Labs for prototyping future mobility data sharing solutions in the cloud

ABCs Navitia Playground

Prepare you datathon in the best way

mobidatalab.eu

MobiDataLab is funded by the EU under the H2020 Research and Innovation Programme (grant agreement No 101006879).

What the Navitia playground is for?

Navitia Playground is a sandbox for testing data and queries. It allows to interpret the raw information generated by Navitia through a graphical interface very easy to use.

The tool is designed for all types of profiles. Anyone can use it: from technical to non-technical backgrounds.The tool is free and accessible to everyone.

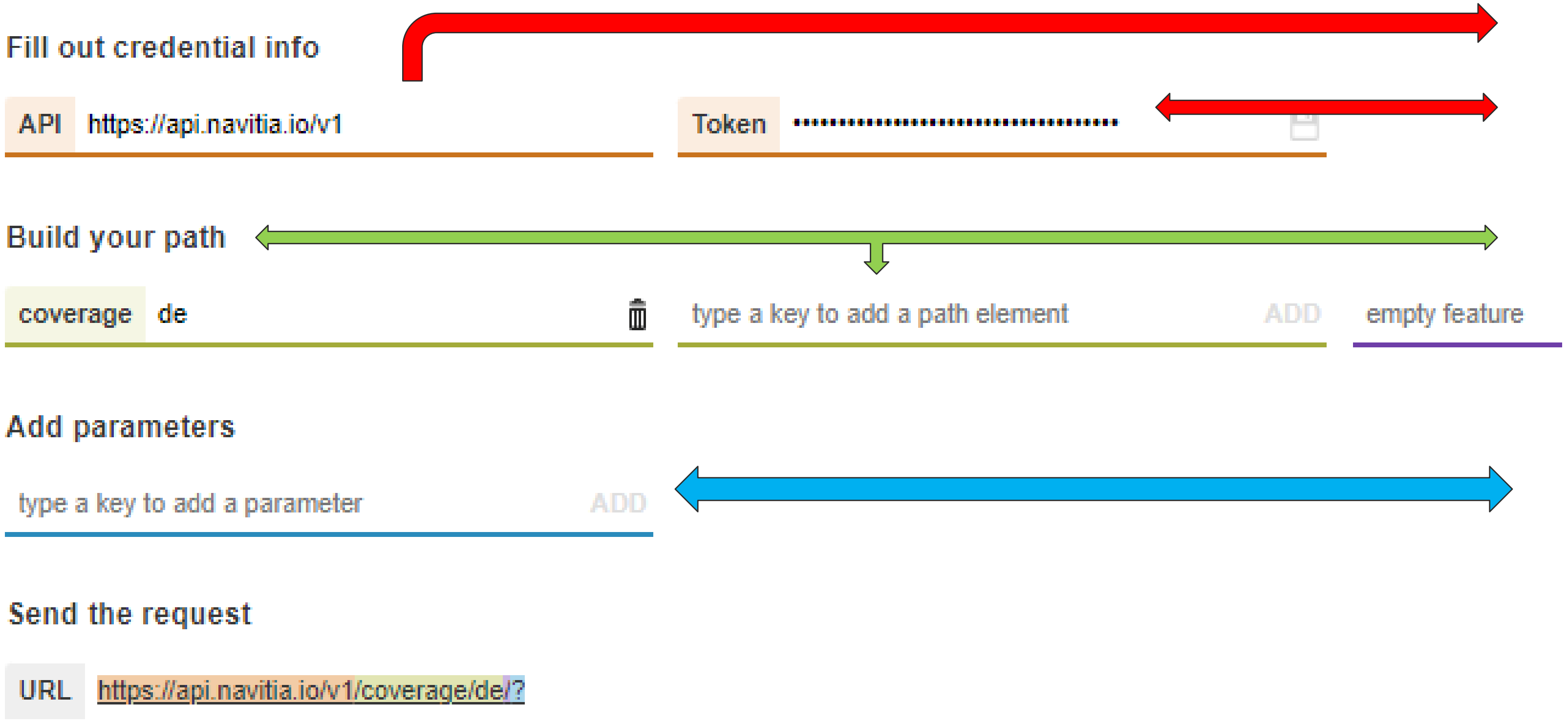
How to get your token?

1) *Create your account – You will receive your token. It will be required in the next slides.*

2) *You're beginner? Use templated examples*

3) *Navitia documentation*

Query fields



Fill out credential info

API https://api.navitia.io/v1

Token ••

Build your path

coverage de

type a key to add a path element ADD

empty feature

Add parameters

type a key to add a parameter ADD

Send the request

URL https://api.navitia.io/v1/coverage/de/?

The root URL of the API ,which is the address of the server.

This is the authentification token (or security token). You will need to edit it when copy/pasting the following URLs with your team URL during the datathon

The path which allows you to specify the search perimeter of the query. Ex : Coverage, stop_point, stop_area **; de (Germany)**

The API access point (/endpoint) to specify what type of information is requested.
Ex : /journeys /stop_schedule /stop_area

Parameters (parameters) to specify what informatin is desired in the API response.
Ex: data_freshness /count /depth

URL Navitia: LINK

Creating a route

We will see how to create a Navitia query to perform a route search.

In this example: **Add parameters**

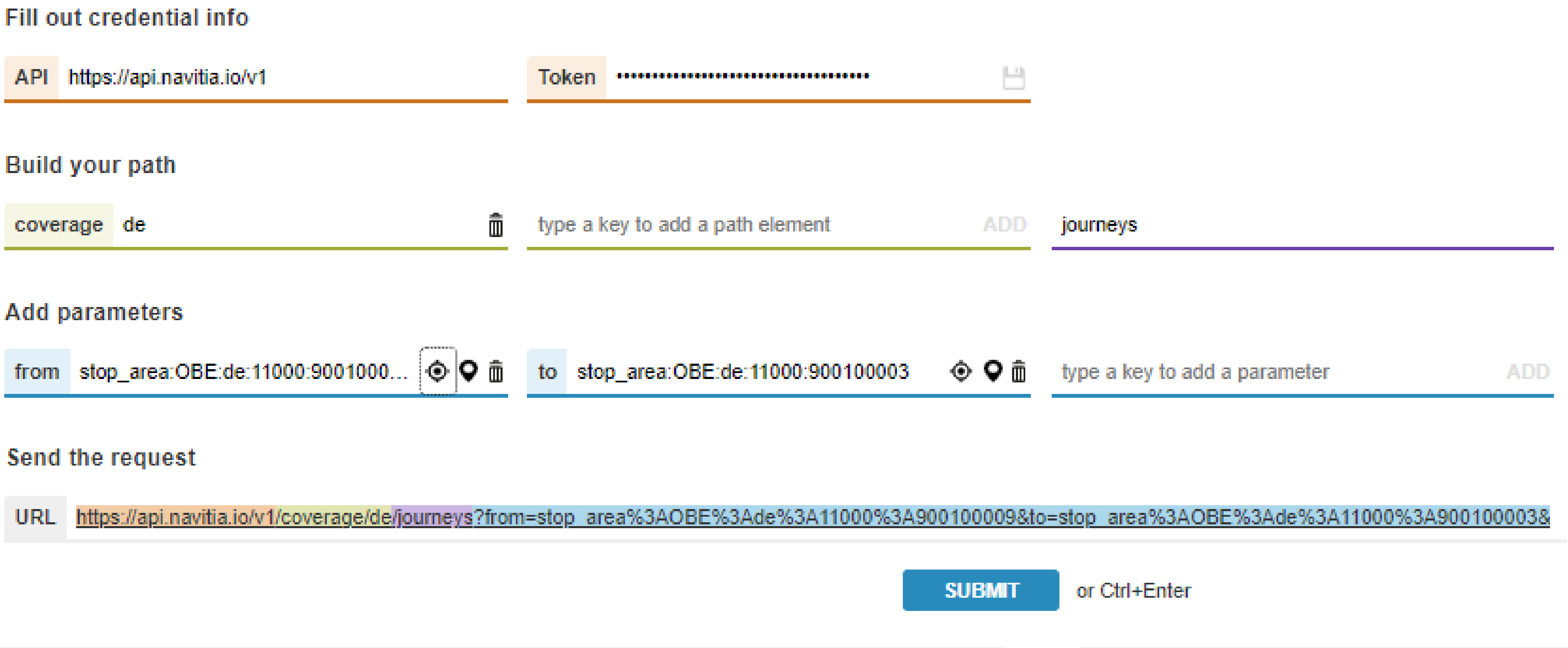
- *From: Stop area = Naturkundenmuseum*
- *To: Stop area = S+U Alexanderplatz (Bhf)*

Route construction

Call settings*:*

- **Endpoint /journeys = (itinary)**
- **Coverage production: sandbox**
- **Coverage validation: sandbox**
- **From (starting point)**
- **To (ending point)**
- **Datetime (predefined date if needed)**

URL Navitia : LINK



Fill out credential info

API https://api.navitia.io/v1

Token ••

Build your path

coverage de

type a key to add a path element ADD

journeys

Add parameters

from stop_area:OBE:de:11000:9001000...

to stop_area:OBE:de:11000:900100003

type a key to add a parameter ADD

Send the request

URL https://api.navitia.io/v1/coverage/de/journeys?from=stop_area%3AOBE%3Ade%3A11000%3A900100009&to=stop_area%3AOBE%3Ade%3A11000%3A900100003&

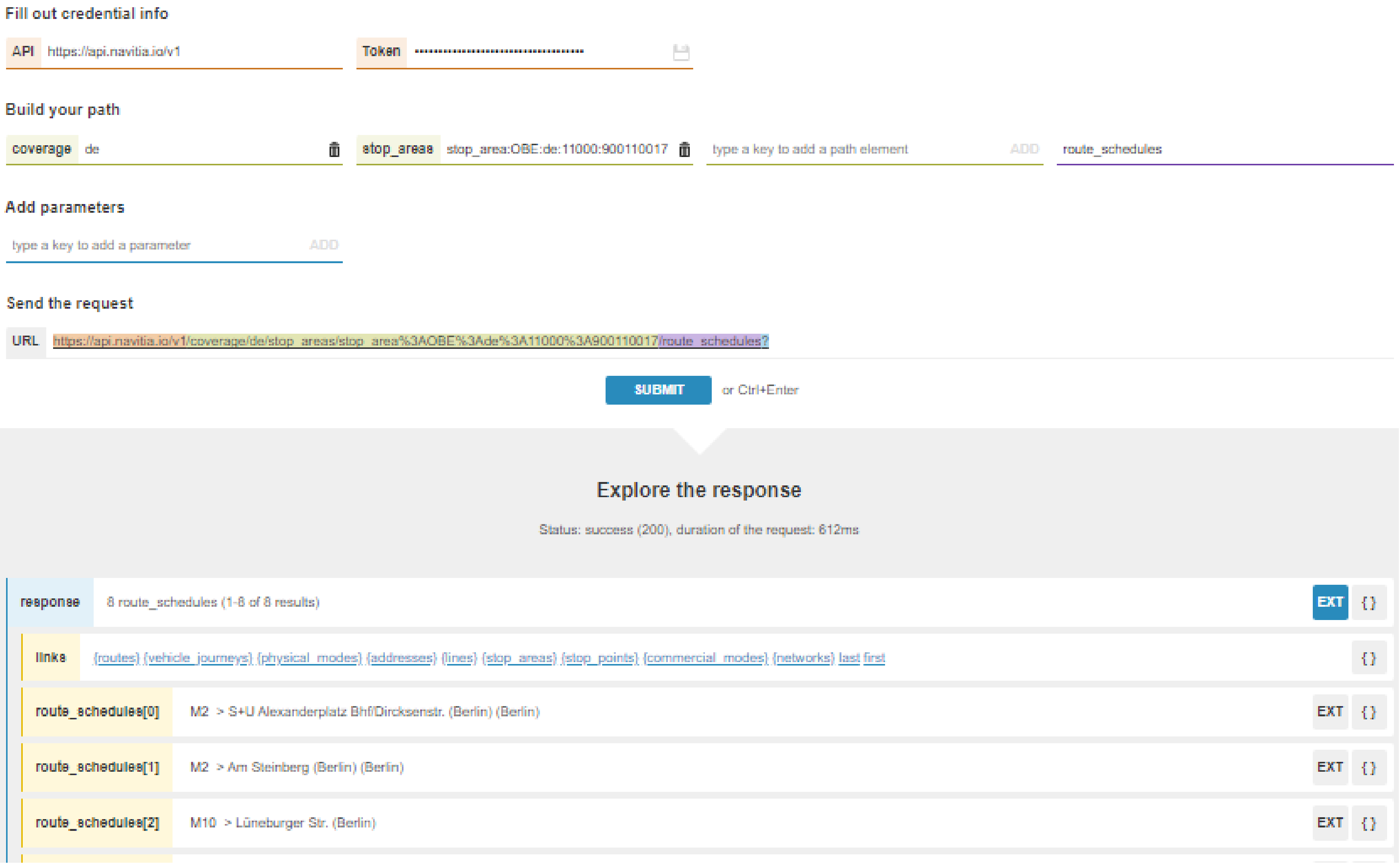
Schedule sheet

API route_schedules:

This API corresponds to the timetable. The purpose of this API is to give the departure times of all the stops of a predefined line.

- *Coverage = de (Germany)*
- *Stop area = Prenzlauer Allee/Danziger StraBe*

URL Navitia : LINK



Fill out credential info

API https://api.navitia.io/v1 | Token ••

Build your path

coverage de | stop_areas stop_area:OBE:de:11000:900110017 | type a key to add a path element ADD | route_schedules

Add parameters

type a key to add a parameter ADD

Send the request

URL https://api.navitia.io/v1/coverage/de/stop_areas/stop_area%3AOBE%3Ade%3A11000%3A900110017/route_schedules?

Explore the response

Status: success (200), duration of the request: 612ms

response 8 route_schedules (1-8 of 8 results)

links {routes} {vehicle_journeys} {physical_modes} {addresses} {lines} {stop_areas} {stop_points} {commercial_modes} {networks} last first

route_schedules[0] M2 > S+U Alexanderplatz Bhf/Dircksenstr. (Berlin) (Berlin)

route_schedules[1] M2 > Am Steinberg (Berlin) (Berlin)

route_schedules[2] M10 > Lüneburger Str. (Berlin)

Stop schedules

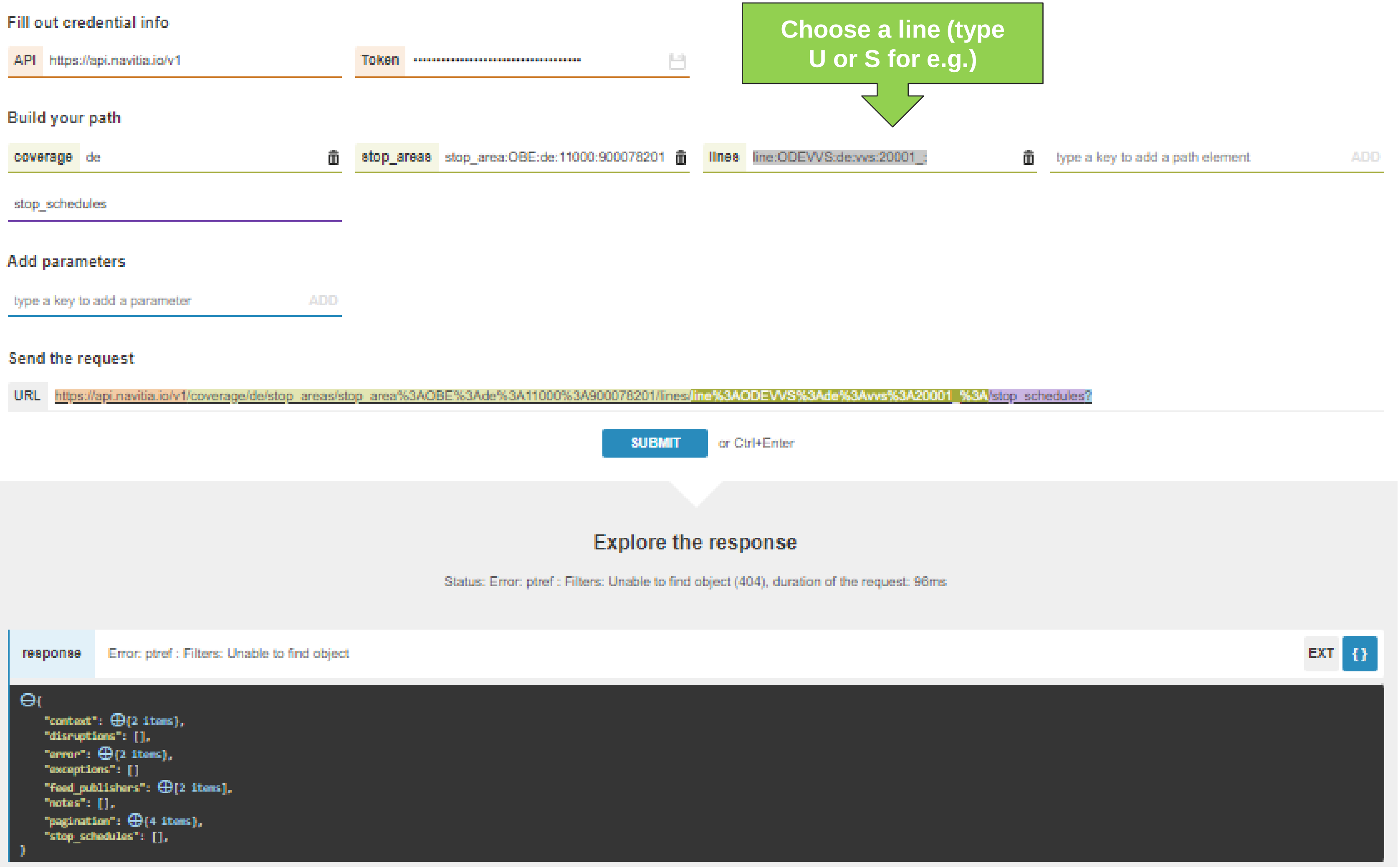
API « stop_schedules » :

- This API is intended to give the times of the next passages on the stops (Neuköln S+U) of a predefined line (U1)

Lines :

- By typping S or U, you find the whole list of S-Bahn and U-Bahn within the Neuköln S+U Bahnof

URL Navitia : LINK



Choose a line (type U or S for e.g.)

Fill out credential info

API https://api.navitia.io/v1

Token ··

Build your path

coverage de

stop_areas stop_area:OBE:de:11000:900078201

lines line:ODEVVS:de:vvs:20001_:

type a key to add a path element ADD

stop_schedules

Add parameters

type a key to add a parameter ADD

Send the request

URL https://api.navitia.io/v1/coverage/de/stop_areas/stop_area%3AOBE%3Ade%3A11000%3A900078201/lines/line%3AODEVVS%3Ade%3Avvs%3A20001_%3A/stop_schedules?

SUBMIT or Ctrl+Enter

Explore the response

Status: Error: ptref : Filters: Unable to find object (404), duration of the request: 96ms

response Error: ptref : Filters: Unable to find object

EXT {}

```
⊖{
    "context": ⊕{2 items},
    "disruptions": [],
    "error": ⊕{2 items},
    "exceptions": []
    "feed_publishers": ⊕[2 items],
    "notes": [],
    "pagination": ⊕{4 items},
    "stop_schedules": [],
}
```

Next Departures

Here we will see how to create a Navitia query in order to have the next departures from a defined stop zone:

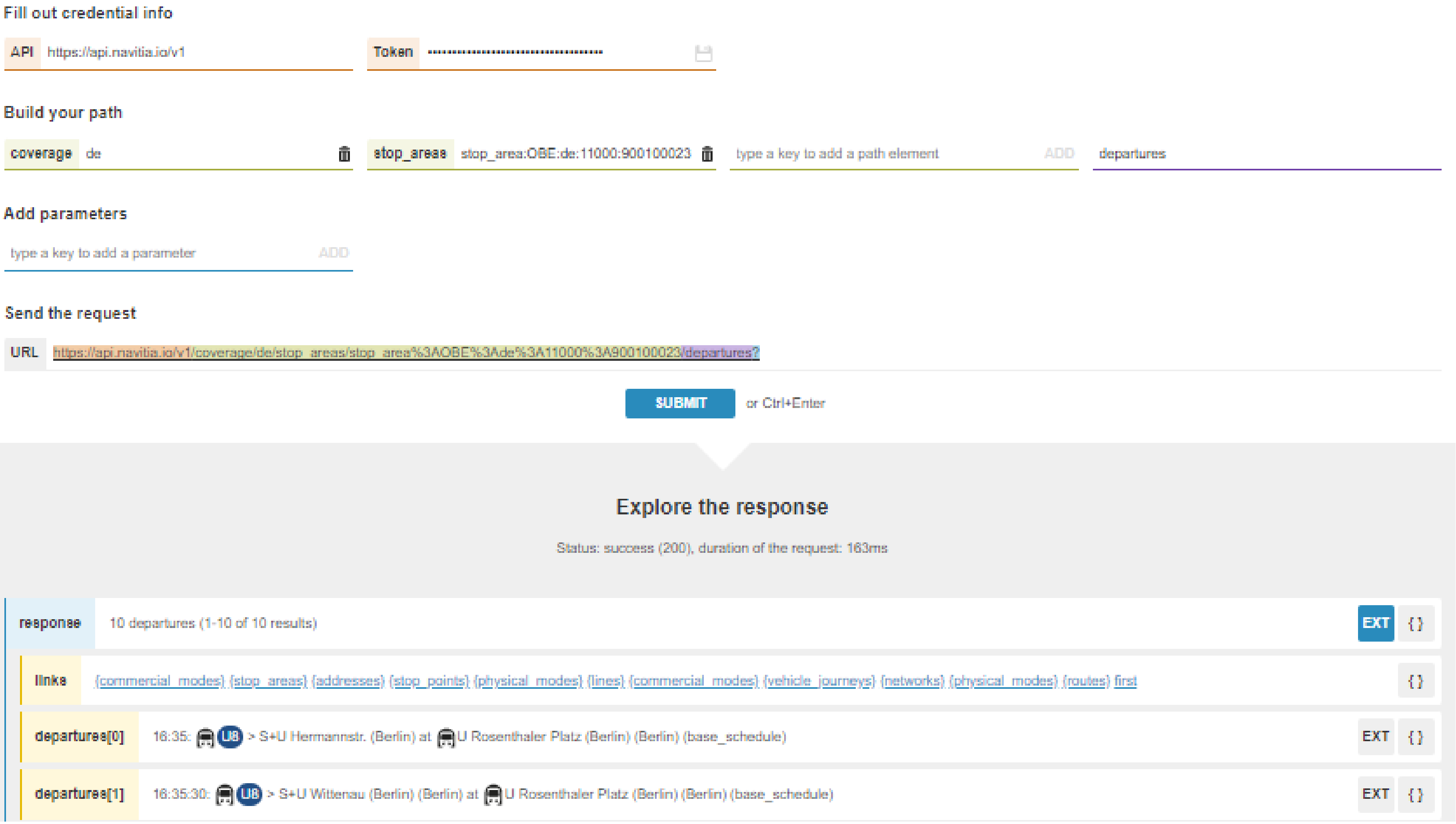
In the Endpoint **you have to add “departures”** (next departures).

Then in the « **path** » you have to add « **stop_area »** and fill in the desired place.

As a reminder, a « **stop_area »** is a group of physical stops closed by.

Screen example : stop_area:U Rosenthaler Platz (Berlin)

URL Navitia : LINK



Fill out credential info

API https://api.navitia.io/v1

Token ••

Build your path

coverage de

stop_areas stop_area:OBE:de:11000:900100023

type a key to add a path element

departures

Add parameters

type a key to add a parameter

Send the request

URL https://api.navitia.io/v1/coverage/de/stop_areas/stop_area%3AOBE%3Ade%3A11000%3A900100023/departures?

Explore the response

Status: success (200), duration of the request: 163ms

response 10 departures (1-10 of 10 results)

links {commercial_modes} {stop_areas} {addresses} {stop_points} {physical_modes} {lines} {commercial_modes} {vehicle_journeys} {networks} {physical_modes} {routes} first

departures[0] 16:35: U8 > S+U Hermannstr. (Berlin) at U Rosenthaler Platz (Berlin) (Berlin) (base_schedule)

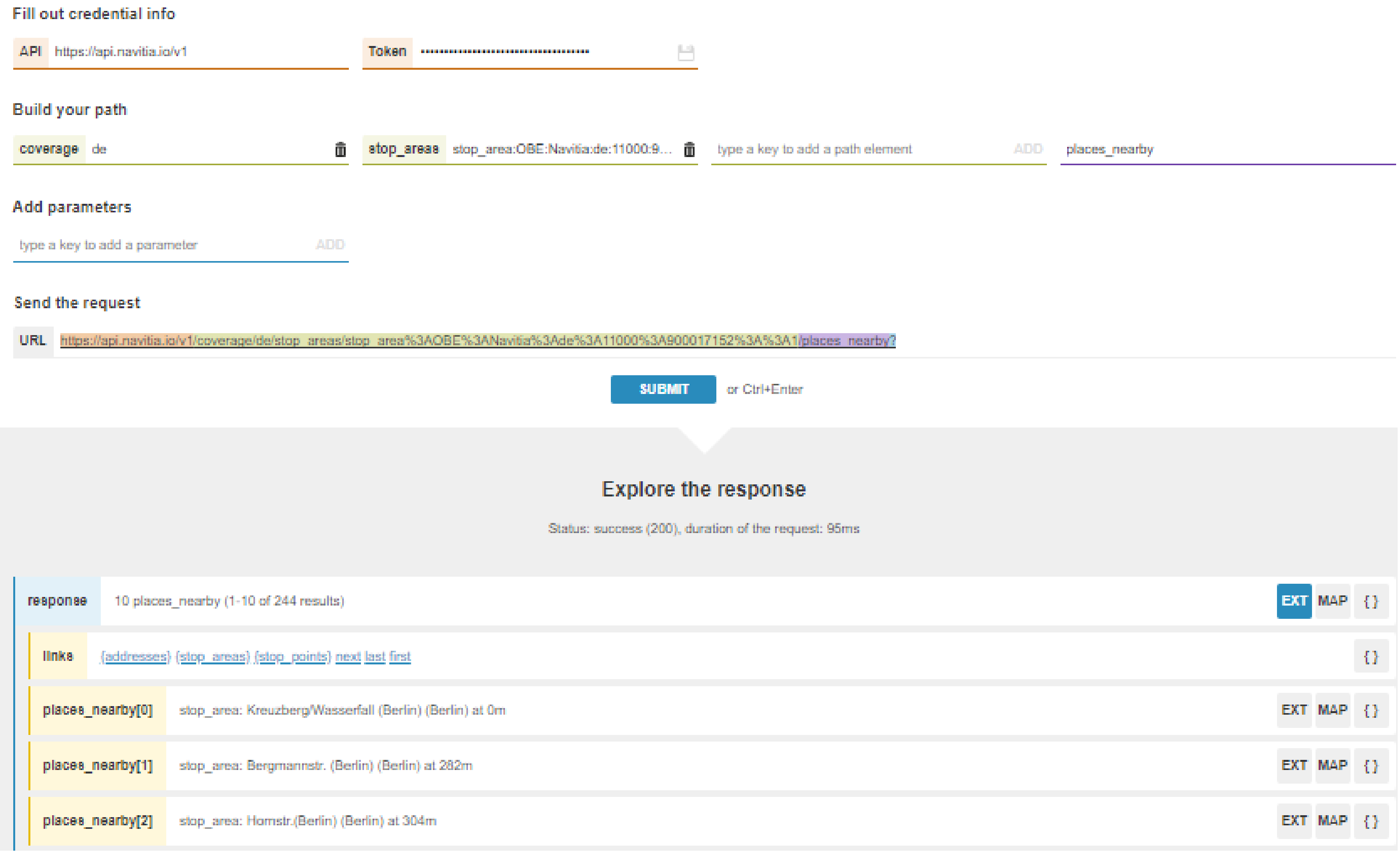
departures[1] 16:35:30: U8 > S+U Wittenau (Berlin) (Berlin) at U Rosenthaler Platz (Berlin) (Berlin) (base_schedule)

Places nearby

The **place_nerby** API displays different transportation options around a location - a GPS coordinate or an address, for example (nearby)

Here, stop area = Kreuzberg/Wasserfall (Berlin)

URL Navitia : LINK



Fill out credential info

API https://api.navitia.io/v1

Token ••••••••••••••••••••••••••••••••••••

Build your path

coverage de

stop_areas stop_area:OBE:Navitia:de:11000:9...

type a key to add a path element ADD

places_nearby

Add parameters

type a key to add a parameter ADD

Send the request

URL https://api.navitia.io/v1/coverage/de/stop_areas/stop_area%3AOBE%3ANavitia%3Ade%3A11000%3A900017152%3A%3A1/places_nearby?

SUBMIT or Ctrl+Enter

Explore the response

Status: success (200), duration of the request: 95ms

response 10 places_nearby (1-10 of 244 results) EXT MAP { }

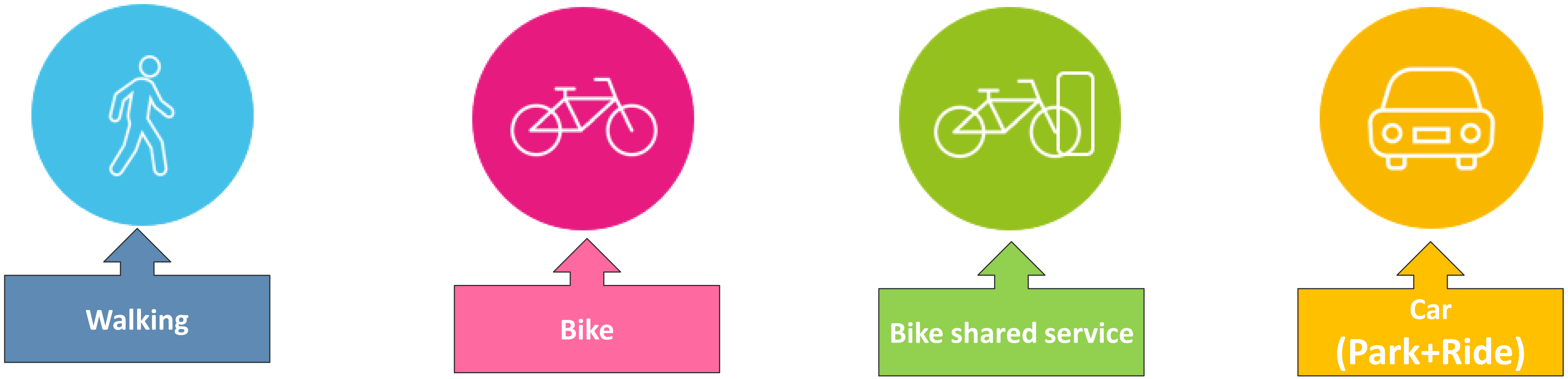
links {addresses} {stop_areas} {stop_points} next last first { }

places_nearby[0] stop_area: Kreuzberg/Wasserfall (Berlin) (Berlin) at 0m EXT MAP { }

places_nearby[1] stop_area: Bergmannstr. (Berlin) (Berlin) at 282m EXT MAP { }

places_nearby[2] stop_area: Hornstr.(Berlin) (Berlin) at 304m EXT MAP { }

Additional parameters



Walking

Bike

Bike shared service

Car
(Park+Ride)

first_section_mode : feeder mode at departure
last_section_mode : feeder mode at the end
direct_path : only end-to-end feeder modes

Additional parameters

Turnaround times

This is the maximum time allowed to reach public transport by feeder mode in seconds

- **max_walking_duration_to_pt**
- **max_bike_duration_to_pt**
- **max_bss_duration_to_pt**
- **max_car_duration_to_pt**

Speed

Speed per mode of reduction in m/s. To get this value in km/h, multiply this by 3.6

- **walking_speed**
- **bike_speed**
- **car_speed**
- **bss_speed**

Accesibility

Setting to use only accessible transit stops in the API calculation

- **wheelchair** = true

Itinerary – Key parameters

Allows you to make a multimodal route from point A to point B

Endpoint: /journeys (itinerary)

Path : coverage = De (Germany), Be (Belgium)…

Key parameter:

- **traveller_type** → traveler profile
- **direct_path** → Only end-to-end feeders modes
- **first_section_mode** → Feeder mode at the start
- **last_section_mode** → Inbound feeder mode
- **data_freshness** → adapted_schedule; base_schedule; real time
- **datetime** → time and date of the route search
- **date_time** → **:** arrival / departure
- **add_poi_infos[]** → Ask in the answer for available spaces bss/parking
- **Forbidden_uris** → prevents the algorithm from using a particular transport object (line/stop_area/network)
- **Allowed_id** → forces the algorithm to use a particular transport object

Description of standard parameters

Description standard	Parameter	PROD
Authorized modes of transport to reach the first section by public transport	**first_section_mode**	walking, bss, bike
Permitted modes of transport to finish the route after the last section by public transport	**last_section_mode**	walking, bss, bike
Speed for the calculation of the duration of the bicycle sections in m/s for the profile	**bike_speed**	4.1
Speed for the calculation of the duration of the bike share sections in m/s for the profile	**bss_speed**	4.1
Speed for the calculation of the duration of the car sections in m/s for the given profile	**car_speed**	11.11
Maximum allowed time to reach public transport by bike in seconds	**max_bike_duration_to_pt**	1800
Maximum time allowed to reach public transport by bike share in seconds	**max_bss_duration_to_pt**	1800
Maximum time allowed to reach public transport by car in seconds	**max_car_duration_to_pt**	1800
Maximum time allowed to reach public transport by foot in seconds	**max_walking_duration_to_pt**	1800
Walking speed in m/s	**walking_speed**	1.1
Wheelchair passenger, true or false.	**wheelchair**	false