Abstract

Apache Bloodhound is a software development collaboration tool recently released on the market. Bloodhound extends Trac with multiple projects management, powerful search functionality, ease of installation and a user-friendly interface. It reduces the effort of tracking projects by providing a clean interface to connect revision control with wiki content and a bug database.

Autocompletion of ticket fields would be a great feature for the Bloodhound, as it saves user’s search time, catch a user’s mistake and allows repeating a favorite search. Typing the whole search word makes finding information slower and requires more typing and also increases the chance of misspelling that leads to incorrect search results which degrades the interface usability.

Description

Autocompletion features can be divided into three components for this project; autocompletion of keywords, autocompletion of usernames and search for duplicate tickets.

Each of these components has a link to a plugin in track-hacks, called KeywordSuggestPlugin, AutocompleteUsersPlugin and DuplicateTicketSearchPlugin respectively. However, we don't want to clutter the Bloodhound project with another plugin for basic features.

Therefore, each of these features can be added to the BloodhoundThemePlugin as an individual component. This will work well since the client-side functionality (i.e. the JavaScript parts) can be added directly in each individual feature and the server-side request handling can also be handled in individual components within the BloodhoundThemePlugin. Moreover, the user will able to enable/disable each feature since they are implemented individually into the BloodhoundThemePlugin.
Implementation Details

The implementation of the project can be split into three distinct parts, the keyword and user suggestion components, and duplicate ticket search component.

The keyword suggestion component of the project will suggest entries that are used in existing tickets while user suggestion component will autocomplete the Owner and Cc fields from a list of valid users. Detecting left mouse button press in the search field will be sufficient to display suggested keywords in an alphabetical list that the user can scroll through. Instead of an alphabetical order, popularity of entries (i.e. frequency of keywords) would be used to order the suggested keywords and also matching would not be case sensitive to give more flexibility to the user in typing. This would be implemented simply by converting the user’s entry to lowercase letter before looking for suggestions.

Detecting left mouse button press would be implemented in JQuery by mousedown() function. For instance,

```javascript
  //list keywords in an alphabetical order
  $('#mousedown').mousedown(function()
  });
```

would be an example of how the detecting left mouse button press code will look like.

Popularity of entries would be implemented by storing the keywords and their frequencies in a table in the database. In more detail, we will create a table in the database with all possible suggestions (and will create another table for all possible usernames), all starts with zero frequency and then when user searches for a keyword/username, we will increase the frequency of that specific word in the database. We will suggest keywords/usernames in an alphabetical order, until we have useful frequency results.

The keyword and user suggestion component of the project will be coded using jQuery (JavaScript library). Since most of Bloodhound’s design is done using the Twitter Bootstrap framework, the autocompletion part of the code can also be designed using Twitter Bootstrap Typeahead plugin.

Suggestions can be stored into the source array and can be retrieved on page load. However, this is really inefficient when there are thousands of keywords/usernames since user needs to wait for a long time for the page to
be displayed. Alternative solution to this would be storing the suggestions in dynamically retrieved server-side (i.e. database), which will provide faster and more efficient results. More advanced approach would be using the dynamically retrieved server-side with a JSON format (JavaScript Object Notation) for human-readable data interchange between client and server.

Using dynamically retrieved server-side for retrieving the suggestions will be faster. However, there will be some latency in retrieving the possible keywords/ usernames. In order to overcome the latency, AJAX (Asynchronous JavaScript And XML) would be used to leave client active while the server retrieves the keywords/ usernames.

The duplicate ticket search component of the project will assist the user to search for the duplicate tickets when entering a new ticket’s summary. One of the possible solution would be determining the duplicates by searching the summary of existing tickets whereas searching the ticket description would be better and powerful solution.

The duplicate ticket search part of the project will be coded using jQuery and AJAX. Since most of Bloodhound’s design has been done using the Twitter Bootstrap framework, this part of the code can also be designed using Twitter Bootstrap. First, the duplicate search feature will be implemented to work with the Trac Search API then it will be manipulated to work with the BloodhoundSearchPlugin since it is much more powerful than Trac Search API. However, the ticket Query API would be another solution to implement duplicate search feature rather than interacting with Search components. The ticket Query API has support for searching ticket summaries and descriptions that contain words and phrases.

Using TicketQuery wiki macro which listing tickets that match certain criteria would be a better solution to duplicate ticket search, since we want to present the duplicate tickets to the user as a list according to ticket ID next to the summary, with each ticket on a separate line. This ticket presentation is one of the format parameters that determine how the list of tickets is presented in TicketQuery wiki macro.

On the other hand, we would need the server to do the work of rendering the macro for us. If there is a macro that gives (or can be made to give) the results in the form that we require, that could be an interesting shortcut but just requesting the data we require should also work.

Additionally, there are going to be a number of areas of the code that deal with tickets including the base implementation from trac and the modifications added by bloodhound. We are going to interact with these through the various
interfaces that are available in trac, like IRequestHandler (for handling web requests), ITemplateProvider, IRequestFilter and others.

Initially, we will start to implement the project with bloodhound without the BloodhoundMultiproduct, since it causes complication, after that support for Multiproduct might be added.

Overall, this project will provide faster search facility to users whereas decrease the chance of misspelling. In other words, it will improve the usability and the core functionality of the Apache Bloodhound. This project would also be a start point for my contribution to the Bloodhound application.

**Deliverables**

- Keyword Suggest Component
- Autocomplete Users Component
- Duplicate Ticket Search Component
- Unit tests for the three main components
- Documentation

**Timeframe**

**May 19 – June 8**
- Implement the Keyword Suggest Component for BloodhoundThemePlugin.

**June 8 – July 8**
- Implement the Autocomplete Users Component for BloodhoundThemePlugin.
- Research the Trac ticket Query API

**July 3**
- Midterm Evaluation

**July 8 – August 8**
- Implement the duplicate ticket feature to work with the ticket Query API
- Add support for the multiproduct API
August 8 – August 16

- Refactor code, finish test and documentation

August 18

- Final Evaluation

Bio

I am a third year Software Engineering student at the University of Sheffield. During my education, I have gained practical and analytical skills on Computer Science fields including various programming languages, such as, Java, Python, HTML, CSS, JavaScript and others. In the past few years I have been developing web applications both in my leisure time as well as part of my coursework and summer internship. Last year, I participated in Personalised Access to Cultural Heritage Spaces (PATHS) EU Project, which is a web-based software that provides users with an interactive and more personalized experience of using digital library collections. It was a great experience and I believe that developing dynamic client-side autocompletion features for the Apache Bloodhound ticket system would give me the opportunity to gain more experience on developing user interactive web applications.

Looking forward to a great Summer of Code!