# @push.rocks/smartjwt

A JavaScript package for creating and verifying JWTs with strong typing support.

- readme.md for @push.rocks/smartjwt
- changelog.md for @push.rocks/smartjwt

# readme.md for @push.rocks/smartjwt

# @push.rocks/smartjwt

a package for handling jwt

# Install

To install @push.rocks/smartjwt, use npm or yarn as follows:

```
npm install @push.rocks/smartjwt --save
# or with yarn
yarn add @push.rocks/smartjwt
```

Make sure you have Node.js installed on your machine. This library uses modern JavaScript features and requires Node.js version 10.x.x or higher.

# Usage

This section illustrates how to use the `@push.rocks/smartjwt` package to handle JSON Web Tokens (JWT) in your TypeScript projects with practical and comprehensive examples. Before diving into the scenarios, ensure you have the package installed and are comfortable with TypeScript and async/await syntax.

## Initializing and Creating a New Key Pair

To utilize `smartjwt` for creating and verifying JWTs, you first need to instantiate `SmartJwt` class and generate or set a key pair (a private and a public key). Let's start by initializing and creating a new key pair:

```
import { SmartJwt } from '@push.rocks/smartjwt';

async function setup() {
  const smartJwt = new SmartJwt();
  await smartJwt.createNewKeyPair();
  console.log('Key pair created successfully');
}
setup();
```

# Creating a JWT

Once you have your key pair ready, you can create JWTs by supplying a payload. The payload is the data you want to encode in your JWT. Here's an example of how to create a JWT:

```
interface MyPayload {
  user_id: number;
  username: string;
}

async function createJwt() {
  const smartJwt = new SmartJwt<MyPayload>();
  await smartJwt.createNewKeyPair();
  const myPayload = {
    user_id: 123,
    username: 'example_username'
  };
  const jwt = await smartJwt.createJWT(myPayload);
  console.log(jwt);
}
createJwt();
```

# Verifying a JWT and Extracting the Payload

Verifying a JWT is crucial for authentication and authorization processes in applications. When you receive a JWT, you need to verify its integrity and authenticity before trusting the contained information. Here’s how to verify a JWT and extract its payload:

```
async function verifyJwt(jwt: string) {
  const smartJwt = new SmartJwt<MyPayload>();
  // In a real application, you should set the public key from a trusted source.
  smartJwt.setPublicPemKeyForVerification('<Your_Public_Key>');

  try {
    const payload = await smartJwt.verifyJWTAndGetData(jwt);
    console.log('JWT verified successfully:', payload);
  } catch (error) {
    console.error('Failed to verify JWT:', error);
  }
}
```

# Handling Public and Private Keys

In scenarios where you have existing keys or receive them from an external source, `smartjwt` allows setting the public and private keys directly instead of generating new ones. Here is an example:

```
async function setExistingKeys() {
  const smartJwt = new SmartJwt<MyPayload>();
  const privateKeyString = '<Your_Private_Key_PEM_String>';
  const publicKeyString = '<Your_Public_Key_PEM_String>';

  smartJwt.setPrivateKeyFromPemString(privateKeyString);
  smartJwt.setPublicKeyFromPemString(publicKeyString);
}
```

# Exporting and Importing Key Pairs

You may need to store your key pairs securely or share them across different parts of your application or with other services securely. `smartjwt` offers a convenient way to export and import key pairs as JSON:

```
async function exportAndImportKeyPair() {
  const smartJwt = new SmartJwt();
  await smartJwt.createNewKeyPair();
  const keyPairJson = smartJwt.getKeyPairAsJson();
```

```
  console.log('Exported Key Pair:', keyPairJson);

  const newSmartJwt = new SmartJwt();
  newSmartJwt.setKeyPairAsJson(keyPairJson);
  console.log('Imported Key Pair Successfully');
}
exportAndImportKeyPair();
```

# Complete Scenario: Signing and Verifying a JWT

Bringing it all together, here is a complete scenario where a JWT is created, signed, and later verified:

```
async function completeScenario() {
  // Creating a new JWT
  const smartJwt = new SmartJwt<MyPayload>();
  await smartJwt.createNewKeyPair();
  const jwt = await smartJwt.createJWT({ user_id: 123, username: 'exampleuser' });
  console.log('Created JWT:', jwt);

  // Verifying the JWT in another instance or part of the application
  const verifier = new SmartJwt<MyPayload>();
  verifier.setPublicPemKeyForVerification(smartJwt.publicKey.toPemString());
  const verifiedPayload = await verifier.verifyJWTAndGetData(jwt);
  console.log('Verified Payload:', verifiedPayload);
}
completeScenario();
```

By following these examples, you can effectively handle JWT creation, signing, verification, and key management in your TypeScript projects using the `@push.rocks/smartjwt` package. Always ensure to manage your keys securely and avoid exposing sensitive information.

For further information and advanced features, consult the project's documentation and source code.

# 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.

# changelog.md for @push.rocks/smartjwt

## 2024-09-05 - 2.2.1 - fix(core)

Add --allowimplicitany flag to build script

- Updated the build script in package.json to include the --allowimplicitany flag for TypeScript compilation.

## 2024-09-05 - 2.2.0 - feat(smartjwt)

Add nested JWT functionality

- Introduce `isNestedJwt` method to check if an object contains a nested JWT
- Implement `verifyNestedJwt` method to validate nested JWTs
- Add `createNestedJwt` method to create nested JWTs with given payload
- Include `nestedJwtGuard` for guarding nested JWT objects

## 2024-08-26 - 2.1.0 - feat(core)

Enhanced JWT handling and key management

- Introduced the `IObjectWithJwt` type for improved type constraints.
- Added support for smartguard to create object guards with JWT validation.
- Upgraded dependencies to latest versions.

## 2024-05-29 - 2.0.4 - Maintenance

Minor updates and configuration changes.

- Updated project description.
- Modified tsconfig settings.
- Updated npmextra.json with githost.

# 2024-02-13 - 2.0.3 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2023-07-11 - 2.0.2 - Organizational Update

Switched to the new organizational scheme.

- Adjusted project structure to align with new organizational policies.

# 2022-12-22 - 2.0.1 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2022-10-24 - 2.0.0 - Major Update

Significant updates including breaking changes.

- **Breaking Change:** Switched project to ECMAScript modules.

# 2021-09-22 - 1.0.14 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2021-02-20 - 1.0.13 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2021-02-09 - 1.0.12 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2021-02-09 - 1.0.11 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2020-03-28 - 1.0.10 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2019-10-01 - 1.0.9 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2019-10-01 - 1.0.8 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2019-10-01 - 1.0.7 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2019-10-01 - 1.0.6 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2019-10-01 - 1.0.5 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2019-10-01 - 1.0.4 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2019-02-23 - 1.0.3 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2019-02-23 - 1.0.2 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.

# 2019-02-23 - 1.0.1 - Bug Fixes

Critical fixes in core functionalities.

- Fixed core issues.