

# Backend Developer assignment

Every enterprise software has a built-in support for organizational charts, in order to represent hierarchies and roles inside a company. A common and convenient way to persist tree-like structures in relational databases is the "Nested Set" model (see [https://en.wikipedia.org/wiki/Nested\\_set\\_model](https://en.wikipedia.org/wiki/Nested_set_model) for further details).

## Database structure

The following MYSQL tables contain an organizational chart, along with the role names in various languages, flattened as per the Nested Set model.

Table "node\_tree"

idNode	level	iLeft	iRight
1	2	2	3
2	2	4	5
3	2	6	7
4	2	8	9
5	1	1	24
6	2	10	11
7	2	12	19
8	3	15	16
9	3	17	18
10	2	20	21
11	3	13	14
12	2	22	23

Table "node\_tree\_names" ("idNode" is Foreign Key referencing "node\_tree.idNode")

idNode	language	nodeName
1	english	Marketing
1	italian	Marketing
2	english	Helpdesk
2	italian	Supporto tecnico
3	english	Managers
3	italian	Managers
4	english	Customer Account
4	italian	Assistenza Cliente
5	english	Docebo
5	italian	Docebo
6	english	Accounting
6	italian	Amministrazione
7	english	Sales
7	italian	Supporto Vendite
8	english	Italy
8	italian	Italia
9	english	Europe
9	italian	Europa

10	english	Developers
10	italian	Sviluppatori
11	english	North America
11	italian	Nord America
12	english	Quality Assurance
12	italian	Controllo Qualità

## Requirements Specification

A frontend application needs to fetch organizational chart nodes and display them as a *tree of folders* and, therefore, depends on a backend API to efficiently obtain such data. The candidate is asked to implement a Python script **api.py** to return organizational chart nodes under a certain parent and support pagination.

The script will be called via HTTP (method GET) through the Flask web server. The candidate can use any Flask add-on libraries for REST handling as needed. Please motivate your choices on any such library.

The REST API thus implemented will receive the following input params:

- node\_id (integer, required): the unique ID of the selected node.
- language (enum, required): language identifier. Possible values: "english", "italian".
- search\_keyword (string, optional): a search term used to filter results. If provided, restricts the results to "all children nodes under node\_id whose nodeName in the given language contains search\_keyword (case insensitive)".
- page\_num (integer, optional): the 0-based identifier of the page to retrieve. If not provided, defaults to "0".
- page\_size (integer, optional): the size of the page to retrieve, ranging from 0 to 1000. If not provided, defaults to "100".

The API should return a JSON with the following fields:

- nodes (array, required): 0 or more nodes matching the given conditions. Each node contains:
  - node\_id (integer, required): the unique ID of the child node.
  - name (string, required): the node name translated in the requested language.
  - children\_count (integer, required): the number of child nodes of this node.
- error (string, optional): If there was an error, return the generated message.

## Constraints

- The proposed solution should properly check that all required params are passed and valid and return the following error messages:
  - "Invalid node id" (if node\_id is not found).
  - "Missing mandatory params" (if any required input param is not passed or has empty value).
  - "Invalid page number requested" (if page\_num is not a valid 0-based index).
  - "Invalid page size requested" (if page\_size is outside the validity range).
- The provided code should be Python 3.x compliant and use a mysql library (e.g. "mysql-connector-python") to query the DB.
- Code quality is not optional: comments, clear and meaningful variable names and well designed Python code will be highly considered in the final evaluation of this test.

## Deliverables

The candidate should deliver a zipped archive with the following "minimum" structure:

- api.py (*entry point*)
- config.py (*configuration file containing DB access credentials*)
- tables.sql (*table definition SQL script*)

- data.sql (*data insertion SQL script*)