# Assessment

General Rules and Recommendations

**Spend at most 8 hours of work on the exercise.**

o Spending less time is ok.

**Implement the exercise using C# and Visual Studio.**

o Either .NET Core or .NET Framework is ok.

o If using .NET is impossible, you can choose another mainstream OO framework/language, like

Java, or a different .NET language.

**Use relevant (free) tools and libraries (NuGet packages or equivalent).**

o Try to use the same tools that you would use in professional coding.

o Don’t reinvent the wheel.

**Write the code professionally**

o Use design patterns, best practices and coding conventions that you’re familiar with.

o Don’t implement the exercise like throw away/proof of concept code.

o Prefer clarity over cleverness.

**Deliver only project and source files.**

o No binary files.

o The project should compile.

**Don’t worry about completing the whole exercise.**

**Implemented components should be as “complete” as possible.**

Exercise: “Payments reconciliation”

Consider the following three datasets provided to you as files:

Purchases.dat

This is a list of fictitious purchase transactions. Each purchase transaction has one customer (CUST) record, one purchase date record (DATE) and one or more item records (ITEM).

A purchase begins with a CUST record and ends before the CUST record of the next block. A CUST record starts with the literal word CUST and is followed by the customer number (8 chars with leading zeroes). DATE and ITEM records are made in the same way.

**CUST**<customer number (with leading zeroes)>

**DATE**<purchase date (“day month year hour minute” in numbers without spaces)>

**ITEM**<item number (with leading zeroes)>

**ITEM**<item number (with leading zeroes)>

…

**CUST**<customer number (with leading zeroes)>

…

Prices.xml

The “Prices” dataset contains a price for each item of the store in XML format.

<ItemPricesRoot>

<ItemPricesList>

<ItemPrice>

<Item>000001</Item>

<Price>27.06</Price>

</ItemPrice>

<ItemPrice>

<Item>000002</Item>

<Price>12.7</Price>

</ItemPrice>

…

</ItemPricesList>

</ItemPricesRoot>

Payments.json

This is a list of monthly payments made by the store customers in favor of the store. Each monthly payment should cover all purchases done by the customer in that month.

[

{

"Customer": "000001",

"Year": 2018,

"Month": 1,

"Amount": 1074.97

},

{

"Customer": "000001",

"Year": 2018,

"Month": 2,

"Amount": 492.31

},

…

Reconciliation

Write a Console application that reads the three files above and creates a json file called

**PaymentsNotMatched.json** containing: customer, year, month, amount due, amount payed and the

difference between due and payed for any customers/month having non zero difference between due and payed (i.e. include only unmatched payments).

Items with the biggest difference between due and payed (in absolute value) should be reported first.

Optionally consider if you can enable saving the PaymentsNotMatched report in a different format – like CSV or narrative text.

Optionally create a web page displaying the PaymentsNotMatched report in tabular format.