readme.md for @push.rocks/webjwt

a package to handle jwt in the web

Install

To install `@push.rocks/webjwt`, ensure you have Node.js and npm installed. Then run the following command in your terminal:

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

This command installs `@push.rocks/webjwt` and adds it to your project's `package.json` dependencies.

Usage

This module provides a streamlined way to handle JSON Web Tokens (JWT) within web contexts, leveraging TypeScript for type safety and developer ergonomics. Let's dive into how you can utilize `@push.rocks/webjwt` in your project.

Importing the Module

First, import the necessary functions from the module in your TypeScript file:

```
import { getDataFromJwtString } from '@push.rocks/webjwt';
```

Decoding JWT

`@push.rocks/webjwt` simplifies the process of decoding JWTs to extract data payloads without dealing with the intricacies of token verification or parsing manually. Here's a basic example of decoding a JWT string to extract its payload:

```
// Example JWT string (Note: This is just an example and not a valid token)
const jwtString: string =
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvZSBEb2UiLCJpYXQi
OjE1MTYyMzkwMjJ9.abcd1234';

// Decode the JWT and extract the data payload
const decodedData = getDataFromJwtString<{ sub: string; name: string; iat: number
}>(jwtString);

// Log the decoded data
console.log(decodedData);
// Output would be an object containing the sub, name, and iat fields from the JWT payload
```

Handling Custom JWT Payloads

`@push.rocks/webjwt` is designed to be flexible, allowing you to define the structure of your JWT payload as needed. The generic `<T>` in the `getDataFromJwtString` function enables you to specify the expected shape of your payload, ensuring type safety:

```
// Define a custom payload structure
interface MyCustomPayload {
  userId: string;
  permissions: string[];
  issuedAt: number;
}

// Example JWT string with a custom payload
const customJwtString: string = 'your.jwt.string.here';

// Decode the JWT with a custom payload
const customDecodedData = getDataFromJwtString<MyCustomPayload>(customJwtString);

// Accessing the custom payload data with full TypeScript support
console.log(`User ID: ${customDecodedData.userId}`);
console.log(`Permissions: ${customDecodedData.permissions.join(', ')}`);
```

Real-World Scenario: User Authentication

In a web application, you might want to use JWTs for user authentication. After the user logs in, you receive a JWT from your authentication server. You can then decode this token on the client side to obtain user-specific information without making additional requests to the server:

```
// This example assumes you have a function to get the auth token, e.g., from local storage or a cookie
const authToken: string = getAuthTokenFromStorage();

// Define the expected structure of your authentication payload
interface AuthPayload {
  userId: string;
  userName: string;
  roles: string[];
}

// Decode the authentication token
const authInfo = getDataFromJwtString<AuthPayload>(authToken);

// Use the decoded information, for example, to customize the UI
console.log(`Welcome, ${authInfo.userName}!`);
// Potentially, check for roles to display certain UI elements
if (authInfo.roles.includes('admin')) {
  console.log('Displaying admin panel...');
}
```

`@push.rocks/webjwt` enables efficient, type-safe handling of JWTs in your TypeScript web projects, simplifying the process of decoding and utilizing token payloads according to your application's requirements.

License and Legal Information

This repository contains open-source code that is licensed under the MIT License. A copy of the MIT License can be found in the license file within this repository.

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.

Revision #3
Created 2026-03-28 11:13:29 UTC by foss.global Team
Updated 2026-03-28 12:20:15 UTC by foss.global Team