# Developer Survey and Coding Exercise

## Technical Expertise

What is your level of experience with the following technologies and frameworks?

Please answer each question with **1** for "no experience", **2** for "some experience" or **3** for "considerable experience".

### Languages

* C#: 1
* Javascript: 3
* Typescript: 3
* HTML/SCSS: 3

### Frameworks and Technologies

* .NET: 1
* .NET Core: 1
* Angular: 3
* NgRx: 3
* GraphQL: 2

## Experience and Training

Please list any computer-related training, qualifications and work experience you have.

* Training and qualifications:
  + Telematics engineer degree
  + UDEMY courses
  + Degree project:
    - Web app that evaluates the credit risky in a credit request using an Artificial Intelligence model and show in a dashboard
    - Frontend development using Angular and SASS
    - Backend development using NestJS (a NodeJS framework) following the microservices pattern
    - The AI model is a Random Forest Regressor developed with SCIKIT in Python
    - The DB is developed with PostgreSQL
    - AWS is used for the cloud of the project, where is used Cognito for authentication, RDS for database, S3 for saving documents and EKS for Docker containers
* Work experience:
  + Fullstack developer intern at Mnemo (Mar 2022 – Mar 2023)
    - Frontend development using Angular as Framework, SASS as CSS
    - Backend development using Express
    - NGINX for web server and Docker for containers
    - Microsoft Azure for cloud system
  + Backend developer at GAC Solutions (Mar 2023 – Jun 2023)
    - Backend development using NestJS as backend framework (a NodeJS framework), MySQL as DB manager and Docker as deployment method, all in AWS Cloud.
  + Fullstack developer at Conten-oh! (Jun 2023 – Oct 2023)
    - Frontend development using React
    - Backend development using Serverless framework with Vanilla NodeJS
  + Product Analyst at Viva Aerobus (Jan 2024 – May 2024)
    - Project management using Monday software.
  + Backend engineer and AI engineer at FP Alpha (May 2024 - Now)
    - Python web development using Flask
    - NodeJS web development using Express and NestJS
    - AI prompts of OpenAI library in Python

## Exercise

The goal is to create a simple server-client "To-Do" application using .NET Core as the server and Angular as the client. The data sent between server and client will be in JSON format. In the ZIP file you received you will find the base source code with instructions on how to compile and run it.

For now, we have only created a basic server controller returning a list of to-do items and a simple client app which shows the list to the user. To-do items must be formatted in title case (the first letter of every word should be capitalized).

Feel free to add any comments that could be useful for the reviewer. In addition, unless otherwise indicated, you can use any library of your choice.

Please reply to the questions of this exercise directly in this document to submit it back for review.

### Pre-requisites

Run the following commands to initialize a local Git repo in the root of the project. You would need to have Git installed.

git init

git add .

git commit -m "Initial commit"

For every section of this exercise, please create a new commit after completing it with these commands:

git add .

git commit -m "EXERCISE\_SECTION"

For example: git commit -m "Refactor"

### Code Review

Review the existing code in the following files:

#### Server

* **src/Todos/Controllers/TodosController.cs**
  + Take note of the ToTitleCase function and its unit tests.
* **tests/Todos.Tests/Controllers/TodosControllerTests.cs**

#### Client

* **client/src/app/todo-list/todo-list.component.ts**

Please, have the following points in mind:

* Standards, good practices you know of
* Code re-usability (for example, we may need to use ToTitleCase in other places too)
* The input data could be very large (your program could be run on a machine with limited memory)

**Please list all the things you would change or improve in the code specifying the line number whenever necessary and explain the reasons.**

* Server:
  + Create CRUD functions and endpoints to separate each functionality
  + Store all the data in a DB (for this case, the DB will be store in JSONs files)
  + Create a DTOs to ensure the data won’t be modified or the server won’t get other data structure
  + Create responses objects for either the successful responses or the error responses
  + Create a global exception handler
  + Following the MVC pattern, create separated files for the model of the to do item, the view will be handled in the frontend, and the controller of the logic of the to do list
  + All the functions that are not related in the To Do main logic will be set in a utils file
  + Use built-in methods.
  + Use the correct encapsulation for every method
* Client:
  + Create a service file where the frontend will request data to the server.
  + Create interfaces file to declare the type of data.
  + Separate HTML and CSS into corresponding files
  + Following the smart - dumb component pattern, create a smart component that handles the logic of the To Do item and a dumb component that will render the To Do list.

**What possible enhancements would you consider?**

* Use a database designed for production use (in some cases when the DB design needs an integer auto-increment primary key, a database system like SQLServer would help with the automatic assign, increment and reassign of IDs if an item is deleted)
* Use roles and permissions
* Implement CSS framework to stylish frontend like [Bootstrap](https://getbootstrap.com/)

### Refactor

Refactor the code in Controllers/TodosController.cs and todo-list/todo-list.component.ts applying the changes and improvements you suggested in the code review.

### Add Create and Delete To-Do Items

Add the ability to create and delete to-do items. User input and stored data should always be valid and safe. You may store data in a text file or in a in-memory database.

**What kind of storage would you use in a real application to store the to-do items? Explain why.**

### Add New Fields

Add a couple of new optional fields to the to-do items: due date and notes.

Create a new detail view to edit a single to-do item. From the current list, the user should be able to click one and navigate to the new view containing:

* Title
* Indication of completion
* Due date
* Notes

The user should be able to edit the fields and save or cancel the changes.

Feel free to enhance the user interface.

### Add a Store (Optional)

Using NgRx, implement a store in the client app.

**What are benefits of using a centralized store? Explain.**

**What factors would you consider when designing the data structure? Explain.**

### Additional Questions

**How would you implement translations?**

**What problems would providing internationalization and localization for many languages have?**

**How would you ensure the accessibility of the app?**

**What types of tests would you implement to ensure QA?**

**How would you implement security in your application? Explain.**

**How would you improve the application to support a high number of concurrent users? Explain.**

## Final Steps

* Run in the root folder a git command to remove all build assets: git clean -dfxi
* Zip up the source files (excluding the node\_modules, binaries and build folders)
* Make sure to include in the zip the hidden .git folder
* E-mail it back to us with this document filled out