JavaSpecs

A Web-Based Platform for Java Specifications

Final Report
Team: May1639
CprE/SE 492
Spring, 2016

Advisor & Client

Dr. Hridesh Rajan

Project Members

Arik Coats (Web Master, Key Concept Holder)
Evan Dye (Team Leader)
Robert Kloster (Communications Leader)

Table of Contents

Project Purpose

Problem Statement

In the world of software development, specifications inform both humans and computers alike how software is intended to behave, and are highly important to the verification process. Checkable, understandable, useful formal specifications can significantly help minimize the cost of developing software which is secure, assured, and reliable. However, such specifications are not widely available.

Interest in formal specifications, or more precisely behavioral interface specifications, lies primarily in their capability to verify the functionality of software in addition to guaranteeing the behavior of an API. Outside of academia, concern often falls on the observable functionality of code over the explicit guarantees offered by formal specifications. There is little motivation for developers to take the time to write formal specifications for their code, due to the time and difficulty involved. As a result, within an academic setting or any environment that requires strict verification of software, there is a noticeable lack of such specifications for commonly available libraries, and available specifications are not necessarily easily accessible.

A solution offered by Dr. Hridesh Rajan proposes the use of data mining to retrieve formal specifications from sites such as Stack Overflow and consolidate them on a web platform. These specifications may then be approved or modified by users of the platform, and missing specifications may be added manually.

Project Solution

Goals

JavaSpecs is a question and answer (Q&A) website dedicated to the creation, discussion, and refinement of formal behavioral specifications for the most commonly used Java libraries. In particular, our goals include the following;

- Increase the availability, discussion, and sharing of knowledge regarding formal specifications.
- Reduce the time and effort required to learn about and make use of formal specifications.
- Encourage the creation and refinement of formal specifications.
- Leverage collective intelligence in order to study specification inference.

Deliverable

The deliverable is JavaSpecs, a Q&A web forum dedicated to formal specifications for commonly used Java libraries, with the following main systems:

- A system for uploading, extracting and embedding source code on the corresponding forum pages.
- A RESTful API system for external users to query for data from the database.
- A System for querying Stack Overflow and retrieving discussion threads to populate a related posts section for each subforum.

Intended Users

- Programmers seeking information about formal behavioral software specifications.
- Experts in formal behavioral software specifications hoping to add and refine specifications.

Requirements

Functional Requirements

- Allow users to ask and answer questions in a manner similar to Stack Exchange.
- Allow users to navigate supported Java libraries in a Library→Package→Class→Method hierarchy.
- Display source code for all classes and methods.
- Provide a RESTful API to allow external users to retrieve website information.
- Display links to relevant Stack Overflow posts.
- Allow administrators to upload new libraries.

Non-Functional Requirements

- The website is extensible for future developers.
- Suggested posts from Stack Overflow are both relevant and related to the page a user is viewing.
- The data returned by the API will prove useful in researching specification inference.

The noted NFRs do not have any measurable goals to be met, such as “must be down for no more than 5 minutes” or “must be able to support at least 25 users in an hour”. Our goal was to make a proof-of-concept web platform that is intended to be passed on to future developers. Specific, measurable NFRs detailing requirements such as connection speeds, minimum down time or maximum concurrent users are not entirely in our control and are not in the domain of this project.

Design

Possible Solutions

At the beginning of the project, the team discussed three possible means of constructing our web-based Q&A discussion platform.

- Create a Q&A platform using an existing internet forums framework
    - There are a variety of available frameworks for the purpose of creating internet forums, including Q&A discussion sites.
    - Notable options include NodeBB and MyBB.
- Construct a new platform from scratch
    - Using a language/framework such as ASP.NET to create our own platform with our necessary functionality.
- Construct a Stack Exchange site dedicated to formal specifications
    - With permission, it is possible to create a Stack Exchange site dedicated to the discussion of topics distinct enough to separate them from already existing Stack Exchange sites.

Our group has settled on the first option of using the MyBB internet forum framework to create our web-based Q&A discussion platform. It was soon discovered, though, that the MyBB forum structure was significantly different than the Q&A structure that was required, and the choice was made to select a more appropriate web framework. The team revisited the issue, and settled on question2answer 1.7.2, an open-source Q&A framework, as the foundation of JavaSpecs.

Solution Assessment

There are several reasons that make the selection of question2answer the best option. While the option to develop an entirely new web-platform was brought up, it was immediately considered too involved. Considering the size of the team and their prior experience with web development, it would prove too time consuming to both learn a new framework and build a web-platform to meet the project requirements from scratch. The option to create a formal Stack Exchange-affiliated website is one that was seriously considered. However, using the Stack Exchange framework may not have given the team the freedom to adequately customize the web platform as need be. So, using a Q&A framework was determined to be the best option, and the team chose question2answer, because it was, and still is, an open-source, free-to-download forum framework with an active and supportive community of developers.

System Analysis

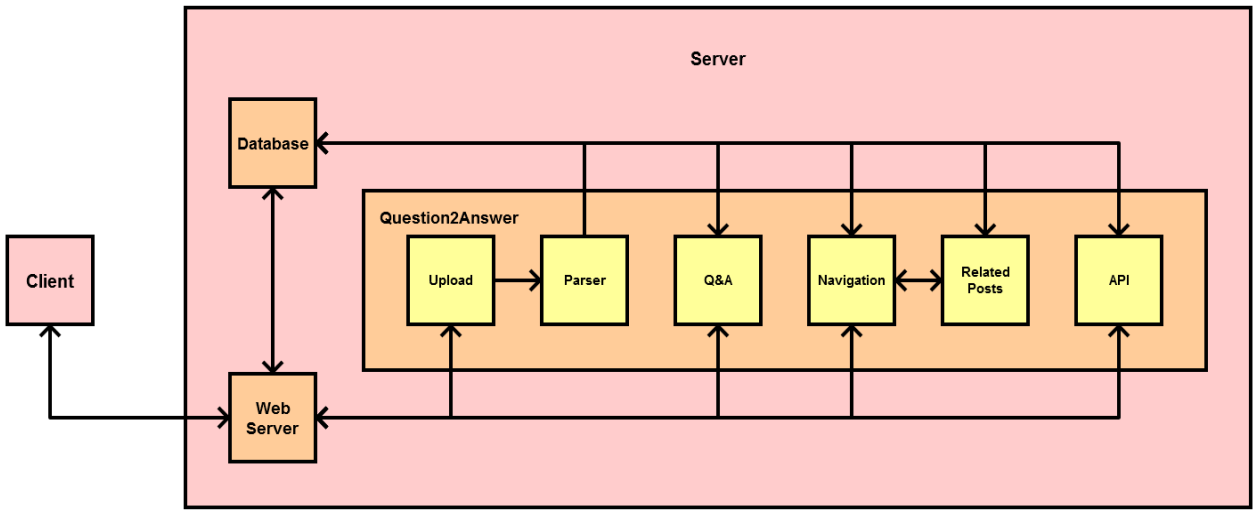

Figure 1: System Block Diagram

The components of the project and their purposes are as follows:

- Client
  - A web browser connecting to the forum website.
- Web Server
  - The intermediate module responsible for assembling the client forum using the information in the project file system and in the project database.
  - In addition, the web server is responsible for performing all interactions with external modules.
- Database
  - A MySQL database that stores information regarding forums, posts, users, and website templates.

- Question2Answer Client Forum
    - The front end display of the project, in the form of a question and answer discussion forum created using the Question2Answer forum software.
        - Upload
            - Allows for the upload of new Java libraries to the forum database.
        - Parser
            - Parses Java files into their base elements (name, return type, arguments, etc.) and stores them in the forum database.
        - Q&A
            - Provides general Q&A functionality; similar to Stack Exchange sites.
        - Navigation
            - Provides Library→Package→Class→Method navigation.
            - Displays source code.
        - Related Posts
            - Provides a list of links to Stack Overflow posts determined to be related to the currently selected method and class.
        - API
            - Enables users to submit complex, flexible queries to the database and acquire useful specification research data.

Interfaces

The major components of the forum interfaces for this project can be broken down as follows:

- Forum Index - The forum homepage.
    - Provides a list of all libraries supported by the forum. Selecting a library navigates to the Library View.
- Library View - Displays relevant information for the currently selected library.
    - Provides a list of all packages in the currently selected library. Selecting a package navigates to the Package View.
- Package View - Displays relevant information for the currently selected package.
    - Provides a list of all classes in the currently selected package. Selecting a class navigates to the Class View.

- Class View - Displays relevant information for the currently selected class.
    - Provides a list of all methods in the currently selected class. Selecting a method navigates to the Method View.
    - Related Discussions - Provides a list of links to Stack Overflow discussions relevant to the currently selected class.
    - Source Code - Displays the source code of this class.
    - Specifications - Displays the currently approved specifications for this class.
- Method View - Displays relevant information for the currently selected method.
    - Provides a list of all discussions concerning the currently selected method. Selecting a discussion navigates to the Discussion View.
    - Related Discussions - Provides a list of links to Stack Overflow discussions relevant to the currently selected method.
    - Source Code - Displays the source code of this method.
    - Specifications - Displays the currently approved specifications for this method.
- Discussion View - Displays the currently selected discussion.
    - Provides a list of all posts in this discussion.

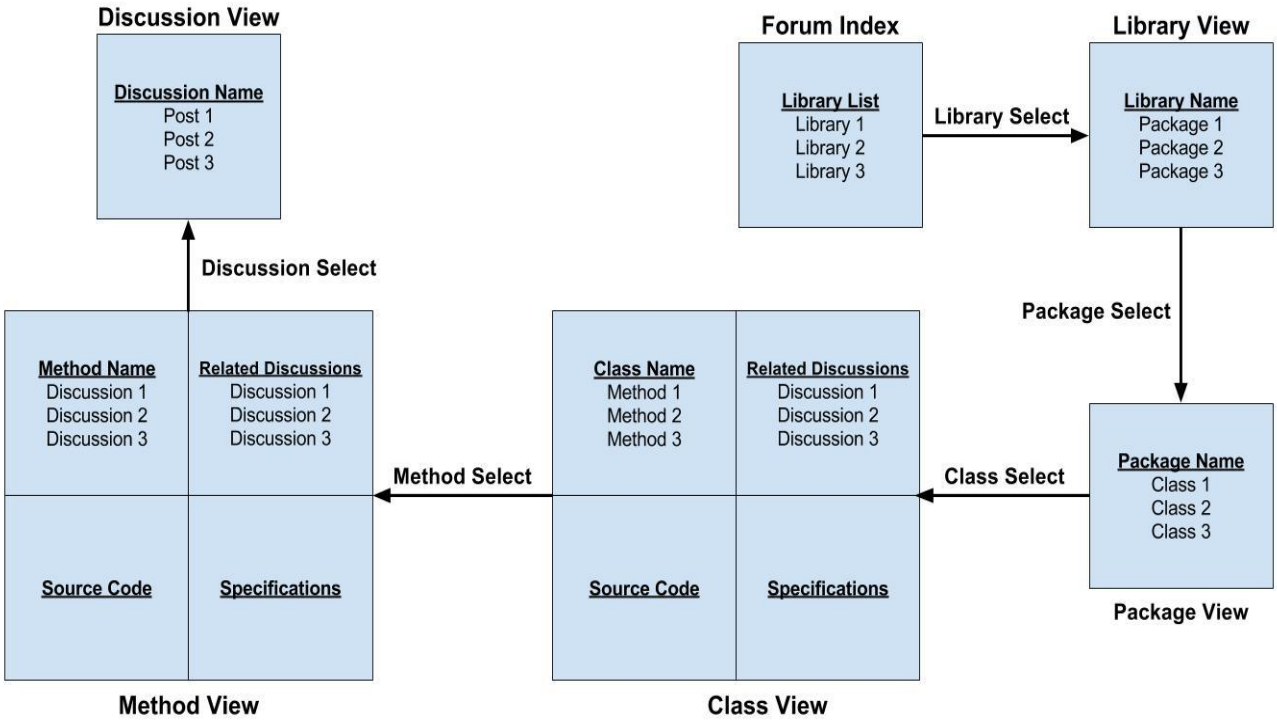

Figure 2: Forum Pageview Diagram

Software Specifications

Software	Use
Java 1.8	Parser development language.
PHP5	Server programing language.
JavaScript	Client side web programing language.
Question2Answer 1.7.2	Provides infrastructure for generic Q&A web platform.
JQuery	JavaScript library for forum GUI behavioral programming.
Eclipse JDT	Java library that provides the core functionality of the parser.
JDBC	Java library used for Communication between the parser and database.
MySQLi	PHP library used for Communication between the forum and project database.

Implementation Details

Parser

The parser is responsible for searching a directory for java files, parsing the java files, extracting their base components, and storing them on the forum database. The program was written in Java 1.8 using Eclipse Mars. The Eclipse JDT library was used to parse java files and separate them into their base components (name, arguments, return type, etc.). The JDBC Library was used to store the extracted source code information on the forum database in tables with the hierarchy Library→Package→Class→Method.

Navigation

The navigation page is used for navigating through parsed libraries stored in the forum database. Initially, a list of names of all libraries are displayed as clickable links. Clicking on a link will update the list to display packages in that library. The new links will follow the same pattern with packages to classes and classes to methods. When a class or method is displayed, a box containing the source code will appear beneath the list of links.

The navigation page also maintains a navigation bar at the top of the page that tracks the user's current location in the library hierarchy. This navigation bar is a series of clickable links that will return the page to their respective views. The bar is formatted to resemble a file path seen in a file explorer.

Lastly, the navigation page contains a search feature for quick navigation. The user can select the type of information desired (library, package, class, method, or any) and enter a name to search for. The search function finds the closest matches within the specified type (or all types starting with method if any was selected) and displays them as a list of clickable links in a file path format identical to the navigation bar. Clicking any of the links will update the page to display the proper content and update the navigation bar.

The Navigation page template is written in html and uses JavaScript to manipulate the DOM. More specifically, jQuery is used to make asynchronous calls to PHP files on the server. The results are passed to JavaScript functions to perform an update to the page content. The PHP scripts called via jQuery use the MySQLi library of functions to perform calls to the forum database. The results are processed and returned in JSON format.

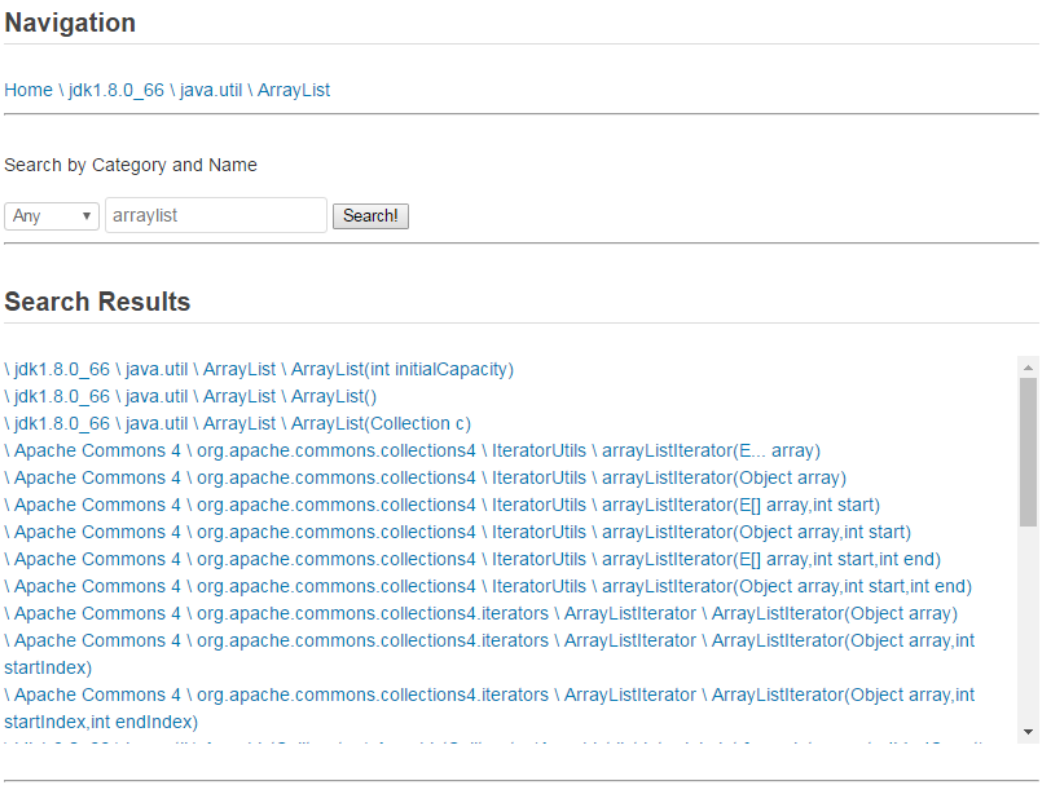

Figure 3: Forum Navigation

Upload

The upload page allows admins to upload new source code archives (.zip or .jar) to the forum to be parsed and added to the database. The new archive will then be browsable on the navigation page. Admins can upload archives in two ways. The first is to click the browse button on the top of the page and upload an archive from their local file system. The second is to copy and paste a download link into the URL box and insert a filename into the name box. This will allow the page to download the linked file and save it to the server with the specified file name. Whichever method is chosen, the admin may then press the corresponding upload button to have the file uploaded to the server.

All uploaded files are displayed in a separate section on the bottom of the page. For each uploaded file, a download link and three buttons are displayed. The text for the download link is the file's name and will open the save file window when clicked. The three buttons are as follows: "Add Archive", "Delete file", and "Remove Archive". The "Add Archive" button will extract and send the contents of the archive to the parser to be processed and added to the database. The "Remove Archive" button removes the archive from the database. The "Delete file" button will delete the uploaded archive file from the server.

The Upload page was written in a similar way to the Navigation page. The page template was written in html and uses JavaScript to manipulate the DOM. Likewise, jQuery was again used to asynchronously call PHP files which would handle database operations and return the results in either JSON or html format. The database communications also used the PHP MySQLi library.

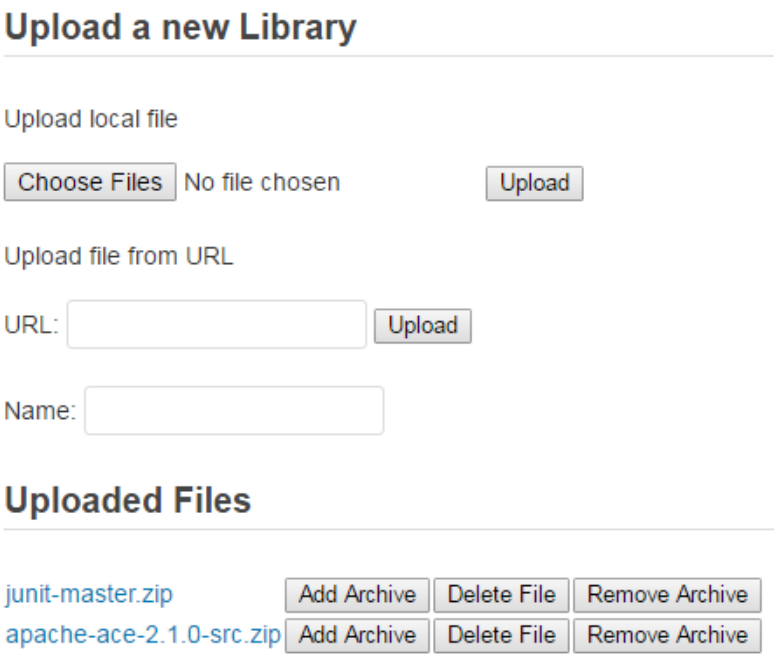

Figure 4: New Library Upload Page (Administrators Only)

Related Posts

When a class or method is selected on the navigation page, the forum will attempt to display of a list of links to Stack Overflow posts related to the selected class or method. The goal of these links is to provide easily accessible resources for researching the functionality and behavior of a class or method to facilitate the creation of software specifications. The two main components of this module involve accessing Stack Overflow post data and determining which posts are most relevant to a given selection.

In order to access Stack Overflow post information, a copy of these posts are stored in the project database. While Stack Exchange provides an extensive API for retrieving post information from its subsidiary sites, it has several limitations, discussed in the Issues and Challenges section below, making it a less useful option. Instead, this project takes advantage of Stack Exchange's quarterly data dump, which makes all Stack Overflow post information available in XML format. This information is then parsed into two database tables. The first table stores all post information, while the second forms a dictionary of post tags and title words.

In order to determine related posts, these posts are ranked based on a variety of factors. Posts must first contain the selected class and method names in the post title or as forum tags. This condition can be easily checked through the dictionary table in the database. Posts meeting these criteria are then further considered based on such factors as the number of post views and the post score. In general, a post with a large number of views or a high score can be considered a quality post.

Figure 5: Related Posts for the ArrayList Class Constructor

API

The API system is responsible for enabling an external user, such as 3rd party software, to submit flexible, complex queries to the database and provide the requested data in JSON format. The user can query for one of 6 different types of data;

- Users
- Questions
- Answers
- Posts (Generalized form of either questions or answers)
- Comments
- Tags

This system's generalized interface design was modeled after the API systems of two other websites;

- GitHub's repository searching API call inspired our API's;
  - Input structure.
  - Query generation format.
- Stack Exchange's API library inspired our API's;
  - Output structure.
  - Object design.
  - Call types.

Implementation Issues and Challenges

Limited Stack Exchange API Calls

Originally, the plan was to implement a system for calling Stack Overflow's API for links to posts that have relevancy to the code the user was looking at. The Stack Exchange API, however, has an API call limit of 300 calls per day per source (10,000 with an access key). While obtaining a key is not an issue and would allow this method to work on the current platform, there is a further problem with a 30 request per second limitation per IP, while determining related posts may require multiple calls per page to obtain enough possible results. This leads to issues when scaling up the scope of the project. This was circumvented by downloading a dump file of Stack Overflow post information and parsing it into a word dictionary in the database. This way, JavaSpecs can both search for related posts and present the relevant links internally.

Parsing Large Amounts of Data

The Stack Overflow data dump allowed local parsing and access to all Stack Overflow posts as of August, 2015 in XML format. At this time, there were over 30 million posts on Stack Overflow, resulting in an approximately 35 GB XML file. This file size is rather unwieldy, as parsing and storing it into a database takes a large amount of time. In addition, ranking post information in real time (on page loads) requires a number of database calls to search through millions of database entries, resulting in slow execution times. Currently, although the scripts are written, the actual XML parsing has been reserved for future users to complete. However, there are still several million parsed posts in the database at the current time, and future users would be required to update the post database periodically anyway, as new data dumps are released quarterly with millions of new posts each time. The slow real-time execution is being addressed by requiring this slow execution the first time a page is loaded, but then storing related post ranking information in the database for faster access in the future.

Server-side Computational Access

The JavaSpecs library parser is designed to parse an incoming Java library or JAR file and insert the proper entries into the database. The server space originally allotted, however, did not enable us to compile or run the parsing program server-side. As such, the team put in a requisition for a new server location and virtual machine to run the parsing software.

API Design Modeling

At its inception, the JavaSpecs API was designed to emulate the Stack Exchange API system all but completely for simple integration with existing data mining software. Over time, it was determined that the resulting API calls were not flexible enough for advanced query processing. Therefore, the decision was made to model the JavaSpecs API calls ‘conditions’ parameter after the GitHub repository search API call in order to expand the versatility of submitted queries.

Testing Processes

Verification Procedures

Test Driven Development

Team May1639 employed the Test Driven Development, or TDD, methodology to develop the multiple components of JavaSpecs. By capitalizing on the incremental addition of functionality, the team was able to generate flexible, powerful software that will prove easy for future developers to extend and improve.

Modular Unit Testing

Given the highly modular nature of the individual components, unit testing comprised the majority of the verification process. As the components have very low coupling in this project, software failures were most likely to occur within a component rather than between components. Thorough testing of each module at the end of a development cycle was performed before integrating with the main repository.

Validation Procedures

Customer Testing

Throughout the development of JavaSpecs, team May1639 met with their client on a regular biweekly (once every other week) basis to demonstrate the progress made by the end of the sprint. This provided us with immediate feedback on the validity of the progress made at that point, as well as which areas should be improved.

Prototype Testing

Team May1639 constantly maintained a valid functional implementation of the JavaSpecs website throughout the development process, extending and adding to it in increments along the way. This granted the team an opportunity to test the effectiveness of the added functionality during the development process, instead of after the software had been completely developed.

Conclusion

The direct result of this project is a web platform to host formal specifications for commonly used Java libraries. In addition, the platform is of a Q&A format allowing human users to validate and edit existing or computer generated specifications as well as propose and manually create new specifications. The platform is planned for use past the departure of the developing senior design students, and development will proceed with the intention of passing the work on to the community for future upkeep and extension.

Currently, there are no such services that meet the needs outlined above, so in addition to providing a working platform with the described functionality, it will also serve as a proof of concept that such platforms are feasible, ideally prompting further development in the area of formal specifications.

References

1. Rajan, Hridesh, Tien N. Nguyen, Gary T. Leavens, and Robert Dyer. "Inferring Behavioral Specifications from Large-scale Repositories by Leveraging Collective Intelligence." “37th International Conference on Software Engineering: NIER Track” May 2015, ICSE’15, Florence, Italy.

Appendix I: Operation Manual

Project Website URL

http://may1639.github.io/

JavaSpecs URL

http://may1639-2.ece.iastate.edu/q2a/

How to use the API

The API can be called using the following URL:

http://may1639-2.ece.iastate.edu/q2a/api.php

All data objects that can be returned from an API call can have one or more variables, and these variables can be either a boolean value, a date and time value, an integer value, and a string value.

API GET Parameters

The ‘conditions’ parameter

This parameter allows the user to enter any number of conditions and/or search terms to filter the results received.

What is a ‘condition’?

A condition is a constraint on a specific variable of the type of object being called. A condition has the general form;

variable_name:argument

For example, the condition “Only return results that have view_count values greater than 10.” is represented as

view_count:>10

Any condition can be negated by prefixing it with -. For example, the condition;

-view_count:>10

means “Only return results that **do not** have view_count values greater than 10.” If the argument contains whitespace, the argument needs to be surrounded with double-quotes. For example, the condition

view_count:“ 20 .. 30 ”

equates to “Only return results that have view_count values between 20 and 30, inclusive.”

A note about datetime or integer conditions: if there is more than one “the value is equal to” condition in the same call, an object only has to be satisfy one of them in order to be returned.

Boolean Conditions

The argument of a valid boolean condition is either “true” or “false”. For example, the condition “is_answered:false” means “Only return results whose is_answered value is false.”

Datetime Conditions

The argument of a valid datetime condition is one of the following;

1. A date (and time, optionally) (same as operator being ‘=’)
    a. Examples:
        i. creation_date:2016-01-01
        ii. last_modified_date:“2016-02-05 12:34:56”
2. An accepted operator followed by a date (and time, optionally)
    a. Accepted operators: ‘=’, (same net effect as no operator,) ‘!=’, ‘<’, ‘>=’, ‘>’, ‘<=’
    b. Example:
        i. last_access_date:<=2014-07-23
3. A date (and time, optionally) followed by the range operator ‘..’ followed by another date (and time, optionally)
    a. Example:
        i. last_edit_date:2000-01-01..2016-01-01

Integer Conditions

The argument of a valid integer condition is one of the following;

1. A number (same as operator being ‘=’)
    a. Examples:
        i. question_id:100
        ii. view_count:“ 125 ”
2. An accepted operator followed by an integer
    a. Accepted operators: ‘=’ (same net effect as no operator), ‘!=’, ‘<’, ‘>=’, ‘>’, ‘<=’
    b. Examples:
        i. answer_id:>9000
        ii. user_id:“ != 42 ”
3. An integer followed by the range operator ‘..’ followed by another integer
    a. Example:
        i. accept_rate:“ 40 .. 400 ”

String Conditions

The argument of a valid string condition is a non-empty string that does not contain a double quote. Only objects which contain the specified argument in the specified string variable will fulfill this condition. Some examples:

- title:search_term_to_look_for
- body:“ search phrase to look for ”

What do we mean by ‘search term’?

A search term is any string in the ‘conditions’ parameter that is not part of a condition, does not consist entirely of whitespace, and does not contain a double quote (“) within it. For an object to be returned from an API call, all search terms must occur at least once in any of the string variables specified in the ‘in’ parameter (defaults to all string variables the object has if the ‘in’ parameter is not specified) Any search term can contain whitespace if contained within double-quotes.

Any search term can be negated by prefixing the string with NOT and a white space; for an object to be returned from an API call, no negated search term can occur even once in any of the string variables specified in the ‘in’ parameter (defaults to all string variables the object has if the ‘in’ parameter is not specified)

The ‘in’ parameter

This parameter contains a comma-delimited list of string variables that specify which string variables get searched in by the search terms. Defaults to all string variables that the object has if the ‘in’ parameter is unspecified.

The ‘page’ parameter

This parameter takes a positive integer representing the page of results the user receives. The default page is 1. (The first page of results)

The ‘pagesize’ parameter

This parameter takes an integer from 1 to 100 (inclusive) representing the number of objects returned per page. Defaults to 30.

The ‘sort’ parameter

This parameter takes a comma-delimited list of sorting arguments, read left to right, that determines how the results will be sorted. Each sorting argument contains a variable name, optionally followed by a colon and either ASC or DESC, which changes the order of the sort. For example, the ‘sort’ parameter “reputation, display_name ASC” means “Sort the results by the reputation value in descending order, and sub-sort by the display_name value in ascending order.” (Default order of sort is DESC)

Callable Data Types

Answer

- API Call URL
  - http://may1639-2.ece.iastate.edu/q2a/api.php/answers
- Primary Key
  - answer_id (Integer)
- Foreign Key(s)
  - owner REFERENCES User (user_id)
  - question_id REFERENCES Question (question_id)
- Boolean Variables (N/A)
- Datetime Variables
  - creation_date The date that the Answer was created.
  - last_activity_date The last time the Answer was created or edited.
  - last_edit_date The last time the Answer was edited.
- Integer Variables
  - answer_id The unique identifier for this Answer.
  - down_vote_count The number of downvotes this Answer has.
  - owner The user_id of the User that owns the Answer.
  - question_id The question_id of the Question that the Answer is for.
  - score The Answer’s score (up_vote_count - down_vote_count).
  - up_vote_count The number of upvotes this Answer has.
- String Variables
  - body The main body of the Answer.

Comment

- API Call URL
  - http://may1639-2.ece.iastate.edu/q2a/api.php/comments
- Primary Key
  - comment_id (Integer)
- Foreign Key(s)
  - owner REFERENCES User (user_id)
  - post_id REFERENCES Post (post_id)
- Boolean Variables
  - edited — Whether or not the Comment has been edited before.
- Datetime Variables
  - creation_date — The date that the Comment was created.
- Integer Variables
  - comment_id — The unique identifier for this Comment.
  - owner — The user_id of the User that owns the Comment.
  - post_id — The post_id of the Post that the Comment is for.
  - score — The score of the Comment.
- String Variables
  - body — The main body of the Comment.

Post

- API Call URL
  - http://may1639-2.ece.iastate.edu/q2a/api.php/posts
- Primary Key
  - post_id (Integer)
- Foreign Key(s)
  - owner REFERENCES User (user_id)
- Boolean Variables (N/A)
- Datetime Variables
  - creation_date The date that the Post was created.
  - last_activity_date The last time the Post was created or edited.
  - last_edit_date The last time the Post was edited.
- Integer Variables
  - down_vote_count The number of downvotes this Post has.
  - owner The user_id of the User that owns the Post.
  - post_id The unique identifier for this Post.
  - score The Post's score (up_vote_count - down_vote_count).
  - up_vote_count The number of upvotes this Post has.
- String Variables
  - body The main body of the Post.
  - post_type "Question" if the Post represents a Question, "Answer" if the Post represents an Answer.
  - title The title of the Post.

Question

- API Call URL
  - http://may1639-2.ece.iastate.edu/q2a/api.php/questions
- Primary Key
  - question_id (Integer)
- Foreign Key(s)
  - accepted_answer_id REFERENCES Answer (answer_id)
  - owner REFERENCES User (user_id)
  - tags REFERENCES (multiple) Tag(s) (name)
- Boolean Variables
  - is_answered — Whether or not the Question has any Answers.
- Datetime Variables
  - creation_date — The date that the Question was created.
  - last_activity_date — The last time the Question was created or edited.
  - last_edit_date — The last time the Question was edited.
- Integer Variables
  - accepted_answer_id — The answer_id of the accepted Answer.
  - answer_count — The number of Answers this Question has.
  - down_vote_count — The number of downvotes this Question has.
  - owner — The user_id of the User that owns the Question.
  - question_id — The unique identifier of this Question.
  - score — The Question's score (up_vote_count - down_vote_count).
  - up_vote_count — The number of upvotes this Question has.
  - view_count — The number of times this Question has been viewed.
- String Variables
  - body — The main body of the Question.
  - tags — A semicolon-delimited list of all included Tags.
  - title — The title of the Question.

Tag

- API Call URL
  - http://may1639-2.ece.iastate.edu/q2a/api.php/tags
- Primary Key
  - name (String)
- Foreign Key(s) (N/A)
- Boolean Variables (N/A)
- Datetime Variables
  - last_activity_date — The last time a Question that has this Tag was created or edited.
- Integer Variables
  - count — The number of Questions this Tag appears in.
- String Variables
  - name — The unique identifying word of this Tag.

User

- API Call URL
  - http://may1639-2.ece.iastate.edu/q2a/api.php/users
- Primary Key
  - user_id (Integer)
- Foreign Key(s) (N/A)
- Boolean Variables (N/A)
- Datetime Variables
  - creation_date — The date that the User was created.
  - last_access_date — The last time the User was created or modified.
  - last_modified_date — The last time the User changed their settings.
- Integer Variables
  - accept_rate — The rate at which this User's Answers are accepted.
  - answer_count — The number of Answers this User owns.
  - badge_count — The number of badges this User owns.
  - down_vote_count — The number of downvotes this User has.
  - question_count — The number of Questions this User owns.
  - reputation — The User's score (up_vote_count - down_vote_count).
  - up_vote_count — The number of upvotes this User has.
  - user_id — The unique identifier for this User.
- String Variables
  - display_name — The displayed name of the User.

Appendix II: Other Considered Designs

Web Platform

- MyBB — Large differences from the required Q&A design
- Custom-built — Time and resource constraints
- Stack Exchange site — Lack of flexibility
- question2answer — Chosen solution

Source Code Display

- Grepcode — Limited searching options
- Reflection API — Insufficient data extraction from Java code
- Custom JDT Parser — Chosen solution

API Design

- Stack Exchange — Limited query abilities
- GitHub — Difference in relevance between API calls
- Stack Exchange + GitHub — Chosen solution

Related Posts

- Stack Exchange API — Limited calls per day
- Local Posts dictionary — Chosen solution

Appendix III: Code

All of the code for the JavaSpecs project has been aggregated in a GitHub repository, the link to which can be found here: https://github.com/may1639/may1639.github.io