**The Test:**

The coding test consists of building a web application which contains Front End and Back End. The coding exercise should not take more than a couple of hours but feel free to take as much time as you want. You can find Both Front End and Back End applications in this repository with some functionality implemented. Your task is to complete the application and make it functional. You can apply any resilience patterns (like Polly), but that is not mandatory. We will be looking and paying attention to the implementation of the code and use of the best practices. We don’t expect to see many design patterns for the entire application but you can apply your Designt Pattern knowledge to the BalanceChecker.cs refactor task (explained below).

**Angular Front End**

The UI should allow the following functionality:

* Display all the Accounts
* Create an Account
* Allow to filter accounts (using dropdown) based on the Account type
* Account type information should be loaded from the REST API and not hard-coded in the UI

The following Account Information should be displayed:

* Account First Name
* Account Last Name
* Account Type
* Account Balance
* Account Address

**Display All Accounts**

Display All Accounts screen should display all the accounts in an HTML table. Observables should be used to trigger filtering of the selected account type in the dropdown component. Do not use Emit for this purpose.

**Create an account**

You only need to save First Name, Last Name and Balance

**C# REST API**

The code contains REST Api project in Visual Studio solution using .net core 3.1. Do not use Mediator and do not use AutoMapper.

The REST API should allow to:

* Get All Account Types
* Get All Accounts
* Create Account

**Account Type**

The account can be of a few types:

* Silver
* Bronze
* Gold

Account type depends on the Balance value and should change based on the following rules:

* Silver < 5,000
* Bronze > 5,001 and < 10,000
* Gold > 10, 001

**Balance**

The source code repository contains *BalanceChecker.cs* file. You need to refactor this class based on best practices and write Unit Tests for it. Call the Process method in the BalanceChecker class to check if it is allowed to save the balance.

**Account Address**

When loading all the Accounts, you need to load a new address for every account.

The code to load the address information has been implemented. You need to change the code which loads the Address using HttpClient, but you can keep the GetCityAndPostCode method. Use only HttpClient to do the request, don’t replace it with any 3rd party libraries, like Flurl or RestShap.

**Testing**

Write all the types of tests you usually do: Unit Tests, Integration Tests, REST API Tests, Angular / Jasmine Tests. Don’t write SpecFlow tests. You don’t need to cover the entire set of APIs (unless you do TDD for all the layers and that’s the normal process you follow).

**Checklist**

This is a short list of the tasks you need to implement:

**Angular**

* account.component
  + Load Accounts from the REST Api
  + Display Accounts in the HTML Table
  + Filter Accounts based on the Account Type
* account-type.component
  + Load Accounts Types from the REST Api
  + Observable should be used to notify the account.component about changes in the selected Type
* new-account.component
  + Save account

**C#**

* Account TypeController
  + Create REST Endpoint to load Account Types
* AccountController
  + Create REST Endpoint to load all the Accounts
  + Load a new address for every account
  + Create REST Endpoint to Save Account
* Refactor BalanceChecker.cs and call its Process method