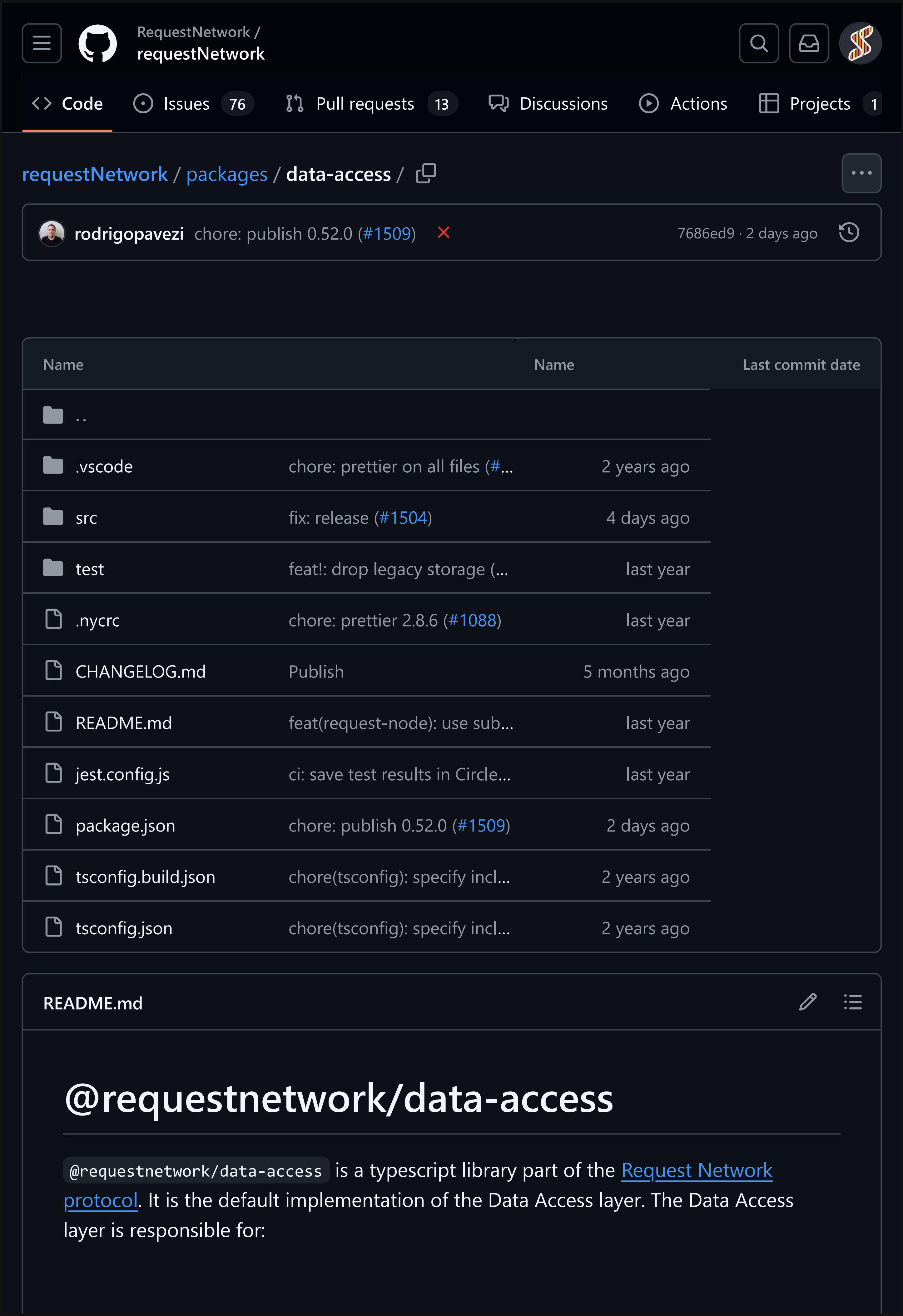

requestNetwork / packages / data-access /

rodrigopavezi chore: publish 0.52.0 (#1509) ✗ 7686ed9 · 2 days ago

Name	Name	Last commit date
..		
.vscode	chore: prettier on all files (#...	2 years ago
src	fix: release (#1504)	4 days ago
test	feat!: drop legacy storage (...	last year
.nycrc	chore: prettier 2.8.6 (#1088)	last year
CHANGELOG.md	Publish	5 months ago
README.md	feat(request-node): use sub...	last year
jest.config.js	ci: save test results in Circle...	last year
package.json	chore: publish 0.52.0 (#1509)	2 days ago
tsconfig.build.json	chore(tsconfig): specify incl...	2 years ago
tsconfig.json	chore(tsconfig): specify incl...	2 years ago

README.md

@requestnetwork/data-access

`@requestnetwork/data-access` is a typescript library part of the Request Network protocol. It is the default implementation of the Data Access layer. The Data Access layer is responsible for:

- Indexing transactions to allow retrieval. In the context of the Request Protocol, examples of transactions are "create a request", "accept a request", "change the expected amount of a request"
- Batching them into blocks to save on cost
- Accessing transactions through a local cache
- Synchronizing with the storage

Transactions indexing (Channel & Topics)

The indexing of the transactions is made by two mechanisms:

- Channel
- Topics

There are two ways of getting transactions:

- by channel id: this gets all the transactions of the channel
- by topic: this gets all the transactions of the channels indexed by this topic

In the context of requests, all the transactions of a channel are the actions of a request. The channel id is the request id. It is possible to get a request from the request id thanks to this. The topics are for example, the identities of the stakeholders of the request. So, you can retrieve all the requests (channels) of an identity.

Installation

```
npm install @requestnetwork/data-access
```

Usage

Persist a Transaction

```
import DataAccess from '@requestnetwork/data-access';
import { DataAccessTypes, SignatureTypes, StorageTypes } from '@requestnetw

// Any implementation of Storage layer, @requestnetwork/ethereum-storage fo
const storage: StorageTypes.IStorage;

const dataAccess = new DataAccess(storage);
await dataAccess.initialize();
```

```
const transactionData = JSON.stringify({
  attribut1: 'some value',
  attribut2: 'some other value',
});

const transactionDataSignature = {
  method: SignatureTypes.REQUEST_SIGNATURE_METHOD.ECDSA,
  value:
    '0xe649fdfe25c3ee33061a8159be9b941141121c5bed8d07664cb67b7912819b453984
};

const transaction: DataAccessTypes.ITransaction = {
  data: transactionData,
  signature: transactionDataSignature,
};

// Channel id is an identifer to group a list of transactions
const channelId = 'myRequest';

// Topics to index the channel
const channelTopics = ['stakeholder1', 'stakeholder2'];

const result = await dataAccess.persistTransaction(transaction, channelId,
```

Get Transactions by topic

```
import DataAccess from '@requestnetwork/data-access';
import { DataAccessTypes, SignatureTypes, StorageTypes } from '@requestnetw

const storage: StorageTypes.IStorage; // Any implementation of Storage laye

const dataAccess = new DataAccess(storage);
await dataAccess.initialize();

const transactionTopic = '010000000000000000000000000000000000000000';

const {
  result: { transactions },
} = await dataAccess.getTransactionsByTopic(transactionTopic);
```

Get Transactions by multiple topics

```
import DataAccess from '@requestnetwork/data-access';
import { DataAccessTypes, SignatureTypes, StorageTypes } from '@requestnetw

const storage: StorageTypes.IStorage; // Any implementation of Storage laye
const dataAccess = new DataAccess(storage);
await dataAccess.initialize();

const topics = [
  '010000000000000000000000000000000000000000',
  '011111111111111111111111111111111111111111',
];

const {
  result: { transactions },
} = await dataAccess.getChannelsByMultipleTopics(topics);
```

Get Transactions by channelId

```
import DataAccess from '@requestnetwork/data-access';
import { DataAccessTypes, SignatureTypes, StorageTypes } from '@requestnetw

const storage: StorageTypes.IStorage; // Any implementation of Storage laye

const dataAccess = new DataAccess(storage);
await dataAccess.initialize();

const channelId = 'myRequest';

const {
  result: { transactions },
} = await dataAccess.getTransactionsByChannelId(channelId);
```

Features

Indexing

Transactions are indexed with `topics`. When persisting a transaction on data-access, topics can be given as second parameters. These topics will allow retrieving transactions later. For example, each transaction made by @requestnetwork/request-logic are indexed with the `requestId`.

Batching

To save on costs, transactions are batched together. This is the responsibility of Data Access layer. This package creates `blocks` that are collections of `transactions`

Synchronization

Blocks can be added into the storage by other peers running their own data-access instance. Therefore, to remain consistent with the global state of the network, this package need to synchronize with these new blocks. `dataAccess.synchronizeNewDataIds()` allows to synchronize manually with all unsynchronized blocks. The synchronization can also be done automatically, `dataAccess.startAutoSynchronization()` allows to automatically synchronize with new blocks, the interval time between each synchronization can be defined in data-access constructor. `dataAccess.stopAutoSynchronization()` allows to stop automatic synchronization.

Architecture

data-access implements the structure of the data on the blockchain.

The architecture is a sorted list of `blocks`. A `block` is a JSON (see packages/data-access/format/) object containing:

- a sorted list of `transactions`
- an index of this transactions (a dictionary referencing the transactions by arbitrary string)

A `transaction` is an object containing the data as a string and the signature of these data. (the data will be the actions from the request logic layer)

Example of a block

```
{
   "header":{
      "index":{
         "0xaaaaaa":[
            0
         ],
         "0xcccccccccccc":[
            0,
            1
         ],
         "0xe53f3ea2f5a8e5f2ceb89609a9c2fa783181e70f1a7508dccf5b770b846a6a8
            0
         ],
         "0x320728cd4063b523cb1b4508c6e1627f497bde5cbd46b03430e438289c6e1d2
            1
         ]
      },
      "version":"0.1.0"
   },
   "transactions":[
      {
            "signature":{
               "method":"ecdsa",
               "value":"0x12345"
            },
            "transaction":"{\"attribut1\":\"plop\",\"attribut2\":\"value\"}"
         },
         {
            "signature":{
               "method":"ecdsa",
               "value":"0x12345"
            },
            "transaction":"{\"attribut1\":\"foo\",\"attribut2\":\"bar\"}"
         }
      ]
   }
```

Contributing

Pull requests are welcome. For major changes, please open an issue first to discuss what you would like to change. Read the contributing guide

License

MIT