**Event Manager**

The following task tests knowledge and experience in C#, .Net, OOP, WPF, XAML, MVVM, TPL, Multithreaded programing and End-to-End Developing.

**Guidelines**

* Do your best without wasting too much time.
* We encourage using NuGet's for frameworks and shortcuts that help implement faster and have a better overall solution, but don’t get carried away.
* Comment only where necessary, you can leave your “debugging code” in a comment, we would like to see it.
* Objects from code in this document are marked with **Blue Bold Text**
* Files from the code in this document are marked with **Green Bold Text**
* Requirements in **bold red** are critical.
* Read the task carefully and fully before starting the actual task is described in the last section.

**Event Manager Application General Description**

* The event manager's application goal is to process and display sensor statuses (which are generated by a mock server), as shown in the picture below.
* A status is represented in a list to the user until the user removes the status using a remove button (trash can button).
* ****

**Event Manager Solution**

You received a solution file with two projects:

1. **SensorServerApi** (class library) : A mock server that generates **SensorStatus** objects and consists of all the model (You can view this code but cannot change it)
2. **EventManager** (wpf app) – A client/server app to process and display the information coming from the SensorServerApi, contains XAML files for views ready to be binded and manipulated.

Within the EventManager project there is main file for you to code in-

1. **MainService.cs** – creates an instance of **SensorServer** and subscribes to **SensorStatus** events.

(Note: You can code another files in **EventManager** solution if you need).

The logic implemented in **MainService.cs**:

* For each **SensorStatus**:
  + Get the **Sensor** for the received status by using the convenient method \_sensorServer.GetSensorById() – please notice this task may take some time.  
    (**Sensor** object has the Sensor name and Id)
  + Signal the sensor status to the UI using the Subject with the following logic: (at most one status per sensor)
    - If a status for the sensor **does not exist** - add It
    - If a status for the sensor **already exists** – replace it with the new one.
    - You differentiate between the sensors by their Id (**Sensor.Id**)

**The Problem**

* The MainService in **MainService.cs** has some problems:
  + Exceptions are thrown in runtime
  + The statuses are not handled in order, this is a sample console output of the program:

got 8a6eeefd-7ebc-4ea2-a42a-d12444664b3c

got 41523714-10cf-4eaa-84e2-39c673de138b

got 9a98ccaf-a725-465d-b4fb-aaa4e0b2d177

got ba42da4f-3d09-4b6e-a238-0bfb96ddc728

got 9d9e00e2-d433-4480-9230-5ea2b864083d

got 6503cf0a-a226-47db-a399-f05d038223d2

got 96e1f51a-585c-4a06-afbb-4e4b798368c3

got 115a364e-beba-451a-b7cf-df4c906ddc0e

got c2cd5eff-ecc3-4809-b0c1-8e1cd11230e3

got 17f5ba20-35f7-4fa4-9a5d-74b8765f12e2

added 115a364e-beba-451a-b7cf-df4c906ddc0e, Sensor 8 to dictionary

added 9d9e00e2-d433-4480-9230-5ea2b864083d, Sensor 5 to dictionary

added ba42da4f-3d09-4b6e-a238-0bfb96ddc728, Sensor 4 to dictionary

added c2cd5eff-ecc3-4809-b0c1-8e1cd11230e3, Sensor 9 to dictionary

added 96e1f51a-585c-4a06-afbb-4e4b798368c3, Sensor 7 to dictionary

added 17f5ba20-35f7-4fa4-9a5d-74b8765f12e2, Sensor 10 to dictionary

added 8a6eeefd-7ebc-4ea2-a42a-d12444664b3c, Sensor 1 to dictionary

added 6503cf0a-a226-47db-a399-f05d038223d2, Sensor 6 to dictionary

added 41523714-10cf-4eaa-84e2-39c673de138b, Sensor 2 to dictionary

added 9a98ccaf-a725-465d-b4fb-aaa4e0b2d177, Sensor 3 to dictionary

* + You can see the status of sensor 1 was received first but was added to the dictionary much later.
    - The expected behavior would be that if received first it will be added first.

**Your Actual Task**

* Your main task is to fix the code that will work as expected, fix all exceptions and provide an explanation of the problem.
* Additional Tasks:
  + Remove every status that is older than 15 seconds automatically, exactly 15 seconds after the received time (**SensorStatus.TimeStamp**)
  + Order the view by Sensor Name.
* You can only change code in EventManager Project, but feel free to investigate the **SensorServerApi** project.
* Some flags and modifications:
  + In MainService line 37: await \_sensorServer.StartServer(Rate.Easy, isContinuous: true);
    - You can lower the rate of the incoming statuses by changing the Rate parameter.
      * **Easy, Medium, Challenging, Hardcore**
    - You can set a continuous mode to the **SensorServerApi**, that will send statuses indefinitely, or send 10 statuses and stop, by changing true and false.
* Good luck.