Pre-Employment Assignment

Document Version: 1.1

Last Updated Date: 01/26/2018

Table of Contents

1. Introduction

The purpose of this document is to provide pre-employment assignment to potential candidates who are willing to join our organization

Document Change Control

Version	Modified By	Date	Comment
0.1	Om Talsania	09/15/2017	Initial draft
0.5	Palak Halvadia	09/25/2017	Adding more details to assignments
0.9	Chandni Patel	09/29/2017	Reviewed content and cosmetic changes
1.0	Om Talsania	10/16/2017	Published
1.1	Om Talsania	01/26/2018	Content changes

2. Assignment

The candidate will need to go through any one (or more) of the relevant assignments in this section (section #2). While performing the assignment, he/she needs to make sure to follow the conventions mentioned in section #3.

After completing the assignment, the code needs to be pushed to github, and the candidate needs to share the link with us.

The code should have a README.md and details of the solution can be mentioned there.

There should be a script for creating database, collection and sample records.

2.1 MEAN Stack (Backend)

SCENARIO

You are working as an engineer for a medical research center and you are tasked with creating an API for your internal IT department. The API should be authenticated via a simple token mechanism, by having a login API that should take username/email and password that generates a token valid for 24 hours or one calendar day. The other APIs should accept this token in header and validate it before rendering any values.

The main set of APIs that your IT department needs is endpoints to register an asset, modify an asset, delete an asset, or search for an asset.

The criteria of search can vary and not just limited to searching by ID.

TEST SCENARIOS

Method	API	Payload	Expected Result

GET	api/login	{
		email: '',
		password: ''
		}
			{
			status: 'Authenticated'
			token: 'abcxyz'
			}

GET	api/assets	token in header	
GET	api/assets/10	token in header	asset with ID 10
GET	api/assets?registration_date[gt]=1/1/2018	token in header	all assets with registration_date greater than 1/1/2018

POST	api/assets	{
		field1: '',
		field2: ''
		}
			Object gets created in DB and returned. You don't pass ID in payload, but you get object with ID in return

PUT	api/assets	{
		status: false
		}
			Deactivates asset

DELETE	api/assets/10	token in header	delete the asset with ID 10

2.2 MEAN Stack (Frontend)

SCENARIO

You are working as an engineer for a medical research center and you are tasked with creating a small asset management application for your internal IT department. You can use the APIs from assignment 2.1 or you mock APIs if your focus is on front-end only.

The system should have one clean login page. Registration or forget password links are not expected. After logging into the system, you need to have a list of assets along with search/filter. You can use either popups or clean screens for adding/updating assets.

3. Development Guidelines

We follow certain guidelines for our coding best practices. The following external guidelines are the basic guidelines that we need to follow along with our specifications.

3.1 MongoDB Conventions

Collection names

- The name should be a plural of the types of documents to be saved.
- Use camelCase. Normally you shouldn't have to because the collection name will be one word (plural of the types of documents saved).
- A collection with "" *empty string* is not a valid collection name.
- A collection name should not contain the *null* character because this defines the end of collection name.
- Collection name should not start with the prefix "system." as this is reserved for internal collections.
- It would be good to not contain the character *"$"* in the collection name as various drivers available for database do not support *"$"* in collection name.

Database names

- Try to have the database named after the project and one database per project.
- Use camelCase.
- A database with *""* *empty string* is not a valid database name.
- Database name cannot be more than 64 bytes.
- Database name are case-sensitive, even on non-case-sensitive file systems. Thus it is good to keep name in lower case.
- A database name cannot contain any of these characters *"/, \, ., ", *, <, >, :, |, ?, $,"*. It also cannot contain a single space or null character.

Field names

- Use camelCase.
- Don't use _ *underscore* as the starting character of the field name. The only field with _ *underscore* should be _id.
- Field names cannot contain *. dots* or *null* characters and must not start with a *$ dollar sign*.

3.2 Angular Coding Conventions

You need to follow the conventions mentioned in the link below

https://angular.io/guide/styleguide

3.3 NodeJS Coding Conventions

You need to follow the conventions mentioned in the link below

https://www.perfomatix.com/nodejs-coding-standards-and-best-practices/

3.4 RESTful API Conventions

You need to follow the conventions mentioned in the link below. This is primarily for .NET Web APIs, but the same concepts can be applied in other stack as well.

https://github.com/Microsoft/api-guidelines/blob/vNext/Guidelines.md

3.5 UI/UX & Content Styling Guidelines

You need to follow the conventions mentioned in the link below

https://styleguide.mailchimp.com/writing-for-accessibility

4. References

1. https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md
2. https://github.com/Microsoft/api-guidelines/blob/vNext/Guidelines.md
3. https://angular.io/guide/styleguide
4. https://styleguide.mailchimp.com/writing-for-accessibility
5. https://www.perfomatix.com/nodejs-coding-standards-and-best-practices/