@apiclient.xyz/medium

Documentation for @apiclient.xyz/medium

- readme.md for @apiclient.xyz/medium
- changelog.md for @apiclient.xyz/medium

readme.md for @apiclient.xyz/medium

An unofficial Medium.com API package that allows interaction with Medium's features using Node.js.

Install

To install the `@apiclient.xyz/medium` package, ensure you have Node.js and npm installed on your system. Then, run the following command in your terminal:

```
npm install @apiclient.xyz/medium
```

Usage

This guide will walk you through how to use the `@apiclient.xyz/medium` package to interact with the Medium API. This includes steps to initialize your Medium account, retrieve account and publication information, and create new posts. The examples will use ESM syntax and TypeScript to ensure you have the best experience with code intelligence and type safety.

1. Importing Required Classes

Start by importing the necessary classes from the `@apiclient.xyz/medium` package.

```
import { MediumAccount, MediumPublication, MediumPost, IPostData } from
'@apiclient.xyz/medium';
```

2. Initializing a Medium Account

To use the Medium API, you'll need to initialize a `MediumAccount` object with your access token.

```
const mediumAccount = new MediumAccount('YOUR_ACCESS_TOKEN_HERE');

// Wait for the account to be ready
await mediumAccount.readyDeferred.promise;
```

3. Fetching Account Information

Retrieve account information such as ID, username, URL, and image URL.

```
const accountInfo = await mediumAccount.getAccountInfo();
console.log(accountInfo);
```

4. Working with Publications

You can interact with publications associated with your account by fetching all publications, getting publications you own, and fetching publications by their names.

Fetch All Publications

```
const publications = await mediumAccount.getAllPublications();
publications.forEach(pub => console.log(pub.name, pub.id));
```

Fetch Publications Owned By You

```
const ownPublications = await mediumAccount.getOwnPublications();
ownPublications.forEach(pub => console.log(pub.name, pub.id));
```

Fetch a Publication by Name

```
const pubName = 'Your Publication Name';
const specificPublication = await mediumAccount.getPublicationByName(pubName);
if (specificPublication) {
  console.log(specificPublication);
} else {
  console.log(`Publication with name ${pubName} not found.`);
}
```

5. Creating and Managing Posts

To create posts within a publication, you need to define the post data and use the publication object to create a post.

Define Post Data

```
const newPostData: IPostData = {
  title: 'My First Post',
  contentFormat: 'markdown',
  content: '# Hello World\nThis is my first post!',
  canonicalUrl: 'https://yourcoolsite.com/first-post',
  tags: ['example', 'first', 'post'],
  publishStatus: 'draft'
};
```

Create a New Post

```
if (specificPublication) {
  const newPost = await specificPublication.createPost(newPostData);
  console.log(newPost);
}
```

6. Error Handling

Effective error handling is essential when interacting with APIs. Make sure to wrap your asynchronous code in try/catch blocks to handle any rejected promises.

```
try {
  const result = await mediumAccount.getAccountInfo();
  console.log(result);
} catch (error) {
  console.error('Failed to fetch account information:', error);
}
```

Complete Example

Here is a complete example script that demonstrates the major functionalities of the `@apiclient.xyz/medium` package.

```
import { MediumAccount, IPostData } from '@apiclient.xyz/medium';

async function run() {
  const mediumAccount = new MediumAccount('YOUR_ACCESS_TOKEN_HERE');
  await mediumAccount.readyDeferred.promise;

  // Fetch and display account information
  const accountInfo = await mediumAccount.getAccountInfo();
  console.log('Account Info:', accountInfo);

  // Fetch and display all publications
  const publications = await mediumAccount.getAllPublications();
  publications.forEach(pub => console.log('Publication Name:', pub.name, 'ID:', pub.id));

  // Fetch and display publications owned by the user
  const ownPublications = await mediumAccount.getOwnPublications();
  ownPublications.forEach(pub => console.log('Own Publication Name:', pub.name, 'ID:',
pub.id));

  // Fetch a specific publication by name
  const pubName = 'Your Publication Name';
  const specificPublication = await mediumAccount.getPublicationByName(pubName);
  if (specificPublication) {
    console.log('Specific Publication:', specificPublication);

    // Define new post data
    const newPostData: IPostData = {
      title: 'My First Post',
      contentFormat: 'markdown',
      content: '# Hello World\nThis is my first post!',
      canonicalUrl: 'https://yourcoolsite.com/first-post',
      tags: ['example', 'first', 'post'],
      publishStatus: 'draft'
    };

    // Create a new post
    const newPost = await specificPublication.createPost(newPostData);
```

```
    console.log('New Post:', newPost);
  } else {
    console.log(`Publication with name ${pubName} not found.`);
  }
}

run().catch(error => {
  console.error('Error:', error);
});
```

Fetching Publication Contributors

If you want to see the contributors of a publication, you can fetch them and display the relevant details:

```
const fetchContributors = async (publicationId: string) => {
  const response = await mediumAccount.request(`/publications/${publicationId}/contributors`,
'GET');
  const contributors: { publicationId: string; userId: string; role: string; }[] =
response.data;
  contributors.forEach(contributor => {
    console.log(`User ID: ${contributor.userId}, Role: ${contributor.role}`);
  });
};

await fetchContributors('SamplePublicationId');
```

Advanced Post Management

Depending on the use case, you might need to fetch posts by specific criteria, update posts, or delete them. For this example, let's assume updating and deleting posts functions are supported by the Medium API, even though those capabilities aren't documented here.

Fetch a Post by ID

```
const fetchPostById = async (publication: MediumPublication, postId: string) => {
  const response = await
mediumAccount.request(`/publications/${publication.id}/posts/${postId}`, 'GET');
  const post = response.data;
```

```
  console.log(post);
};

await fetchPostById(specificPublication, 'SamplePostId');
```

Update a Post

Let's assume we have an endpoint for updating posts. The method might look like this:

```
const updatePost = async (publication: MediumPublication, postId: string, newData:
Partial<IPostData>) => {
  const response = await
mediumAccount.request(`/publications/${publication.id}/posts/${postId}`, 'PUT', newData);
  console.log('Updated Post:', response.data);
};

const updatedData: Partial<IPostData> = {
  title: 'Updated Title',
  content: '# Updated Content'
};

await updatePost(specificPublication, 'SamplePostId', updatedData);
```

Delete a Post

Similarly, assuming an endpoint exists for deleting posts:

```
const deletePost = async (publication: MediumPublication, postId: string) => {
  const response = await
mediumAccount.request(`/publications/${publication.id}/posts/${postId}`, 'DELETE');
  console.log('Deleted Post', response.statusCode);
};

await deletePost(specificPublication, 'SamplePostId');
```

By following this guide, you should be able to leverage the extensive features provided by the `@apiclient.xyz/medium` package. From account management to creating and managing posts within publications, this guide offers the complete scenarios you need in order to interact effectively with the Medium API using Node.js and TypeScript.

Extensions

Consider creating more complex interactions by combining different endpoint requests. For example:

- Creating a scheduled post by integrating with a scheduling library like node-schedule.
- Fetching and analyzing post stats if such endpoints are provided by Medium.
- Automating cross-platform publication by combining Medium's API with other platforms' APIs.

The flexibility and power of Node.js along with the typed safety of TypeScript provide a robust environment for developing applications that interact with web APIs efficiently and effectively.

Start by running the complete example script to get familiar with the basic functionalities. Build upon these examples to create feature-rich applications that make extensive use of Medium's capabilities.

Always ensure to consult Medium's API documentation for any updates or additional features that may be available for further enhancement and optimization of your integrations. undefined

changelog.md for @apiclient.xyz/medium

2024-07-02 - 1.0.7 - fix(metadata)

Update project metadata and readme for new scope and descriptions

- Updated package name and description in package.json
- Updated repository and scope information in npmextra.json
- Expanded readme with detailed usage instructions and examples

2024-07-01 - 1.0.6 - fix(medium)

Fix various bugs and improve async handling.

- Update @push.rocks/qenv dependency to ^6.0.5
- Fix bug in test: await getEnvVarOnDemand
- Improve error handling in getAccountInfo
- Simplify fetching publications by refactoring methods in MediumPublication and MediumAccount classes

2024-07-01 - 1.0.5 - fix(core)

Fixed module name inconsistencies and documentation links

- Corrected module names in package.json and VSCode launch configuration.
- Updated npm package name from '@pushrocks' to '@push.rocks' in readme.md and package.json.
- Fixed test imports and improved test scripts.
- Added updated TypeScript configuration file tsconfig.json.

2020-11-17 - 1.0.4 - No significant changes

No significant changes made in this version update.

2020-11-16 - 1.0.3 to 1.0.1 - Core Updates

Routine maintenance and updates in the core functionality.

- fix(core): update