readme.md for @push.rocks/smartrouter

@push.rocks/smartrouter

A JavaScript library providing routing capabilities for web applications with support for path parameters, query parameters, and programmatic navigation

Install

To install `@push.rocks/smartrouter`, run the following command in your project directory:

```
pnpm install @push.rocks/smartrouter --save
```

This will add `@push.rocks/smartrouter` to your project's dependencies and enable you to use it within your application.

Usage

`@push.rocks/smartrouter` provides a versatile routing solution for websites, leveraging modern Web APIs to manipulate browser history and handle URL paths intelligently. Below are examples demonstrating how to use `@push.rocks/smartrouter` effectively in a TypeScript project, taking advantage of ESM syntax.

Basic Setup

First, ensure you've installed the package as described in the Install section above. Next, import `SmartRouter` from `@push.rocks/smartrouter` in your application's entry point or any module where routing is required.

```
import { SmartRouter } from '@push.rocks/smartrouter';
```

Initialize the Router

Create an instance of `SmartRouter` and optionally provide configuration options. If your application requires debugging information, `debug` can be set to `true`.

```
const router = new SmartRouter({
  debug: true, // Enables debugging. Optional and false by default.
});
```

Define Routes

Define your application routes using the `on` method, which takes a URL pattern and a handler function. The handler function will be called when the application navigates to a URL that matches the pattern.

```
router.on('/home', async (routeInfo) => {
  console.log('Home route accessed', routeInfo);
  // Handle the home route
  // You can load a page component, change document title, etc.
});

router.on('/about', async (routeInfo) => {
  console.log('About route accessed', routeInfo);
  // Handle the about route
});
```

Path Parameters

`@push.rocks/smartrouter` supports dynamic path parameters. Define path parameters within your route strings using the `:` prefix, and access their values from the `routeInfo.params` object in your handler function.

```
router.on('/user/:userId', async (routeInfo) => {
  console.log(`User Profile for ID: ${routeInfo.params.userId}`, routeInfo);
  // Load and display user profile based on userId
});
```

Query Parameters

Query parameters can be accessed through the `routeInfo.queryParams` object, making it easy to handle complex routing scenarios with optional parameters.

```
router.on('/search', async (routeInfo) => {
  console.log('Search Query:', routeInfo.queryParams.query);
  // Perform a search operation using the provided query parameter
});
```

Programmatic Navigation

Navigate programmatically using the `pushUrl` method. This method allows you to change the URL without reloading the page, and optionally pass state information.

```
// Navigate to the about page
router.pushUrl('/about');

// Navigate to a user profile with URL parameters
router.pushUrl('/user/12345');

// Navigate with state data
router.pushUrl('/dashboard', { previousPage: 'home' });
```

Managing Query Parameters

`@push.rocks/smartrouter` provides methods for managing URL query parameters, enabling dynamic URL manipulation for filter settings, pagination, and other use cases.

```
// Set a query parameter (replace by default)
router.queryParams.setQueryParam('key', 'value');

// Set a query parameter with push (adds to history)
router.queryParams.setQueryParam('key', 'value', 'push');

// Get a query parameter
const value = router.queryParams.getQueryParam('key');
```

```
// Get all query parameters as an object
const allParams = router.queryParams.getAllAsObject();

// Delete a query parameter
router.queryParams.deleteQueryParam('key');

// Delete with push to history
router.queryParams.deleteQueryParam('key', 'push');
```

Sub-Routers

Create sub-routers with a specific base path for modular routing:

```
// Create a sub-router for admin routes
const adminRouter = router.createSubRouter('/admin');

// Routes will be prefixed with /admin
adminRouter.on('/dashboard', async (routeInfo) => {
  // This handles /admin/dashboard
});

adminRouter.on('/users', async (routeInfo) => {
  // This handles /admin/users
});
```

Route Management

The `on` method returns a function that can be used to remove the route:

```
// Add a route
const removeRoute = router.on('/temporary', async (routeInfo) => {
  console.log('Temporary route accessed');
});

// Later, remove the route
removeRoute();
```

Cleanup

When you're done with a router instance, clean it up properly:

```
// Destroy the router, removing all event listeners and routes
router.destroy();
```

This module enables complex routing scenarios, simplifying the handling of navigational logic in modern web applications. By leveraging `@push.rocks/smartrouter`, developers can implement detailed routing mechanisms, manipulate browser history thoughtfully, and maintain cleaner URL structures, enhancing the user experience and making web apps more accessible.

License and Legal Information

Trademarks

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.

Revision #2
Created 2026-03-28 11:11:44 UTC by foss.global Team
Updated 2026-03-28 11:41:19 UTC by foss.global Team