**Report**

**1.Documentation.**

System has been designed based on an Object-Oriented approach which required good planning and design before implementation, therefore, a code is more robust with fewer errors and flaws.

Structure fo the project contains concrete classes like *Data, Order, Store*which implements system objects that are used in more sophisticated classes like *DataLoader*  or *MainWindow*

for data retrieving, storing, processing or displaying.

All concrete classes are private and contains interface methods that comply with Object Oriented design standards involving need of encapsulating objects.

DataLoader class has been designed to be able to read appropriate CSV files. The location of the folder including files with CSV extension can be selected from the computer storage.

There have been appropriate structures used to reassure that this application is a responsive and efficient in terms of loading and processing substantial amount of the data. In order to achieve this goal an appropriate tool of Task Parallel Library (TPL) has been used. For storing data retrieved from files TPL structures as a  *ConcurrentDictionary*and ConcurrentQueue have been used.

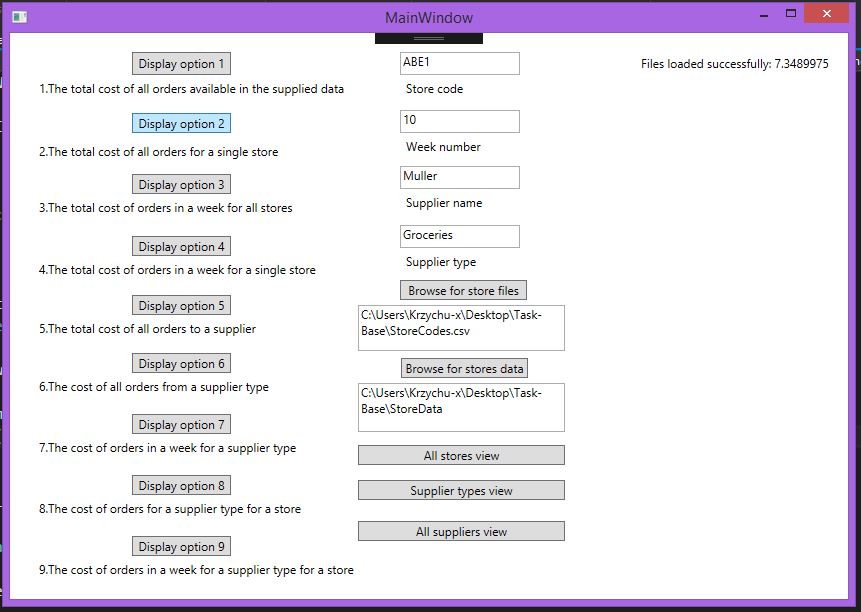
These structures have been specifically designed to simplify thread-safe access to shared data and increase the performance and scalability of multi-threading applications.

For processing data parallel *foreach* loop has been used to decrease data loading times.

*MainWindow* class is the system GUI that is making use of WPF application.

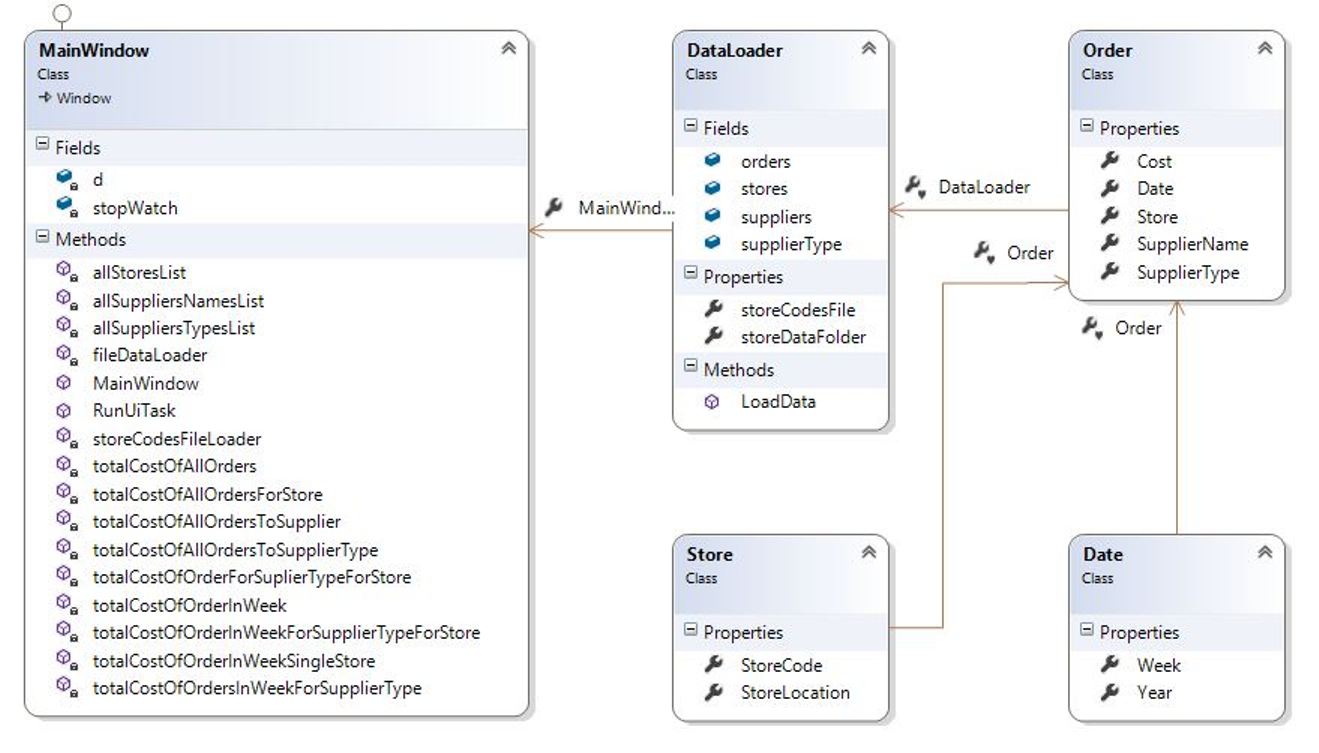
WPF implements Dispatcher class that is invoking different tasks. Each Task is designed to process one specific data request and return its value. Collection of different tasks analyzing data with different criteria and displaying them in the main window as a total value. This creates a queue of work Tasks and Dispatcher which invokes and schedule these Tasks.

**WPF form Graphical User Interface.**

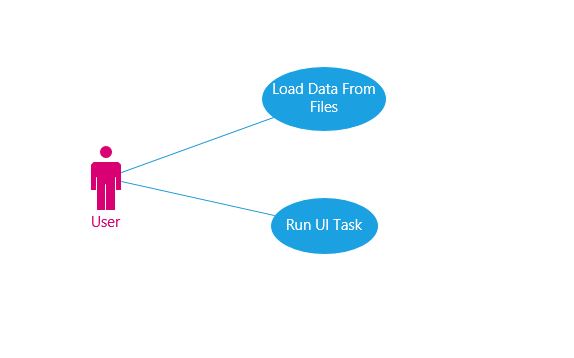


**2.Design Diagramas.**

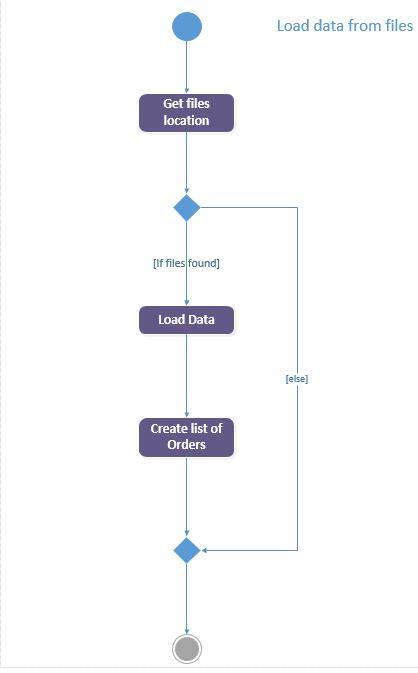
**UML Diagram.**

****

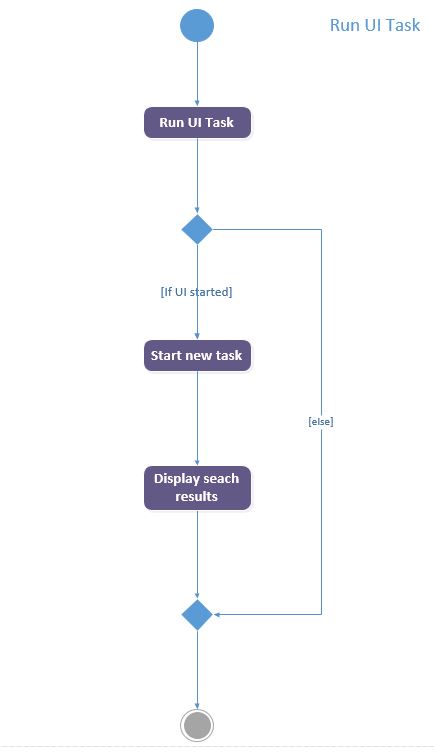
**Use Case Diagram.**

****

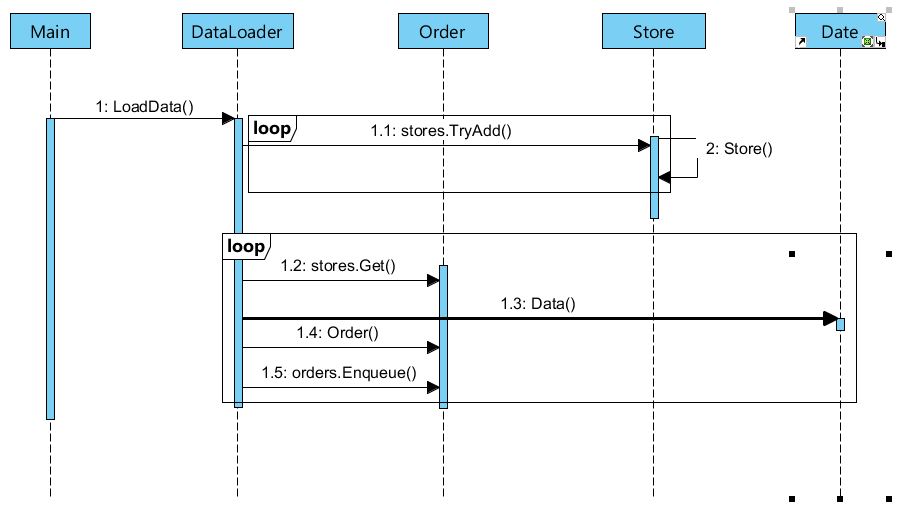
**Activity Diagram 1.**

****

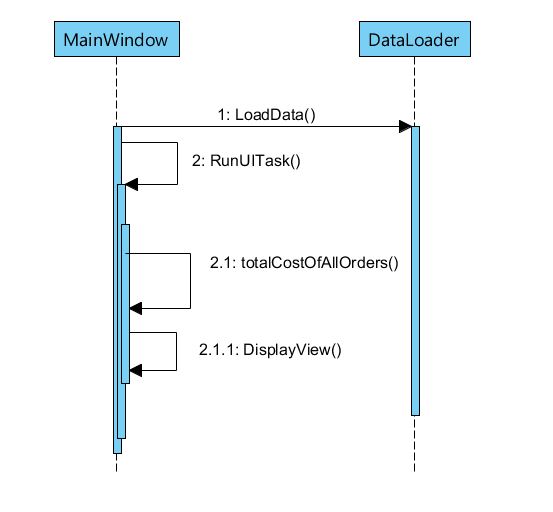
**Activity Diagram 2.**

****

**Sequence Diagram 1.**

****

**Sequence Diagram 2.**

****

**3.Performance Profiling and Evaluation.**

In order to increase readability and more clear design structure of the code the separate classes for each object has been used and use case method to comply with good Object Oriented programming practices has been created. This in turn, enabled to improve responsiveness of the entire system triggered by a robust design.

Range of improvements has been implemented to the standard example data loader by adding parallel *for each* loop to read data more efficient. The improvement achieved this way is 6 seconds while the prior result was  15 seconds to read all files.

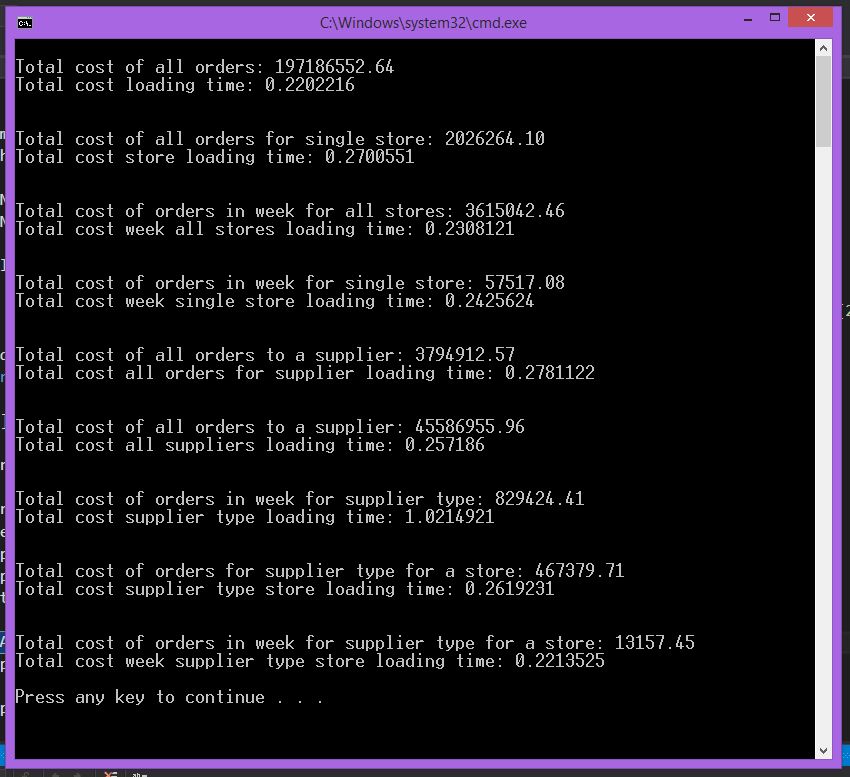
Further steps were taken to assure thread-safe access to shared data and increase the performance and scalability of multi-tasking by implementing ConcurrentBag instead of List for reading files and storing them. It is crucial for Task Parallel Library to be able to work efficiently and without generating errors. The great improvement has been noticed in terms of counting orders costs accuracy, but also slightly longer UI response times for *LoadingFile()*  from 6 sec. to 7.45 sec.  and for sequent tasks methods from 0.1 sec to approximately 0.2 sec. But considering achieved this way benefits in terms of accuracy it has been essential to implement ConcurrentBag.

WPF form has been implemented as a graphical interface for the application. The interface is using  *Task Parallel Library*and implements tasks, base on *Task.Factory*method and Dispatcher, to invoke particular *tasks.* Graphical User Interface implemented based on WPF form is twice as responsive as standard command line UI in all tasks. Times of loading files takes from 7 up to 8 seconds, similar to command line UI results. Sequent tasks times are very responsive range from 0.1 to 0.16 second.

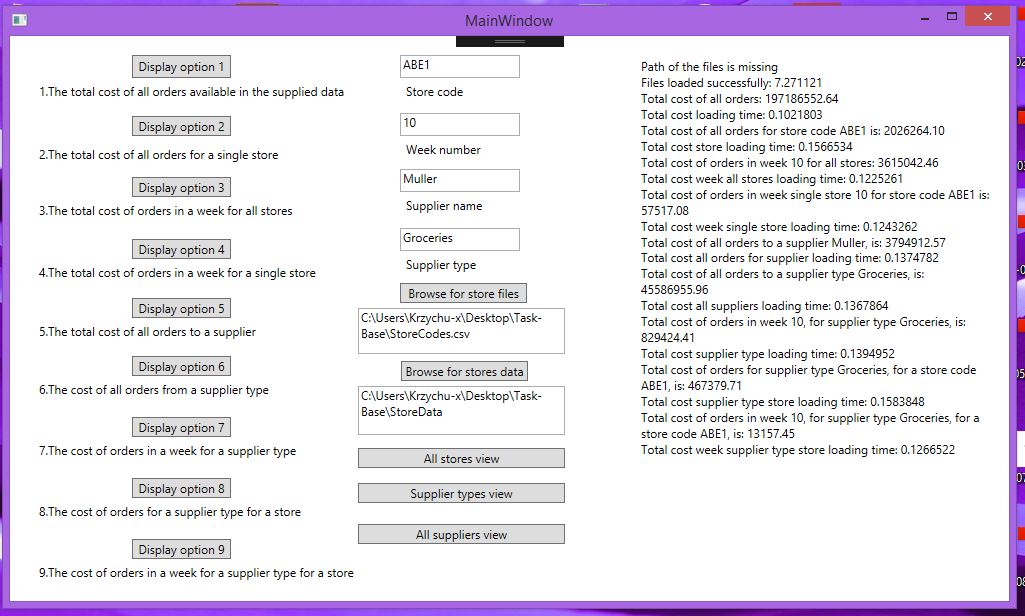
**Methods loading times table for standard command line UI versus WPF GUI application.**

**With same parameters for each data being processed.**

**Command line output**

****

**WPF GUI output**

****

When parallel *foreach* loop has been used inside sequent tasks invoked by Dispatcher in WPF form the processed results were generated not accurately. For producing more accurate results it was crucial to use regular for each loop for iteration through the data file.

Some another interesting example of decreasing operation time by implementing different solutions have been discovered during code implementation. Examples not always been used. Replacing *try-catch* block searching for null pointer exceptions by if statement decreased process time from 0.48 second to incredible 0.07 second.  

Using string variable rather then textBox.Text method for strings comparison when searching for specific data in files is more effective in terms of time. With textBox.Text being used 0.4 sec for method execution. With string variable being used  0.14 sec for method execution.