Practical Test – NodeJS

3 Hours

Parker Inc, is a company situated in United Kingdom. It needs a simple API to **create** and store basic employee details in a database. The IT department has prepared the following requirement which needs to be implemented in the initial stage.

1. You are expected to create an API as per the following specification.

Field	Type	Validation Rules
name	String	Required Max characters: 150
email	String	Required Max characters: 75
profile_picture	Base64	Optional Maximum file size: 10MB
created_at	Timestamp	Time should be in "Europe/London"
modified_at	Timestamp	Time should be in "Europe/London"
status	ENUM("Active", "Deleted")	

   a. Design the database tables as required. You may use any database server.
   b. You can use NodeJS with any framework.
   c. Appropriate HTTP status codes need to implement for responses.
   d. Unit testing should be implemented as required.

Notes: Keep in mind the following points when creating the API:

- Use most appropriate data types for database fields.
- Use best practices and standards for coding.
- Use OOP concepts.
- Use proper validations where required.
- You will not be allowed to use your previous codebase to develop this application.
- Try to provide complete solutions within the given time to meet this requirement.

Submission Guidelines:

The submission should be in the form of a link to a GitHub/Bitbucket repo containing your code. Please include instructions in your README for setting up and executing your code. High-level descriptions regarding your approach would also be welcome.

General Guidelines

- We expect well-designed and clean code. Remove all unwanted/unused codes.
- Use proper naming convention throughout. Use meaningful names that convey the purpose of the variables. Choose names that are easy to pronounce and avoid cryptic abbreviations.
- Documentation and comments should be included as appropriate. Comments are structured as sentences with proper grammar and punctuation and without spelling mistakes. If time is a constraint, provide at least one sample section. Also, you are free to use code formatting tools.
- Use abstraction and modularity as needed, separate application and presentation logic, implement design patterns where appropriate. Consider the reusability, maintainability, testability, and changeability. Consider the S.O.L.I.D design principles.
- Do not reference unnecessary libraries, modules, or classes. Pay attention to small things that can improve building time, minimize chance for mistakes.
- Make sure to follow good programming logic with best practices and coding standards wherever possible.
- Don't ignore or suppress errors. Make sure they don't happen in the first place.
- Avoid long chunks of code (for instance large methods).
- Make sure your application is easily configurable.

Note: ***While using online resource is recommended as an efficient use of time, directly copying and pasting code without attribution during an exercise is not recommended when testing your coding ability.***