# Candidate Interview Project: Node

## Project Overview

Welcome to the interview project! This project is designed to assess your skills in building a backend API using Node.

## Goals

* Understand your proficiency with Node.
* Assess your ability to design and implement a scalable API.
* Evaluate your coding practices and problem-solving approach.
* Test your ability to work with different ORMs and databases.

## Project Details

The project includes two SQLite databases, one using Prisma ORM and the other using TypeORM. Both ORMs are already configured within the project. You are free to choose any of them for your test.

The database consists of a single table:

**Table "pokemons"**

* id: Int
* name: String
* type: String
* created\_at: Datetime

## Setup Instructions

1. Clone the Repository: <https://github.com/MatheusGNachtigall/Backend-Node-Test>
2. Follow installation guide on README.MD.
3. Run the Project.
   1. (If using GraphQL): Access GraphQL Playground: Navigate to <http://localhost:4000/graphql>.
   2. (If using REST): Navigate to <http://localhost:4000/hello>

## Task Instructions

* First of all, we don’t expect you to complete everything we’ve proposed here; we just want you to do your best and showcase your skills.
* Our evaluation will focus on the quality of your solution, the code you write, as well as your efficiency and time management.
* You don’t need to complete all the 'Bonus' features in order. Work on whichever tasks you feel comfortable with and believe you can accomplish within the defined deadline.

### Tasks

* **Required:** Implement CRUD Operations (as resolvers or REST endpoints) for the pokemons Table:
  + createOnePokemon
  + updateOnePokemon
  + deleteOnePokemon
  + findManyPokemon
* (Bonus) Enhance the API with Extra Features
  + Query filters (e.g., filter by type, partial name).
  + Pagination.
  + Sorting (by name in ascending or descending order)
  + Rate limiting.
  + Caching for performance optimization.
* (Bonus) Data Validation and Error Handling:
  + Ensure input validation using class-validator.
  + Handle errors and return appropriate responses.
* (Bonus) Testing:
  + Write unit tests.
* (Bonus) Convert the “type” field into a many-to-many table

**Table "types"**

id: Int

name: String **(unique)**

created\_at: Datetime

* (Bonus) Create an “importPokemonById” mutation/rest endpoint: This endpoint should fetch data from the official PokeAPI site (<https://pokeapi.co>) and add/update the record based on its ID.
  + For instance, if we add the pokemon with id “158”, the endpoint should add “totodile” (<https://pokeapi.co/api/v2/pokemon/158>) to our database, **OR** update the record (let’s say we previously added a pokemon with id 158 with the “createOnePokemon” resolver).
* (Bonus) **Go Wild! Do you feel like there’s anything else you would like to include? Do so! Add as many extra features as you want. Write down in the README what you added and why.**

## Evaluation Criteria

* Code Quality and Structure.
* Error handling and input validation.
* Test coverage.
* Ability to implement extra features.
* Clear and maintainable code.

## Submission

* Create a public repository on your personal GitHub and send the link to the reviewers
* Include a brief README with any assumptions or additional instructions.

**Good luck!**