# # .NET Technical Test #

This document explains the technical test, the candidate must go through in order to demonstrate programming, analysis, organization capabilities.

Candidate must obey the following rules:

- Candidate may use freely online documentation to achieve the goals.

- Candidate must do this exercise alone, without receiving external help.

- Evaluators may contact the candidate to explain the approach, strategy and key elements.

## ### Pre-requisites ###

\* Visual Studio or VS Code prepare to build this project successfully.

\* Email address to send BitBucket temporal invitation.

\* .NET Framework 4.7.2

## ### Goal Summary ###

***\*\*First goal\*\****: candidate must implement a kind of banking-system, which will be capable of some basic operations that will be further detailed and explained in this document.

This banking system must be able to perform the required business logic, on the expected terms and observing the restrictions set.

***\*\*Second goal\*\****: the functionality provided by the banking system must be properly tested, by making use of unit tests.

## ### Business entities ###

The program will manage 2 system entities:

- **\*\*Banking-system\*\*** itself: providing business functions explained below.

- **\*\*Customer\*\***: are the entities managed or operated by the banking-system and the target of this application.

## ### Banking System ###

The banking-system is an entity that manages a static list of customers and is able to perform some business operations, that are listed below:

\* **BF1** - Initialize and load data

\* **BF2** - Transfer money between customers

\* **BF3** - Calculate fees

\* **BF4** - Count customers

\* **BF5** - Search customers (by name or id)

All these business functions are explained below.

### #### BF1 - Initialize and load data ####

In order to operate, the banking system needs, first of all, to load and process a list of sample customers stored in a file in the form of .csv data. See \*Customer database\* sub-section below.

As the data is loaded, the customer entities will be created in memory.

**\*\*Initialization must happen once, and only once\*\***. If the data is tried to be loaded more than once time, the banking-system must raise and exception.

#### ##### Customer database #####

There is a customer database provided in the form of a csv file **\*customers.csv\*** that candidate must load onto the banking-system, as part of the initialization process.

Candidate must treat the file taking into account the following points:

- Field delimiter: **;**

- Row delimiter: **CRLF**

- First row contains column names: **YES**

### #### BF2 - Transfer money between customer ####

It's a business function whose purpose is transferring money from one customer to another.

|  |  |
| --- | --- |
| **Argument** | **Meaning** |
| Giver Id | Id of the customer that \*gives\* the money |
| Receiver Id | Id of the customer that \*receives\* the money |
| Amount | Amount to be transferred |

The banking-system will:

\* Validate the operation observing the conditions explained below.

\* Calculate the fees according to the fees below.

\* Deduct the amount (including fees) from the \*giver's\* balance.

#### ##### Transfer Fees #####

In the CSV there is a flag, indicating whether a customer is **\*internal\*** or **\*external\***. By **\*Internal\*** means, is owned by the banking-system, being **\*external\*** the contrary.

Depending on this feature of a customer, there are following **\*fees\*** to apply:

|  |  |  |
| --- | --- | --- |
| **Type of transfer** | **Amount** | **Fees** |
| Giver is internal, receiver also | From 1 to 100 | 0 Fees |
| Giver is internal, receiver also | From 100 | 5 Fees |
| Giver is internal, receiver external | any amount | 10 Fees |
| Giver is external | error - not possible | error - not possible |

#### ##### Conditions #####

**\* \*Giver\*** should have enough funds available, also to cover the transfer FEES. If not, the banking system should raise an error.

\* Transfer money function should raise an error if he **\*giver\*** customer is not an internal one (owned by the banking-system).

### #### BF3 - Calculate fees ####

The banking system must expose a function that must return the FEES applicable for transferring money.

Function takes the following arguments.

|  |  |
| --- | --- |
| **Argument** | **Meaning** |
| Giver Id | Id of customer that \*gives\* the money |
| Receiver Id | Id of customer that \*receives\* the money |
| Amount | amount of money to be transferred |

### #### BF4 - Count customers ####

The banking system must be able to return the total of customers loaded in the system.

### #### BF5 - Search customers (by name or id) ####

**\*\*Search by name\*\***

Banking-system must be able to search by name, using a \*wildcard\* kind of search. Additionally, the search must be partial and case insensitive.

For the search, the Banking-system, must take into account both First Name and Last Name properties of a customer; so, the search term could be either found on first name or in last name.

**\*\*Search by Id\*\***

Taking as input a number; the banking-system must return the matching customer. It is an exact match.

### ### CUSTOMER MODEL ###

A customer on the banking-system will have, at least, the following properties:

\* Id (number, unique)

\* First name, last name.

\* Phone1, phone2.

\* Type of customer: whether the customer is internal or external.

\* Balance: the amount of money the customer has on its account.

## ### UNIT TESTS ####

The banking system must be properly tested. Candidate must implement, at least, the following unit tests:

- Banking-system must not transfer money if the \*giver\* is non-internal customer.

- Amount to transfer must be positive and greater than 0.

- Banking-system can only be initialized once in during the program lifetime.

- Using the banking-system search by name function, count the number of customers having on any part of the name "meaghan".

- Transferring money from customer with id 187, to customer id 188 should not work, as 187 is not an internal customer.

- Test that the fees for transferring an amount of 50 money from customer 34 to customer id 22 is free of charge.

- Test that the fees for transferring an amount of 100 money from customer 34 to customer id 22 is 5.

- Test that the fees for transferring an amount of 75 money from customer 34 to customer id 76 is 10.

- Test that transfer from customer id 41 to customer id 75, for an amount of 40, is not possible, because the total amount (amount + fees to be charged), exceeds the customer balance.

- Test that transfer from customer id 41 to customer id 75, for an amount of 10, is possible and that final balance of the customer 41 is equals to 30, and balance of customer id 75 is equals to 84.

## ## DURATION ##

- 15 minutes reading, understanding and QA.

- 1 hour for implementation.

## ## LIBRARIES ##

Libraries might be used, if needed, but they may cause the candidate to lose time searching and installing. Thus, we generally advise not to use any library and employ time on that.

## ## VCS - COMMITTING & PUSHING CHANGES ##

Please, candidates must be, from time to time, pushing changing to the respective individual forks.

## ## VERY NICES TO HAVES ##

- Standard C# coding style.

- Well-structured and responsibilities of each object correctly assigned.

- Simplicity of the solution.

- Proper management of exceptions.

- Logging.

- Messages on commits.