@push.rocks/searchquery

A module for parsing and handling search queries with debounce and customization options.

- readme.md for @push.rocks/searchquery

readme.md for @push.rocks/searchquery

a module for dealing with searchqueries

Install

To use `@push.rocks/searchquery` in your project, you need to install it via npm. You can do so by running the following command in your terminal:

```
npm install @push.rocks/searchquery --save
```

This will add `@push.rocks/searchquery` to your project's dependencies and allow you to start using it in your code.

Usage

`@push.rocks/searchquery` is designed to enhance the handling of search queries within your application. It leverages the power of `search-query-parser` and `@pushrocks/smartrx` for parsing and reactive management of search queries. In this guide, we'll explore how to effectively use this module to its full potential.

Initializing SearchQuery

First, let's start by importing and initializing the `SearchQuery` class from `@push.rocks/searchquery`. You will need to provide it with appropriate options.

```
import { SearchQuery } from '@push.rocks/searchquery';

const searchQueryOptions = {
  debounceMs: 300, // Milliseconds to debounce the search input
  searchParserOptions: { // Options passed to 'search-query-parser'
    keywords: ['tag', 'author'],
```

```
    ranges: ['date'],
    tokenize: true,
    alwaysArray: true,
  },
};
const mySearchQuery = new SearchQuery(searchQueryOptions);
```

The `debounceMs` option allows you to set a debounce time for processing the search query, which can help in reducing the frequency of search operation execution, making it more efficient especially for applications that perform real-time search query processing.

The `searchParserOptions` are directly passed to the `search-query-parser`. In this example, we specify that we want to parse for keywords such as `tag` and `author`, and for range queries like `date`. We also configure the parser to always return the results as an array and to tokenize the input.

Feeding the Search String

To process a search string, you'll need to feed it into your `SearchQuery` instance. Typically, you'd do this in response to a user action, such as typing in a search input on a UI. Below is an example of how you might do this:

```
// Placeholder function to simulate user input
async function simulateUserInput(input: string) {
  // Here you would feed the user input to the search query
  // In a real application, this might be triggered by an event listener on a search input
field
}

// Example usage
simulateUserInput('tag:important author:John');
```

Subscribing to Search Query Changes

`SearchQuery` utilizes reactive programming principles by using a Subject from the `@pushrocks/smartrx` package. You can subscribe to the search query's changes and get notified whenever the search query is updated. This is particularly useful for implementing real-time search features where the search results are dynamically updated as the user types.

```
mySearchQuery.querySubject.subscribe({
  next: (searchQueryResult) => {
    console.log('New search query result:', searchQueryResult);
    // Here you would typically update the search results in your UI based on the new
searchQueryResult
  },
  error: (err) => {
    console.error('Something went wrong with the search query subscription:', err);
  },
});
```

This setup allows you to build highly interactive and responsive search experiences in your web application while keeping the complexity of handling search queries and results manageable.

Advanced Usage

`@push.rocks/searchquery` is designed to be flexible and extensible. You can further customize its behavior by exploring additional options available in the `search-query-parser` library and integrating more features from the `@pushrocks/smartrx` package for advanced reactive programming patterns.

Remember to explore the APIs and documentation of these underlying libraries to fully leverage the power of `@push.rocks/searchquery` in your projects.

Conclusion

`@push.rocks/searchquery` offers a powerful and flexible way to handle search queries in your application. By combining the capabilities of `search-query-parser` for parsing complex queries and `@pushrocks/smartrx` for reactive programming, it provides a robust solution for managing and responding to user-generated search queries in real time. Whether you are building a search-intensive application or just need a sophisticated mechanism to deal with search queries, `@push.rocks/searchquery` is a valuable tool to have in your development toolkit.

License and Legal Information

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH and are not included within the scope of the MIT license granted herein. Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines, and any usage must be approved in writing by Task Venture Capital GmbH.

Company Information

Task Venture Capital GmbH
Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.