COGNIGY.AI

Coding Challenge
For the backend developer position - variant 2

Cognigy GmbH

January 16, 2020

Chapter 1

Introduction

The current file contains all information you will need in order to work on our coding-challenge. Please focus on the contents of this file - don't waste your time and make the challenge more complex as it is! If you finished your first implementation, feel free to extend your solution, but ensure, that you fullfil at least the minimal requirements listed.

Chapter 2

Your challenge

2.1 The tools

Build a Node.JS based application and use the following:

- modern Node.JS (node 12), ES7 / ES8
- use a linter
- use the mongoose package from NPM
- use the express package from NPM
- use the ajv package from NPM

Containerize your solution by writing a valid **Dockerfile** which packages your application on top of a public NodeJS Docker-base-image. Please also provide a **docker-copmose** file which references your Docker image and the official MongoDB docker image.

Feel free and optionally use the following:

- use Typescript
- use TSLint instead of ESLint

2.2 Your task

Build an application which uses **the Express.JS framework** to expose a RESTful API to manage **cars**. Cars is an artificial resource which you can freely design. A car should have at least 5 different properties, e.g. brand, color, model etc. You can freely decide on the properties of a car - please ensure that you implement the following functionality in form of valid RESTful HTTP endpoints:

- Create a new car.
- Read meta-data of all cars in the system. This endpoint should really only return meta-data - there is no pagination required.
- Read the full data of an individual car.
- Delete an individual car.
- Updaten single properties of a single car (not a full replace operation!)

Your backend application should connect to MongoDB using the mongoose package. Establish the connection on startup of your application. There is no need to configure access-control for MongoDB - feel free to use a fully un-authenticated database. Your individual API handlers have to update the data-set within the database. Ensure that you implement JSON-schema validation using the **ajv** package. You must ensure that the user of your API can't send additional properties into your API (and the database) by providing additional keys in the API requests and their JSON-payloads. There is no need for you to implement access-control using some sort of API-key mechanism - your API can be completely public!

Bonus task Use Typescript for your whole application and add a simple **x-api-key** based authentication mechanism to your application.

2.3 What you should focus on

The following things are important and we will have a look at specifically them:

- is your implementation functional, can it be dockerized, started and can it connect to MongoDB?
- naming variables & files
- usage of comments
- common file indentation for all of your files
- is there any information on how to use your implementation (e.g. README.md)? We want to get your application running, so give us a step-by-step guide of what we need to do!