The MapOSMatic Rendering API

Render printable maps without a lot of clicking

Hartmut Holzgraefe

hartmut@php.net

SOTM 2022 - Aug. 19, 2022

Speaker notes

Outline

Speaker notes

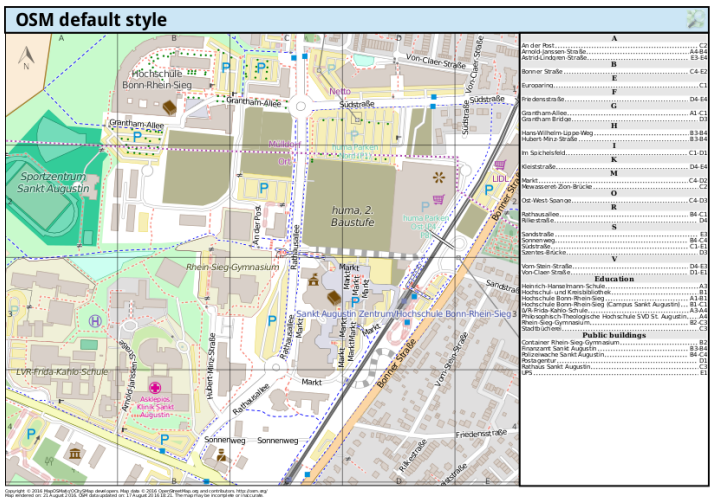
Who am I?

- Hartmut Holzgraefe
- from Bielefeld, Germany
- studied electric engineering and computer science
- OpenStreetMapper since 2007
- Database Support Engineer for MariaDB Corp.
  (and prev. MySQL, Sun, Oracle, SkySQL)

Speaker notes

What is MapOSMatic?

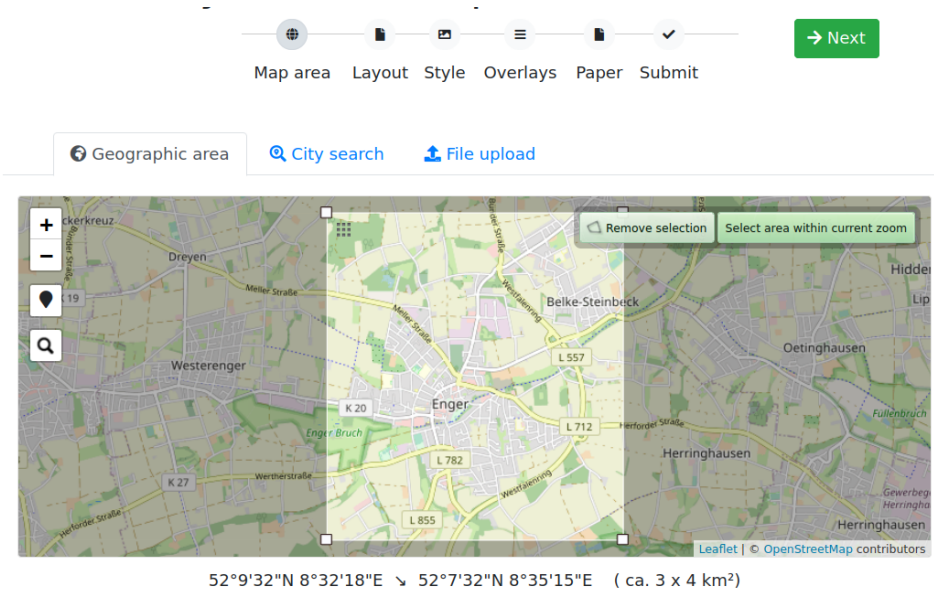
A web frontend and backend renderer infrastructure for rendering large format printable maps in various file formats.



Speaker notes

- Originally started by a team of French students
- Open Source
- Using Python / Django
- I sort of became the new maintainer around 2017

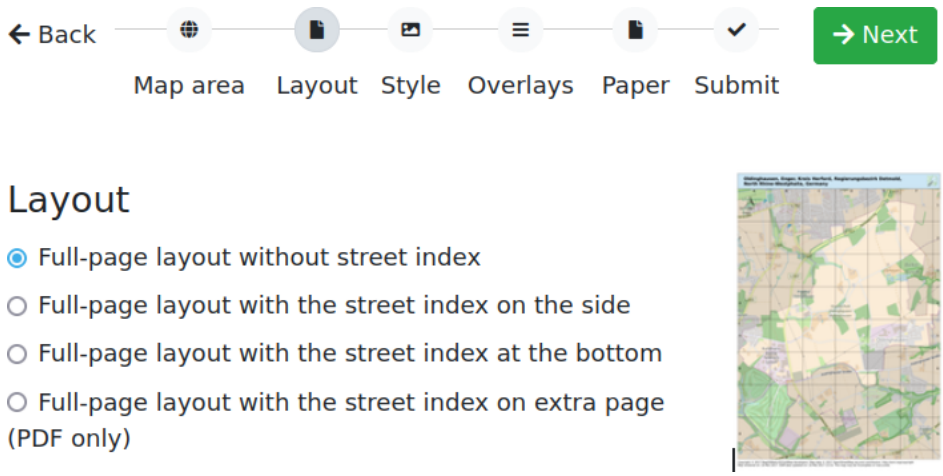
Map Area



Speaker notes

Let's do a quick walkthrough of the web interface workflow.
Starting by selecting the map area to render.

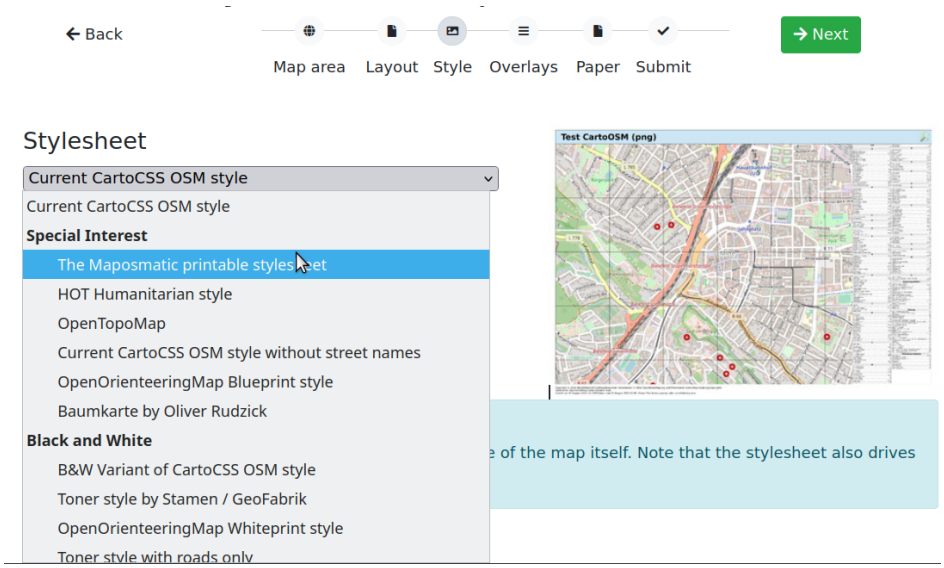
Layout



Speaker notes

Next we can choose from several single and multiple page layouts.

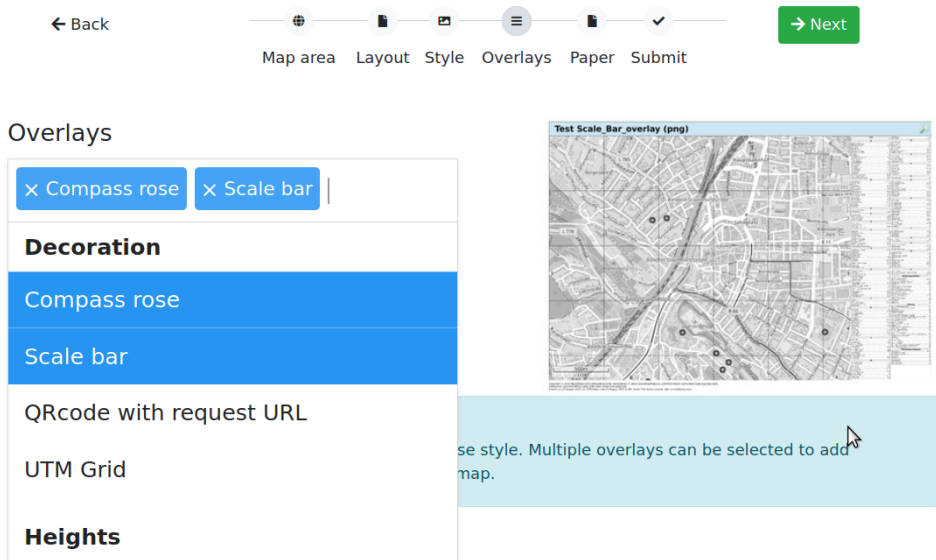
Map Base Style



Speaker notes

As MapOSMatic uses the same Mapnik render library as the OpenStreetMap tile servers we can use the same rendering styles.

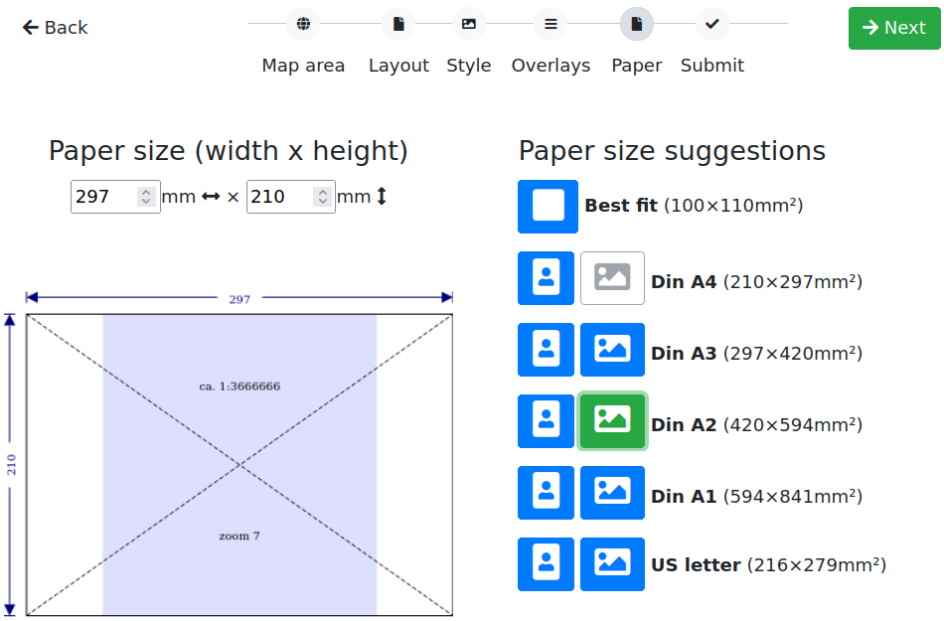
Overlay Styles



Speaker notes

On top of the base map style we can add one or more overlay styles that add extra data on a transparent background so that the base map style still shines through.

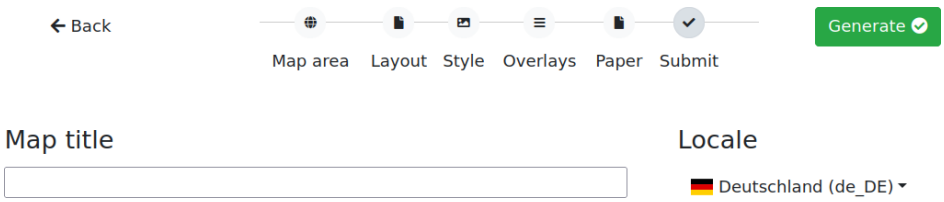
Paper Size



Speaker notes

It is possible to choose from several pre-defined paper sizes, or to provide custom paper width and height.

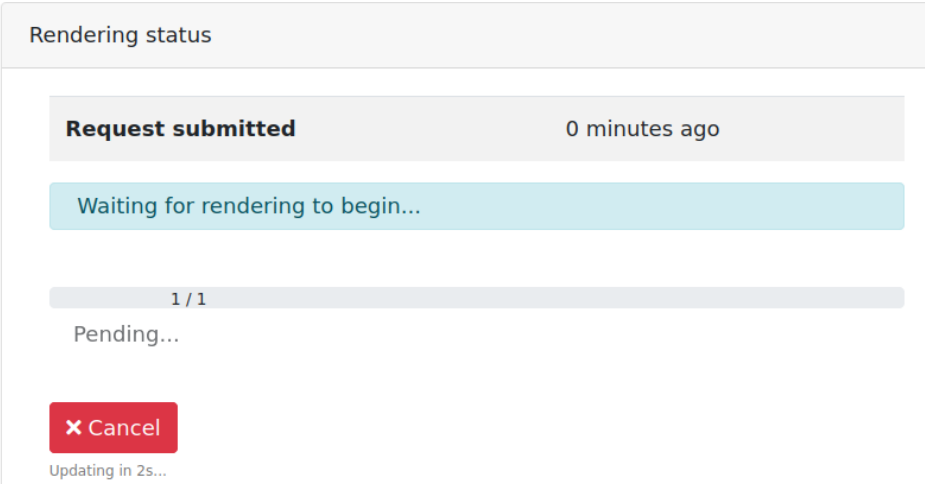
Final Step



Speaker notes

We finish by choosing an optional title text and the language to use for map annotations (not all text snippets may be translated, so English text may still appear in some places)

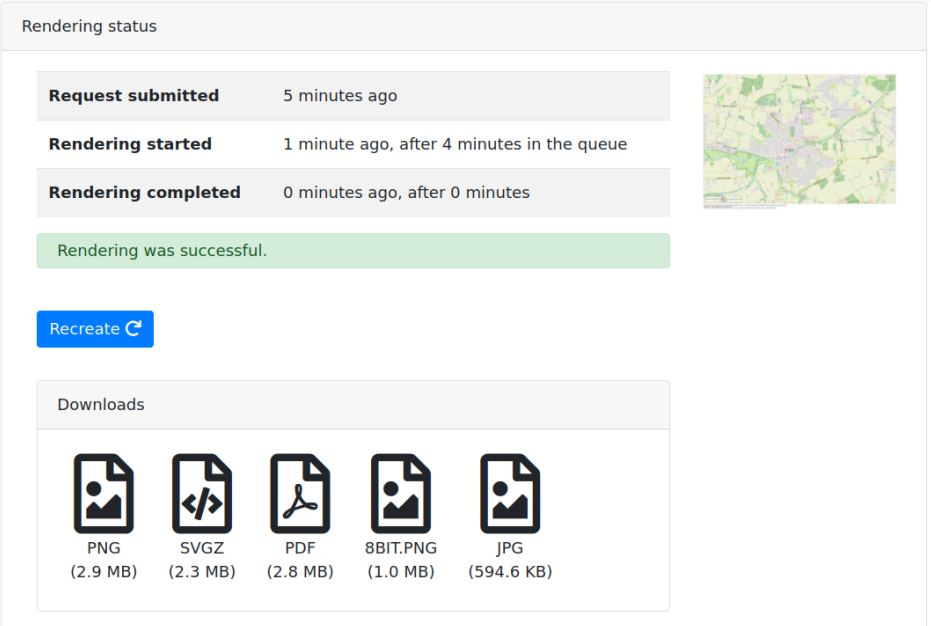
Rendering in Progress



Speaker notes

Now we are sent to a status page that updates itself every 15 seconds ...

Finished



Speaker notes

.. until the rendered results are available for download (or in case of failure: error messages are available)

General API Information

To allow for automated requests without having to click through the user interface a HTTP API has been addded with following properties:

- Request parameters (if any) are passed as JSON
- Results are passed as JSON, too
- Most calls are stateless
- Actual render call is returning state information though
- ... to be used in further calls

Speaker notes

TODO

A simple request

Using the curl tool to submit HTTP requests a most basic rendering request may look like this:

```
curl --form job='{"bbox": [52.0, 8.5, 52.02, 8.52]}' \
     https://api.get-map.org/apis/v1/jobs
```


Speaker notes

Curl takes care of creating a proper HTTP POST request from the parameters given to it.

I'm using it here as it is almost always available on Linux systems and does not require any prior programming or HTTP knowledge.

The First Reply

Returning a status reply like this on success:


```
{
  "id": 230035,
  "queue_size": 11,
  "status": 0,
  "status_msg": "Submitted",
  "files": {},
  "interactive": "https://print.get-map.org/maps/230035",

  "language": "en_US.UTF-8",
  "bbox_bottom": 52.02,
  "bbox_left": 8.52,
  "bbox_right": 8.5,
  "bbox_top": 52.0,
  "layout": "plain",
  "paper_height_mm": 297,
  "paper_width_mm": 210,
  "style": "CartoOSM",
  "title": ""
}
```


Speaker notes

This is slightly edited, the fields are actually returned in alphabetical order.

The reordered output above separates job status information in the upper half from layout information in the lower part.

The First Reply - Key Parts

The status information enlarged:


```
{
  "id": 230035,
  "queue_size": 11,
  "status": 0,
  "status_msg": "Submitted",
  "files": {},
  "interactive": "https://print.get-map.org/maps/230035'
  ...
}
```


Speaker notes

We see that the job has been transmitted with id 230035 and is waiting in the rendering queue with 10 other jobs still ahead of it.

No result files are available yet, and an interactive user may be redirected to the render status page we've seen earlier.

Checking The Status

The job moves closer to the head of the queue:

```
curl https://api.get-map.org/apis/v1/jobs/230035
```

```
{
  "id": 230035,
  "queue_size": 6,
  "status": 0,
  "status_msg": "Submitted",
  "files": {},
  ...
}
```


Speaker notes

Reload the status URL on a regular basis to check on its progress.

Do not do this too frequently though, once per minute should be more than sufficient in most cases.

Checking The Status - Again

```
curl https://api.get-map.org/apis/v1/jobs/230035
```

Now the job is getting rendered:

```
{
  "id": 230035,
  "status": 1,
  "status_msg": "In Progress",
  "files": {},
  ...
}
```


Speaker notes

The job has reached the head of the queue and is now being processed by the rendering daemon.

Checking The Status - Final

And now the job is done and we can retrieve the results:


```
{
  "id": 230035,
  "status": 2,
  "status_msg": "Done",
    "files": {
        "8bit.png": "https://print.get-map.org/results/...",
        "jpg": "https://print.get-map.org/results/...",
        "pdf": "https://print.get-map.org/results/...",
        "png": "https://print.get-map.org/results/...",
        "svgz": "https://print.get-map.org/results/..."
    },
  ...
}
```


Speaker notes

I had to shorten the result URLs here, but you get the idea.

A more complex request

```
curl --form job='{
  "bbox": [52.0, 8.5, 52.02, 8.52],
  "title": "curl test",
  "language": "de_DE.UTF-8",
  "layout": "single_page_index_bottom",
  "style": "OsmBright",
  "overlays": ["ContourOverlay", "MaxspeedOverlay"],
  "paper_size": "Din A1",
  "orientation": "landscape",
  }' \
  https://api.get-map.org/apis/v1/jobs
```


Speaker notes

Lets try a more complex case now, requesting specific layout, styles, and paper size.
Most of these fields may accept different values based on the actual configuration and version of the rendering server.

Next slides will show how to retrieve the possible values.

Page Layouts

https://api.get-map.org/apis/v1/layouts


```
{
  "multi_page": {
    "description": "A multi-page layout.",
    "preview_url": "https://api.get-map.org/media/img/layout/multi_page.png"
  },
  "plain": {
    "description": "Full-page layout without index.",
    "preview_url": "https://api.get-map.org/media/img/layout/plain.png"
  },
 "single_page_index_bottom": {
    "description": "Full-page layout with the index at the bottom.",
    "preview_url": "https://api.get-map.org/media/img/layout/single_page_index_bottom.png"
  },
  "single_page_index_side": {
    "description": "Full-page layout with the index on the side.",
    "preview_url": "https://api.get-map.org/media/img/layout/single_page_index_side.png"
  }
}
```


Speaker notes

Layout choices are not configurable per se, but additional ones may show up in future versions.

Base Layer Styles

https://api.get-map.org/apis/v1/styles


```
{
  "CartoOSM": {
    "annotation": "OpenStreetMap Carto standard style",
    "description": "CartoCSS OSM standard style",
    "preview_url": "https://api.get-map.org/media/img/style/CartoOSM.png"
  },
  "GermanCartoOSM": {
    "annotation": "German OSM style based on OSM Carto",
    "description": "German OSM style",
    "preview_url": "https://api.get-map.org/media/img/style/GermanCartoOSM.png"
  },
  [...]
}
```


Speaker notes

Available base layers depend on the actual render server configuration, on my server instance I try to provide as many different open styles as I can find and get working.

Overlay Styles

https://api.get-map.org/apis/v1/overlays


```
{
  "OpenRailwayMap_Overlay": {
    "annotation": "OpenRailwayMap overlay",
    "description": "OpenRailwayMap rail line overlay",
    "preview_url": "https://api.get-map.org/media/img/style/OpenRailwayMap_Overlay.jpg"
  },
  "Scale_Bar_overlay": {
    "annotation": "",
    "description": "Map scale bar"
    "preview_url": "https://api.get-map.org/media/img/style/Scale_Bar_overlay.jpg"
  },
  [...]
 }
```


Speaker notes

Available overlay style choices also depend on the actual render server configuration.

There are a few 'special' overlays that do not render actual map data but rather add decorations like a compass rose or a scale bar that should always be available as they are implemented in the renderer code itself.

Paper Formats

https://api.get-map.org/apis/v1/styles/paper_formats

```
{
  "Best fit": {
    "height": null, "width": null
  },
  "Din A4": {
    "height": 297, "width": 210
  },
  "US letter": {
    "height": 279, "width": 216
  },
  [...]
}
```


Speaker notes

Available predefined paper size choices are also configurable. You're always free to submit your own 'paper_width' and 'paper_height' in millimeters instead of choosing a size by name.

Import File Support

Like the web frontend the API allows to add files that provide additional data to render on top of the base map.

- Supports GPX, general GeoJSON and Umap exports
- Files can be transmitted as direct uploads
- ... or via external URLs
- Bounding box and titles can be determined automatically

Speaker notes

GeoJSON support is pretty rudimentary and not providing for any kind of styling.

Umap support is a bit better, it should support most of the static styling features, but does not have support for importing any external data files.

GPX Tracks from URL

```
curl --form job='{
          "style": "OsmBright",
          "paper_size": "Din A1",
          "orientation": "portrait",
          "import_urls": [
             "https://get-map.org/example1.gpx",
             "https://get-map.org/example2.gpx"
             ]
          }' \
     https://api.get-map.org/apis/v1/jobs
```


Speaker notes

This is the least complex way to import files, just pass URLs and let the render process download the data locally.

GPX Tracks from local files

```
curl --form job='{ "paper_size": "Din A1",
                   "orientation": "portrait" }' \
     --form file1=@example1.gpx \
     --form file2=@example1.gpx \
     https://api.get-map.org/apis/v1/jobs
```


Speaker notes

It is also possible to directly upload local files.
This is pretty straight forward when using curl using the '@' prefix to add information from local files.

Using other tools or languages your mileage may vary.

PHP Example

```
<?php
require_once 'HTTP/Request2.php';

define('BASE_URL', 'https://api.get-map.org/apis/v1/');
define('GPX_FILE', 'x.gpx');

$data = ["style"       => "OsmBright",
         "paper_size"  => "Din␣A1",
         "orientation" => "portrait"];

$request = new HTTP_Request2(BASE_URL . "jobs");

$request->setMethod(HTTP_Request2::METHOD_POST)
        ->addPostParameter('job', json_encode($data))
        ->addUpload('track', GPX_FILE);

$reply = json_decode($request->send()->getBody());

echo $reply->interactive."\n";
```


Speaker notes

Just to quickly show how things may look in an actual programming language. Most other languages have conceptually similar HTTP libraries ...

Proof-of-concept code only, no error checking etc., just the bare minimum ...

Neighborhood POIs

- Alternative Web Frontend
- Allows for interactive entry of POIs
- Submits actual render request to MapOSMatic
- Forwards user to interactive result page

`https://around.get-map.org/`

Speaker notes

This was my first real use case, originally it would call the command line rendering script directly and users had to wait without getting any status feedback.

Now jobs are submitted to the general rendering queue now, and users are redirected to the interactive status page we've seen earlier.

City Hiking Atlas

This is a proof-of-concept script for now that:

- Takes OSM id of a city
- Retrieves hiking routes via OverPass API
- Submits render requests for each route
- ... using WayMrakedTrails route GPX URLs
- Waits for all requests to complete
- Stitches results together into one PDF

It may become a full interactive application at a later date ...

`https://github.com/hholzgra/maposmatic-hiking-atlas`

Speaker notes

TODO

Cancel submitted requests

The web user interface allows to cancel submitted jobs as long as they are still waiting in the queue.
A similar API call is still missing.

Speaker notes

TODO

Multiple jobs via single request

Less API calls needed when requesting multiple related maps.
Also makes clear that certian jobs are related to each other, and may allow to cancel them all together.

Speaker notes

TODO

Job priorization

Change job handling from “first come, first serve” to a more ‘clever‘ scheduling scheme.

Speaker notes

TODO

User authentication

Limit API access to registered users only.
Also combined with job priorization allows for more fair resoure allocation.

Speaker notes

TODO

Wrapping it up

- Try it out and provide feedback :)
- But try to not overload the server
- Consider to run your own instance for more intensive use cases

Speaker notes

So far this is mostly in "works for me" status.
Try to not submit multiple requests quickly, wait for a request to finish before submitting the next one for now.

You can create your own instance rather easily using the maposmatic-vagrant project.

Questions? Suggestions? Wishes?

Speaker notes

References

API documentation https://print.get-map.org/about/api/

My MapOSMatic Instance https://print.get-map.org/

GitHub Projects

maposmatic web interface
https://github.com/hholzgra/maposmatic

maposmatic render script
https://github.com/hholzgra/ocitysmap

maposmatic vagrant VM https://github.com/hholzgra/maposmatic-vagrant

hiking atlas https://github.com/hholzgra/maposmatic-hiking-atlas

Speaker notes