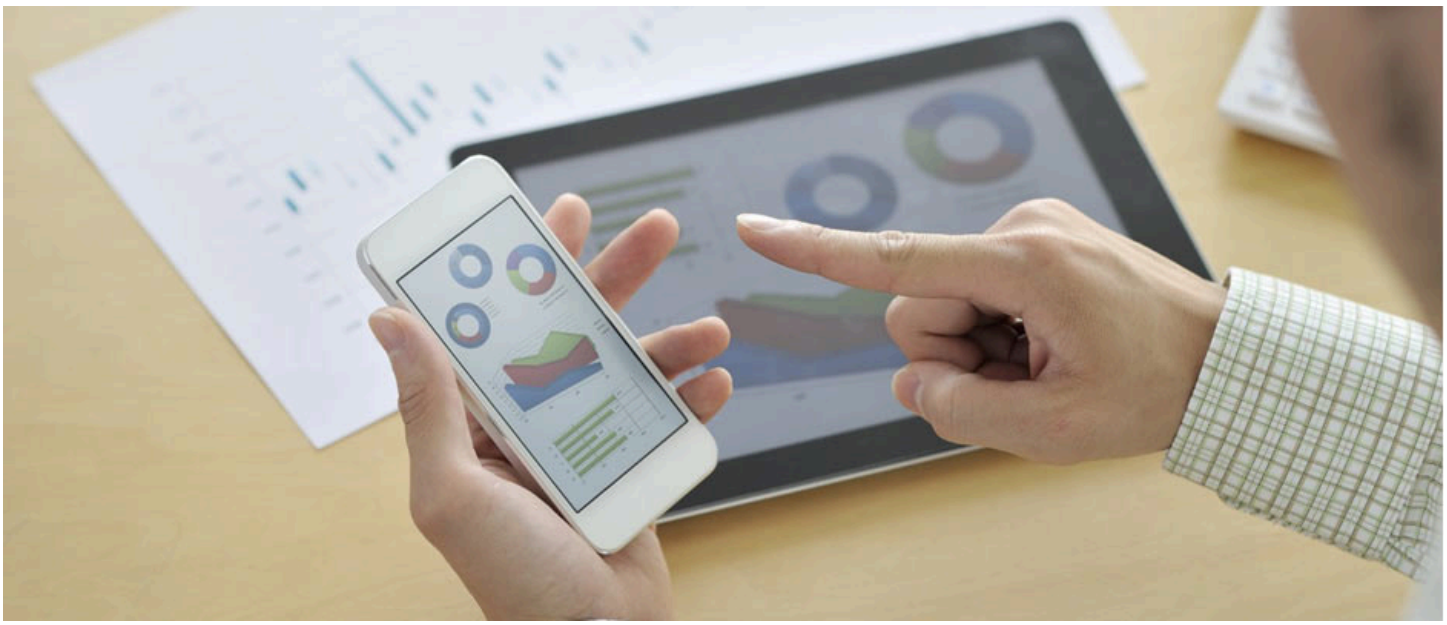
scientiamobile

"The first step in a great mobile experience"



WURFL QUERY API (QUWURFL) GETTING STARTED GUIDE

Support

The ScientiaMobile Enterprise Support Portal is open to all WURFL users, both commercial license holders and evaluation users. It represents the combined knowledge base for the WURFL community. Commercial licensees are invited to post questions in the forum using the account to which their licenses are associated. This may mean faster handling of those posts by ScientiaMobile's personnel.

For commercial license holders, there are tiered support levels to address a variety of business support needs. After logging into your account, commercial licensees with support options can access the Enterprise Support portal to post tickets. These tickets will receive expedited attention.

To inquire about support plans, use our License Inquiry or our General Inquiry form.

Update Notifications

If you would like to be notified of our API updates, major data updates, and other technical changes, please subscribe to our ScientiaMobile Announcements list.

scientiamobile

www.scientiamobile.com

Tel +1.703.310.6650

E-mail: sales@scientiamobile.com

Copyright © 2026 ScientiaMobile, all rights reserved. WURFL Cloud, WURFL OnSite, WURFL and, InFuze WURFL InSight and respective logos are trademarks of ScientiaMobile. Apache is the trademark of the Apache Software Foundation. NGINX is the trademark of Nginx Software Inc. Varnish is the trademark of Varnish Software AB

WURFL Query API (QuWURFL) Getting Started Guide

The WURFL Query API, or QuWURFL for short, lets developers query WURFL using device names and filters on capability values.

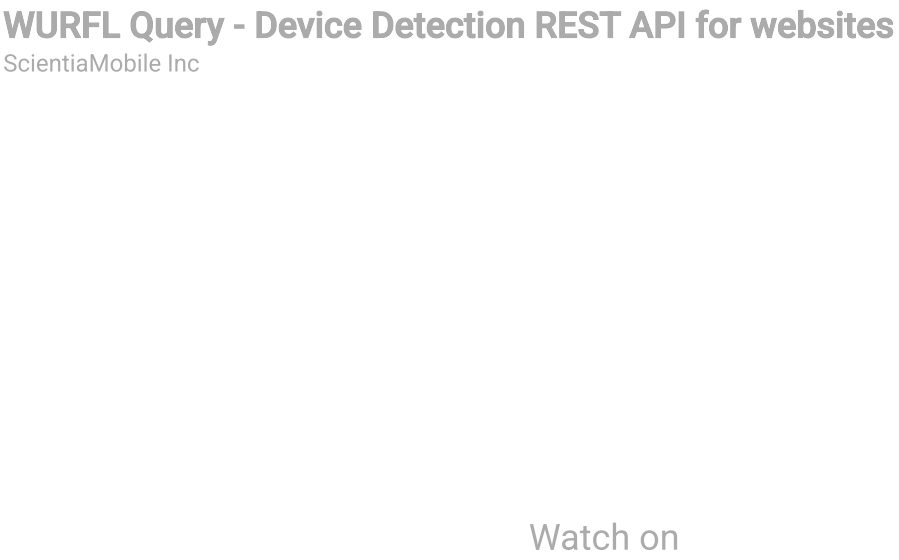
Invoking the API will return "ranges" of devices or other WURFL data structures (in JSON format) that match the query.

This novel approach to using WURFL represents a significant departure from the traditional method. In the past, programmers needed an HTTP request or, at the very least, a User-Agent string to query WURFL for device properties. The WURFL Query REST API enables your website or your applications to programmatically access WURFL information based on device names and features.

For example, applications can now access the following device data in real time:

- Retrieve a list of all devices from a specific make (brand)
- Find a device by model name or marketing name (using model_name or marketing_name capabilities, respectively)
- Find the device "SM T113"
- Find the device with the marketing name "Galaxy Tab 3 Lite"
- Find a list of devices based on their release date
- Find all iOS tablets that support OS versions 9.1 through 10.4
- Combine any of the queries above as needed.

The product is designed for easy integration with other WURFL products, including **WURFL.js Business Edition**.



Video: *This video will give you high level understanding of the WURFL Query REST API before you delve into the details.*

Quick aside: APIs and REST APIs

This document assumes that you are familiar with the Application Programming Interface (API) concepts. If you heard of WURFL, you probably already know that it's an API to access mobile device information in various scenarios, including batch processing of data and real-time analysis.

When the term API was first introduced, it primarily referred to the use of a programming library that implemented specific functions, shielding the user (i.e., the engineer adopting that library) from knowing more than the bare minimum about what was happening behind the scenes.

As the world has increasingly moved to the Cloud, the term API tends to be conflated with REST API for practical purposes.

REST stands for REpresentational State Transfer and was created by computer scientist Roy Fielding. **A REST API (also known as RESTful API)** is a "web API", meaning that instead of calling a local library's function, a program can access those functions remotely through HTTP calls. These calls follow certain conventions to interact with RESTful web services.

Most WURFL products come in the form of an API, i.e., a series of functions that you can access locally on your computer. However, that's not the only incarnation of WURFL, as other ScientiaMobile SaaS products require programs to interact with our servers through HTTP to access the latest and greatest device information and API logic.

QuWURFL is a REST API

QuWURFL - the WURFL Query REST API - does not rely on the assumption that you have HTTP information readily available. Instead, it allows you to query WURFL using the values of device capabilities as query parameters. Furthermore, you can retrieve devices through their brand, model, and marketing names through various strategies tailored to address common use cases, which we will illustrate in the following sections.

Note: QuWURFL utilizes the Openapi 3.0.0 specification format — an industry standard for defining REST APIs that many developers are likely to be familiar with. You can access the API definition at: **https://api.query.wurflcloud.com/**

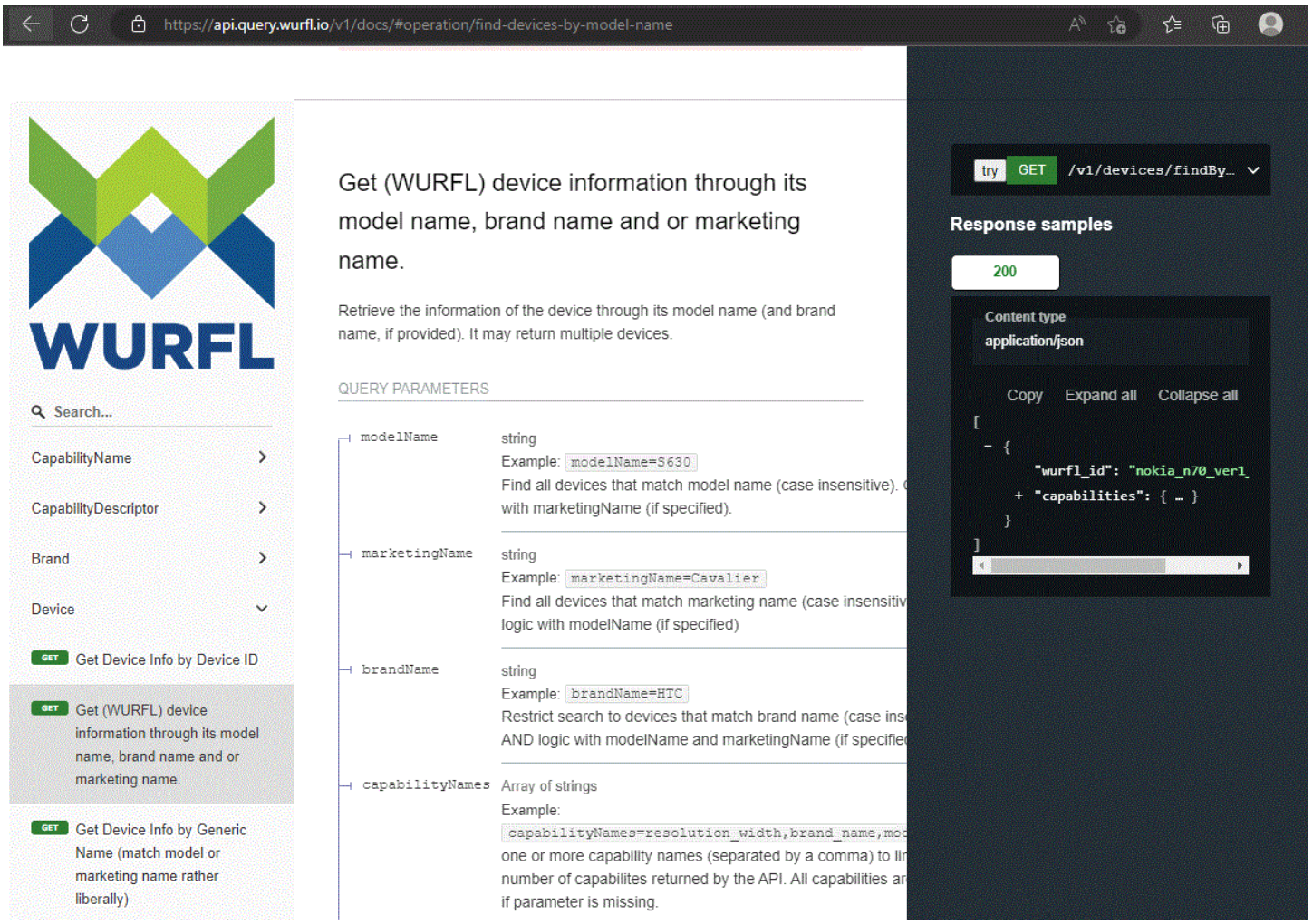


Figure: *Official QuWURFL API definition, available at: https://api.query.wurflcloud.com/*

This document provides an overview of the REST endpoints that let you query device information.

Authenticating QuWURFL requests

QuWURFL is available to ScientiaMobile customers who have licensed the service. You will find your API keys in your customer vault once you log in on the www.scientiamobile.com website (you will also be able to revoke keys and generate new ones). With those keys, you will be able to access QuWURFL with almost anything that speaks HTTP:

```
$ curl "https://api.query.wurflcloud.com/v1/capabilityNames" --header 'Authorization:
Basic MTE5OTAwOkUxSXFMQ1hJd7FqQXZobWY4MTR4R3FBZHpkRHRSQ0tN'
[
    "brand_name",
    "can_assign_phone_number",
    "device_os",
    "device_os_version",
    "form_factor",
    "is_google_glass",
    "is_smartphone",
    ...
```

The capabilities available to you depend on your account.

Telling you how to go from an API key to the Basic Authentication token is outside of the scope of this document, but here you go:

```
$ echo -n '112200:E3IqLCXIwajAvhi4f812xGqLbzdDtRCKM' | base64
MTE5OTAwOkUxSXFMQ1hJd7FqQXZobWY4MTR4R3FBZHpkRHRSQ0tN
```

QuWURFL Resources (aka “Nouns”)

Over the last decade there has been significant standardization of conventions used when designing and implementing REST APIs. Without going into details that would be outside the scope of this document, REST API designers typically begin by identifying the "resources" (i.e., the nouns representing clear and recognizable entities) in their domain.. Those resources are the entities that resonate with potential API users — the ‘objects’ that API users will want to create, retrieve, update and delete (#CRUD).

Note: *Products, Students, Buildings, Pets and Invoices are all good examples of “resources” that illustrate - in literature - how REST APIs are built these days. Selecting the right nouns from the outset helps guide the definition of specific endpoints that act on those resources. Actions performed on the actual resources are then inherited from the HTTP Spec (GET, POST, UPDATE, DELETE). More fine-grained action can then be tacked on to those endpoint URLs to handle more specific use-cases that relate. Any recent tutorial on the design of REST APIs (**this one for example**).*

QuWURFL defines the following entities:

- `capability name`
  - Name of WURFL capability.
- `capability descriptor`
  - Name, type, description of the capability
  - Type is one of: `boolean, enumerable, numeric, string`
  - Regardless of the Type, the API will always return strings. The responsibility of casting the string to the relevant type lies with the client consuming the API. For example, `"is_tablet":"false"` would be cast to a JS boolean false for ease of consumption in JavaScript.
  - Example:

    ```
    {"name": "resolution_width",
    "type": "numeric",
    "description": "This field represents the screen width expressed in pixels."}
    ```

- `brand`.

    - Name of device brand.
    - Example: `Nokia`, `Samsung`, `Apple`,....
- `device` (WURFL device)
    - WURFL ID: `"wurfl_id":"oneplus_cph2389_ver1"`
    - List of capabilities

```
{
  "brand_name":"OnePlus",
  "resolution_width":"960",
    …
}
```

Note: A "range" of devices - a common response from the `/v1/devices/*` endpoints - is simply a list of devices.

Not only are these resources guaranteed to resonate with anyone who is familiar with WURFL, but they are also virtually guaranteed to make sense to everyone technologically savvy.

QuWURFL endpoints that return those resources directly reference those names, as specified in the `wurfl-query.yaml` file:

```
$ grep "/v1/" wurfl-query.yaml | sed s/\'//g | sed s/://
  /v1/capabilityNames
  /v1/capabilityDescriptors
  /v1/capabilityDescriptors/{capabilityName}
  /v1/brands
  /v1/brands/findByDeviceCount/{top}
  /v1/brands/autofillBySubstring
  /v1/devices/{deviceId}
  /v1/devices/findByModelName
  /v1/devices/findByGenericName
  /v1/devices/search
  /v1/devices/autofillBySubstring
  /v1/devices/notify/{channel}
```

Note: if you are not familiar with OpenAPI definitions of REST APIs, you can **read this**. QuWURFL has adopted the OpenAPI 3.0.3 format for its definitions.



Figure: *Top part of the OpenAPI 3.0 YAML file that specifies the WURFL Query REST API.*

We recommend that you read this document to get a general understanding of the service, but we encourage you to refer to **https://api.query.wurflcloud.com/** for the documentation that is directly derived from the OpenAPI YAML specification. The documentation will also allow you to build requests against QuWURFL.

WURFL Query REST API: Demo Dashboard

While describing the API is theoretically enough to enable usage in any application, showing those endpoints in action in a real application is a powerful way to make a point about QuWURFL's capabilities.

We built a single page application using Vue.js 3 that demonstrates how QuWURFL can be used in practice. Endpoints were accessed using the **Axios** NPM library.

The demo dashboard is available at **https://demo.query.wurfl.io/**

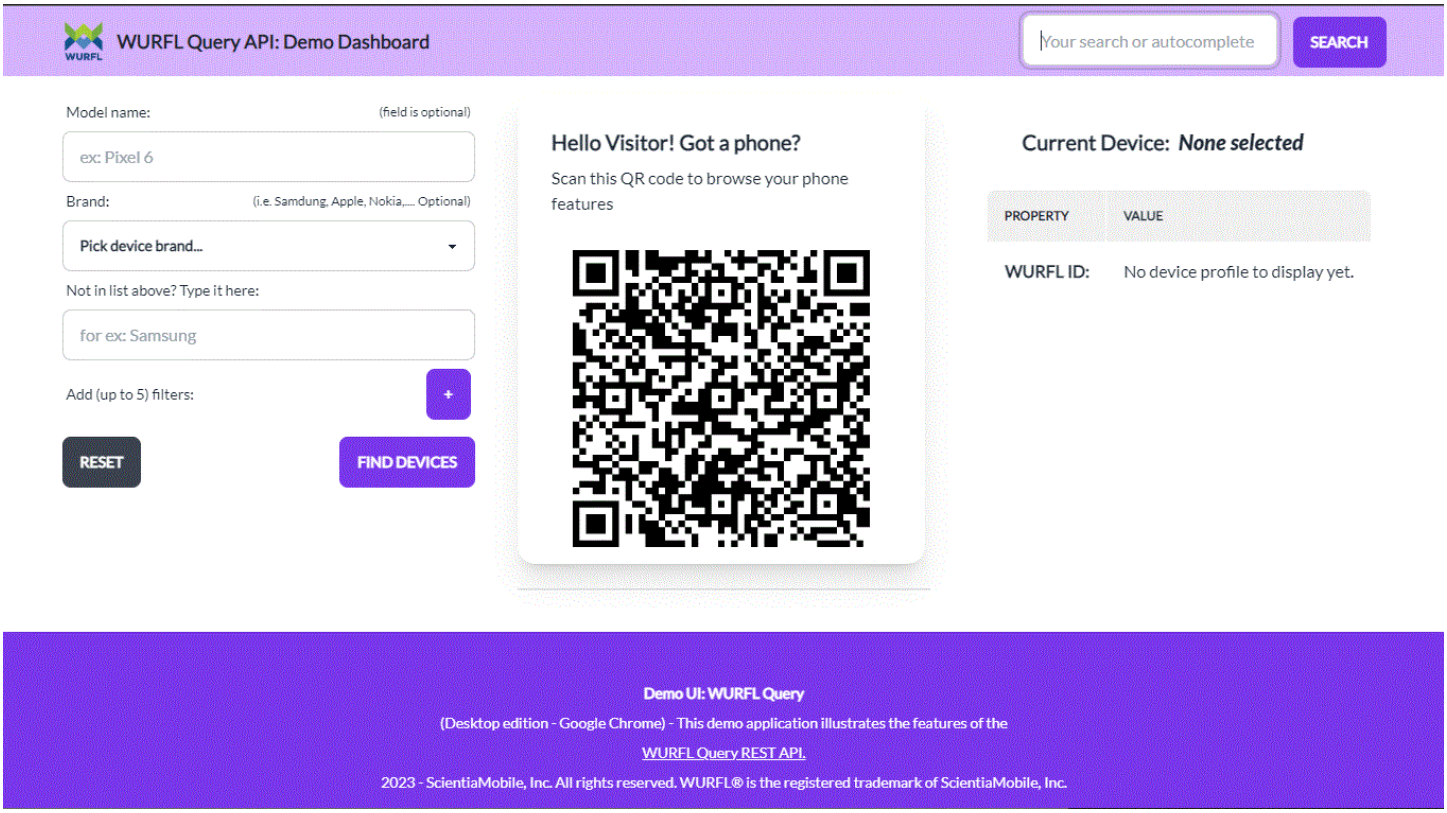


Figure: *Demo application that utilizes QuWURFL endpoints (https://demo.query.wurfl.io/)*

Resource: CapabilityName

There is always one endpoint that returns capability names.

ENDPOINT: `/v1/capabilityNames`

Method: GET

This endpoint returns a JSON array of strings representing supported capabilities. It is useful if you need to populate a list of available capabilities dynamically (rather than hard-wiring capability names in your UI).

Usage Example

To show how you can invoke this endpoint, we use `curl`, but you can of course use any other library that builds HTTP requests and receives HTTP responses in any programming language.

```
$ curl 'https://api.query.wurflcloud.com/v1/capabilityNames' --header 'Authorization:
Basic <your-token>'
[
    "brand_name",
    "model_name",
    "is_wireless_device",
    "device_os",
    "mobile_browser",
    "mobile_browser_version",
    "device_os_version",
    "pointing_method",
    "release_date",
    "marketing_name",
    "can_assign_phone_number",
    "is_tablet",
    "xhtml_support_level",
    "preferred_markup",
    "resolution_width",
    "resolution_height",
    "physical_screen_width",
    "physical_screen_height",
    "is_smarttv",
    "ux_full_desktop",
    "is_smartphone",
    "form_factor"
]
```

This is not the only way to trigger an endpoint response. The **Postman REST client** is a popular (and powerful) free tool to build requests to a REST endpoint and analyze the responses.

If you are ok with something less sophisticated that will still do the job, our online documentation offers a handy ‘Try” button next to every endpoint entry:

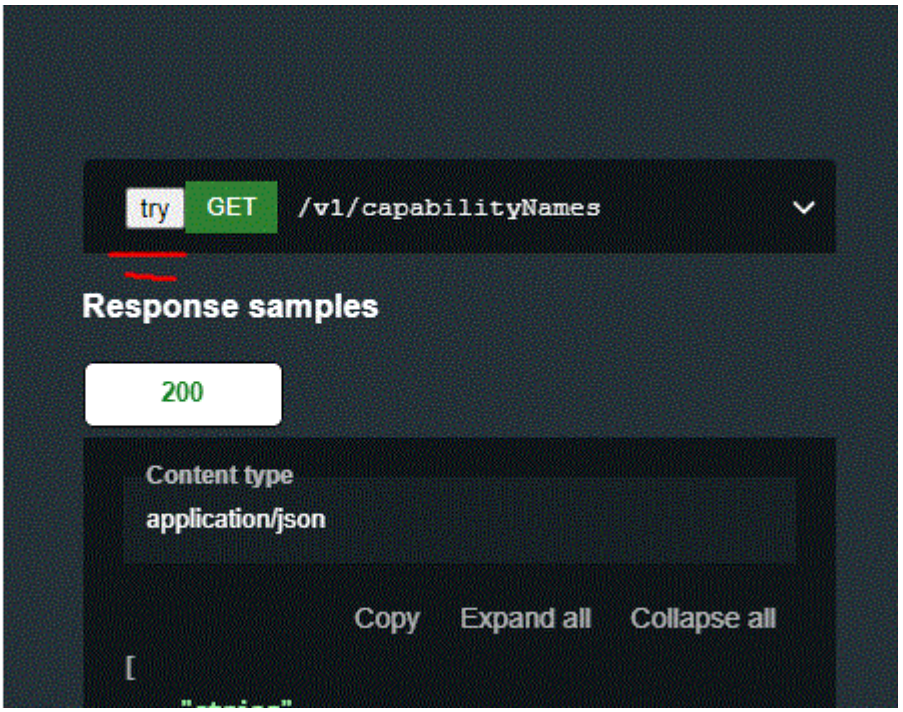


Figure: *The Try button next to the endpoint will open up a form that submits against the endpoint.*

Clicking on the Try button will open up an HTML form that will submit against that endpoint:

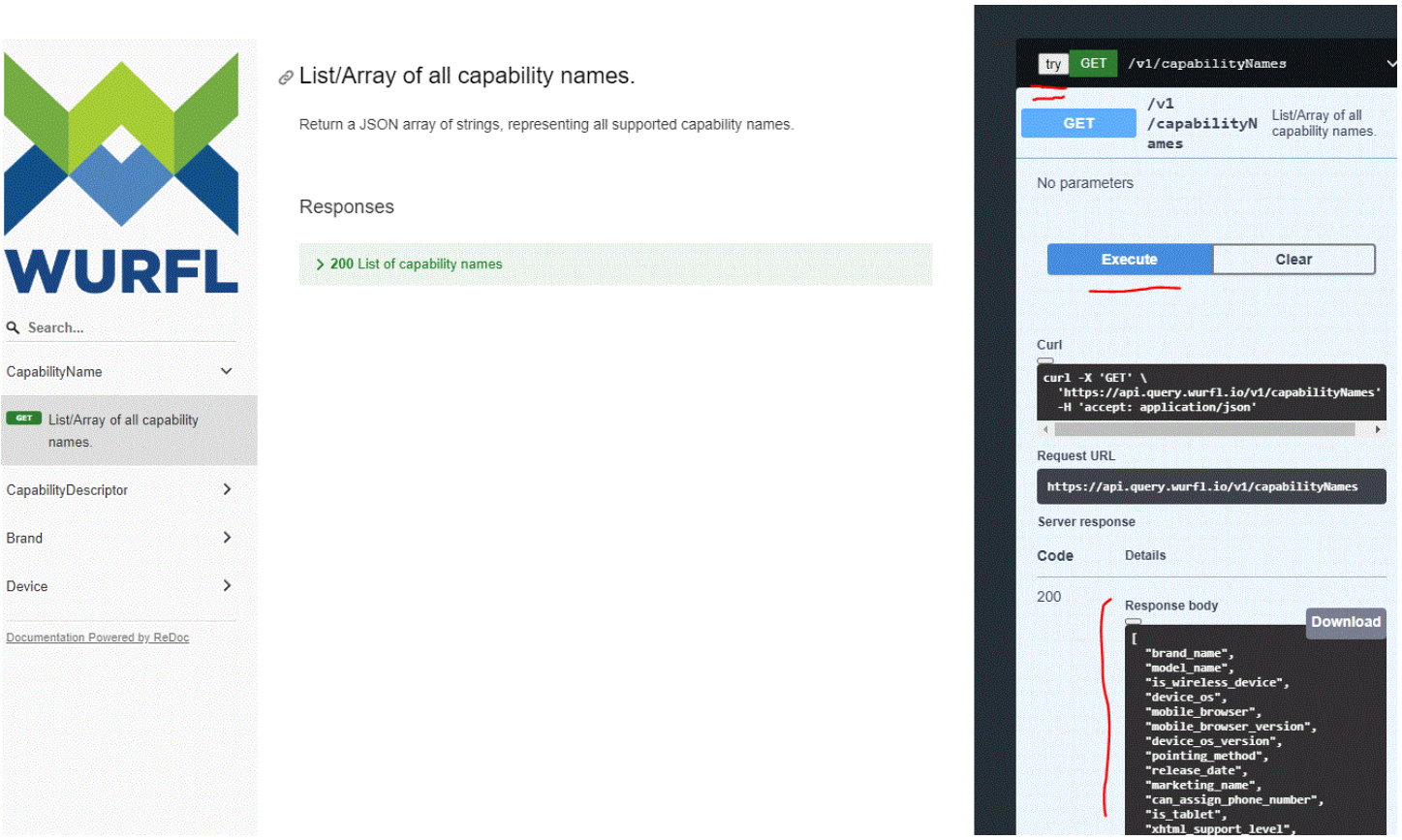


Figure: *You can trigger an endpoint and observe the returned data directly from the documentation page at https://api.query.wurflcloud.com/*

Usage in Demo Dashboard

Most endpoints have been used in the demo dashboard to illustrate the respective usages. /v1/capabilityNames is no exception. The UI that lets you add filters to build a query will rely on this endpoint to suggest capability names:

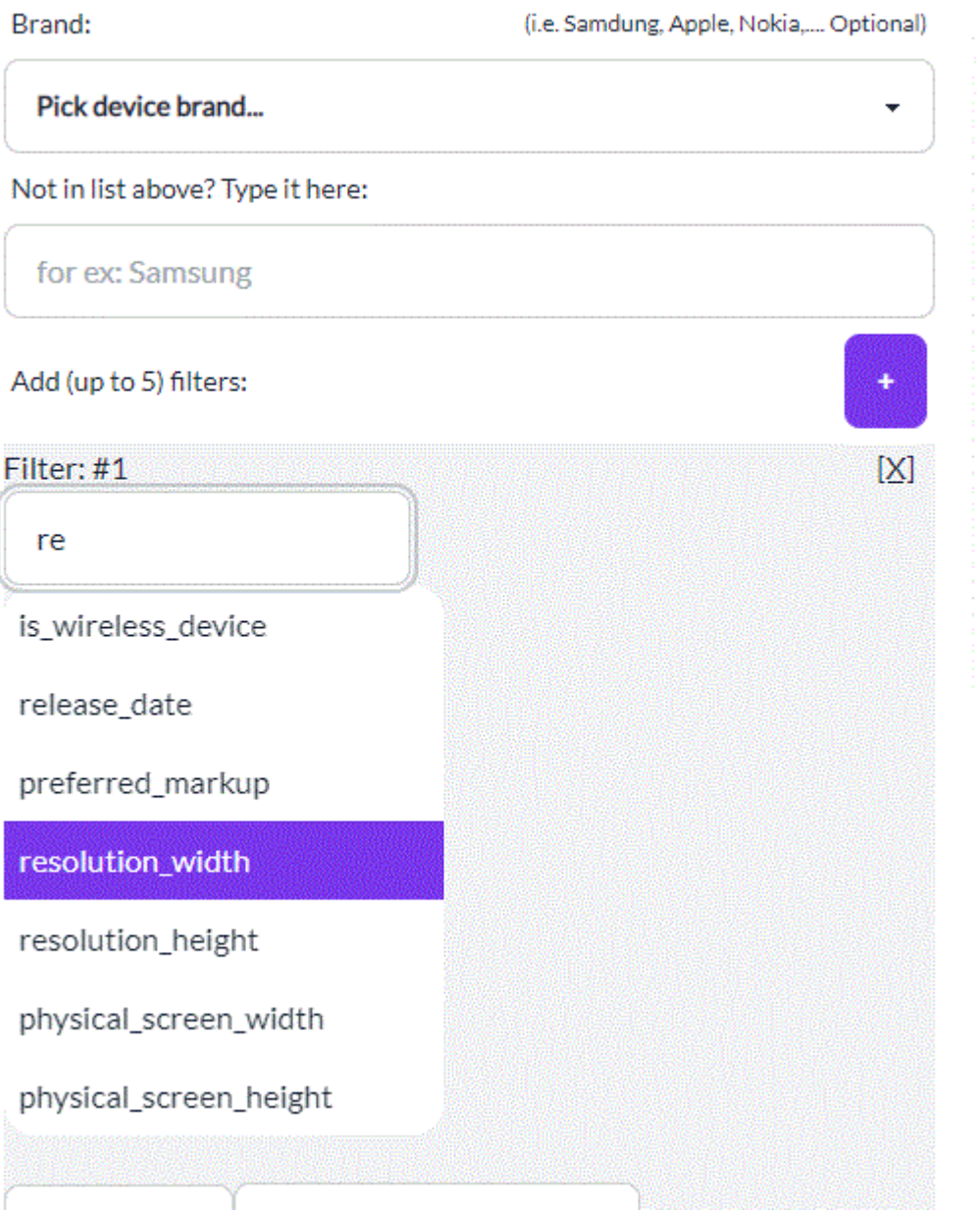


Figure: *capability name autofill as it appears in the demo application.*

```
console.log(`Retrieving list of supported capas at ${this.realBaseUrl}/v1/capabilityNames`);
axios({
  method: 'get',
  url: `${this.realBaseUrl}/v1/capabilityNames`
})
.then(res => {
  let capaList = res.data;

  this.setAvailableCapabilities(capaList);
})
.catch(err => {
    // error
    console.log(err);
    this.errors.push(err)
})
```

```
let item = this.filters[counter];
// don't trigger autocomplete unless you have at least two chars
// console.log("capa field " + counter + " : " + this.filters[counter].capaname.length)
if (this.filters[counter].capaname.length < 2) {
  return;
}

item.results = this.availableCapabilities.filter(capastring => capastring.toLowerCase().
indexOf(item.capaname.toLowerCase()) > -1);
item.isAutoCompleteOpen = true;
```

Figure: *Code snippets that show how the endpoint is used in JavaScript (Vue.js)*

We will show how other endpoints are used elsewhere in the demo application.

Resource: CapabilityDescriptor

Endpoints related to this resource are useful for introspection..

ENDPOINT: `/v1/capabilityDescriptors`

Method: GET

This endpoint returns a JSON array of objects representing supported capabilities. It may be useful for introspection (i.e. discovering what type a capability is and also obtaining a short english description).

Parameters:

- `fields`. Whether we only want the capability type, the description or both (default if parameter is missing)

Usage Example

```
$ curl "https://api.query.wurflcloud.com/v1/capabilityDescriptors?
fields=type,description" --header 'Authorization: Basic <your-token>'
[
    {
        "capability_name": "brand_name",
        "type": "string",
        "description": "Brand (ex: **Nokia**)"
    },
    {
        "capability_name": "model_name",
        "type": "string",
        "description": "Model (ex: **N95**)"
    },
    {
        "capability_name": "is_wireless_device",
        "type": "boolean",
        "description": "Tells you if a device is wireless or not. Specifically a
mobile phone or a PDA are considered as wireless devices, but a desktop PC or a
laptop are not."
    },
    {
        "capability_name": "device_os",
        "type": "enumerable",
        "description": "Information about hosting OS."
    },
    :
]
```

ENDPOINT: `/v1/capabilityDescriptors/{capabilityName}`

Method: GET

Return a JSON object with information about a specific capability.

Parameters:

- `fields`. Whether we only want the capability type, the description or both (default if parameter is missing)

Usage Example

```
$ curl "https://api.query.wurflcloud.com/v1/capabilityDescriptors/resolution_width?
fields=type,description" --header 'Authorization: Basic <your-token>'
{
    "capability_name": "resolution_width",
    "type": "numeric",
    "description": "This field represents the screen width expressed in pixels."
}
```

The endpoint will return an HTTP 404 if the capability is not found.

Resource: Brand

Query device brands available in WURFL.

ENDPOINT: `/v1/brands`

Method: GET

Endpoint returns a JSON array of strings representing all device brands found in WURFL. As list will be very long - there are 2500 brands in WURFL - , you may want to paginate it using the `offset` and `limit` query parameters.

Parameters:

- `offset`. Element position in list that indicates beginning of 'page'
- `limit`. Number of elements per 'page'

Usage Example

The following example will pick 10 brand names out of a list of 2500 starting from element at position 350.

```
$ curl "https://api.query.wurflcloud.com/v1/brands?offset=350&limit=10" --header
'Authorization: Basic <your-token>'
[
    "Cellution",
    "Celmi",
    "Cendyne",
    "Centric",
    "Century Star",
    "Challenger",
    "ChangJia",
    "ChangJiang",
    "Changhong",
    "Cheer"
]
```

ENDPOINT: `/v1/brands/findByDeviceCount/{top}`

Method: GET

This endpoint is a specialization of the previous one (`/v1/brands/`). Not all brands are equally important. There are major manufacturers and manufacturers that have released just a few devices. The `{top}` parameter is an integer that indicates how many of the top brands should be returned by the endpoint. This ranking is calculated by counting the number of devices in WURFL that have that brand_name. This count can be used as a proxy for a brand's popularity.

Usage Example

The following example will retrieve the top 10 brands by "popularity".

```
$ curl "https://api.query.wurflcloud.com/v1/brands/findByDeviceCount/10" --header
'Authorization: Basic <your-token>'
[
    "Samsung",
    "Apple",
    "LG",
    "Motorola",
    "ZTE",
    "Huawei",
    "Nokia",
    "Alcatel",
    "HTC",
    "DoCoMo"
]
```

Usage in Demo Dashboard

The UI will show the top 20 brands by popularity. Here is how it's done.

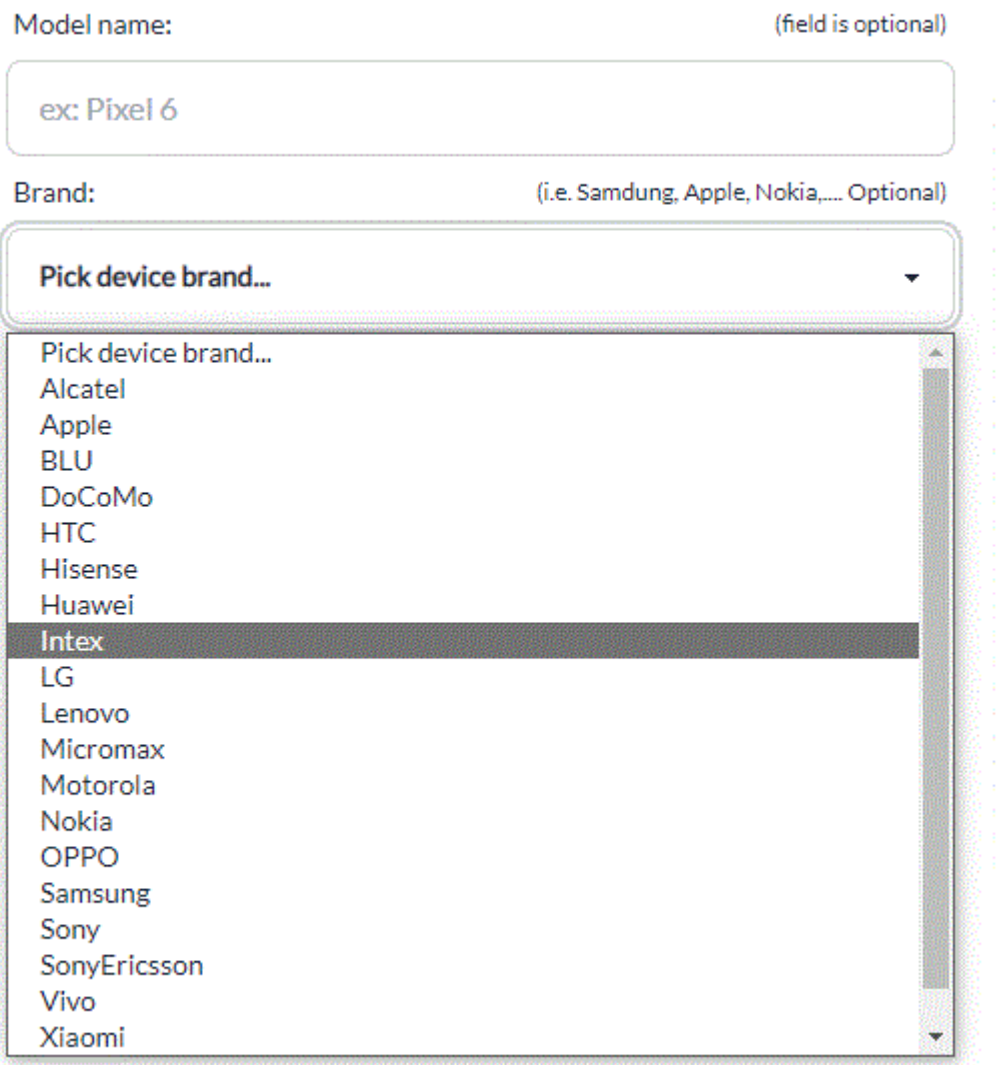


Figure: *Display list of popular brands in demo application.*

```
// Fetches posts when the component is created.
created() {
  this.resetHTTPErrors();
  axios.get(`${this.realBaseUrl}/v1/brands/findByDeviceCount/20`)
    .then(response => {
      // JSON responses are automatically parsed.
      this.posts = response.data.sort()
    })
    .catch(err => {
      console.log("Logging server ERROR: " + err)
      this.setHTTPErrorMessage(err)
    })
},
```

```
<select id="brand_pick" name="brand_pick" v-model="formdata.brand_pick" v-if="posts &&
posts.length" class="select select-bordered w-full">
  <option value="Null" selected><b>Pick device brand...</b></option>
  <option v-for="post of posts" :key="post" :value="post">{{post}}</option>
</select>
```

Figure: *Example of retrieval of popular brands with JavaScript and Vue.js.*

ENDPOINT: `/v1/brands/autofillBySubstring`

Method: GET

This endpoint returns a list of brands that match a substring (case insensitive match). It is handy in web UIs and dashboards. End-users just need to enter a few characters and the endpoint will return a list of suggested brands:

Parameters:

- `key (*)`. Find all brand names in WURFL that match parameter (case insensitive).
  - `key=samsung`
- `maximum`. Limit/Extend maximum number of expected matches (default 10)
  - `maximum=5`

Usage Example

```
$ curl "https://api.query.wurflcloud.com/v1/brands/autofillBySubstring?key=sam" --
header 'Authorization: Basic <your-token>'
[
    "SAMART",
    "Samsung",
    "Samwell",
    "Samzong"
]
```

Resource: Device

Endpoints that return devices are the “main course” offered by QuWURFL. Unsurprisingly, there are many ways to look for devices. QuWURFL supports specialized endpoints to:

- Retrieve specific devices whose WURFL ID is known.
- Selecting a range of devices (DeviceRange) based on their features, i.e. testing conditions on capability values to determine if a device should be selected or not.
- Selecting devices based on their names.

In turn, this last category includes multiple endpoints as we need to cover for different situations.

The user that requests the data may be a professional who has a good grasp of what the brand and model name of a device are, and can build a very specific query for that device.

Other times, the use-case may be to make sense of end-user input, i.e. offer website visitors support to identify their mobile device when an approximate recollection of the device name is all they have. To add to that, the difference

between model name (say `KB2005`) and marketing name (`8T`) may be obscure. For those cases, some endpoints are forgiving and search both fields irrespective of capitalization (and without being picky of certain symbols - such as dashes and spaces - that would normally interfere with a verbatim search).

Making sense of unstructured strings that conflate the brand, the model name and/or the marketing name is also one of the supported use cases.

Of course, QuWURFL may return bulky device ranges (i.e. long lists of devices) in some cases. To handle this, endpoints support pagination and aggregation to make long device ranges easier to manage on the client side.

- Endpoints that can return potentially large lists of elements, support pagination, i.e. the ability to split up the list in pages starting from an element position of our choice, and deciding the number of elements that each page should have.
- A device make and model can still identify multiple WURFL devices that represent variations of the same model across geographies, custom firmware subversions and other minor aspects. By default, device endpoints will "aggregate", i.e. return only one representative for each device along with a sublist of WURFL IDs that account for all other devices (should one need to access them). This behavior can be overridden by setting the `expand` query parameter to true to request the full list of devices with all the capabilities expanded for each device.

Supporting instantaneous searches among WURFL devices is a-nice-to-have. One of the endpoints freely errs on the side of matching a lot of device names,... but fast. You can use this feature to provide iTunes-que autofills to users of your application: users will select the right one - out of the many that are suggested - with one click.

ENDPOINT: `/v1/devices/{WURFL ID}`

{#endpoint-v1-devices-wurfl-id}

METHOD: GET:

Endpoint retrieves the device with the given WURFL ID. All WURFL capabilities are returned, unless a specific subset is selected through the `capabilityNames` attribute.

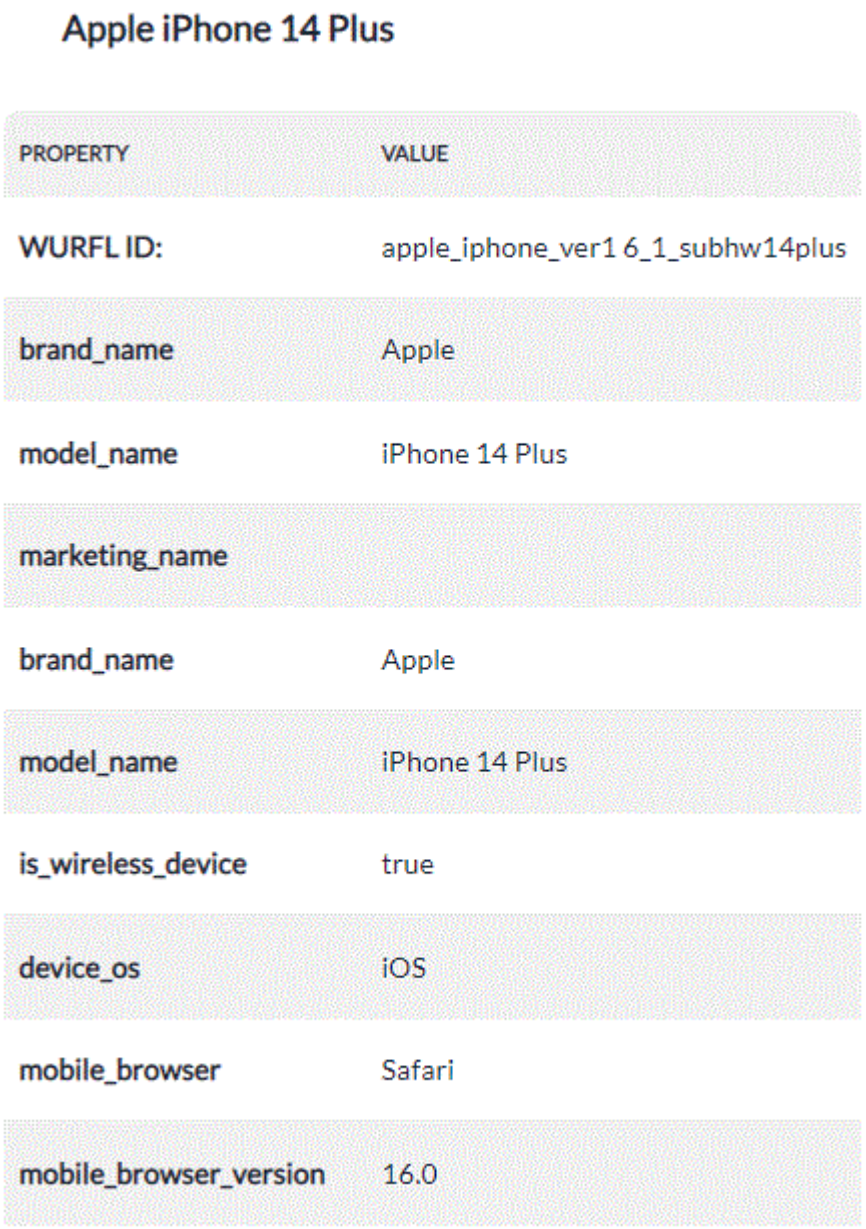
Endpoint will throw a 404 error if the device is not found in WURFL.

Usage Example

```
$ curl "https://api.query.wurflcloud.com/v1/devices/oneplus_kb2005_ver1?
capabilityNames=resolution_width,brand_name,model_name" --header 'Authorization:
Basic <your-token>'
{
    "capabilities": {
        "brand_name": "OnePlus",
        "model_name": "KB2005",
        "resolution_width": "1080"
    },
    "wurfl_id": "oneplus_kb2005_ver1"
}
```

Usage in Demo Dashboard

The UI relies on this endpoint whenever information about a specific devices (identified by a WURFL ID) needs to be displayed.



Apple iPhone 14 Plus

PROPERTY	VALUE
WURFL ID:	apple_iphone_ver1 6_1_subhw14plus
brand_name	Apple
model_name	iPhone 14 Plus
marketing_name	
brand_name	Apple
model_name	iPhone 14 Plus
is_wireless_device	true
device_os	iOS
mobile_browser	Safari
mobile_browser_version	16.0

Figure: *Example of how a single device information is displayed in the demo app.*

```
setCurrentDeviceThroughREST(wid) {
    console.log("setCurrentDeviceThroughREST clicked : " + wid);
    // console.log("this.realBaseUrl: ", this.realBaseUrl)
    axios({
        method: 'get',
        url: `${this.realBaseUrl}/v1/devices/${wid}`,
      })
      .then(res => {

        this.selectedDevice = res.data
        // this.setCurrentDeviceList(res.data);
        this.retrieveStatus = '';
      })
      .catch(err => {
        // error. To be honest we could improve usability by making the error message a modal.
        console.log("Logging server ERROR: " + err + " - " + err.response.data)
        // refactor to show diagnostics
        // this.retrieveStatus = '';
        this.errors = []
        this.errors.push(err)
        this.errors.push(err.response.data)
      })
},
```

Figure: *Code that implements displaying device information given its WURFL ID.*

ENDPOINT: `/v1/devices/findByModelName`

METHOD: GET:

Return all devices that match the provided model name

Parameters:

- `modelName`. Find all devices that match the `model_name` (case insensitive).
  - `modelName=KB2005`
- `marketingName`. Find all devices that match the `marketing_name` (case insensitive).
  - `marketingName=8T`
- `brandName`. Find all devices that match the `brand_name` (case insensitive).
  - `brandName=OnePlus`
- `capabilityNames`. Array of strings representing list of capability names. Device objects in response should contain only those capabilities.
  - `capabilityNames=resolution_width,release_date`
- `expand.` Show list of primary and secondary devices (default is `false`, i.e. only primary devices are returned along with the list of WURFL IDs of secondary devices).
- `offset`. Element position in list that indicates beginning of ‘page’
- `limit`. Number of elements per ‘page’

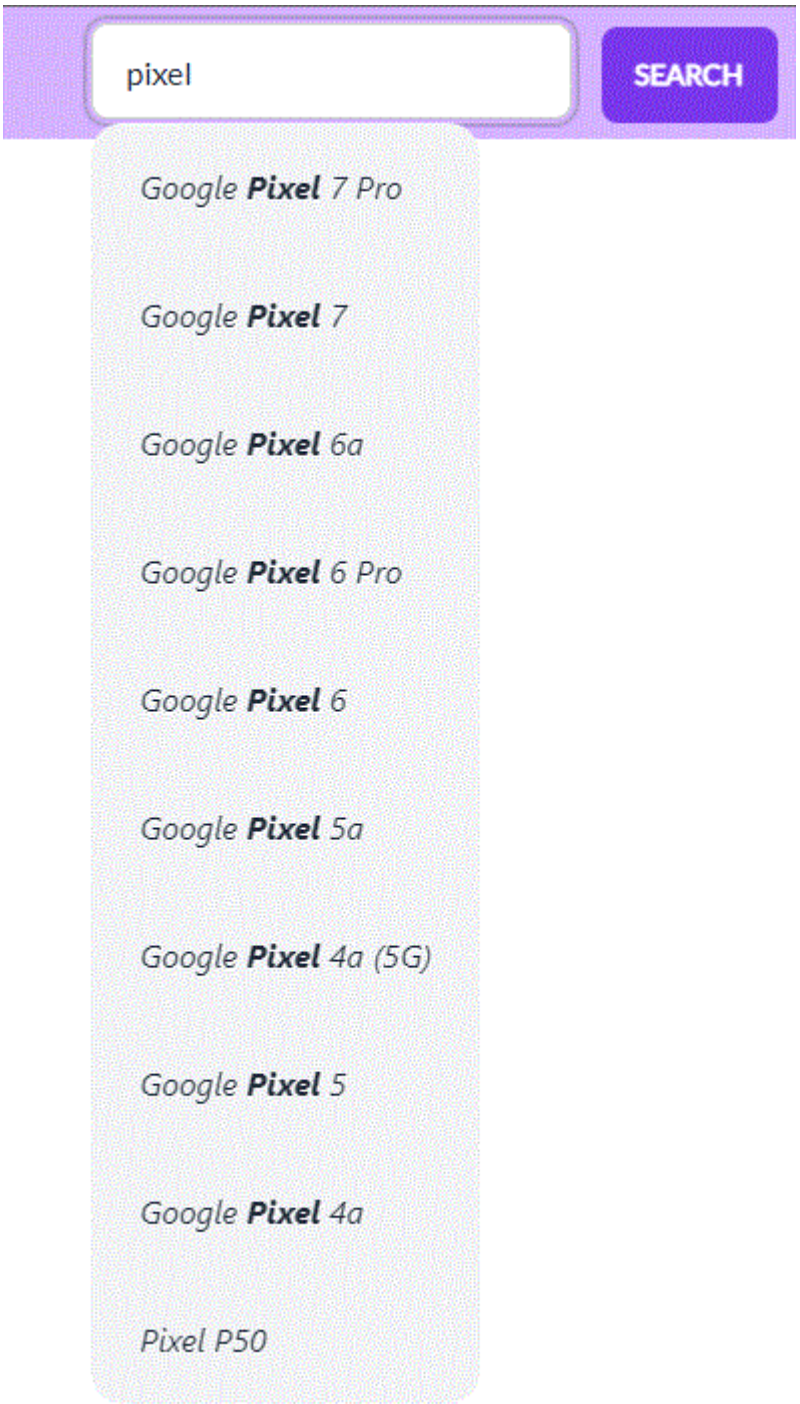
Usage Example

```
$ curl "https://api.query.wurflcloud.com/v1/devices/findByModelName?
modelName=KB2005&marketingName=8T&brandName=OnePlus&capabilityNames=resolution_width,
brand_name,model_name&expand=false" --header 'Authorization: Basic <your-token>'
[
    {
        "primary": true,
        "capabilities": {
            "brand_name": "OnePlus",
            "model_name": "KB2005",
            "resolution_width": "1080"
        },
        "wurfl_id": "oneplus_kb2005_ver1",
        "secondary_wurflids": [
            "oneplus_kb2005_ver1_suban120u3k13",
            "oneplus_kb2005_ver1_subuakb2001u3k12",
            "oneplus_kb2005_ver1_suban130kb2001u3k13",
            "oneplus_kb2005_ver1_suban120",
            "oneplus_kb2005_ver1_suban130kb2001",
            "oneplus_kb2005_ver1_suban120kb2001u3k13",
            "oneplus_kb2005_ver1_subuakb2003u3k13",
            "oneplus_kb2005_ver1_subu3k13",
            "oneplus_kb2005_ver1_suban120kb2005",
            "oneplus_kb2005_ver1_subuakb2003",
            "oneplus_kb2005_ver1_subuakb2001u3k13",
            "oneplus_kb2005_ver1_subuakb2007u3k13",
            "oneplus_kb2005_ver1_suban130kb2007",
            "oneplus_kb2005_ver1_suban120kb2007u3k13",
            "oneplus_kb2005_ver1_suban130kb2003u3k13",
            "oneplus_kb2005_ver1_suban130kb2000",
            "oneplus_kb2005_ver1_subuakb2007",
            "oneplus_kb2005_ver1_subuakb2001",
            "oneplus_kb2005_ver1_suban120kb2000",
            "oneplus_kb2005_ver1_subua2000",
            "oneplus_kb2005_ver1_subuakb2001u3k11",
            "oneplus_kb2005_ver1_suban130",
            "oneplus_kb2005_ver1_suban130kb2003",
            "oneplus_kb2005_ver1_subua2000u3k13",
            "oneplus_kb2005_ver1_suban120kb2007",
            "oneplus_kb2005_ver1_suban130u3k13",
            "oneplus_kb2005_ver1_subuakb2003u3k12",
            "oneplus_kb2005_ver1_suban120kb2003",
            "oneplus_kb2005_ver1_suban120kb2003u3k13",
            "oneplus_kb2005_ver1_suban120kb2000u3k13",
            "oneplus_kb2005_ver1_subua2000u3k12",
            "oneplus_kb2005_ver1_suban120kb2001"
        ]
    }
]
```

Usage in Demo Dashboard

The UI component that uses this endpoint is actually relying on other two endpoints as well: '`/v1/devices/autofillBySubstring`' and '/`v1/devices/findByGenericName`' respectively.

As users type in the search bar at the top, the autofill will suggest devices that might match what the user is looking for. If the user picks one of the suggested devices, then '/`v1/devices/findByModelName`' is used to zero in on the selected device. On the other hand, users that prefer to try their luck with their suggestion will be dealt the result of the generic match endpoint (see later)



```
let endpoint = `${this.realBaseUrl}/v1/devices/findByModelName?modelName=${this.
formdata.model_name}&marketing_name=${this.formdata.marketing_name}&brandName=$
{this.formdata.brand_name}&expand=false`
if (!this.formdata.model_name)
  endpoint = `${this.realBaseUrl}/v1/devices/findByGenericName?genericName=${this.
  formdata.completeModelName}`

axios({
  method: 'get',
  url: endpoint
})
.then(res => {
  this.setCurrentDeviceList(res.data);
  this.retrieveStatus = '';
  // if the response contains devices, show the first one
  if (res.data.length >= 1) {
    let wid = res.data[0].wurfl_id
    if (wid) {
      this.setCurrentDeviceThroughREST(wid)
    }
  }
})
.catch(err => {
    // error
    this.retrieveStatus = '';
    console.log(err);
    this.errors.push(err)
})
```

Figure: *Example of autofill of WURFL Device names is achieved visually and with JS code (Vue.js).*

ENDPOINT: `/v1/devices/findByGenericName`

METHOD: GET:

This endpoint is conceptually very similar to `/v1/devices/findByModelName` but is way more "lenient", as it will try several strategies to make sense of unstructured input.

For example, assuming the input from a user is "oneplus 8t", that string will be associated with the `genericName` parameter. `FindByGenericName` will go out of its way to determine that 'oneplus' is in fact a device manufacturer, and hence the '8t' part is a string that needs to be matched against both marketing and model names of OnePlus devices.

Parameters:

- `genericName`. Find all devices that match the `model_name` (case insensitive).
  - `modelName=KB2005`
- `marketingName`. Find all devices that match the `marketing_name` (case insensitive).
  - `marketingName=8T`

- `brandName`. Find all devices that match the `brand_name` (case insensitive).
  - `brandName=OnePlus`
- `capabilityNames`. Array of strings representing list of capability names. Device objects in response should contain only those capabilities.
  - `capabilityNames=resolution_width,release_date`
- `expand.`Show list of primary and secondary devices (default is `false`, i.e. only primary devices are returned along with the list of WURFL IDs of secondary devices).
- `offset`. Element position in list that indicates beginning of 'page'
- `limit`. Number of elements per 'page'

Usage Example The example below retrieves all Nokia devices with generic name "Lumia" and brand name "Nokia" (just first part of output is shown)

```
$ curl -s "https://api.query.wurflcloud.com/v1/devices/findByGenericName?
genericName=Lumia&brandName=Nokia&capabilityNames=brand_name,model_name,marketing_nam
e" --header 'Authorization: Basic <your-token>'
[
    {
        "primary": true,
        "capabilities": {
            "brand_name": "Nokia",
            "marketing_name": "Lumia 1320",
            "model_name": "1320"
        },
        "wurfl_id": "nokia_1320_ver1",
        "secondary_wurflids": [
            "nokia_1320_ver1_subos10",
            "nokia_1320_ver1_subos81u3k4",
            "nokia_1320_ver1_subos10nokia",
            "nokia_1320_ver1_subos81"
        ]
    },
    {
        "primary": true,
        "capabilities": {
            "brand_name": "Nokia",
            "marketing_name": "Lumia 1020",
            "model_name": "Lumia 1020"
        },
        "wurfl_id": "nokia_lumia_1020_ver1",
        "secondary_wurflids": [
            "nokia_lumia_1020_ver1_subua9091",
            "nokia_lumia_1020_ver1_subuarm875",
            "nokia_lumia_1020_ver1_subos81id300",
            "nokia_lumia_1020_ver1_subua909u3k4",
            "nokia_lumia_1020_ver1_subos81909u3k4",
            "nokia_lumia_1020_ver1_subuarm876",
            "nokia_lumia_1020_ver1_subos81909",
            "nokia_lumia_1020_ver1_subos81nokia909u3k4",
            "nokia_lumia_1020_ver1_subos81rm877",
            "nokia_lumia_1020_ver1_subu2k4",
            "nokia_lumia_1020_ver1_subos81rm875",
            "nokia_lumia_1020_ver1_subos10909",
            "nokia_lumia_1020_ver1_subua909",
            "nokia_lumia_1020_ver1_subuarm877",
            "nokia_lumia_1020_ver1_subos81nokia909"
        ]
    },
   :
]
```

WURFL IDs of all Lumia devices:

```
$ curl -s "https://api.query.wurflcloud.com/v1/devices/findByGenericName?
genericName=Lumia&brandName=Nokia&capabilityNames=brand_name,model_name,marketing_nam
e" --header 'Authorization: Basic <your-token>' | grep wurfl_id
        "wurfl_id": "nokia_1320_ver1",
        "wurfl_id": "nokia_lumia_1020_ver1",
        "wurfl_id": "nokia_lumia_1320_ver1",
        "wurfl_id": "nokia_lumia_1520_ver1",
        "wurfl_id": "nokia_lumia_430_ver1",
        "wurfl_id": "nokia_lumia_435_ver1",
        "wurfl_id": "nokia_lumia_505_ver1",
        "wurfl_id": "nokia_lumia_510_ver1",
        "wurfl_id": "nokia_lumia_520_ver1",
        "wurfl_id": "nokia_lumia_525_ver1",
        "wurfl_id": "nokia_lumia_530_ver1",
        "wurfl_id": "nokia_lumia_535_ver1",
        "wurfl_id": "nokia_lumia_610_ver1",
        "wurfl_id": "nokia_lumia_620_ver1",
        "wurfl_id": "nokia_lumia_625_ver1",
        "wurfl_id": "nokia_lumia_630_ver1",
        "wurfl_id": "nokia_lumia_635_ver1",
        "wurfl_id": "nokia_lumia_638_ver1",
        "wurfl_id": "nokia_lumia_640_ver1",
        "wurfl_id": "nokia_lumia_640_xl_ver1",
        "wurfl_id": "nokia_lumia_710_ver1",
        "wurfl_id": "nokia_lumia_720_ver1",
        "wurfl_id": "nokia_lumia_730_ver1",
        "wurfl_id": "nokia_lumia_735_ver1",
        "wurfl_id": "nokia_lumia_800_ver1",
        "wurfl_id": "nokia_lumia_810_ver1",
        "wurfl_id": "nokia_lumia_820_ver1",
        "wurfl_id": "nokia_lumia_822_ver1",
        "wurfl_id": "nokia_lumia_830_ver1",
        "wurfl_id": "nokia_lumia_900_ver1",
        "wurfl_id": "nokia_lumia_920_ver1",
        "wurfl_id": "nokia_lumia_925_ver1",
        "wurfl_id": "nokia_lumia_929_ver1",
        "wurfl_id": "nokia_lumia_930_ver1",
        "wurfl_id": "nokia_rm941_ver1",
        "wurfl_id": "nokia_rm978_ver1",
```

If we use “Nokia Lumia” as generic name, and omit the brand, the endpoint will figure out that Nokia is the brand and still converge to the same result:

```
$ curl -s "https://api.query.wurflcloud.com/v1/devices/findByGenericName?
genericName=Nokia%20Lumia&capabilityNames=brand_name,model_name,marketing_name" --
header 'Authorization: Basic <your-token>' | wc -l
819
$ curl -s "https://api.query.wurflcloud.com/v1/devices/findByGenericName?
genericName=Lumia&brandName=Nokia&capabilityNames=brand_name,model_name,marketing_nam
e" --header 'Authorization: Basic <your-token>' | wc -l
819
```

Usage in Demo Dashboard

Usage of this endpoint was illustrated above ('/`v1/devices/findByModelName`'). For example, if the user enters "Nokia Lumia" without picking any of the autofill suggestions, here's the result.

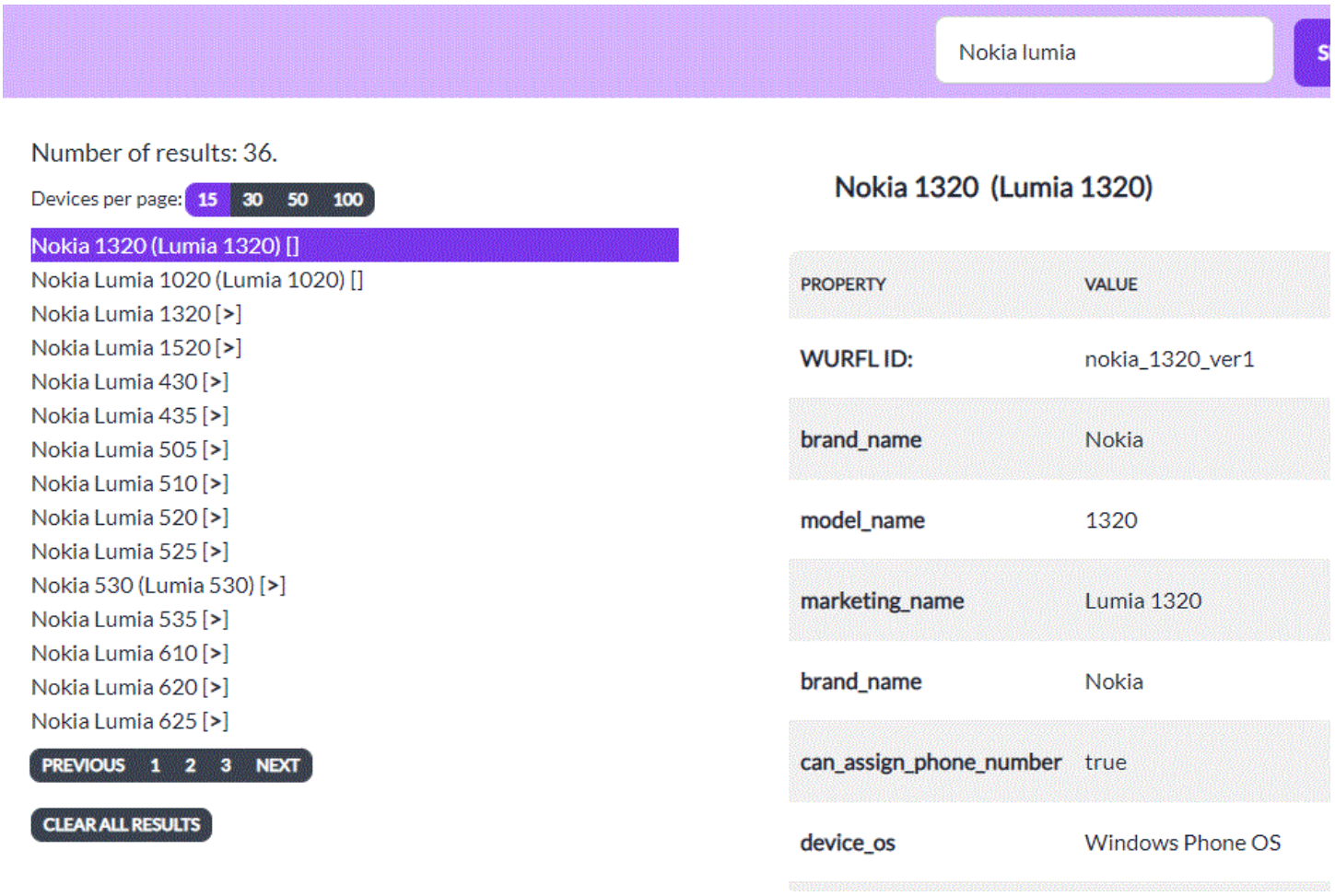


Figure: *Finding all devices generically names Nokia Lumia.*

ENDPOINT: `/v1/devices/search`

METHOD: GET/POST

This endpoint is arguably one of the most critical, as it enables the creation of complex queries by combining multiple "filters," each applying a condition on a capability value. This endpoint allows us to request all OPPO devices released

between January 2020 and December 2022 with a resolution width of 960 or greater. There could be a use case for incorporating a complex query into a single clickable URL, which is why we support the GET method. However, a long list of filters can become bulky and encounter limitations on the maximum length of a URL. For this reason, the API supports the POST method as well.

Note: *A discussion on whether GET is more appropriate than POST in this kind of scenario has already occurred on StackOverflow.* ***https://stackoverflow.com/questions/14202257/design-restful-query-api-with-a-long-list-of-query-parameters***

Note 2: *It appears that the OpenAPI specification (version 3) is not powerful enough to properly represent an arbitrary number of filters that can be passed as parameters to a query (ref:* ***openapi 3.0 specify complex filter in parameters - Stack Overflow*** *). To address this, filters need to be created in JSON format and passed to the endpoint as a string. This will work and be consistent with the API specification. Nonetheless, if OpenAPI had provided a way to serialize our filter queries, we would have been happier.*

All parameters are optional, but endpoint may return an error if not enough information is provided or certain consistency constraints are violated. For example, the endpoint will return an error if two filters in the same request refer to two different brand_names.

Parameters:

- `capabilityNames`.
  - Array of strings representing list of capability names. Device objects in response should contain only those capabilities.
  - `capabilityNames=resolution_width,release_date`
- `filters`.
  - Actually a string representing a JSONified list of filters (arrary of objects)
  - `filters=[{filter1},{filter2},...,{filterN}]`
  - A single filter is an object with three fields: capability name, operator, target value.
  - `{"capaname":"resolution_width","operator":"EqualOrGreaterThan","targetvalue":"640"}`
- `expand.` Show list of primary and secondary devices (default is `false`, i.e. only primary devices are returned along with the list of WURFL IDs of secondary devices).
- `offset`. Element position in list that indicates beginning of ‘page’
- `limit`. Number of elements per ‘page’

When it comes to the possible values for operators in each query filter, this is the list of possible values:

- "Equal"
- "NotEqual"
- "LessThan"
- "GreaterThan"
- "EqualOrLessThan"
- "EqualOrGreaterThan"
- "CaseInsensitiveEqual"
- "NearlyEqual"
- "Contains"

Not all capabilities can be used with all QueryTypes (see definitions after table):

QueryType	Brand, Model and Marketing Name*	Special Handling*	Numeric*	General*
Equal	O	O	O	O
NotEqual	O	O	O	O
LessThan		O	O	
GreaterThan		O	O	
EqualOrLessThan		O	O	
EqualOrGreaterThan		O	O	
CaseInsensitiveEqual	O			
NearlyEqual	O			
Contains	O			

(*) Definitions:

- Brand, Model and Marketing Name: `brand_name`, `model_name` and `marketing_name`.
- Special Handling: `release_date`, `mobile_browser_version`, `device_os_version`.
- Numeric: `colors`, `max_data_rate`, `max_image_height`, `max_image_width`, `physical_screen_height`, `physical_screen_width`, `release_msrp`, `resolution_height`, `resolution_width`.
- General: all capabilities not mentions above.

A note about WURFL virtual capabilities

WURFL supports static and virtual capabilities. Virtual capabilities are capabilities whose value is calculated at run time based on both the values of static capabilities and other aspects of the User-Agent string and/or other HTTP headers (Client-Hint, UA string made available in other headers, such as in the case of so-called “side-loaded” browsers)

This mechanism is super useful in many scenarios. In the case of WURFL Query, though, calculating the value of a virtual capability may or may not make sense. This has to be assessed for each and every vcap.

Usage Example

Let’s use `curl` to speak POST to our endpoint for a change. We are looking for devices that have 7000 pixel wide screen:

```
$ curl -X 'POST'   'https://api.query.wurflcloud.com/v1/devices/search?
capabilityNames=resolution_width,brand_name,model_name&offset=0&limit=5&expand=true'
--header 'Authorization: Basic <your-token>'   -d '[
  {
    "capability_name": "resolution_width",
    "operator": "EqualOrGreaterThan",
    "capability_value": "4320"
  }
]'
[
    {
        "primary": true,
        "capabilities": {
            "brand_name": "Samsung",
            "model_name": "GQ65Q800TGTXZG",
            "resolution_width": "7680"
        },
        "wurfl_id": "samsung_gq65q800tgtxzg_ver1",
        "secondary_wurflids": null
    },
    {
        "primary": true,
        "capabilities": {
            "brand_name": "Samsung",
            "model_name": "QN55Q900RBFXZA",
            "resolution_width": "7680"
        },
        "wurfl_id": "samsung_qn55q900rbfxza_ver1",
        "secondary_wurflids": null
    },
    {
        "primary": false,
        "capabilities": {
            "brand_name": "Samsung",
            "model_name": "QN55Q900RBFXZA",
            "resolution_width": "7680"
        },
        "wurfl_id": "samsung_qn55q900rbfxza_ver1_subuahulu11120",
        "secondary_wurflids": null,
        "ref_to_primary": "samsung_qn55q900rbfxza_ver1"
    },
    {
        "primary": true,
        "capabilities": {
            "brand_name": "Samsung",
            "model_name": "QN65Q800TAFXZA",
            "resolution_width": "7680"
        },
        "wurfl_id": "samsung_qn65q800tafxza_ver1",
```

```
        "secondary_wurflids": null
    },
    {
        "primary": false,
        "capabilities": {
            "brand_name": "Samsung",
            "model_name": "QN65Q800TAFXZA",
            "resolution_width": "7680"
        },
        "wurfl_id": "samsung_qn65q800tafxza_ver1_subuahulu11120",
        "secondary_wurflids": null,
        "ref_to_primary": "samsung_qn65q800tafxza_ver1"
    }
]
```


Usage in Demo Dashboard

The Dashboard UI lets users add filters, each one of them representing a condition of a search query. The screenshot below shows how we can ask QuWURFL to give us the list of all Kyocera tablets that run Android OS:

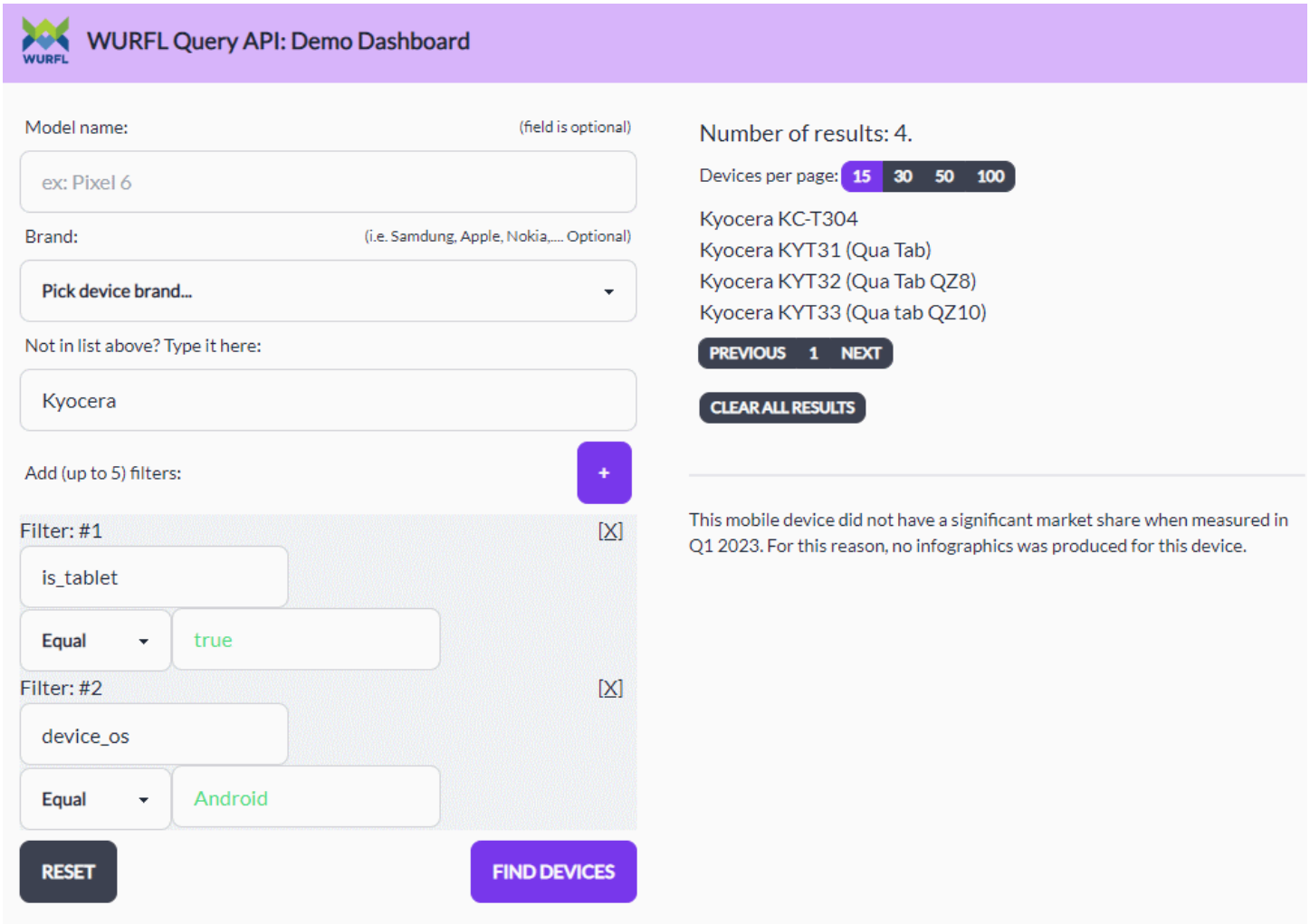


Figure: *Demo application lets users add filters to create a complex WURFL Query.*

Here's the salient JS part that connects UI and endpoint behind the scenes:

```
// Model
if (this.formdata.model) {
  console.log("model field is NOT Null")
  let modelFilter = {
          capability_name: "model_name",
          operator: "NearlyEqual",
          capability_value: this.formdata.model
        }
  allFilters.push(modelFilter)
}

// finally the filters that user may have added manually
this.filters.forEach((filter) => {

  if (filter.capaname.trim() && filter.targetvalue.trim()) {
    allFilters.push({
          capability_name: filter.capaname,
          operator: filter.queryType,
          capability_value: filter.targetvalue
          })
  }
})
```

Figure: *JavaScript code that packages filters into formdata ready for a (Axios) POST request.*

ENDPOINT: `/v1/devices/autofillBySubstring`

Method: GET

This endpoint is useful to support autofill in web applications. A short string (2 chars minimum) is enough to make the API rush to return a list of devices that match our language.

Endpoint returns a list of JSON objects with 4 data points: model name, marketing name, brand name, and WURFL ID. This list has been obtained by matching the `key` parameter against brand, model and marketing name (with some secret sauce resolving the case when the key matches in multiple situations: the endpoint will send representatives from both sets of matches in that case).

Note: *there is a 5th data point called relevance that the algorithm uses internally to rank device matches that are more assumed to be relevant. The algorithm that decides which autofill entries are more relevant may change over time. Currently, newer devices are prioritized over older devices, based on the assumption that users are probably looking for information on newer devices.*

The number of devices returned by the autofill is dictated by the number of actual matches. For this reason the endpoint will limit the number of returned matches to 10, unless a different limit is set through the `maximum` query parameter.

Parameters:

- `key`. Find all brand names in WURFL that match our parameter (case insensitive).

    - `key=samsung`
- `maximum`. Limit/Extend maximum number of expected matches (default 10)
    - `maximum=5`

Usage Example

The "`pixel`" substring is found in the name of all the Google Pixels released by Google over time. There is also a (minor) device manufacturer called Pixel, though.

In the following example, we limit the number of devices to 4.

```
$ curl "https://api.query.wurflcloud.com/v1/devices/autofillBySubstring?
key=pixel&maximum=4" --header 'Authorization: Basic <your-token>'
[
    {
        "model_name": "Pixel 7 Pro",
        "marketing_name": "",
        "brand_name": "Google",
        "wurfl_id": "google_pixel_7_pro_ver1",
        "relevance": 273
    },
    {
        "model_name": "Pixel 7",
        "marketing_name": "",
        "brand_name": "Google",
        "wurfl_id": "google_pixel_7_ver1",
        "relevance": 273
    },
    {
        "model_name": "Pixel 6a",
        "marketing_name": "",
        "brand_name": "Google",
        "wurfl_id": "google_pixel_6a_ver1",
        "relevance": 270
    },
    {
        "model_name": "P50",
        "marketing_name": "",
        "brand_name": "Pixel",
        "wurfl_id": "pixel_p50_ver1",
        "relevance": 153
    }
]
```

Please observe how:

- The endpoint only returned 4 elements

- While elements were almost exclusively Google Pixel devices, one refers to the P50, a device by device manufacturer Pixel.
- Most relevant devices bubbled to the top.

Usage in Demo Dashboard

Usage of this endpoint was illustrated above along with '/`v1/devices/findByModelName`'.

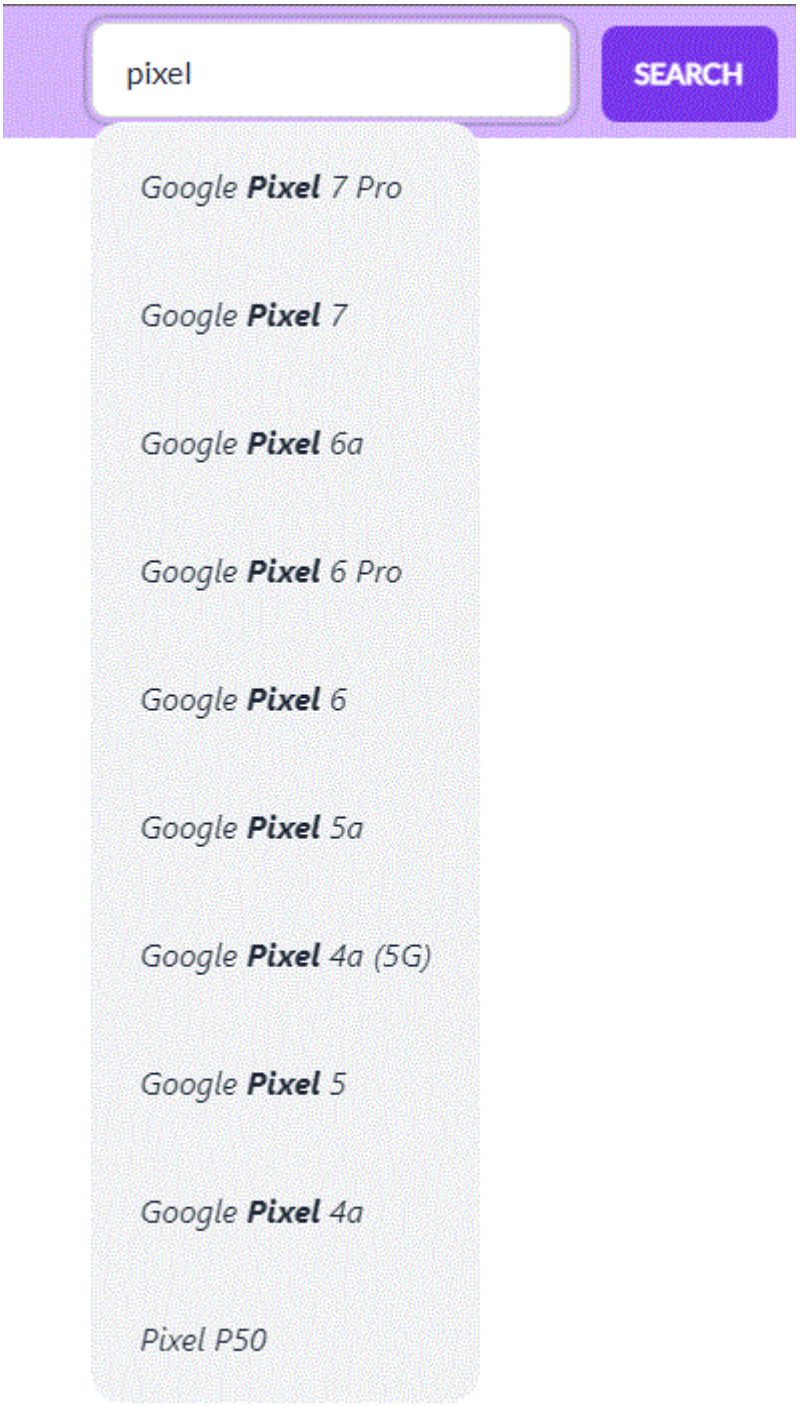


Figure: *If the autofill suggestion is accepted, findByModelName will be used to query the devices (instead of findByGenericName).*

```
axios({
  method: 'get',
  url: `${componentdata.realBaseUrl}/v1/devices/autofillBySubstring?key=${key}&
  maximum=10`
})
.then(res => {
  let suggestions = res.data
  for (let i = 0; i < suggestions.length; i++) {
    let modelName = suggestions[i].model_name
    modelName = highlightKey(modelName,key)
    let marketingName = suggestions[i].marketing_name + ""
    marketingName = highlightKey(marketingName, key)
    let brandName = suggestions[i].brand_name
    brandName = highlightKey(brandName, key)
    let completeModelName = brandName +" " + modelName
    if (marketingName.trim().length > 0) {
      completeModelName += " (" + marketingName +")"
    }
    let deviceObj = { ...suggestions[i]} // spread operator to clone object
    deviceObj.completeModelName = completeModelName
    componentdata.results.push(deviceObj)
  }
  componentdata.isAutoCompleteOpen = true
  componentdata.isLoadingModelNames = ""
})
.catch(err => {
    // error
    console.log(err);
    // componentdata.errors.push(err)
})
```

Figure: _Code that loads autofill data from the endpoint and places it in the 'results' reactive Array. Vue.js will take care of visualizing the list.

ENDPOINT: `/devices/notify/{channel}`

Method: GET

This endpoint can be used to push a message to a channel. Clients (such as Apps or web applications) that are listening to that channel will get an event and can trigger actions based on the information they have received (such as displaying information about a given device make and model). The system uses the Pusher.com API, which - in turn - relies on WebSockets.

Parameters:

- `channel`. String that identifies a channel. Can be used to share device information from a session on a mobile device to a different session on a desktop web browser. Choice of string is arbitrary but should be unique per user-session and - more importantly - it should be the same used to receive events.

- `wurfl_id`: notify clients of a WURFL ID (typically a mobile phone that has accessed our application and whose WURFL ID has been captured with the help of WURFL.js)

In order to harness this endpoint, programmers will need the following Pusher.com public keys to enable their client application to listen to notifications:

```
PUSHER_KEY=4cbafd91f7216ef6edce
PUSHER_CLUSTER=us2
```

Usage Example

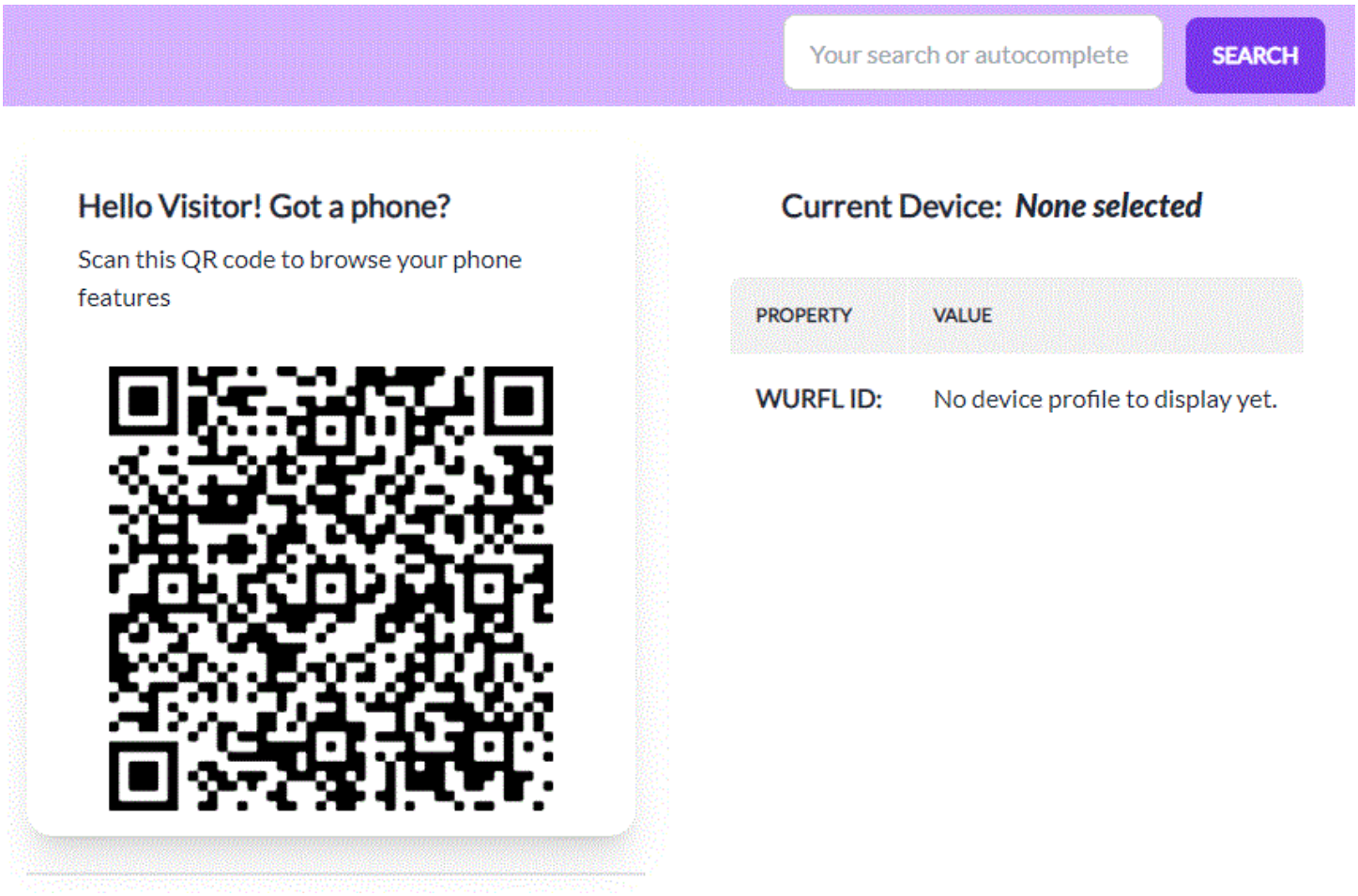
The notification part can be invoked like this:

```
$ curl "https://api.query.wurflcloud.com/v1/devices/notify/1679232120530-r02kbtwuo?
wurfl_id=oneplus_hd1925_ver1" --header 'Authorization: Basic <your-token>'
{
    "status": "success"
}
```

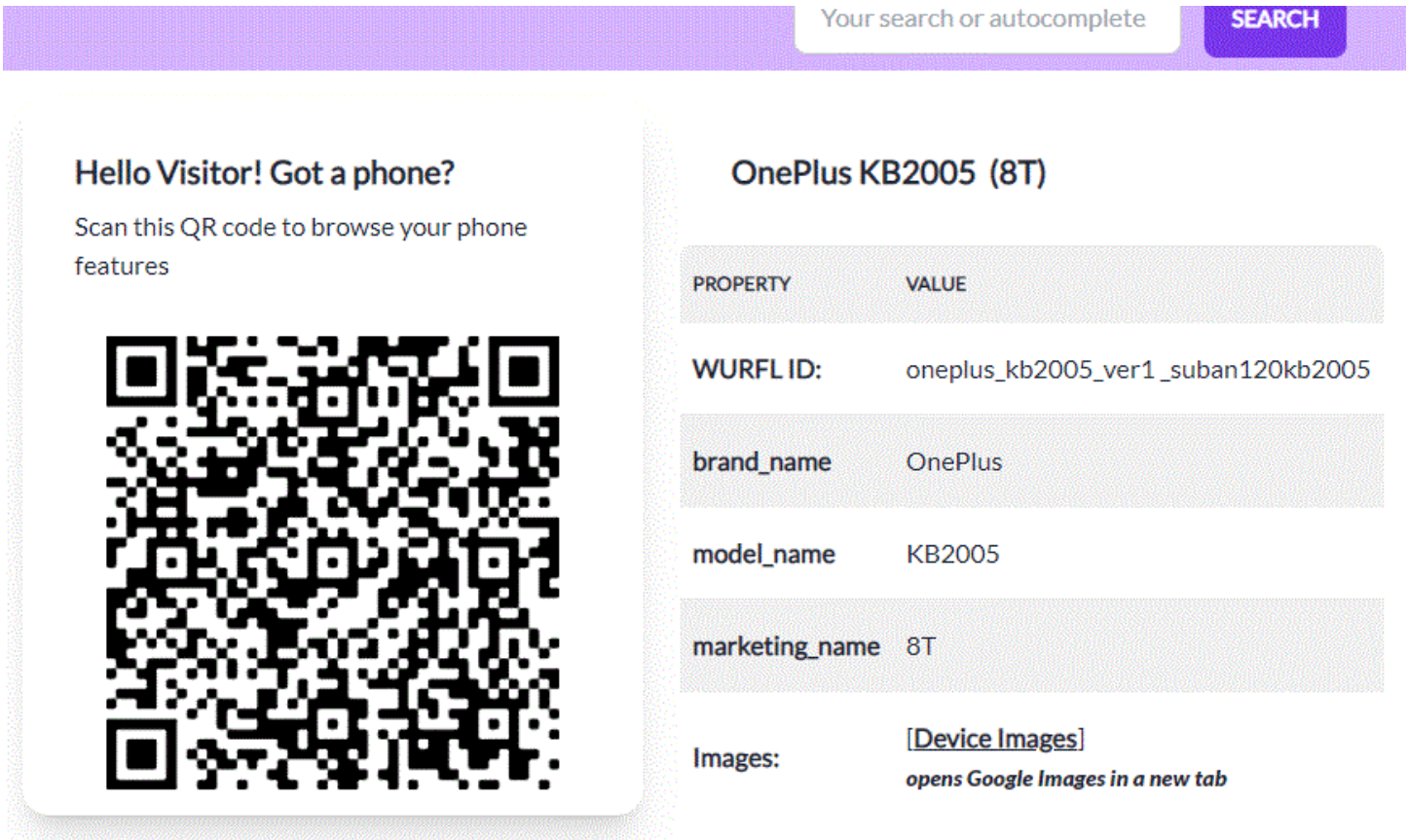
Of course the successful status only means that the endpoint has received the request to dispatch the message to all clients listening on that channel. Whether this actually happens should be verified independently, as there are multiple reasons why a message is ultimately not dispatched.

Usage in Demo Dashboard

The Dashboard uses this endpoint to implement a nifty feature: the QR code that captures device information. If you pull up the QR code on a freshly loaded **https://demo.query.wurfl.io/** on your PC and scan it with a smartphone (or any device for that matter), WURFL.js will recognize your phone and notify the application running on the desktop. In practice, this means that your page will go from looking like this:

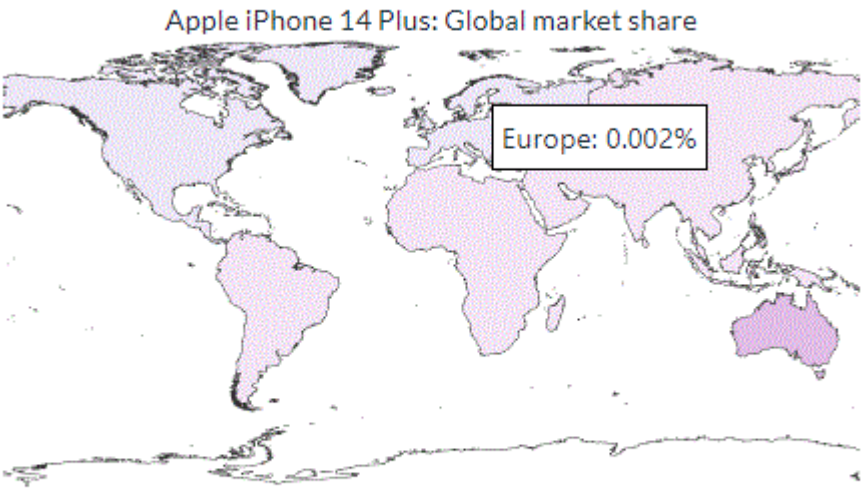


to displaying the information of the device that has scanned the QR code (a OnePlus 8T, in this case):



Bonus feature: Choropleth

Not really a feature that requires any of the endpoints, but the demo dashboard will also display a map with device market data for popular devices.



We added this feature because the demo app had way too much whitespace, but it is not relying on any specific endpoint. If you have WURFL, you can use the API to give a name to the mobile traffic on your website as we did to produce this `data.csv` file.

public > data.csv

```
complete_device_name,north_america,south_central_america,europe,asia,africa,australia
Apple iPhone 11,0.025,0.041,0.031,0.02,0.022,0.037
Apple iPhone 13 Pro Max,0.018,0.007,0.013,0.014,0.012,0.031
Apple iPhone 7 Plus,0.002,0.007,0.002,0.006,0.012,0.003
Apple iPhone 12 Pro Max,0.013,0.005,0.01,0.012,0.011,0.024
Huawei JNY-LX1 (nova 7i),0.0,0.0,0.0,0.0,0.011,0.0
Apple iPhone XR,0.009,0.013,0.011,0.006,0.01,0.012
Apple iPhone X,0.003,0.004,0.005,0.005,0.009,0.005
Apple iPhone 11 Pro Max,0.006,0.004,0.006,0.008,0.009,0.016
Apple iPhone 8 Plus,0.004,0.011,0.004,0.005,0.008,0.005
Apple iPhone 13,0.015,0.01,0.02,0.01,0.008,0.023
Apple iPhone 7,0.002,0.008,0.004,0.004,0.008,0.003
Apple iPhone XS Max,0.003,0.003,0.003,0.006,0.008,0.006
Samsung SM-A207F (Galaxy A20s),0.0,0.0,0.0,0.0,0.007,0.0
Apple iPhone 12,0.011,0.009,0.017,0.008,0.006,0.023
Samsung SM-A307FN (Galaxy A30s),0.0,0.0,0.001,0.001,0.006,0.0
Samsung SM-A127F (Galaxy A12 Nacho),0.0,0.0,0.003,0.002,0.006,0.002
Samsung SM-A325F (Galaxy A32),0.0,0.0,0.002,0.003,0.006,0.002
Samsung SM-A217F (Galaxy A21s),0.0,0.0,0.003,0.002,0.005,0.001
Apple iPhone 6S Plus,0.0,0.001,0.0,0.002,0.005,0.0
Apple iPhone 6S,0.001,0.004,0.002,0.003,0.005,0.001
Apple iPhone 8,0.003,0.007,0.007,0.003,0.005,0.005
Samsung SM-A125F (Galaxy A12),0.0,0.0,0.003,0.002,0.005,0.002
Apple iPhone 11 Pro,0.003,0.003,0.007,0.005,0.005,0.01
Apple iPhone 12 Pro,0.007,0.003,0.008,0.007,0.004,0.015
Samsung SM-A032F (Galaxy A03 Core),0.0,0.0,0.0,0.0,0.004,0.0
Apple iPhone 13 Pro,0.01,0.004,0.011,0.008,0.004,0.019
```

From there to the animated map was easy peasy with D3.js

Important note about the use of Virtual Capabilities in QuWURFL

If you are familiar with WURFL, you already know the difference between regular (static) capabilities and virtual (run-time) capabilities. In short, static capabilities are found in the `wurfl.xml` file, either in the definition of the device itself or “inferrable” through the fallback chain. Virtual capabilities are different: they depend on the value of other capabilities and “run-time” values inferred from the HTTP request (be it UA string, Client Hints or other headers/conditions).

WURFL has a way to always honor the request for a capability, but the response does not always make sense in the absence of an HTTP request. For example, the virtual capability that determines whether a device should be considered a smartphone can be relied on when calculated “statically”. On the other hand, the capability that returns the micro version number of an older Android device is not very significant, as there is no UA string to extract that number from.

What follows is a list of virtual capabilities that are not significant for use with QuWURFL:

- *advertised_browser*: use *mobile_browser* instead.
- *advertised_browser_version*: use *mobile_browser_version* instead

- *advertised_device_os*: use _device_os _instead.
- *advertised_device_os_version*: use _device_os_version _instead
- *advertised_app_name*: do NOT use. QuWURFL is meant to query devices, not apps.
- *is_app, is_app_webview* : do NOT use. Returned value is insignificant unless a real UA string and other headers are provided.
- *is_robot* : do NOT use. Not significant.

The following virtual capabilities are derived from static capabilities and can be used to create meaningful queries.

- form_factor
- is_android
- Is_ios
- is_mobile
- complete_device_name
- is_smartphone
- is_full_desktop
- is_touchscreen
- is_largescreen
- is_windows_phone
- pixel_density
- is_phone

QuWURFL Support

Adopters of the WURFL Query REST API can get support and provide feedback through our (newly established) **Discord** server available at:

https://discord.gg/wurfl

While Discord support does not come with a **SLA** of any kind, the channel will be attended routinely. You are welcome to join our server and introduce yourself!

Conclusions

QuWURFL enables you to programmatically access WURFL information based on device names and features. Applications can now access device data in real time with queries.