Userfeeds Documentation

Release 0.1.0

Grzegorz Kapkowski

Feb 22, 2018

Contents

Contents:

CHAPTER 1

Overview

1.1 What is Userfeeds Platform?

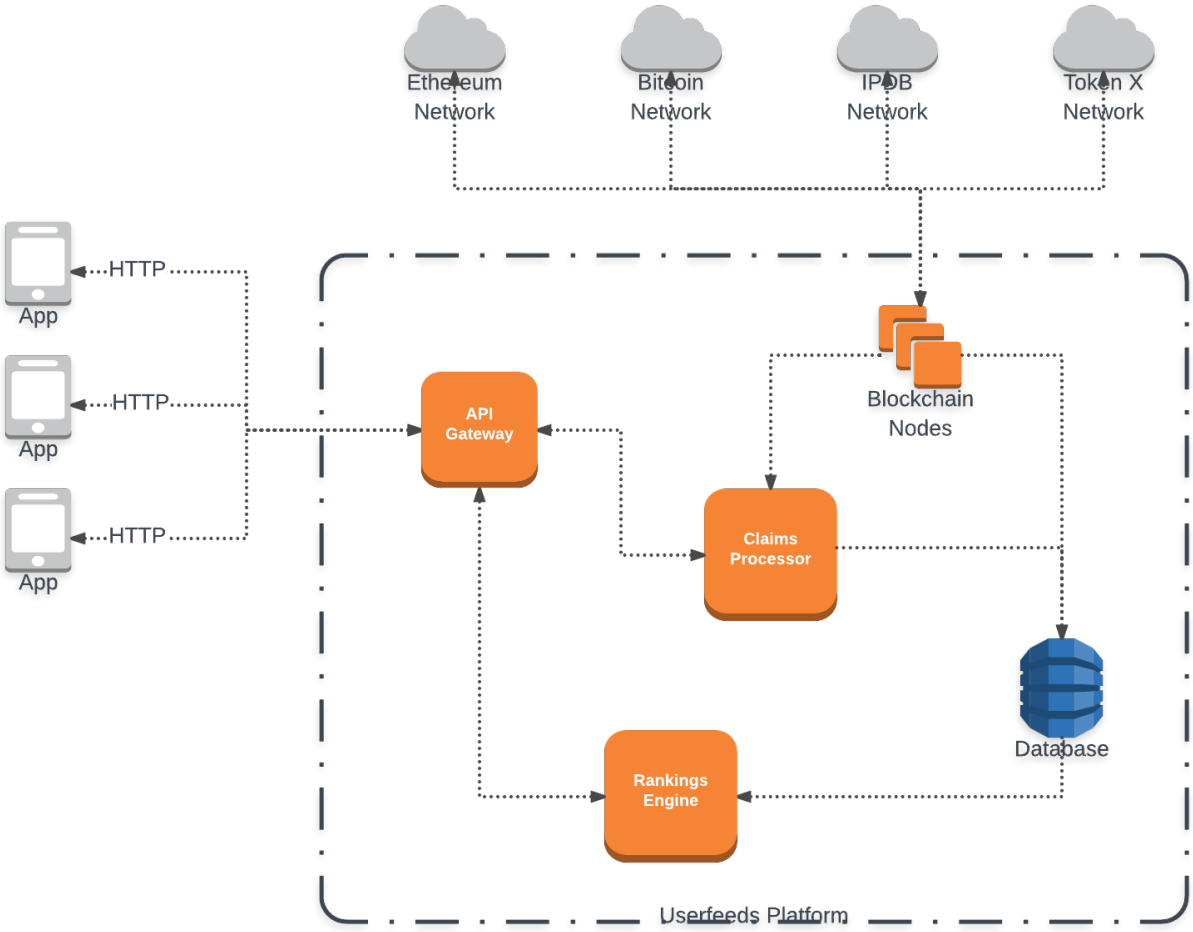
Userfeeds Platform was envisioned as a infrastructure allowing developers to easily use and build better content rankings for their clients. We want to help developers by providing them ready to use APIs, libraries and algorithms that they can incorporate into their software.

1.2 What is a content ranking?

Some examples of content rankings are list of products on amazon.com, songs in playlist on spotify, news on nytimes, and all other places that have some content (links, articles, songs) sorted in particular way that you can access.

1.3 How Userfeeds Platform works?

Userfeeds Platform is composed of couple of parts that interact together to deliver rankings through HTTP APIs to your application.



First Userfeeds Platform reads all the information from supported blockchins and stores it in internal DB. We store all transactions and contract calls in Graph database and current balances in standard SQL database. When someone sends *Claim* data to Userfeeds Platform through HTTP or through Ethereum network, it is processed and inserted into Graph database for use in ranking algorithms. When application makes HTTP request to our Ranking API endpoint, an Ranking Engine is taking requested algorithm and applies it to current data stored in Graph database and SQL database and returns sorted entries for application to display to its user.

1.4 What blockchains are supported?

Currently we support Ethereum, Bitcoin, IPDB

1.5 What is *Claim*?

Claim is basic data entity in Userfeeds Platform. It may represent *endorsement*, *like*, *upvote* for given URL/Text/Identifier that is signed by one of supported signing methods.

1.6 What are supported signing methods?

You can *sign Claim* with ECDSA and send it through HTTP, or make Ethereum transaction with *Claim* data and it will be treated as signed.

1.7 Why should I use Userfeeds Platform?

- Our algorithms are open source and can be improved every time an issue arises
- You are able to create your own custom rankings
- We use blockchain as a source of ranking signals, eg. how much tokens are connected to given content, how stable were token holders endorsing given content, how involved they are in given token.
- You can monetize your app/site easily by providing *sponsored* ranking on your app/site
- You can deliver superior experiance for your users by customizing our ranking output based on your needs.

1.8 I'm a developer. How can I start using Userfeeds Platform?

You should go to *Quick Start* and *API Reference* to checkout how to use our read-only APIs and how to use Ethereum blockchain for pushing information into Userfeeds Platform. Later you can go to http://api.userfeeds.io/portal/, register as developer, and go to http://api.userfeeds.io/portal/apis/ API Catalog to request API Key. When you have API Key you can start using all of our HTTP APIs.

CHAPTER 2

Features

2.1 Done

- **Links Exchange**
  - `userfeeds-links` widget
  - Whitelisting
  - Link bidding
- **Rankings**
  - Links ranking
  - Sponsored ranking
- **Basic data synchronization**
  - **Ethereum - ether**
    - mainnet
    - ropsten
    - rinkeby
- Simple *sponsored* Items App
- **Claim transports**
  - HTTP
  - Ethereum Contract
- Continuous deployment (dev environment)

2.2 Planned

- **Stable data synchronization**
  - **Ethereum - ether, tokens**
    - mainnet
    - ropsten
    - kovan
    - rinkeby
  - Bitcoin - value transactions
  - BigchainDB (IPDB) - asset/value transactions
- **Claim transports**
  - Ethereum Whisper
  - BigchainDB (IPDB)
- Integration Tests Suite
- **Link Exchange**
  - Production ready look&feel
  - **More `userfeeds-links` widget types**
    - X links
  - Integrations with partners
- Status.im integration
- **Pairing**
  - Auth Service
- Simple `userfeeds-button` widget
- Simple Governance App
- Simple Attention Guard App
- **Rankings**
  - Hold
- **Public Database Access**
  - Data dumps
  - Public instances
- **External algorithm providers**
  - Github
  - Gitlab
  - Gist
  - Bitbucket
- External fraud-detection data providers

2.3 Under consideration

- Distributed, decentralized, verifiable *claims/data* storage

CHAPTER 3

Roadmap

3.1 Applications

3.1.1 Links

- Finish current views
- ShareApp - allow posting links to given contexts

3.1.2 Labels

- Web app gallery with pictures with labels, grouping based on labels
- Mobile gallery app with picture gouping based on labels
- ShareApp - allow sharing pictures with attached labels

3.1.3 StateOfTheDapps

- Integrate sponsored ranking for dapps
- Add sorting by *hold* for ether and other tokens

3.1.4 Status.im

- Integrate sponsored dapps page
- userfeeds-button *endorse/star* on each dapp

3.1.5 CryptoAuth

- claims signing
- claims sending through HTTP
- claims sending through web3.js (MetaMask, Mist)
- pairing support
- Support for import/export of identity to mobile app (ShareApp)

3.1.6 Clouds

- bring back clouds ui for most popular tokens

3.2 Algorithms

3.2.1 Labels

- Targets for context with their labels
- All labels attached to identifier

3.2.2 Hold

- Postgresql filled with hold data for tokens and ether

3.2.3 Piping

- Change algorithms to be albe to pipe | them through - links:contextID|timedecay:7days|groupby:id - links:contextID|hold|groupby:id - claims:contextID|hold - authored:contextID|timedecay:7days - labels:contextID|groupby

3.3 Widgets

3.3.1 userfeeds-links

- link list widget

3.3.2 userfeeds-button

- allow selecting transport
- connect with cryptoauth.io

3.4 Data synchronization

3.4.1 Tokens

- Add token transactions support
- Verify if postgresql fillers are working correctly
- Add token-hold data to postgresql

3.4.2 Claims

Move claims to blocks reader

- **Based on Keccak-256 hash of:**
  - Claim(address,string)
  - Claim(address,address,string)
- Look through all transaction recipts if it generates such topics and treat them as proper claims

CHAPTER 4

Basic Concepts

4.1 Context

Identifier in format `networkName:networkAddress` specifing to where given claims are destined. Can be arbitrary string without special meaning. (needs to follow format)

More comples formats are possible, eg. `networkName:networkAddress:Tag` depending on use case.

CHAPTER 5

Quick Start

APIs that attempt to create Claims using HTTP require API key for access, before you start using them you should register and get your API Key at https://api.userfeeds.io/portal/apis/.

Read-only APIs do not require API key.

5.1 Get simple ranking for Ethereum

Those code snippets will return list of news shared to Ethereum context sorted according to our *simple* algorithm.

5.1.1 In python

```
import requests

RANKING_URL = "https://api.userfeeds.io/ranking/ethereum/claims/"

response = requests.get(RANKING_URL).json()

print("Simple ranking for Ethereum:\n")

for index, item in enumerate(response['items']):
    print("{0}. {1}".format(index, item["target"]))
    print("Score: {0}".format(item['score']))
    print()


# Simple ranking for Ethereum:
#
# 0. http://example.com/one
# Score: 123.1234324
#
```

```
# 1. http://example.com/two
# Score: 32.234542343
```

5.1.2 In JavaScript (node.js)

```
const https = require('https');

const options = {
  hostname: 'api.userfeeds.io',
  port: 443,
  path: '/ranking/ethereum/claims/',
  method: 'GET'
};

console.log('Simple ranking for Ethereum:\n')

const req = https.request(options, (res) => {
  let data = "";

  res.setEncoding('utf8');
  res.on('data', (chunk) => {
    data += chunk;
  });
  res.on('end', () => {
    let ranking = JSON.parse(data);
    for (let index in ranking.items) {
      console.log(`${index}. ${ranking.items[index].target}`);
      console.log(`Score: ${ranking.items[index].score}\n`);
    }
  });
});

req.on('error', (e) => {
  console.error(e);
});

req.end();


// Simple ranking for Ethereum:
//
// 0. http://example.com/one
// Score: 123.1234324
//
// 1. http://example.com/two
// Score: 32.234542343
```

5.1.3 In JavaScript (browser)

```
<!DOCTYPE html>
<html>
<head>
  <title>Simple ranking for Ethereum:</title>
</head>
```

```
<body>
  <script>
    fetch('https://api.userfeeds.io/ranking/ethereum/claims/')
    .then(function(response) {
      return response.json();
    })
    .then(function(ranking) {
      var div = document.getElementById('ranking');

      for (var i = ranking.items.length - 1; i >= 0; i--) {
        var item = ranking.items[i];
        div.innerHTML += i.toString() + '.<a href="' + item.target + '">' + item.
↪target + '</a><br/>';
        div.innerHTML += 'Score: ' + item.score + '<br/><br/>';
      }
    });
  </script>
  <h1>Simple ranking for Ethereum</h1>
  <div id="ranking"></div>
</body>
</html>
```

Demo:

5.2 Get available algorithms for Ethereum

Sometimes you want to give your users more than one view onto the same dataset. You can get algorithms available for Ethereum context using those simple code snippets.

5.2.1 In python

```
import requests

RANKING_URL = "https://api.userfeeds.io/ranking/ethereum/"

response = requests.get(RANKING_URL).json()

print("Available algorithms for Ethereum:\n")

for index, item in enumerate(response['items']):
    print("ID:", item["identifier"])
    print("Summary:", item['description'])
    print()


# Available algorithms for Ethereum:
#
# ID: simple
# Summary: Simple algorithm
#
# ID: sponsored
# Summary: Sponsored content algorithm
```

5.2.2 In JavaScript (node.js)

```
const https = require('https');

const options = {
  hostname: 'api.userfeeds.io',
  port: 443,
  path: '/ranking/ethereum/',
  method: 'GET'
};

console.log('Available algorithms for Ethereum:\n')

const req = https.request(options, (res) => {
  let data = "";

  res.setEncoding('utf8');
  res.on('data', (chunk) => {
    data += chunk;
  });
  res.on('end', () => {
    let ranking = JSON.parse(data);
    for (let item of ranking.items) {
      console.log(`ID: ${item.identifier}`);
      console.log(`Summary: ${item.description}\n`);
    }
  });
});

req.on('error', (e) => {
  console.error(e);
});

req.end();


// Available algorithms for Ethereum:
//
// ID: simple
// Summary: Simple algorithm
//
// ID: sponsored
// Summary: Sponsored content algorithm
```

5.2.3 In JavaScript (browser)

```
<!DOCTYPE html>
<html>
<head>
  <title>Available algorithms for Ethereum:</title>
</head>
<body>
  <script>
    fetch('https://api.userfeeds.io/ranking/ethereum/')
    .then(function(response) {
      return response.json();
```

```
    })
    .then(function(ranking) {
      var div = document.getElementById('ranking');

      for (var i = ranking.items.length - 1; i >= 0; i--) {
        var item = ranking.items[i];
        div.innerHTML += 'ID: ' + item.identifier + '<br/>';
        div.innerHTML += 'Summary: ' + item.description + '<br/><br/>';
      }
    });
  </script>
  <h1>Available algorithms for Ethereum</h1>
  <div id="ranking"></div>
</body>
</html>
```

Demo:

5.3 Allow your users to sponsor content on your webpage

5.3.1 In JavaScript (browser)

```
<!DOCTYPE html>
<html>
<head>
  <title>Sponsred ranking for Ethereum:</title>
  <style type="text/css">
    ul#choices li {
      list-style: none;
      display: inline-block;
    }

    ul#choices li img {
      width: 200px;
      height: auto;
    }

    div#ranking img {
      width: 180px;
      height: auto;
      margin-right: 20px;
    }

    ul.winners li {
      display: inline-block;
    }
  </style>
</head>
<body>
  <h1>Winners:</h1>
  <div id="ranking"></div>

  Ranking address:<input type="text" id="address" value=
↪"rinkeby:0xfe5da6ae3f65e7d5ce29121a9f5872ffd83b45e6"/>
  <button onclick="init();">Refresh</button>
```

```
<h1>Which is the best animal?:</h1>
<ul id="choices">
  <li>
    <img id="cat" src="img/cat.png">
    <br/>
    <button onclick="promote(this, 0.01);">Promote by 0.01 Ether</button><br/>
    <button onclick="promote(this, 0.1);">Promote by 0.1 Ether</button><br/>
    <button onclick="promote(this, 1);">Promote by 1 Ether</button>
  </li>
  <li>
    <img id="dog" src="img/dog.png">
    <br/>
    <button onclick="promote(this, 0.01);">Promote by 0.01 Ether</button><br/>
    <button onclick="promote(this, 0.1);">Promote by 0.1 Ether</button><br/>
    <button onclick="promote(this, 1);">Promote by 1 Ether</button>
  </li>
  <li>
    <img id="horse" src="img/horse.png">
    <br/>
    <button onclick="promote(this, 0.01);">Promote by 0.01 Ether</button><br/>
    <button onclick="promote(this, 0.1);">Promote by 0.1 Ether</button><br/>
    <button onclick="promote(this, 1);">Promote by 1 Ether</button>
  </li>
  <li>
    <img id="bird" src="img/bird.png">
    <br/>
    <button onclick="promote(this, 0.01);">Promote by 0.01 Ether</button><br/>
    <button onclick="promote(this, 0.1);">Promote by 0.1 Ether</button><br/>
    <button onclick="promote(this, 1);">Promote by 1 Ether</button>
  </li>
</ul>

<script>
  var rankingAddress = null;
  var content = {};

  function init() {
    rankingAddress = document.getElementById('address').value;

    var choices = document.getElementById('choices').getElementsByTagName("li");

    for (var i = choices.length - 1; i >= 0; i--) {
      var img = choices[i].getElementsByTagName('img')[0];
      content[img.id] = img.src;
    }

    displayRanking();
  }

  function displayRanking() {
    fetch(`https://api.userfeeds.io/ranking/${rankingAddress}/sponsored/`)
    .then(response => response.json())
    .then(ranking => {
      let div = document.getElementById('ranking');

      let html = '<ul class="winners">';
```

```
        for (let i = 0; i <= 3; i++) {
          let item = ranking.items[i];
          if (item) {
            let src = content[toAscii(item.target)];
            let score = item.score;
            if (src) {
              html += `<li><img src="${src}"><br/>Score: ${score}</li>`;
            }
          }
        }
        html += '</ul>';

        div.innerHTML = html;
      });
    }

    function promote(button, value) {
      var img = button.parentElement.getElementsByTagName('img')[0];

      // Send identifier as transaction data
      var data = web3.fromAscii(img.id);

      web3.eth.sendTransaction({to: rankingAddress.split(':')[1], data: data, value:␣
↪web3.toWei(value, 'ether')}, function(err, address) {
        if (!err)
          alert("Thank You!");
      });
    }

    function toAscii (hex) {
        var str = '',
            i = 0,
            l = hex.length;
        if (hex.substring(0, 2) === '0x') {
            i = 2;
        }
        for (; i < l; i+=2) {
            var code = parseInt(hex.substr(i, 2), 16);
            if (code === 0) continue; // this is added
            str += String.fromCharCode(code);
        }
        return str;
    };

    init();
  </script>
</body>
</html>
```

Demo:

CHAPTER 6

API Reference

API root is https://api.userfeeds.io/

Note: APIs that attempt to create Claims using HTTP require API key for access, before you start using them you should register and get your API Key at https://api.userfeeds.io/portal/apis/.

Read-Only APIs do not require API key.

6.1 Ranking

Get ranking for given *context* according to *algorithm*.

read-only

Available algorithms are described in *Algorithms* section.

```
$ curl https://api.userfeeds.io/ranking/ethereum/all/
```

GET /ranking/(**string:** *context*)**/**
string: *algorithm***/ Example request**:

```
GET /ranking/ethereum/all/ HTTP/1.1
Host: api.userfeeds.io
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "items": [
```

```
    {
      "target": "https://ethereum.org/",
      "score": 123
    },
    {
      "target": "https://userfeeds.io/",
      "score": 88
    }
  ]
}
```

6.2 Algorithms

Get all available algorithms for given *context*

read-only

```
$ curl https://api.userfeeds.io/ranking/ethereum/
```

GET /ranking/(string: *context*)/

Example request:

```
GET /ranking/ethereum/ HTTP/1.1
Host: api.userfeeds.io
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "items": [
    {
      "description": "Simple hold-based algorithm",
      "source": "https://github.com/Userfeeds/Algorithms/simple/"
      "identifier": "simple"
    },
    {
      "description": "All claims sorted by time",
      "source": "https://github.com/Userfeeds/Algorithms/all/"
      "identifier": "all"
    }
  ]
}
```

6.3 Storage

Post `claim` to Userfeeds Platform.

Note: This API requires Authorization header. Register at https://api.userfeeds.io/portal/apis/ to get one.

```
$ curl -X POST https://api.userfeeds.io/storage/ \
  -H "Content-Type: application/json" \
  -H "Authorization: API_KEY" \
  --data @/path/to/claim.json
```

POST /storage/

Example request:

```
POST /storage/ HTTP/1.1
Host: api.userfeeds.io
Content-Type: application/json
Authorization: API_KEY
Content-Length: 104
```

Example response:

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json
Content-Length: 27

{"status": "Accepted"}
```

6.4 Verification

Verify `claim` signature.

Note: This API requires Authorization header. Register at https://api.userfeeds.io/portal/apis/ to get one.

```
$ curl -X POST https://api.userfeeds.io/verify/ \
  -H "Content-Type: application/json" \
  -H "Authorization: API_KEY" \
  --data @/path/to/claim.json
```

POST /verify/

Example request:

```
POST /verify/ HTTP/1.1
Host: api.userfeeds.io
Content-Type: application/json
Authorization: API_KEY
Content-Length: 104
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 27

{"valid": true, "error": null}
```

```
HTTP/1.1 400 BAD REQUEST
Content-Type: application/json
Content-Length: 27
```

```
{"valid": false, "error": "Invalid message format"}
```

6.5 Contexts

Get contexts supported by Userfeeds Platform

TODO

6.6 Tokens

Get token balance for given identifier

TODO

CHAPTER 7

Tutorials

7.1 React App

TODO

7.2 Angular App

TODO

7.3 Android App

TODO

CHAPTER 8

Guides

In this section we will try to introduce you to more in-depth look at Userfeeds Platform

8.1 API Authorization

APIs that write to Userfeeds Platform require `Authorization` header to be present with API key as a value.

Note: API key can be requested at https://api.userfeeds.io/portal/apis/, after registration. API key requests are currently approved manually so it may take up to 24h for you request to be approved.

Example of Authorized request would be:

cURL:

```
$ curl -X POST https://api.userfeeds.io/verify/ \
  -H "Content-Type: application/json" \
  -H "Authorization: YourAPIKey123" \
  --data @/path/to/claim.json
```

Python:

```
import json
import requests

claim = open('claim.json').read()
claim = json.loads(claim)

response = requests.post(
  "https://api.userfeeds.io/storage/",
  json=claim,
  headers={
      "Authorization": "YourAPIKey123"
```

```
  })

print(response.content)
```

8.2 Claims Signatures

Signed claims are created by adding special `signature` object to claim.

All claims inside Userfeeds Platform databases are signed (in some way). Claims sent to Userfeeds Platform through HTTP needs to be signed prior to being sent in contrast to claims sent via Ethereum transaction which signature is created on Userfeeds Platform side based on transaction containing the claim.

`signature` object structure inside `claim` looks like this:

```
{
  "context": "...",
  "claim": {
    "target": "..."
  },
  "signature": {
    "type": "TYPE",
    "creator": "CREATOR",
    "signatureValue": "SIGNATURES_VALUE"
  }
}
```

8.2.1 `type`

It descries what kind of signature we are dealing with.

Full list of supported types can be found at *claim.signature.type* reference.

8.2.2 `creator`

Identifier of entity that created the signature.

8.2.3 `signatureValue`

Signature allowing for verification that `creator` signed this `claim`.

Create signed Claim (ECDSA)

Following code will show you how to create *signed claim* with signature type of *ecdsa.**.

Simple signing script written in Python.

```
import ecdsa
import json
import sys
import binascii
import hashlib
```

```
## Read claim from file
claim = open(sys.argv[-1]).read()
claim = json.loads(claim)

## Generate Keys
private_key = ecdsa.SigningKey.generate(curve=ecdsa.NIST256p)
public_key = private_key.get_verifying_key()

## Sign claim
message = json.dumps(claim, separators=(',', ':'), sort_keys=True).encode('utf8')

creator = public_key.to_string()
creator = binascii.hexlify(creator)
creator = creator.decode("utf8")

signature = private_key.sign(message, hashfunc=hashlib.sha256, sigencode=ecdsa.util.
↪sigencode_der)
signature = binascii.hexlify(signature)
signature = signature.decode("utf8")

claim["signature"] = {
    "type": "ecdsa.prime256v1",
    "creator": creator,
    "signatureValue": signature
}

## Print signed claim
print(json.dumps(claim, separators=(',', ':'), sort_keys=True))
```

Save this code to `sign.py` and run it like this:

```
$ pip install ecdsa
$
$ python sign.py /path/to/claim.json
```

Simple signing script written in Javascript.

```
let KJUR = require('jsrsasign');
let serialize = require('canonical-json');
let fs = require('fs');

// Read claim from file
let claim = fs.readFileSync(process.argv[process.argv.length - 1]);
claim = JSON.parse(claim);

// Generate Keys
let curve = "secp256k1";
let keypair = KJUR.KEYUTIL.generateKeypair("EC", curve);
let sig = new KJUR.crypto.Signature({"alg": "SHA256withECDSA"});

// Sign claim
let message = serialize(claim);

let creator = keypair.pubKeyObj.pubKeyHex;

sig.init(keypair.prvKeyObj);
sig.updateString(message);
```

```
let signature = sig.sign();

claim. signature = {
  type: "ecdsa." + curve,
  creator: creator,
  signatureValue: signature
};

// Print signed claim
console.log(serialize(claim));
```

Save this code to `sign.js` and run it like this:

```
$ npm install canonical-json jsrsasign
$
$ node sign.js /path/to/claim.json
```

Verify claim signature

```
$ curl -X POST https://api.userfeeds.io/verify/ \
  -H "Content-Type: application/json" \
  -H "Authorization: API_KEY" \
  --data @/path/to/claim.json
```

8.3 Submiting claims

TODO

8.3.1 HTTP

TODO

8.3.2 Ethereum Transaction

TODO

8.3.3 Whisper message

TODO

8.4 Sponsored vs Organic Rankings

TODO

8.5 Organic ranking with interface token

Warning: NOT IMPLEMENTED!

You can pass additional token to organic rankings to inject sponsored results into organic rankings. Those results will be marked with *sponsored: true* key in *items* list.

example:

```
GET /ranking/ORGANIC_TOKEN/SIMPLE_RANKING/?sponsored=INTERFACE_TOKEN

{
    "items": [
        {
            "value": "http://organic.com/art/1",
            "score": 1234
        },
        {
            "value": "http://organic.com/art/1",
            "score": 0,
            "sponsored": true
        },
        {
            "value": "http://organic.com/art/1",
            "score": 1100
        },
        ...
    ]
}
```

TODO...

CHAPTER 9

Data Model

Claims are the smallest self contained piece of information inside Userfeeds Platform. All claims are cryptographically signed or have reference to their cryptographic origin (eg. transaction on blockchain)

9.1 Structure

9.1.1 Overview

```
{
  "context": "CONTEXT",
  "type": ["TYPEA", "TYPEB", "..."],
  "claim": {
    "target": "TARGET",
    "additional": "fields",
    "go": ["here", "..."]
  },
  "credits": [
    {
      "type": "interface",
      "value": "INTERFACE IDENTIFIER"
    }
  ],
  "signature": {
    "type": "TYPE",
    "creator": "CREATOR",
    "signatureValue": "SIGNATURE"
  }
}
```

9.1.2 context

optional

Context field is used to denote *destination* of given claim. It can be interpreted as name of any topic/thing that people might want to share information about. Usually it will have something to do with blockchain space (but it doesn't have to). For example if you would like to share some information to all people interested in *Ethereum* blockchain you will send claim with `ethereum` as context.

Other examples of contexts might be:

- ethereum
- ethereumclassic
- bitcoin
- ethereum:0x4564567890abcdef....abc
- ethereumclassic:0x4564567890abcdef....abc
- myspecialcontext
- companyx
- companyx:departmenty

`ethereum:0x123....456` context identifiers are interpreted as *object 0x123....456 on ethereum* and usually will be used to share information about contracts/addresses on given blockchain.

Special contexts starting with *userfeeds:* are *technical* context and have special meaning in Userfeeds Platform. eg. *userfeeds:pairing* will create special *PAIRED* relationship allowing one to connect their cryptocurrency holdings with additional public key that will only be used to sign claims.

9.1.3 type

optional

Type describes additional data present in *claim* object.

Example can be *labels* type, with this type *claim* object needs to have *labels* key with array of values.

Description of all supported types can be found at *Types*

9.1.4 claim

This key is used to store user provided information and it is mandatory for all claims. `claim` object will always have `target` key and additional fields depending on `type` array.

target

`target` value identifies target object that user wants to share or tag with additional information.

Some examples of proper `target` values:

- http://some.url/path/
- text:base64:base64encodedtext
- ipfs:somehash

- ipdb:somehash
- claim:claimsignaturehash
- mediachain:somehash
- isbn:0451450523
- isan:0000-0000-9E59-0000-O-0000-0000-2
- btih:c12fe1c06bba254a9dc9f519b335aa7c1367a88a
- ethereum:0x4564567890abcdef...abc
- bitcoin:0x4afebcdef...123

Additional fields

Depending on `type` array additional keys might be present in `claim` object. See *Types* for supported types.

9.1.5 `signature`

This field is generated to denote ownership of claim and can be cryptographic signature or a pointer to cryptographically secure origin of claim.

`type`

Describes what type of signature we are dealing with. It could be `ecdsa.secp256r1` for elliptic curve signature or `ethereum.transaction` for claim origination from Ethereum blockchain.

Supported Signature Types

`ecdsa.prime192v1` In python: ecdsa.NIST192p

`ecdsa.secp256r1` / `ecdsa.prime256v1` In python: ecdsa.NIST256p

`ecdsa.secp224r1` In python: ecdsa.NIST224p

`ecdsa.secp384r1` In python: ecdsa.NIST384p

`ecdsa.secp521r1` In python: ecdsa.NIST521p

`ecdsa.secp256k1` In python: ecdsa.SECP256k1

`ethereum.transaction` Claims posted on ethereum blockchain will be verified by comparing blockchain content with claim content.

`creator`

This identifies public key or address that signed claim.

Format: identifier

- hex:04861127b14bf0036e...ef7127b114988057
- rinkeby:0x1234567890abcdef...1456
- ethereum:0x1235...145

- bitcoin:0x123456...1234

signatureValue

This key holds raw signature value as produced by signing algorithm, or it can be transaction hash or any valid identifier of externally verifiable origin of claim.

9.2 Types

9.2.1 Basic

Basic claim:

```
{
  "claim": {
    "target": "http://some.url/path/"
  },
  "context": "ethereum:0x4564567890abcdef....abc",
  "signature": {
    "creator":
↪"94d1aa6655d931294d524cf52b0df866976f89774bac38a730cf20e2d51dd24d34efc2bbb4d5bba91a7a6582511491dde1
↪",
    "signatureValue":
↪"304402203dac2176721d7e05cd8c580a27a504b64b0a8ee171b18a07630201cbed979ac7022013faf8735f90b957ca4656
↪",
    "type": "ecdsa.prime256v1"
  }
}
```

With acknowledgment of interface from which claim was created:

```
{
  "claim": {
    "target": "http://some.url/path/"
  },
  "context": "ethereum:0x4564567890abcdef....abc",
  "credits": [
    {
      "type": "interface",
      "value": "http://blog.example.com/path/"
    }
  ],
  "signature": {
    "creator":
↪"0df1d4915347bcae90a0696c9efd6300e33b610d31130c3049d329fa61af138de7a7ee55f99057fd8d39c4664be9f1c343
↪",
    "signatureValue":
↪"304502206c243684007c9e412612b5d1a371b20eb146652e4b149bb1fc0e6da437e7f728022100b8c77983949feac478de
↪",
    "type": "ecdsa.prime256v1"
  }
}
```

9.2.2 `link`

Additional keys:

- `title` **string**
- `summary` **string**


```
{
  "claim": {
    "summary": "summary",
    "target": "http://some.url/path/",
    "title": "title"
  },
  "context": "rinkeby:0xfe5da6ae3f65e7d5ce29121a9f5872ffd83b45e6",
  "credits": [
    {
      "type": "interface",
      "value": "http://blog.example.com/path/"
    }
  ],
  "signature": {
    "creator":
↪"ffc8c2f39e8a302bf9ca37b06fa9014f0cd3c85900c3d8b771f31a91ce33c050948faedad14d73a4f6c41f5937c3a010b5
↪",
    "signatureValue":
↪"304602210093c55c30be1868de005def995aefb391d7ff20f235457b37a01e6921427701f80221008e5fc2afc2c912466
↪",
    "type": "ecdsa.prime256v1"
  },
  "type": [
    "link"
  ]
}
```


9.2.3 `labels`

Additional keys:

- `labels` **array** of **string**


```
{
  "claim": {
    "labels": [
      "Good",
      "Book",
      "Cats"
    ],
    "target": "http://some.url/path/"
  },
  "context": "rinkeby:0xfe5da6ae3f65e7d5ce29121a9f5872ffd83b45e6",
  "signature": {
    "creator":
↪"de9965ce03cf6f960a7efe423633409a0052ad8f9f2100e27026ad94551d4d69058c0a263dbd0cacf999ca3e97ddcc0afa
↪",
    "signatureValue":
↪"3044022069457927f1fc06b26467a7cc93c99085efea4d8811c6979ffab9ba2196be5ad702201587d3f88d2e9058d6ce48
↪",
```

```
    "type": "ecdsa.prime256v1"
  },
  "type": [
    "labels"
  ]
}
```

9.3 Value Transfer

Along with claims Userfeeds Database contains normalized data about transfer of assets (tokens and other)

If transfer of assets was accompanying a claim they will be connected with *TRANSFER* relation.

CHAPTER 10

Transports

10.1 HTTP

You can send signed claims through Userfeeds Platform HTTP Gateway

Properties:

In-transport secrecy the claim cannot be sniffed on transport thanks to HTTPS connection and will only be available to outside world after it's incorporated into Userfeeds Platform through rankings APIs and database dumps.

Independent distribution the claim exists only inside Userfeeds Platform database and will be distributed with Userfeeds Platform database dumps.

Example of posting claim directly to Userfeeds Platform through HTTP transport

```
$ curl \
  -X POST https://api.userfeeds.io/storage/ \
  -H "Content-Type: application/json" \
  -H "Authorization: 59049c8fdfed920001508e2a03414df648e34ea665f544a17d5c113b" \
  -d '{"claim":{"target":"http://some.url/path/"},"context":
↪"ethereum:0x4564567890abcdef....abc","signature":{"creator":
↪"82fb68fc14719b94b36e99e588c9988458ca187d4791463164285bb064458232c4bb5bb638158096f4ed957e275e2e157
↪","signatureValue":
↪"c675d123fb1e99ffe59e7626c655806ebe47ec1ca4100b953029731159c2e14f4461e2ebf7f43a071b847b57cbcbd66bd
↪","type":"ecdsa.prime256v1"}}'
```

10.2 Ethereum Transaction

You can send claim through Ethereum Blockchain transaction.

Note: Userfeeds Platform does not yet monitor every transaction for potential claims

To send a claim through transaction you need to call special contracts that are monitored by Userfeeds Platform.

Properties:

In-transport secrecy The claim is available from the moment it's distributed through Ethereum network and can be sniffed before it reaches desired ranking.

Independent distribution The claim will be a part of Etherum Blockchain and will be available from all copies of the blockchain.

10.2.1 Contracts

With value transfer

Code

```
pragma solidity ^0.4.11;

contract Userfeeds {

    event Claim(address sender, address userfeed, string data);

    function post(address userfeed, string data) payable {
        userfeed.transfer(msg.value);
        Claim(msg.sender, userfeed, data);
    }
}
```

ABI

```
[
  {
    "constant":false,
    "inputs":[
      { "name":"userfeed", "type":"address" },
      { "name":"data", "type":"string" }
    ],
    "name":"post",
    "outputs":[],
    "payable":true,
    "type":"function"
  },
  {
    "anonymous":false,
    "inputs":[
      { "indexed":false, "name":"sender", "type":"address" },
      { "indexed":false, "name":"userfeed", "type":"address" },
      { "indexed":false, "name":"data", "type":"string" }
    ],
    "name":"Claim",
    "type":"event"
  }
]
```

Addresses

- Mainnet: ...
- Rinkeby: 0x0a48ac8263d9d79768d10cf9d7e82a19c49f0002

- Ropsten: 0xa845c686a696c3d33988917c387d8ab939c66226
- Kovan: ...

Without value transfer

Code

```
pragma solidity ^0.4.11;
contract Userfeeds {

    event Claim(address sender, string data);

    function post(string data) {
        Claim(msg.sender, data);
    }
}
```

ABI

```
[
  {
    "constant":false,
    "inputs":[
      { "name":"data", "type":"string" }
    ],
    "name":"post",
    "outputs":[],
    "payable":false,
    "type":"function"
  },
  {
    "anonymous":false,
    "inputs":[
      { "indexed":false, "name":"sender", "type":"address" },
      { "indexed":false, "name":"data", "type":"string" }
    ],
    "name":"Claim",
    "type":"event"
  }
]
```

Addresses

- Mainnet: ...
- Rinkeby: 0x09dcdf34e0c28b106fdfe51009cb71ae92bf8bbc
- Ropsten: 0x5c3fe6b94b57c1e294000403340f12f083e71b83
- Kovan: ...

10.3 Whisper Protocol

TODO: Direct message TODO: Broadcast

10.4 IPDB (BigchainDB)

TODO

CHAPTER 11

Algorithms

Userfeeds Platform allows running community written algorithms on too of data gathered from all supported sources like Ethereum blockchain (mainnet, ropsten, rinkeby, kovan, whisper messages), IPDB (BigchainDB) or direct messages.

Algorithms can be referenced by their identifier that depends on where the author decided to share them.

Some of algorithm sources and naming schemes are:

- Github - `github:username.repository.filename`
- Bitbucket - `bitbucket:username.repository.filename`
- Gist - `gist:hash`

Note: More esoteric sources are possible, like downloading code from blockchain or encrypted algorithm code but their support is not planned at the moment.

Warning: Current support is limited to built-in algorithms through their *short* identifiers.

11.1 Built-in algorithms

Source code: https://github.io/Userfeeds/algorithms

All algorithms require one information: `context`. They will get all claims intended for given `context` and sort them according to their specification.

Note: Built-in algorithms can be referenced normally via eg. `github:username.repo.filename` or by their short name eg. `latest`

11.1.1 Claims (TODO)

Identifier: `claims`, `github:Userfeeds.algorithms.claims`

The simples algorithm of all. It just returns 100 latest claims for given `context`

Parameters

type

- **string**
- **optional**

Type of claims to return

Example data


```
{
  "items": [
    {
      "created_at": 1499676056549,
      "id": "0xd10409e340b5f41be0c0c8db85a61b2db0e5f4fb6c87fd3a5e4b73adc9bee9bf",
      "score": 1000000000000000000,
      "target": "https://userfeeds.io/"
    },
    {
      "created_at": 1499676053781,
      "id": "0xcd05844f64d472d8aeceeafc1c14f377bc664a9f94574ee820128254c962bfdc",
      "score": 5000000000000000000,
      "target": "https://example.com/"
    },
    {
      "created_at": 1499676051855,
      "id": "0xaa6b523d72d1f0e9b00689f494ac622804e0552e55c939cabc97381bd401732b",
      "score": 0,
      "target": "0x64537b9f7c9d85bae5c52cb4fa2307e7da14c6f1cae6d710b66a4ddcc059e1a8"
    },
    {
      "created_at": 1499676051757,
      "id": "0x64537b9f7c9d85bae5c52cb4fa2307e7da14c6f1cae6d710b66a4ddcc059e1a8",
      "score": 100000000000000,
      "target": "https://some.path/"
    }
  ]
}
```


11.1.2 Authored (TODO)

Identifier: `authored`, `github:Userfeeds.algorithms.authored`

This algorithm return all claims that were *authored* by *context*. In other words all claims that have `AUTHORED` relation to `Account` node with `id` equal to `context`.

Parameters

type

- **string**
- **optional**

Type of claims to return

Example data


```
{
  "items": [
    {
      "created_at": 1499676051855,
      "id": "0xaa6b523d72d1f0e9b00689f494ac622804e0552e55c939cabc97381bd401732b",
      "score": 0,
      "target": "0x64537b9f7c9d85bae5c52cb4fa2307e7da14c6f1cae6d710b66a4ddcc059e1a8"
    },
    {
      "created_at": 1499675901184,
      "id": "0xc3921abdd00ca53925e5a5b8559537d44e19165ff3ad4e89daf0bb034a4afa21",
      "score": 10000000000000000,
      "target": "Lorem ipsum dolor sit amet, consectetur adipisicing elit"
    },
    {
      "created_at": 1499156515246,
      "id": "0x85b6643f69e746c7525fc3ed88e1f18241068bcff6516fbf2dd0d967ebdaf390",
      "score": 0,
      "target": "0x794d0c42fc35af7eba9216261ea3799e19d079d24dcf795fda94acf20ad298f8"
    }
  ]
}
```


11.1.3 Sponsored

Identifier: `sponsored`, `github:Userfeeds.algorithms.sponsored`

This algorithm is different form other as it does not use `claims` as a source of information but more low level `transaction.input` and as such works only for contexts like `ethereum`.

It just takes whatever is inside `transaction.input` sums up all `transaction.value`, sorts the result and returns those inputs as `target`.

The idea is that developer would send inside `transaction.input` only values that make sense for their application and only this application can display appropriate content for each entry.

An example application can be a gallery of images with identifiers like `img1` `img2`... Using those identifiers as `transaction.input` values, developer can create special *sponsored* gallery in his application that will display images sorted by overall value of ether used to promote them.

Example application can be found at *Quick Start*.

Example data


```
{
  "items": [
    {
      "bids": 2,
      "score": 1.01,
      "target": "0x636174"
    },
    {
      "bids": 1,
      "score": 1.0,
      "target": "0x62697264"
    },
    {
      "bids": 1,
      "score": 0.333,
      "target": "0x6b616e6761726f6f"
    },
    {
      "bids": 1,
      "score": 0.01,
      "target": "0x646f67"
    }
  ]
}
```


11.1.4 Links

Identifier: `links`, `github:Userfeeds.algorithms.links`

Time based algorithm that collects all claims with type `link` and sorts them according to formula:

```
period = 7 days

score = Sum(
  if now - claim.create_at > period:
    0
  else:
    claim.value / period * (now - claim.create_at)
)
```

Parameters

`period`

- **string**
- **optional**
- default: 7 days

Period of time decay to use during calculations

`whitelist`

- **string**

- **optional**

When present all claims type `whitelist` and `claim.signature.creator` equal to `whitelist` parameter will be used as filtering mechanism. Whitelisting claim needs to specify link claim `claim.signature.signatureValue` in it's `claim.target`.

Whitelisting

Ads algorithm has special property that allows for arbitrary list filtering based on `whitelist` parameter passed to it. It works like that: Algorithm gathers all claims with type `link` intended for given `context` and compares their `signature.signatureValue` with `claim.target` value of claims with `signature.creator` equal to `whitelist` parameter. Example:

Link claims

context	type	target	signatureValue
ethereum:0xA	link	http://some.url/1	hashABC
ethereum:0xA	link	http://some.url/2	hashDEF
ethereum:0xA	link	http://some.url/3	hashGHI

Whitelist claims

target	creator
hashABC	ethereum:0xjohn
hashGHI	ethereum:0xjohn
hashABC	ethereum:0xalice
hashDEF	ethereum:0xbob

Above combination will produce two links with `whitelist` parameter equal to `ethereum:0xjohn`:

target
http://some.url/1
http://some.url/3

with `whitelist` equal to `ethereum:0xalice` it will produce one link:

target
http://some.url/1

and with `ethereum:0xbob`:

target
http://some.url/2

if you omit `whitelist` in api call the algorithm will produce full set of links:

target
http://some.url/1
http://some.url/2
http://some.url/3

Example data


```
{
  "items": [
    {
      "id": "0xdddf6852e4e239cba808ebdb620535a30189cb5e66b2a4bdbc46e41c844543d6",
      "score": 340000000000000000,
      "summary": "Has nice pipelines",
      "target": "https://bitbucket.org",
      "title": "Bitbucket"
    },
    {
      "id": "0xbc7dfcd2a2148cda1f763fb2d4713911c55c9f0d3fa56d62264c085f550a2586",
      "score": 321923548932000000,
      "summary": "Best social platform",
      "target": "https://github.com",
      "title": "GitHub"
    },
    {
      "id": "0x943e5c024f04acd0352095a09c71b4e5ce6e5d789da588ad6aacdec859917cd6",
      "score": 10000000000000000,
      "summary": "Userfeeds Platform",
      "target": "https://userfeeds.io/",
      "title": "Userfeeds"
    },
    {
      "id": "0xb4ed77fd554929f28c255279c307aedbd39510437e405693e5d84fd6a69b8fca",
      "score": 10000000000000000,
      "summary": "Status",
      "target": "https://status.im/",
      "title": "Status.im"
    }
  ]
}
```


11.1.5 Hold (TODO)

Identifier: `hold`, `github:Userfeeds.algorithms.hold`

Simple algorithm that collects all claims and sorts them according to cumulative `hold`. Hold is `holdings*time` for given token. If someone has 10 ETH for 1 day then we can say he has hold of value 10, if he still has 10 ETH during next 6 days his `hold` will raise to 70.

11.2 Writing ranking algorithms

The best way to start writing your own algorithms is to look at the built-in one at https://github.io/Userfeeds/algorithms

TODO: ^^ more effort here please...

11.2.1 Data available for ranking algorithms

Database dumps can be found here:

- Neo4j: http://some.s3.or.smemthing/neo4j/latest

- Postgresql: http://some.s3.or.something/postgresql/latest

Those data dumps are updated `TODO: daily/hourly/constantly?` and can be used for development and data analysis.

TODO: Import instructions

Note: Direct database access will be possible to our read-only instances for projects that require lower delays.

Claims

TODO

Balances and Aggregated data

TODO

11.2.2 Helper functions

TODO

11.2.3 Testing your algorithm

11.2.4 Importing database

11.2.5 Connecting your algorithm to Userfeeds Platform

TODO

Publishing your algorithm

Supported source are:

- Github
- Bitbucket
- Gist
- ...

TODO: ...

Registering algorithm on Userfeeds Platform

TODO

CHAPTER 12

Apps

In this section we will describe basic applications built using Userfeeds Platform.

Apps overview can be found at https://userfeeds.io/apps.html

12.1 Links

12.1.1 Quick Start

Widgets configurator: https://userfeeds.io/apps/widgets/#/configurator/ Widgets demo: https://userfeeds.io/demo

12.1.2 Overview

Links app allows Publishers to integrate widgets into their web and mobile apps for presenting links and Advertisers to buy presentation space directly from given Publisher. This simple scheme allows Publishers to monetize their interfaces directly.

Whitelisting

Publisher has an option to choose which links he wants to display. He can either self-manage links by whitelisting them by hand or outsource this task to someone else. All he needs to do is to set *whitelist* option to desired identifier. This identifier will be used to query Claim.signature.creator field. It can be set to public address from metamask/mist if Publisher wants to manage links by himself or to any other identifier that will be used by some bot/AI to post *whitelist* claims.

12.1.3 Widgets

Links App is based on *userfeeds-links* widget.

12.1.4 Algorithms

Link App uses custom algorithm *ads* to provide basic *advertise-like* experiance.

Example response:

```
{
  "items": [
    {
      "target": "http://target.one/",
      "score": 123,
      "title": "Title One",
      "summary": "Lorem ipsum dolor incididunt officia cillum aute incididunt nisi␣
↪exercitation voluptate elit.",
      "bids": 10,
      "id": "abc"
    },
    {
      "target": "http://target.two/",
      "score": 123,
      "title": "Title Two",
      "summary": "Lorem ipsum dolor incididunt officia cillum aute incididunt nisi␣
↪exercitation voluptate elit.",
      "bids": 10,
      "id": "abcd"
    },
    {
      "target": "http://target.three/",
      "score": 123,
      "title": "Title Three",
      "summary": "Lorem ipsum dolor incididunt officia cillum aute incididunt nisi␣
↪exercitation voluptate elit.",
      "bids": 10,
      "id": "abcde"
    }
  ]
}
```

Source: https://github.com/Userfeeds/Algorithms/links/

This algorithm uses time-decay with 7 days decay period. (TODO)

Score key represents cumulated ETH value calculated based time-decay. Time decay is calculated from the moment of posting link so if publisher is using whitelisting mechanism it can have lower score if publisher takes long time to whitelist link. We encourage advertisers to post links with low amount of ether attached first and only after whitelisting adding more ether to them.

For example if someone pays 10 ETH for link it will have *score* 10 ETH at this moment. After 1 day the *score* will drop linearly to 10 * 6/7 ~ 8.57. If at this moment strengthening claim will be created with link claim id as *target* the *score* will increase to 18.57.

(TODO: add graphs)

12.1.5 Claims

Links App uses type='link' claim for posting new link and basic claims for approving (whitelisting).

Posting Link Claim:

```
{
  "context": "ethereum:0x4564567890abcdef....abc",
  "type": ["Link"],
  "claim": {
    "target": "http://some.url/path/",
    "title": "title",
    "summary": "summary"
  },
  ...
}
```

This claim type has two additional keys compared with basic claim.

title will be used as link title *summary* will be used as extended text (might not be displayed on some widgets)

Whitelisting claim is essentially a basic claim:

```
{
  "claim": {
    "target": "claim:signatureOfLinkClaim",
  },
  "signature": {
    "type": "...",
    "creator": "ethereum:0x1234567890abcdef....123",
    "signatureValue": "..."
  }
}
```

CHAPTER 13

Widgets

This section describes details about widgets options and implementation.

Widgets configurator: https://app.linkexchange.io/direct/configurator

13.1 linkexchange-link

Source: https://github.com/Userfeeds/Apps/tree/master/widgets/linkexchange-link

Latest version is available here: https://cdn.jsdelivr.net/npm/@linkexchange/widgets@latest

Common parameters:

`asset`

- **string**
- **required**
- **values**: ethereum | rinkeby | ropsten | kovan | ethereum:erc20TokenContractAddress

specifies the asset being used in the widget

`recipient-address`

- **string**
- **required**
- **values**: recipientAddress

specifies the address to which the asset will be sent on each transaction

`whitelist`

- **string**
- **optional**
- **values**: whitelistAddress

specifies the address that will be used as a whitelist for links. If specified all the links shown all page will need to be first accepted by the whitelist address

algorithm

- **string**
- **optional**
- **default**: ads
- **values**: *Algorithms*

name of the algorithm to use for links filtering/sorting

size

- **string**
- **optional**
- **default**: rectangle
- **values**: rectangle, leaderboard

specifies the size of the widget

widget-title

- **string**
- **required**

specifies the title of widget shown in details

description

- **string**
- **required**

specifies the description of widget shown in details

contact-method

- **string**
- **required**

specifies a contact method shown in widget details

slots

- **number**
- **optional**
- **default**: 10

specifies a number of links shown in widget

timeslot

- **number**
- **optional**
- **default**: 5

seconds each link fitting into the slots will be shown in widget

translations

- **string**
- **optional**

name of the object in global window object which contains translations mapping

translations-url

- **string**
- **optional**

url to json file which contains translations mapping

open-details

- **"modal" | "tab"**
- **optional**
- **default**: "modal"
- **since** 0.0.158

defines how widget details will be opened

Sample usage

```
<linkexchange-link
  algorithm="links"
  slots=2
  timeslot=7
  size="leaderboard"
  asset="rinkeby:0xd5cfec7...eec72aced241e5e"
  recipient-address="0xcD335186...41909D3448BC60f0665"
  whitelist="0xcD335186...5215215215abcdabcdabcd"
  widget-title="First Widget"
  description="I accept only links that are about science and technology. I like␣
↪trains"
  impression="N/A"
  contact-method="office@linkexchange.io"
>
</linkexchange-link>
<script src="https://cdn.jsdelivr.net/npm/@linkexchange/widgets@latest"></script>
```

```
<linkexchange-link
  algorithm="links"
  slots=10
  size="leaderboard"
  asset="ethereum"
  recipient-address="0xcD335186...41909D3448BC60f0665"
  widget-title="Real widget"
  description="True widget"
  impression="50k - 100k"
  contact-method="contact@realwidget.realwidget"
>
</linkexchange-link>
<script src="https://cdn.jsdelivr.net/npm/@linkexchange/widgets@latest"></script>
```

13.1.1 Custom validations

The widget allows you to register and unregister your own custom validators during runtime. To do that you can use *addValidation* and *removeValidation* methods defined on linkexchange-link html element.

`addValidation`

```
addValidation(
  formName: string,
  inputName: string = 'form',
  callback: (fieldName: string, value: any) => Promise<string | null> | string | null,
) : void
```

`addValidation` is a method that registers a validator that returns a string when validation did not pass. It can also return a Promise that resolves with string when error occurs.

Sample usage:

```
const widget = document.querySelector("linkexchange-link");
widget.addValidation("add-link", "title", (fieldName, value) => {
  return value === "Inappropriate" ? "Title can't be appropriate" : null;
});
```

`removeValidation`

```
removeValidation(
  formName: string,
  inputName: string = 'form',
  callback: (fieldName: string, value: any) => Promise<string | null> | string | null,
) : void
```

`removeValidation` is a method that unregisters a validator.

Sample usage:

```
const widget = document.querySelector("linkexchange-link");
const validator = (fieldName, value) => {
  return value === "Inappropriate" ? "Title can't be appropriate" : null;
};
widget.addValidation("add-link", "title", validator);
widget.removeValidation("add-link", "title", validator);
```

Current forms

Currently supported forms with fields withcustom validations are:

- `add-link`
  - `title`
  - `summary`
  - `target`
  - `value`

13.1.2 Translations

Translations can be provided using `transactions` or `transactions-url` attribute of the widget. When using remote file with translations widget rendering is defered.

Sample usage:

```
<script>
  window.my-custom-translations = {
    'banner.sponsoredWith': 'Gesponsert mit',
    'banner.noLinks': 'Keine Links verfügbar',
    'menu.buyLink': 'Kaufe einen Link',
  };
</script>
<linkexchange-link
  transactions="my-custom-translations"
  size="leaderboard"
  type="text"
  asset="ropsten"
  ...
></linkexchange-link>
```

Currently supported default transactions:

```
{
  "list.slots.title": "Slots",
  "list.approved.title": "Approved",
  "list.algorithm.title": "Algorithm",
  "widgetSpecification.title": "Widget Specification",
  "userfeedsAddressInfo.title": "Userfeed Address",
  "list.header.no": "NO",
  "list.header.probability": "Probability",
  "list.header.content": "Content",
  "list.header.score": "Current score",
  "list.header.bids": "Bids",
  "widgetSummary.openInNewWindow": "New Window",
  "widgetSummary.addLink": "Create new link",
  "widgetSummary.declaredImpression": "Declared impressions",
  "widgetSummary.sourceDomain": "Source Domain",
  "widgetSummary.contact": "Contact",
  "widgetSummary.validTill": "Valid till",
  "sideMenu.slots": "Slots",
  "sideMenu.approved": "Approved",
  "sideMenu.algorithm": "Algorithm",
  "sideMenu.specification": "Specification",
  "sideMenu.userfeed": "Userfeed",
  "menu.buyLink": "Buy a Link",
  "banner.sponsoredWith": "Sponsored With",
  "banner.noLinks": "No links available"
}
```

CHAPTER 14

Indices and tables

- genindex
- modindex
- search

HTTP Routing Table

/ranking

GET /ranking/(string:context)/, 26
GET /ranking/(string:context)/(string:algorithm)/, 25

/storage

POST /storage/, 27

/verify

POST /verify/, 27