# Guide for Installing and Setting up Mediawiki with Restbase and Mathoid

For this guide a clean install of Ubuntu Desktop 16.04 64-bit was used. In this guide we use Mediawiki 1.28.2, the procedure should however be similar to other versions of Mediawiki which support Restbase and Mathoid.

## Setup Environment

Update and upgrade apt:

sudo apt update

sudo apt upgrade

Install/Setup Lamp server:

sudo apt install tasksel

sudo tasksel install lamp-server

#Set password for mysql

Install additional packages and tools:

sudo apt install php7.0-mbstring php7.0-xml php7.0-curl

# Restart Apache

sudo /etc/init.d/apache2 restart

sudo apt install git curl nodejs-legacy npm librsvg2-dev screen

## Setup Mediawiki

Download Mediawiki

wget https://releases.wikimedia.org/mediawiki/1.28/mediawiki-1.28.2.tar.gz

Unpack and move

tar -xzf mediawiki-1.28.2.tar.gz

sudo mv mediawiki-1.28.2/\* /var/www/html/

Setup wiki through Web Installer

Place generated LocalSettings.php into mediawiki installation folder

## Downloading and Extracting Math Pages

For this installation of Mediawiki we shall be importing the math pages of the simple English Wikipedia Dump. For this purpose, we shall be using the tool WikiFilter developed by Moritz Schubotz to extract the pages containing mathematical formulas.

Download WikiFilter

git clone https://github.com/physikerwelt/wikiFilter

Add line at end of wikiFilter.py

split\_xml( args.file, args.size, args.dir, args.tag, args.template)

Download and extract math pages

mkdir wout

wget https://dumps.wikimedia.org/simplewiki/latest/simplewiki-latest-pages-articles.xml.bz2

./wikiFilter.py -f simplewiki-latest-pages-articles.xml.bz2

## Import dump into Mediawiki

Extract xml from bz2 file

cd wout

bzip2 -d chunk-1.xml.bz2

Import into mediawiki

sudo php /var/www/html/maintenance/importDump.php < chunk-1.xml

sudo php /var/www/html/maintenance/rebuildall.php

## Install Math extension

Download math extension

wget https://extdist.wmflabs.org/dist/extensions/Math-REL1\_28-1097ee7.tar.gz

Extract to extension folder

sudo tar -xzf Math-REL1\_28-1097ee7.tar.gz -C /var/www/html/extensions

Add the following to LocalSettings.php

wfLoadExtension( 'Math' );

Run update.php

sudo php /var/www/html/maintenance/update.php

## Setup Mathoid

Download and install Mathoid

git clone https://github.com/wikimedia/mathoid

cd mathoid

npm install

Run mathoid in screen

screen -S mathoidScreen

nodejs server.js

## Setup Restbase

Download and install Restbase

git clone https://github.com/wikimedia/restbase

cd restbase

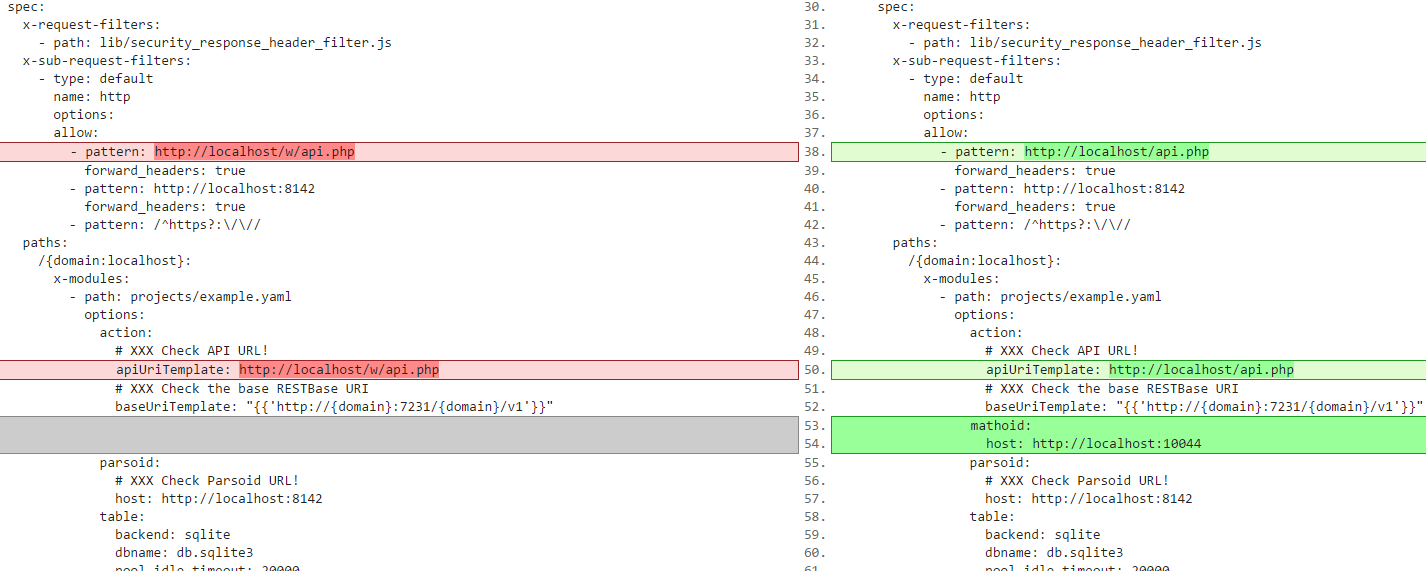
npm install

The following steps describe the minimum changes that are needed to get restbase + mathoid up and running. We assume both mathoid and restbase are running locally. Adapt and adjust files to cover needs.

Copy the config file

cp config.example.yaml config.yaml

Adjust the Config file as shown below.



1: Created using https://www.diffchecker.com

The file below reflects the changes as shown above.

# RESTBase config for small wiki installs

#

# - sqlite backend

# - parsoid at http://localhost:8142

# - wiki at http://localhost/w/api.php

#

# Quick setup:

# - npm install

# If you see errors about sqlite, you might have to `apt-get install

# libsqlite3-dev`.

# - cp config.example.yaml config.yaml

# - double-check and possibly modify lines marked with XXX, then start restbase with

#

# node server

#

# - If all went well, http://localhost:7231/localhost/v1/page/html/Main\_Page

# should show your wiki's [[Main Page]].

services:

- name: restbase

module: hyperswitch

conf:

port: 7231

salt: secret

default\_page\_size: 125

user\_agent: RESTBase

ui\_name: RESTBase

ui\_url: https://www.mediawiki.org/wiki/RESTBase

ui\_title: RESTBase docs

spec:

x-request-filters:

- path: lib/security\_response\_header\_filter.js

x-sub-request-filters:

- type: default

name: http

options:

allow:

- pattern: http://localhost/api.php

forward\_headers: true

- pattern: http://localhost:8142

forward\_headers: true

- pattern: /^https?:\/\//

paths:

/{domain:localhost}:

x-modules:

- path: projects/example.yaml

options:

action:

# XXX Check API URL!

apiUriTemplate: http://localhost/api.php

# XXX Check the base RESTBase URI

baseUriTemplate: "{{'http://{domain}:7231/{domain}/v1'}}"

mathoid:

host: http://localhost:10044

parsoid:

# XXX Check Parsoid URL!

host: http://localhost:8142

table:

backend: sqlite

dbname: db.sqlite3

pool\_idle\_timeout: 20000

retry\_delay: 250

retry\_limit: 10

show\_sql: false

# Finally, a standard service-runner config.

info:

name: restbase

logging:

name: restbase

level: info

Adjust the example.yaml file (/projects/example.yaml)



2: Created using https://www.diffchecker.com

The file below reflects the changes as shown above.

# RESTBase config for small wiki installs

#

# - sqlite backend

# - parsoid at http://localhost:8142

# - wiki at http://localhost/w/api.php

#

# Quick setup:

# - npm install

# If you see errors about sqlite, you might have to `apt-get install

# libsqlite3-dev`.

# - cp config.example.yaml config.yaml

# - double-check and possibly modify lines marked with XXX, then start restbase with

#

# node server

#

# - If all went well, http://localhost:7231/localhost/v1/page/html/Main\_Page

# should show your wiki's [[Main Page]].

# First, we define some project templates. These are referenced / shared

# between domains in the root\_spec further down.

paths:

/{api:v1}:

x-modules:

- spec:

info:

version: 1.0.0

title: Wikimedia REST API

description: Welcome to your RESTBase API.

x-route-filters:

- path: ./lib/normalize\_title\_filter.js

options:

redirect\_cache\_control: '{{options.purged\_cache\_control}}'

paths:

/media:

x-modules:

- path: v1/mathoid.yaml

options: '{{options.mathoid}}'

/page:

x-modules:

- path: v1/content.yaml

options:

purged\_cache\_control: '{{options.purged\_cache\_control}}'

- path: v1/common\_schemas.yaml # Doesn't really matter where to mount it.

/transform:

x-modules:

- path: v1/transform.yaml

options: '{{options}}'

/{api:sys}:

x-modules:

- spec:

paths:

/table:

x-modules:

- path: sys/table.js

options:

conf: '{{options.table}}'

/key\_value:

x-modules:

- path: sys/key\_value.js

/key\_rev\_value:

x-modules:

- path: sys/key\_rev\_value.js

/key\_rev\_latest\_value:

x-modules:

- path: sys/key\_rev\_latest\_value.js

/page\_revisions:

x-modules:

- path: sys/page\_revisions.js

/post\_data:

x-modules:

- path: sys/post\_data.js

/action:

x-modules:

- path: sys/action.js

options: '{{options.action}}'

/page\_save:

x-modules:

- path: sys/page\_save.js

/mathoid:

x-modules:

- path: sys/mathoid.js

options: '{{options.mathoid}}'

/parsoid:

x-modules:

- path: sys/parsoid.js

options:

parsoidHost: '{{options.parsoid.host}}'

response\_cache\_control: '{{options.purged\_cache\_control}}'

/events:

x-modules:

- path: sys/events.js

options: '{{merge({"skip\_updates": options.skip\_updates}, options.events)}}'

options: '{{options}}'

In mathoid.yaml (/v1/mathoid.yaml) adjust uri paths eg. /Wikimedia.org/sys/key\_value/mathoid.mml

The file below reflects the changes with all uris pointing to localhost

# Mathoid - math formula rendering service

tags:

- name: Math

description: formula rendering

paths:

/math/check/{type}:

post:

tags: ['Math']

summary: Check and normalize a TeX formula.

description: |

Checks the supplied TeX formula for correctness and returns the

normalised formula representation as well as information about

identifiers. Available types are tex and inline-tex. The response

contains the `x-resource-location` header which can be used to retrieve

the render of the checked formula in one of the supported rendering

formats. Just append the value of the header to `/media/math/{format}/`

and perform a GET request against that URL.

Stability: [stable](https://www.mediawiki.org/wiki/API\_versioning#Stable).

produces:

- application/json

parameters:

- name: type

in: path

description: The input type of the given formula; can be tex or inline-tex

type: string

required: true

enum:

- tex

- inline-tex

- chem

- name: q

in: formData

description: The formula to check

type: string

required: true

responses:

'200':

description: Information about the checked formula

'400':

description: Invalid type

schema:

$ref: '#/definitions/problem'

default:

description: Error

schema:

$ref: '#/definitions/problem'

x-monitor: true

x-amples:

- title: Mathoid - check test formula

request:

params:

domain: localhost

type: tex

body:

q: E=mc^{2}

response:

status: 200

headers:

content-type: /^application\/json/

x-resource-location: /.+/

cache-control: 'no-cache'

body:

success: true

checked: /.+/

x-request-handler:

- get\_from\_sys:

request:

method: post

uri: /localhost/sys/mathoid/check/{type}

headers: '{{ request.headers }}'

body: '{{ request.body }}'

/math/formula/{hash}:

get:

tags: ['Math']

summary: Get a previously-stored formula

description: |

Returns the previously-stored formula via `/media/math/check/{type}` for

the given hash.

Stability: [stable](https://www.mediawiki.org/wiki/API\_versioning#Stable).

produces:

- application/json

parameters:

- name: hash

in: path

description: The hash string of the previous POST data

type: string

required: true

minLength: 1

responses:

'200':

description: Information about the checked formula

'404':

description: Data for the given hash cannot be found

schema:

$ref: '#/definitions/problem'

default:

description: Error

schema:

$ref: '#/definitions/problem'

x-monitor: false

x-request-handler:

- get\_from\_sys:

request:

method: get

uri: /localhost/sys/mathoid/formula/{hash}

headers: '{{ request.headers }}'

/math/render/{format}/{hash}:

get:

tags: ['Math']

summary: Get rendered formula in the given format.

description: |

Given a request hash, renders a TeX formula into its mathematic

representation in the given format. When a request is issued to the

`/media/math/check/{format}` POST endpoint, the response contains the

`x-resource-location` header denoting the hash ID of the POST data. Once

obtained, this endpoint has to be used to obtain the actual render.

Stability: [stable](https://www.mediawiki.org/wiki/API\_versioning#Stable).

produces:

- image/svg+xml

- application/mathml+xml

- image/png

parameters:

- name: format

in: path

description: The output format; can be svg or mml

type: string

required: true

enum:

- svg

- mml

- png

- name: hash

in: path

description: The hash string of the previous POST data

type: string

required: true

minLength: 1

responses:

'200':

description: The rendered formula

'404':

description: Unknown format or hash ID

schema:

$ref: '#/definitions/problem'

default:

description: Error

schema:

$ref: '#/definitions/problem'

x-monitor: false

x-setup-handler:

- init\_svg:

uri: /localhost/sys/key\_value/mathoid.svg

body:

keyType: string

valueType: string

- init\_mml:

uri: /localhost/sys/key\_value/mathoid.mml

body:

keyType: string

valueType: string

- init\_png:

uri: /localhost/sys/key\_value/mathoid.png

body:

keyType: string

valueType: blob

x-request-handler:

- check\_storage:

request:

method: get

uri: /localhost/sys/key\_value/mathoid.{$.request.params.format}/{$.request.params.hash}

headers:

cache-control: '{{ cache-control }}'

catch:

status: 404

return\_if:

status: '2xx'

return:

status: 200

headers: "{{ merge({ 'cache-control': options.cache-control }, check\_storage.headers) }}"

body: '{{ check\_storage.body }}'

- postdata:

request:

uri: /localhost/sys/mathoid/formula/{request.params.hash}

- mathoid:

request:

method: post

uri: /localhost/sys/mathoid/render/{request.params.format}

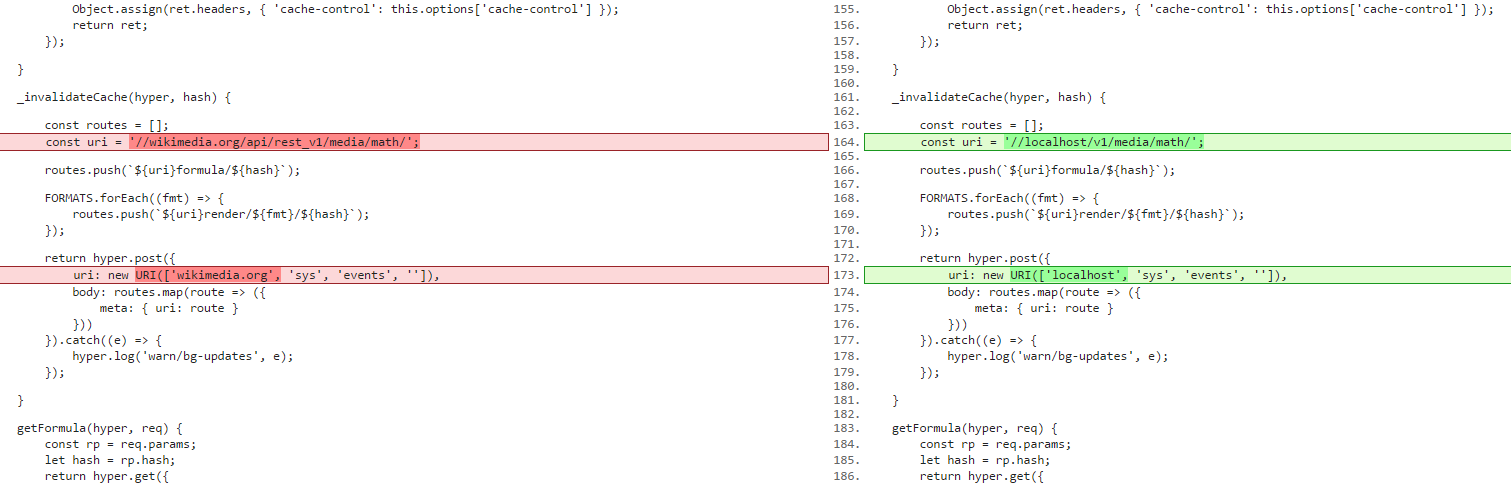
headers:

content-type: application/json

x-resource-location: '{{ request.params.hash }}'

body: '{{postdata.body}}'

Adjust the mathoid.js file (/sys/mathoid.js) as shown below



3: Created using https://www.diffchecker.com

The file below reflects the changes as indicated above.

'use strict';

const P = require('bluebird');

const HyperSwitch = require('hyperswitch');

const URI = HyperSwitch.URI;

const HTTPError = HyperSwitch.HTTPError;

const FORMATS = ['mml', 'svg', 'png'];

class MathoidService {

constructor(options) {

this.options = options;

}

checkInput(hyper, req) {

const rp = req.params;

let hash;

let origHash;

let checkRes;

// start by calculating the hash

return hyper.post({

uri: new URI([rp.domain, 'sys', 'post\_data', 'mathoid.input', 'hash']),

body: { q: req.body.q, type: rp.type }

}).then((res) => {

hash = origHash = res.body;

// short-circuit if it's a no-cache request

if (req.headers && /no-cache/.test(req.headers['cache-control'])) {

return P.reject(new HTTPError({ status: 404 }));

}

// check the post storage

return hyper.get({

uri: new URI([rp.domain, 'sys', 'key\_value', 'mathoid.check', hash])

}).catch({ status: 404 }, () => // let's try to find an indirection

hyper.get({

uri: new URI([rp.domain, 'sys', 'key\_value', 'mathoid.hash\_table', hash])

}).then((hashRes) => {

// we have a normalised version of the formula

hash = hashRes.body;

// grab that version from storage

return hyper.get({

uri: new URI([rp.domain, 'sys', 'key\_value', 'mathoid.check', hash])

});

}));

}).catch({ status: 404 }, () => // if we are here, it means this is a new input formula

// so call mathoid

hyper.post({

uri: `${this.options.host}/texvcinfo`,

headers: { 'content-type': 'application/json' },

body: {

q: req.body.q,

type: rp.type

}

}).then((res) => {

checkRes = res;

// store the normalised version

return hyper.put({

uri: new URI([rp.domain, 'sys', 'post\_data', 'mathoid.input', '']),

headers: { 'content-type': 'application/json' },

body: {

q: res.body.checked,

type: rp.type

}

});

}).then((res) => {

let indirectionP = P.resolve();

hash = res.body;

// add the indirection to the hash table if the hashes don't match

if (hash !== origHash) {

indirectionP = hyper.put({

uri: new URI([rp.domain, 'sys', 'key\_value', 'mathoid.hash\_table',

origHash]),

headers: { 'content-type': 'text/plain' },

body: hash

});

}

// store the result

checkRes.headers = {

'content-type': 'application/json',

'cache-control': 'no-cache',

'x-resource-location': hash

};

return P.join(

hyper.put({

uri: new URI([rp.domain, 'sys', 'key\_value', 'mathoid.check', hash]),

headers: checkRes.headers,

body: checkRes.body

}),

indirectionP,

this.\_invalidateCache.bind(this, hyper, hash),

() => checkRes

);

}));

}

\_storeRenders(hyper, domain, hash, completeBody) {

let idx;

const len = FORMATS.length;

const reqs = new Array(len);

for (idx = 0; idx < len; idx++) {

const format = FORMATS[idx];

// ensure that we have a proper response for a given format

if (!completeBody[format]

|| !completeBody[format].headers

|| !completeBody[format].body) {

return P.reject(new HTTPError({

status: 500,

body: {

type: 'server\_error#empty\_response',

description: `Math: missing or malformed response for format ${format}`

}

}));

}

// construct the request object that will be emitted

const reqObj = {

uri: new URI([domain, 'sys', 'key\_value', `mathoid.${format}`, hash]),

headers: Object.assign(

completeBody[format].headers, { 'x-resource-location': hash }),

body: completeBody[format].body

};

if (format === 'png' && reqObj.body && reqObj.body.type === 'Buffer') {

// for png, we need to convert the encoded data manually

// because we are receiving it wrapped inside a JSON

reqObj.body = new Buffer(reqObj.body.data);

completeBody[format].body = reqObj.body;

}

// store the emit Promise

reqs[idx] = hyper.put(reqObj);

}

// invalidate the cache

reqs.push(this.\_invalidateCache(hyper, hash));

// now do them all

return P.all(reqs).then(() => completeBody);

}

requestAndStore(hyper, req) {

const rp = req.params;

const hash = req.headers['x-resource-location'];

// first ask for all the renders from Mathoid

return hyper.post({

uri: `${this.options.host}/complete`,

headers: { 'content-type': 'application/json' },

body: req.body

}).then(res => // now store all of the renders

this.\_storeRenders(hyper, rp.domain, hash, res.body)).then((res) => {

// and return a proper response

const ret = res[rp.format];

ret.status = 200;

Object.assign(ret.headers, { 'cache-control': this.options['cache-control'] });

return ret;

});

}

\_invalidateCache(hyper, hash) {

const routes = [];

const uri = '//localhost/v1/media/math/';

routes.push(`${uri}formula/${hash}`);

FORMATS.forEach((fmt) => {

routes.push(`${uri}render/${fmt}/${hash}`);

});

return hyper.post({

uri: new URI(['localhost', 'sys', 'events', '']),

body: routes.map(route => ({

meta: { uri: route }

}))

}).catch((e) => {

hyper.log('warn/bg-updates', e);

});

}

getFormula(hyper, req) {

const rp = req.params;

let hash = rp.hash;

return hyper.get({

uri: new URI([rp.domain, 'sys', 'post\_data', 'mathoid.input', hash])

}).then((res) => {

res.headers['x-resource-location'] = hash;

return res;

}).catch({ status: 404 }, () => // let's try to find an indirection

hyper.get({

uri: new URI([rp.domain, 'sys', 'key\_value', 'mathoid.hash\_table', hash])

}).then((hashRes) => {

// we have a normalised version of the formula

hash = hashRes.body;

// grab that version from storage

return hyper.get({

uri: new URI([rp.domain, 'sys', 'post\_data', 'mathoid.input', hash])

}).then((res) => {

res.headers['x-resource-location'] = hash;

return res;

});

}));

}

}

module.exports = (options) => {

const mathoidSrv = new MathoidService(options);

return {

spec: {

paths: {

'/formula/{hash}': {

get: {

operationId: 'getFormula'

}

},

'/check/{type}': {

post: {

operationId: 'checkInput'

}

},

'/render/{format}': {

post: {

operationId: 'requestAndStore'

}

}

}

},

operations: {

getFormula: mathoidSrv.getFormula.bind(mathoidSrv),

checkInput: mathoidSrv.checkInput.bind(mathoidSrv),

requestAndStore: mathoidSrv.requestAndStore.bind(mathoidSrv)

},

resources: [

{

uri: '/{domain}/sys/post\_data/mathoid.input'

}, {

uri: '/{domain}/sys/key\_value/mathoid.hash\_table',

body: { valueType: 'string' }

}, {

uri: '/{domain}/sys/key\_value/mathoid.check',

body: { valueType: 'json' }

}

]

};

};

Run restbase in screen

screen -S restbaseScreen

nodejs server.js

Add the following lines to LocalSettings.php

$wgDefaultUserOptions['math'] = 'mathml';

$wgMathFullRestbaseURL= '<Link to your Restbase>';