Dot Net Entrance Evaluation

Nexlesoft Cooperation, May 2022

# OVERVIEW & PURPOSE

This is the entrance test for Backend Genius. Have a good day and good luck :)

# REQUIREMENT

**Objective**

Your task is to create a Dot Net API that serves Authentication feature. The code must be clean, readability and maintainable.

**Requirements**

* You will need to implement the following 4 end-points. For detail, you can view ***Reference 2***
  + Signup
  + Sign in
  + Sign out
  + Refresh Token
* Technologies
  + [mysql](https://www.npmjs.com/package/mysql) database. For tables on MySQL, you can view ***Reference 1***
  + [Entity Framework Core](https://docs.microsoft.com/en-us/ef/core/) library to access to the mysql database
  + [JwtBearer](https://www.nuget.org/packages/Microsoft.AspNetCore.Authentication.JwtBearer) library for token validation
  + [ASP.NET Core Identity](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity) library to manages users, passwords, profile data, roles, claims, tokens, email confirmation, and more.
  + [Microsoft.NET.Test.Sdk](https://www.nuget.org/packages/Microsoft.NET.Test.SDK) and [xUnit](https://xunit.net/) for unit testing
* The implementation is uploaded to github and send back to Nexle the repo link.
* It’s a plus if the code implementation is covered by some unit tests

# Reference 1: Database MySQL

* The mysql database has these tables
  + Users
    - id: int
    - firstName: varchar(30)
    - lastName: varchar(30)
    - email: varchar(250)
    - password: varchar(250)
    - updatedAt: datetime
    - createdAt: datetime
  + Tokens
    - id: int
    - userId: int (ref to Users (id) table)
    - refreshToken: varchar(250)
    - expiresIn: varchar(64)
    - updatedAt: datetime
    - createdAt: datetime
* There’s a mysql server for development which you can access using the following info
  + IP: 178.128.109.9
  + Username: test01
  + Password: PlsDoNotShareThePass123@
  + Database: entrance\_test

# Reference 2: APIs

## Sign up

* HTTP method: POST
* Inbound
* email
* password
* firstName
* lastName
* Outbound:
  + Response is an object which has these fields as shown in the following example

*{*

*"id": "<id of the user in the database>",*

*"firstName": "<user first name>",*

*"lastName": "<user last name>",*

*"email": "<user email>",*

*"displayName": "<firstName + last Name>"*

*}*

* + Response http code
    - 201 http code on success
    - 400 http code on validation error
    - 500 http code on internal error
* Validation
  + The api should validate if the email is available for the signup or not.
  + The api validate if the email in in correct email format or not.
  + Password must be between 8-20 characters
* Other requirements
  + The password must be encrypted using bcrypt before saving to the database

## Sign In

* HTTP method: POST
* Inbound
* email
* password
* Outbound:
  + Response is an object which has these fields as shown in the following example

*{*

*"user": {*

*"firstName": "<user first name>",*

*"lastName": "<user last name>",*

*"email": "<user email>",*

*"displayName": "<firstName + last Name"*

*},*

*"token": "<jwt token>",*

*"refreshToken": "<jwt refresh token>"*

*}*

* + Response http code
    - 200 http code on success
    - 400 http code on validation error
    - 500 http code on internal error
* Validation
  + The api validate if the email in in correct email format or not.
  + Password must be between 8-20 characters
* Other requirements
  + The token expires in one hour
  + The refreshToken expires in 30 days
  + The refreshToken once created, it must be saved in **tokens** table in database

## Sign out

* HTTP method: POST
* Inbound: none
* Outbound:
  + Response: none
  + Response http code
    - 204 http code on success
    - 500 http code on internal error
* Validation
  + None
* Other requirements
  + The Api should remove all the refresh tokens belong to the user account in refreshToken table.

## Refresh Token

* HTTP method: POST
* Inbound
* refreshToken
* Outbound:
  + Response is an object which has these fields as shown in the following example

*{*

*"token": "<jwt token>",*

*"refreshToken": "<jwt refresh token>"*

*}*

In which

* + - token: is a new token which will expire in one hour
    - refreshToken: is a new refresh token which will expires in 30 days
  + Response http code
    - 200 http code on success
    - 404 if the supplied refreshToken in the inbound does not exist.
    - 500 http code on internal error
* Validation
  + The api validates if the refreshToken in the inbound exists or not.
* Other requirements
  + Invalidate the old refreshToken
  + None