Arman Daniel Marcus Robert

Code Search Engine Fully Dressed Use Cases

Use Case 1: Log in to user account

The user logs in to their account.

Primary Actor

The user - A researcher or programmer interested in a GitHub repository

Stakeholders

The user - Needs accurate, fast results

The people the searcher is searching on behalf of (ex: their team)

Preconditions

The user has an account with the system.

Success Guarantee

The user is logged into the system.

Main Success Scenario

1. The user clicks login to proceed to the login page.
2. The user provides their username and password.
3. The user clicks log in.
4. The system checks both of these values against the stored values for the account.
5. The system redirects the user to a page where they are logged in.

Extensions

The user enters incorrect data (invalid username, wrong password, or missing information):

1. The user clicks login to proceed to the login page.
2. The user provides their username and password.
3. The user clicks log in.
4. The system checks both values against the stored values for the account.
5. The system displays an error indicating that the user has not logged in.

Special Requirements

-The user must be logged into the proper account.

-The user's password must not be shown.

-The user's account information must be securely transmitted.

Technology and Data Variation List

-An HTML form is used for the front end.

-A scripting language such as JavaScript or PHP will be used for verification.

Frequency of Occurrence

Once every 30 days (expiration of the login)

Use Case 2: Change syntax formatting:

Change the color scheme for a particular results page.

Primary Actor

The user - A researcher or programmer interested in a GitHub repository

Stakeholders

The user - Needs results displayed in a useful way

The people the searcher is searching on behalf of (ex: their team)

Preconditions

The user is viewing a page with results

Success Guarantee

The lines are formatted with the new, desired style.

Main Success Scenario

1. The user clicks a change formatting button.
2. The user selects a new format.
3. The system displays the lines on the page with the new formatting.

Extensions

None

Special Requirements

None

Technology and Data Variation List

-HTML and CSS will be used for styling. An appropriate framework may be used.

Frequency of Occurrence

As desired by the user.

Use Case 3: Search a repository:

The given repository is searched for any lines matching the query.

Primary Actor

The user - A researcher or programmer interested in a GitHub repository

Stakeholders

The user - Needs accurate, fast results

The people the user is searching on behalf of (ex: their team)

Preconditions

The user has already submitted a valid GitHub URL and query

Success Guarantee

The system accurately displays any lines in the repository that match the query.

Main Success Scenario

1. The system generates a request for the GitHub api
2. The system submits the request to the GitHub api
3. The system accepts results from GitHub.
4. The system shows results on the page.

Extensions

The GitHub api is down:

1. The system generates a request for the GitHub api
2. The system submits the request to the GitHub api
3. The system receives either a timeout or an error from the api.
4. The system displays an error to the user.

No results were found:

1. The system generates a request for the GitHub api.
2. The system submits the request to the GitHub api.
3. The system receives no results from the api.
4. The system generates a message indicating the lack of results.
5. The system displays the message to the user.

Special Requirements

-Results of all sizes must be properly accepted and displayed.

Technology and Data Variation List

-The appropriate data structure may be an array or other structure.

-A scripting language such as JavaScript or PHP will be used for querying..

Frequency of Occurrence

Once per search

Use Case 4: Limit query to code changes or to commit messages:

Only show a result if it is a code change or a commit message, depending on the user's preference.

Primary Actor

The user - A researcher or programmer interested in a GitHub repository

Stakeholders

The user - Needs results displayed in a useful way

The people the searcher is searching on behalf of (ex: their team)

Preconditions

The user has already submitted a valid GitHub URL and query

Success Guarantee

The system returns only results that match the user's request.

Main Success Scenario

1. The user provides a URL and Query to the system.
2. The user chooses whether to search for code changes or commit messages.
3. The system generates the appropriate api call based on the user's input.
4. The system queries the api.
5. The system shows the returned results.

Extensions

No results were found:

1. The user provides a URL and Query to the system.
2. The user chooses whether to search for code changes or commit messages.
3. The system generates the appropriate api call based on the user's input.
4. The system queries the api.
5. The system generates a message indicating the lack of results.

Special Requirements

-Results of all sizes must be properly displayed.

Technology and Data Variation List

-HTML and CSS will be used for styling. An appropriate framework may be used.

-A scripting language such as JavaScript or PHP will be used for verification.

Frequency of Occurrence

Once per search.

Use Case 5: Filter what is being searched for

Primary Actor

Searcher (user): A researcher or programmer interested in a GitHub repository

Stakeholders and interests:

The searcher - Needs accurate, fast results
The people the searcher is searching on behalf of (ex: their team)

Preconditions

The user has the URL of a Github repository of interest and a query in mind

Success Guarantee

Results are filtered based on the options (filters) the user selected

Main Success Scenario

1. The user inputs the URL of the desired GitHub repository

2. The user types in the query

3. The user selects filters to show results by (ex: show only results from commits the past week and/or from function names)

4. After clicking submit, the system finds all results for the query matching both the keyword and filters.

5. The system displays these results to the user

Extensions

4a. The repository doesn't exist: The system generates an error and displays it to the user

4b. No results were found:
    1. The system generates an error and displays it to the user

    2. The system suggests alternate search filters based on what it did find if any

Special Requirements

-Results must be returned quickly and there should be a variety of relevant filters

-Filters should not make the screen looked cluttered and difficult for mobile browser users

Technology and Data Variation List

3a. Filtering may be done in a few ways, but chiefly either by filtering as it searches or first finding all the textual matches, then filtering

3a. Filtering may be done in JavaScript, PHP, or other applicable languages

4a. Searching may use JavaScript, PHP, or other applicable languages.
4b. Searching may be done for a query string or keywords

Frequency of Occurrence

Whenever the uses decides to search

Use Case 6: Enable/disable suggested search terms based on history

Primary Actor

Searcher (user): A researcher or programmer interested in a GitHub repository

Stakeholders and interests:

Searcher - wants their search history to help them with future searches
People the searcher is searching on behalf of (ex: their team)

Preconditions:

The user has made searches and the system has stored the search terms and or filters

Success Guarantee:

The user sees suggested search terms based on their history or not if they disable the feature

Main Success Scenario

1. The user starts typing in the search box

2. Suggested search terms appear based on their search history as they type

3. User selects one of the suggested searches

4. The filters associated with that suggested search become selected on the UI

5. The system searches with the suggested search term(s) and its associated filters (if any) and display results

OR

I. The user selects an option to disable suggested search terms

II. The user starts typing in the search box

III. Suggested search terms do not appear

Extensions

2a. If the user types a totally new search query, no suggested searches should display

Special Requirements

-suggested search terms need to appear very quickly on the UI to be effective as the user types

-suggested search terms should not be take too much of the screen

Technology and Data Variations List

Ia. The option to enable or disable can be checkbox or toggle on the main page of the program or in a separate settings page- may depend on how much screen clutter there is after implementing the majority of the UI

Frequency of Occurrence:

Whenever the user is typing in the search box and there exist relevant search terms and the user has not disabled the feature

Use Case 7: Interact with Visualizations

Primary Actor

Searcher (user): A researcher or programmer interested in a GitHub repository

Stakeholders and interests:

Searcher - wants useful ways to understand the information of results
People the searcher is searching on behalf of (ex: their team)

Preconditions:

The system has found relevant results for a search

Success Guarantee:

The user can choose to display or hide visualizations and can hover their mouse or click on visualizations for more useful information if any exists.

Main Success Scenario:

1. The user clicks to display visualization

2. The visualization is displayed

3. The user hovers their mouse over visualization and additional info is displayed

4. The user clicks on the on the additional info and the system shows where in the code or commit message the search criteria was found or other relevant info depending on the search and the type of visual

Extensions

2a. If no relevant results were found from the search, no visualization should appear and a message such as “No results found” should display instead

Special Requirements

-Visualizations should scale well on many different monitor sizes

Technology and data variation List:

2a. The visualization can be viewed in a new browser tab or window

3a. Additional info should be suitable to the visualization as well as the search, but specifics will be dependent on other implementation details (what filters there will be for example)

Frequency of Occurrence:

Anytime the user decides to search, but visualizations only appear when there are results returned by the system

Use Case 8: Enable/Disable sound

Primary Actor:

Searcher (user): A researcher or programmer interested in a GitHub repository

Stakeholders and interests:

Searcher - wants sound because it offers a unique type of feedback

Preconditions:

- Searching is working

- UI is behaving in expected ways

Success Guarantees:

When the user has selected the sound options he or she wants to hear, they should hear them when they perform the relevant action

Main Success Scenario:

1. The user goes to a settings page

2. The user selects the sounds he wants to be played

3. The user performs actions that cause the sounds to trigger based on his selection in step 2

Extensions:

1a. If the user finishes selecting settings, they can navigate to other pages of the app

3a. If the user searches with invalid input, rejection sound should play

3b. The user may conduct a search and hear “audibilization” of result data as an option

3c. If a sound option is disabled, sound does not play when the user performs the associated action

Special Requirements:

-Whatever implementation means used for sound should be very compatible with modern browsers

-Sounds should be consistent in overall loudness, with none being too loud or quite in comparison to one another

Technology and data variation list:

3a. For more reusability, there should be many types of sound files supported (ex: .mp3, .wav, .aif, .midi), however more file support should not be of greater value than efficient playback (ex: .mp3, .midi)

Frequency of Occurrence:

Whenever the user performs an action that is associated with a sound and they have that sound option selected

Use Case 9: Refine existing query and update results

Description: The user can optimize the query to make it faster or change what the query searches to change the results from the query

Primary Actor

A researcher or programmer interested in a GitHub repository

Stakeholders

The searcher - Needs faster results or more specific results.

The people the searcher is searching on behalf of (ex: their team)

Preconditions

A query must already exist in the database.

Success Guarantee

The speed of the user receives the results is faster. The user receives new results.

Main Success Scenario

1. The user selects filters to show results by (ex: show only commits from the past week or by X contributor)

2. After clicking submit, the system finds all results for the query matching both the keyword and filters.

3. The system displays these results to the user

4. The user refines the searched query.

Extensions

2a. the repository doesn't exist:

1. The user inputs the URL of the desired Github repository

2. The user then inputs a query.

3. After clicking "submit", the system attempts to search the repository, but fails due to the repository not being found.

4. The system generates an error and displays it to the user

3a. the query was not found: An error message should display that the filters don't match any current query.

Special Requirements:

-Results must be returned quickly and there should be a variety of relevant filters

-Filters should not make the screen looked cluttered and difficult for mobile browser users

-A message should display showing the success of the user's actions.

Technology and Data Variation List

1a. Filtering may be done in a few ways, but chiefly either by filtering as it searches or first finding all the textual matches, then filtering

1b. Filtering may be done in JavaScript, PHP, or other applicable languages

2a. Searching may use JavaScript, PHP, or other applicable languages.

2b. Searching may be done for a query string or keywords

Frequency of Occurrence

Not often, just whenever a user wants to refine a query.

Use Case 10: Sort Results

Description: The user is able to filter the results by specific key words.

Primary Actor

A researcher or programmer interested in a GitHub repository

Stakeholders

The searcher – Wants specifics from search results

The people the searcher is searching on behalf of (ex: their team)

Preconditions

The output of the search must have results that match the filters.

Success Guarantee

The user is able to sort the results and is able to find what he or she needs faster due to sorting.

Main Success Scenario

1. The user inputs the URL of the desired GitHub repository
2. The user types in the query
3. After clicking submit, the system finds all results for the query matching the key word
4. The system displays these results to the user
5. The user selects filters to sort the results.

Extensions

3a. the repository doesn't exist:

1. The user inputs the URL of the desired Github repository

2. The user then inputs a query.

3. After clicking "submit", the system attempts to search the repository, but fails due to the repository not being found.

4. The system generates an error and displays it to the user

4b. No results were found:

1 . The user inputs the url of the desired Github repository

2. The user then inputs a query

3. After clicking submit, the system attempts to search the repository

but does not find any results

4. The system generates an error and displays it to the user

4a. the system suggest alternate search filters based on what it did find

Special Requirements:

-Results must be returned quickly and there should be a variety of relevant filters

-Filters should not make the screen looked cluttered and difficult for mobile browser users

-A message should display showing the success of the user's actions.

Technology and Data Variation List

3a. Searching may use JavaScript, PHP, or other applicable languages.

3b. Searching may be done for a query string or keywords

5a. Filtering must be done after the search is over.

5b. Filtering may be done in JavaScript, PHP, or other applicable languages

Frequency of Occurrence

Whenever the user wants to make the search more efficient by sorting the results

Use Case 11: Display/hide search history log file.

Description: The user is able to hide the history log file while typing in a search.

Primary Actor

A researcher or programmer interested in a GitHub repository

Stakeholders

The searcher – Does/doesn't want the past results to show while searching.

Preconditions

The log file is not empty.

Success Guarantee

The log history is displayed how the user desires.

Main Success Scenario

1. The user begins a new search.
2. The history log will start to show while the user is inputting the search.
3. An option to hide the file will appear above the first result in the log file.
4. The log file will disappear when the option is clicked or reappear if it is already hidden.
5. The user completes the search by using one of the past reaches or by searching something new.

Extensions

2a. the log file doesn't exist:

1. No history will show and the search will continue
2. The option to hide the log file will still be present.

Special Requirements:

Technology and Data Variation List

1a. Searching may use JavaScript, PHP, or other applicable languages.

1b. Searching may be done for a query string or keywords

2a. The log file will be help in a table of past results.

Frequency of Occurrence

Very frequently. Every time a search is done the function should apply.

Use Case 12: Clear the log file

Description: The user is able to remove all past searches from the log file.

Primary Actor

A researcher or programmer interested in a GitHub repository

Stakeholders

The searcher – Wants to remove all previous searches from the history log file.

Preconditions

The log file is not empty.

Success Guarantee

The log history will be empty and the data will not be retrievable after clearing the file.

Main Success Scenario

1. The user begins a new search.

2. The history log will start to show while the user is inputting the search.

3. An option to clear the log file will appear below all results from the file.

4. The option to clear the file is selected and the file is cleared.

Extensions

2a. the log file is empty:

1. No history will show and the search will continue
2. The option to hide the log file will still be present.

3a. A message to confirm the users request will appear before clearing the file. This way, the user is less likely to accidentally clear the history.

Special Requirements:

Technology and Data Variation List

1a. Searching may use JavaScript, PHP, or other applicable languages.

1b. Searching may be done for a query string or keywords

2a. The log file will be help in a table of past results.

Frequency of Occurrence

Whenever the user wants to clear the history in the log file.

Use case 13: Create new query from attribute of current result

Primary actor:

Service user

Stakeholders and interests:

Service user: wants to search along an attribute seen in current results

Preconditions:

The service user is viewing a non-empty result set.

Success guarantee:

The service displays the results of performing a query on an attribute seen in the original result set.

Main success scenario:

1. The user hovers over an attribute of a displayed result.
2. The service displays a context menu giving options for what to do with the hovered-over attribute.
3. The user selects the "Search on this attribute" option.
4. The service queries on the hovered attribute and displays the results.

Extensions:

None.

Special requirements:

The user must be on a device that supports hovering.

Technology and data variation list:

None.

Frequency of occurrence:

Once per displayed result set.

Use case 14: Export results to file

Primary actor:

Service user

Stakeholders and interests:

Service user: wants to show results to others or store them for later viewing

(Possible) others: want to view exported results

Preconditions:

The service user is viewing a result set.

Success guarantee:

A plaintext file containing the exported results is created and saved to the user's computer.

Main success scenario:

1. The user selects the "Export to File" option from the results page.
2. The service creates a plaintext file containing the results.
3. The service opens a dialog windox asking the user to select a location to which to save the file and a name for the file.
4. The user selects a save location and a file name.
5. The user selects the "Save" option from the save dialog window.
6. The service saves the export file to the selected location.

Extensions:

5a: The user selects to "Cancel" in the save dialog window rather than "Save"

1. The service closes the save dialog window and clears the file it created.

6a: Invalid file name or save location

1. The service informs the user that their selected file name or save location is invalid, and prompts them to change it.

Special requirements:

1. The service should be able to save result sets of any size, even large ones or empty ones.

Technology and data variation list:

None.

Frequency of occurrence:

As often as the user desires—no benefit will be gained after the first occurrence per result set.

Use case 15: Import results from file

Primary actor:

Service user

Stakeholders and interests:

Service user: Wants to see results of a previous search

Preconditions:

The service user has previously exported a result set, and is able to access it.

Success guarantee:

The result set represented by the file on the user's computer is accurately displayed as though it had just been searched.

Main success scenario:

1. The user selects the "Import" option.
2. The service opens a file selection dialog window.
3. The user navigates to a file containing a previously-exported result set and selects it.
4. The user selects the "Open" option from the file selection dialog window.
5. The service imports the file and creates a result set containing the file's data.
6. The service displays the created result set.

Extensions:

4a: The user selects to "Cancel" in the file selection dialog window rather than "Open"

1. The service closes the file selection dialog window.

5a: File is selected with invalid data

1. The service informs the user that the selected file does not contain a valid result set.

Special requirements:

1. The service should be able to load result sets of any size, even large ones or empty ones.

Technology and data variation list:

None

Frequency of occurrence:

As often as the user desires.

Use case 16: Re-search query from imported file

Primary actor:

Service user

Stakeholders and interests:

Service user: wants to see updated results for a previously-saved query

Preconditions:

The service user is displaying the contents of an imported result set.

Success guarantee:

The service displays the results of executing the query of the imported result set on the most recent data.

Main success scenario:

1. The user selects the "Autofill query" option on the results page.
2. The service populates the query text field with the query that produced the displayed result set.
3. The user selects the "Query" option.
4. The service displays the results of running the query on the most recent data.

Extensions:

None.

Special requirements:

None.

Technology and data variation list:

None.

Frequency of occurrence:

Whenever the user desires—no benefit after the first occurrence per imported file.