Table	Attribute	
One-to-Many	Accepted_Edit	edit_id	
One-to-Many	Rejected_Edit	edit_id	

Accepted_Edit			
Attribute Name	Type	Indexed	Required
edit_id	Integer	Yes, Primary Key	Yes
redis_edit_id	Integer	Yes	Yes
edit_text	Text	No	Yes
edit_rationale	Text	No	No
content_part	Text	No	Yes
author_type	Text	No	Yes
timestamp	Timestamp	No	Yes
acc_timestamp	Timestamp	No	Yes
Foreign Relationships			
Type	Table	Attribute	
Many-to-One	User	user_id	

Citation			
Attribute Name	Type	Indexed	Required
citation_id	Integer	Yes, Primary Key	Yes
citation_text	Text	No	Yes
timestamp	Timestamp	No	Yes
Foreign Relationships			
One-to-Many	Accepted_Edit	edit_id	
One-to-Many	Rejected_Edit	edit_id	

User			
Attribute Name	Type	Indexed	Required
user_id	Integer	Yes, Primary Key	Yes
user_type	Text	No	Yes
user_name	Text	Yes	Yes
email	Text	Yes	Yes
confirmed_timestamp	Timestamp	No	No
pass_hash	Text	Yes	Yes
pass_hash_type	Text	No	Yes
pass_salt	Text	Yes	Yes
remember_id	Text	Yes	No
remember_token_hash	Text	Yes	No
remember_hash_type	Text	No	No
timestamp	Timestamp	No	Yes
deleted_timestamp	Timestamp	No	No

Rejected_Edit			
Attribute Name	Type	Indexed	Required
edit_id	Integer	Yes, Primary Key	Yes
redis_edit_id	Integer	Yes	Yes
edit_text	Text	No	Yes
edit_rationale	Text	No	No
content_part	Text	No	Yes
author_type	Text	No	Yes
timestamp	Timestamp	No	Yes
rej_timestamp	Timestamp	No	Yes
Foreign Relationships			
Type	Table	Attribute	
Many-to-One	User	user_id	

Vote			
Attribute Name	Type	Indexed	Required
vote_id	Integer	Yes, Primary Key	Yes
vote	Text	No	Yes
content_part	Text	No	Yes
timestamp	Timestamp	No	Yes
close_timestamp	Timestamp	No	Yes
Foreign Relationships			
Type	Table	Attribute	
One-to-One	Accepted_Edit	edit_id	
One-to-One	Rejected_Edit	edit_id	
Many-to-Many	User	user_id	

User_Report			
Attribute Name	Type	Indexed	Required
report_id	Integer	Yes, Primary Key	Yes
report_text	Text	No	Yes
report_type	Text	No	Yes
author_type	Text	No	Yes
admin_report	Text	No	Yes
timestamp	Timestamp	No	Yes
res_timestamp	Timestamp	No	Yes
Foreign Relationships			
Type	Table	Attribute	
Many-to-One	User	user_id	
Many-to-One	User	user_id	
Many-to-One	Content_Piece	content_id	

7.1.1 Query API

The following is a rough layout of the Storage Query API.

Selection Queries

Content

- Retrieve a piece of content.
- Retrieve a list of alternate names.
- Retrieve an edit and list of edits.
- Retrieve a vote and list of votes.

Users

- Retrieve a user.
- Retrieve a list of user email addresses.

Reporting

- Retrieve a user report and list of user reports.

Action Queries

Content

- Store a piece of content and associated content parts.
- Store/update an edit, the associated content part, a vote, and the corresponding list of authors.
- Update a piece of content's deleted_timestamp.

Users

- Store a new user.
- Update a user's user name, email address, confirmation status, password, and remember login data.

Reporting

- Store a user report.

7.2 Search

7.3 Content

Since most application functions (submitting content pieces, searching and retrieving content pieces, etc.) expect immediate retrieval and delivery of data, most of the Content API will be synchronous. Certain functions though, most notably parts of the edit validation and voting process and the delivery of update notifications and reminders to authors, should be asynchronous and run outside of the main request-response event loop.

Hence, Celery will be used for asynchronous task execution, with RabbitMQ selected as the broker. Redis[38] will be used to store results, but manually, not via Celery's result store mechanisms. Redis is chosen over the PostgreSQL database because Redis is more optimized for read and write performance.

7.3.1 Edit Validation and Voting

Asynchronous processing will almost exclusively[39] be needed for certain parts of the edit validation and voting process. The process will roughly proceed as follows (note that, where necessary, we mark events as processed synchronously or asynchronously by writing "sync" or "async", respectively):

1. An edit of a content piece is submitted and received by the Content API. The edit[40] is stored in Redis (sync) and notifications are emitted to both the account pages and emails of all authors of the content piece (async). A periodic task is started which emits an additional email notification every 3 days to all authors who have not yet voted (async). Another periodic task is started which checks after 5 days and 10 days have passed whether the time-dependent conditions have been met for the vote to close (async) (if so, see 3).

38. Configured for maximum persistance by writing every change to disk.
39. It will also be used for password recovery, at the least.
40. And all associated metadata.

2. All relevant pages check Redis for the existence of an edit each time the associated Content API functions are called. All authors of the content piece are now directed to vote whenever they visit the author page (directly, via an email link, a link from their account page, etc.).

3. Upon submission of an author's vote, Redis (specifically the entry containing the edit) is updated with all the vote's information (sync). Calculations are then run to determine whether the vote has ended (async). If not, the process continues and no further changes are made. If the vote has ended, the edit and vote data is stored in the database and deleted from Redis, and, if the edit was accepted, the associated content piece is updated (async). Additionally, all remaining periodic tasks (from 1) are cancelled (async).

7.3.2 Backend-linking API Layout

The following roughly documents the layout of the backend-linking Content API; we denote class and static methods by "class" and "static", respectively.

ContentError

ApplicationError

Name

json_ready (property) - JSON serializable dictionary for transmission over HTTP.

storage_object (property)

Text

json_ready (property) - JSON serializable dictionary for transmission over HTTP.

storage_object (property)

UserData

load_email

json_ready (property) - JSON serializable dictionary for transmission over HTTP.

Content - instantiation of a Content object can create a new content piece or retrieve an existing content piece.

_check_legal (static)

_transfer

bulk_retrieve (class)

get_content_types (class)

check_uniqueness (class)

filter_by (class)

search (class)

autocomplete (class)

store

update (class)

_delete

json_ready (property) - JSON serializable dictionary for transmission over HTTP.

DuplicateError

Edit - instantiation of an Edit object can create a new edit or retrieve an existing edit.[41]

_check_legal (static)

_retrieve_from_storage

_retrieve_from_redis

_transfer

edits_validating (class)

bulk_retrieve (class)

start_vote

save

validate

_accept

_compute_merging_diff

apply_edit

_reject

_notify

conflict (property)

json_ready (property) - JSON serializable dictionary for transmission over HTTP.

AuthorVote - instantiation of an AuthorVote object can create a new author vote or retrieve an existing author vote.

_transfer

_retrieve_from_storage

_retrieve_from_redis

bulk_retrieve (class)

unpack_vote_summary (class)

41. From Redis or the database.

get_vote_summary (class)

check_vote_order

save

votes_needed (class)

json_ready (property) - JSON serializable dictionary for transmission over HTTP.

An email package separate from the Content API will provide email notification functionality and a diff module will provide for the calculation of edit diffs and the merging of edits with existing content parts to produce new versions. A Redis module will also provide for the storage, retrieval, and deletion of edits and votes from Redis.

7.3.3 Client-facing API Layout

The following roughly documents the layout of the client-facing Content View API. Whereas the backend-linking Content API provided classes and methods closely mapping application objects and their related functions, properties, and operations, the Content View API will essentially map to the "pages" actually requested by clients; in other words, it will provide a "view" of the backend for the client-communicating RESTful Pyramid API to consume.

ContentView - instantiation creates a new content piece or retrieves an existing content piece.

user_content (class)

get_content_types (class)

search (class)

filter_by (class)

autocomplete (class)

recent_activity (class)

validation_data (class)

EditView - instantiation creates a new edit or retrieves an existing edit.

bulk_retrieve (class)

conflict (property)

VoteView - instantiation creates a new vote or retrieves an existing vote.

7.4 User

The following roughly documents the layout of the User API.

PasswordError

UserNameError

EmailAddressError

RegisteredUser - instantiation creates a new user or authenticates and retrieves an existing user.

- _check_legal (static)
- _transfer
- register
- store
- send_welcome
- initiate_confirm
- request_confirm
- expire_confirm
- process_confirm (class)
- remember_user
- update (class)
- delete (class)
- json_ready (property)

Admin - instantiation creates a new admin or authenticates and retrieves an existing admin.

- promote (class)
- demote
- json_ready (property)

Report - instantiation creates a new report or retrieves an existing report.

- _check_legal
- _transfer
- bulk_retrieve (class)
- submit
- assign_admin
- resolve
- save
- store
- json_ready (property)

7.4.1 Client-facing API Layout

UserView - instantiation creates a new user or authenticates and retrieves an existing user.

confirm (class)

update (class)

delete (class)

AdminView - instantiation creates a new admin or authenticates and retrieves an existing admin.

ReportView - instantiation creates a new report or retrieves an existing report.

bulk_retrieve (class)

resolve (class)

8 Pyramid Web Layer

The following approximately documents the public web layer, built using the Pyramid framework. To reduce redundancy and promote a functionally-optimal stack layout, the web layer will be implemented as a thin layer on top of the client-facing Content and User APIs primarily only handling request routing, templating, data serialization, and response transmission. The web layer will be split into two separate components, the web API and the UI server.

Note that we will assume a top-level domain of https://kdb.com and denote a placeholder for an attribute, say id, by {id}.

8.1 Web API

Approximate RESTful web API reference:

API Endpoint – https://api.kdb.com

/content

GET – Retrieve a list of all content pieces, paginated with 10 results per page and sorted by timestamp of creation.

GET ?sort={parameter} – Retrieve a list of all content pieces, paginated with 10 results per page and sorted in descending order by {parameter} (created_at, last_edited_at).

GET ?{part}={string} – Retrieve a list of all content pieces with a {part} (keyword, content type, name, or citation) matching {string}, paginated with 10 results per page and sorted by timestamp of creation.

GET ?q={string} – Search all content pieces by query {string}, resulting list of top matching 1000 content pieces is paginated with 10 results per page and sorted in descending order by query match strength.

POST – Creates a new content piece.

/{id}

GET – Retrieve the content piece with content_id {id}.

/authors

GET – Retrieves a list of all authors of this content piece.

/edits

GET – Retrieve a list of recent validating edits (sorted in descending order by creation date) and a list of recently closed edits (accepted and rejected, sorted in descending order by validation date), both paginated with 10 results per page. Also include a count of the number of validating edits and the number of closed edits.

POST – Submit an edit of this content piece.

/{id}

GET – Retrieve the edit of this content piece with edit_id {id}

GET ?check_conflict={bool} – Retrieve the edit of this content piece with edit_id {id} and, if its a validating edit and {bool} is true, include a boolean conflict indicator (indicating whether this edit likely conflicts with other validating edits).

/votes

GET – Retrieve the vote summary for this edit.

POST – Submit a vote for or against this edit.

/edit_activity

GET – For this content piece, retrieves a list of all authors, and two paginated lists (with 10 results per page): the first contains recent currently validating edits and the second contains recent closed edits, including both accepted and rejected edits. Also includes a list indicating which validating edits the requesting author still needs to vote on.

/names

GET ?complete={string} – Retrieve a list in descending order of the 10 best matching name or alternate name completions of the fragment {string}.

/content_types

GET – Retrieve a list of all content types.

/keywords

GET – Retrieve a list of all keywords, paginated with 50 results per page and sorted in descending alphabetical order.

GET ?complete={string} – Retrieve a list in descending order of the 10 best matching keyword completions of the fragment {string}.

/citations

GET – Retrieve a list of all citations, paginated with 50 results per page and sorted in descending alphabetical order by first author last name.

GET ?complete={string} – Retrieve a list in descending order of the 10 best matching citation completions of the fragment {string}.

/users

POST – Register a new user.

PATCH – Alter the state of this user. Options include changing the password, username, or email address, or confirming an email address.

DELETE – Delete a user account.

/sessions

POST – Create a new session.

DELETE – Delete a session.

/{id}

GET – Redirects to /recent_activity

/recent_activity

GET – Retrieve a list of recent edit metadata for content pieces authored by this user, including closed and validating edits, paginated with 20 results per page and sorted in descending order by submission timestamp or, if present, validation timestamp.

/content

GET – Retrieve a list of metadata for all content pieces authored by this user, including validating edit and vote required indicators.

/edits

GET – Retrieve a list of this user's validating edits and a list of this user's closed edits, both sorted in descending chronological order, as well as the sizes of both lists.

/admins

/{id}

GET – Redirects to /users/{id}.

/reports

GET – Get a list of the most 10 recent reports.

GET ?{id_type}={id}&report_status={status} – Retrieve a list of all user reports with {id_type} (user_id, admin_id, content_id, or ip_address) matching {id} and report status matching {status} (open or resolved), paginated with 10 items per page and sorted in descending chronological order by creation timestamp or, if present, resolution timestamp.

POST – Submit a new report.

/{id}

GET – Retrieve the user report with report_id {id}.

/admin_reports

POST – Submit a new admin report detailing the resolution of this report.