Project Part A

Objective

Over two days you will be working on **Project Part A** individually or together in group. Assignment A1 has 3 parts to be completed and the Assignment A2 has 4 parts. You may decide to do part of the work in the group and complete it individually. In any case the project part is individual and should be send in individually. Pls send in your solutions, compressed into zip format, so I can easily un-compress them and open in Visual Studio 2019 and run the code. **Pls see deadline in "Kursplaneringen".**

The objective is by using the Model (and Service) separation of MVVM create a service that communicates with a web api service using Json. You should use as much as possible of what you learned during the C# AOOP theory, in particular Tasks, async/await, events, Lambda Expression, Linq, Collections, Serialization. But also, of course basics like classes, tuples, enumerations.

For **Assignment A1**, I have provided hints and a simple template in code for you to not get stuck on trivial matters. For **Assignment A2** is of somewhat similar nature but, only some basic data types are provided, and only one hint.

Grade criterias for this part of the project:
Godkänt (G) : Completion of all assignments part of "Project Part A: Assignment A1", using the C# methodology you learned in C# AOOP theory part of the course.
Väl godkänt (VG): Completion of all assignments as part of "Project Part A: Assignment A2", using the C# methodology you learned in C# AOOP theory part of the course.

The services are Weather webapi service (https://openweathermap.org/) and News webapi service (https://newsapi.org/). You should have already registered yourself to the two web-api services following my document "Setting up a .NET cross-platform development environment in Windows 10".

Note: For the weather web-api on the free plan you have 60 free web-api calls per minute, and for the news web-api you have 100 calls per day. For the news web-api, I provide in the template directory sample data and code that you can use to test and debug before calling for live data.

Project Part A: Assignment A1

Use the Console Solution Project Part A template.

Assignment A1.1

In class *OpenWeatherService* you should create an async method, *GetForecastAsync(..)* that implements following functionality:

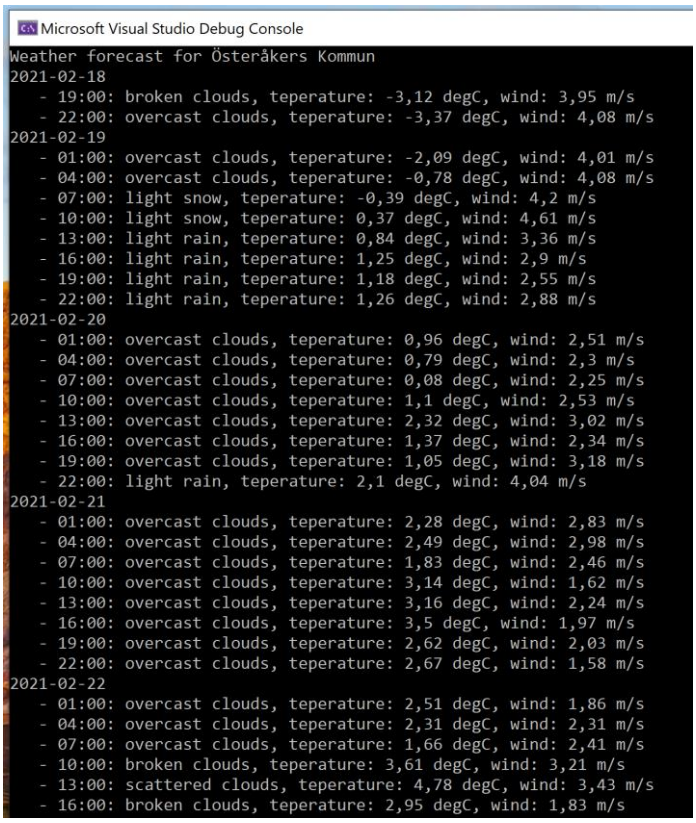
- Requests weather data from https://api.openweathermap.org using your apiKey.
- Read the Json response string async.
- Convert the Json string into the class *WeatherApiData.*
- Create a response object of type *Forecast* from *WeatherApiData* using *Linq*.
- return the *Forecast* object to the caller.

The caller, *Program.Main* should implement following functionality:

- Call the *GetForecastAsync(..)* using the asynchronous design methodology.
- Wait for the task to be completed.
- Group the Forecast according to Dates using *Linq*.
- Write the weather data to the console.

Hint:
The timestamp from the weather service is in Unit time stamp format. In one of my examples you see how to convert. *Linq* operator *Select* is a perfect tool together with Lambda Expression to convert one collection to another. *Linq* operator *GroupBy* is perfect for grouping. Remember how to Wait for all tasks to complete. *ReadFromJsonAsync<>()* extension is an excellent way to get the Json response string.

An output from *Main* on the Console could be something like:



Output from Assignment A1.1

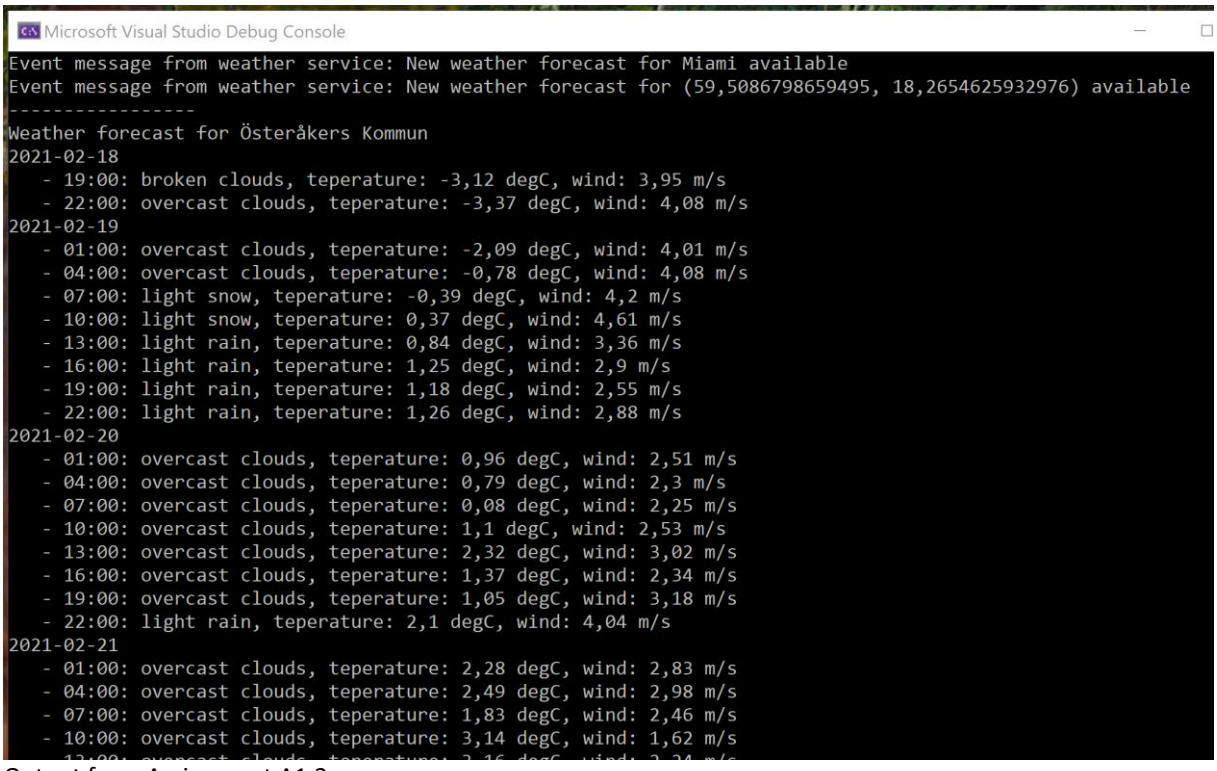
Assignment A1.2

Modify *OpenWeatherService* so that you can also get a *Forecast* for a city. Once a Forecast is received, either by GeoLocation or by City, the service should fire an *event* with a message.

Modify the caller, *Program.Main,* so that two weather forecasts are requested; one based on GeoLocations, one based on a City name. Once both forecasts are received, asynchronously, the forecasts should be printed. Exception handling should be implemented so incase a forecast is not available for a city an appropriate message should be printed. *Program.Main* should be a subscriber of the event from *OpenWeatherService* and print out the message.

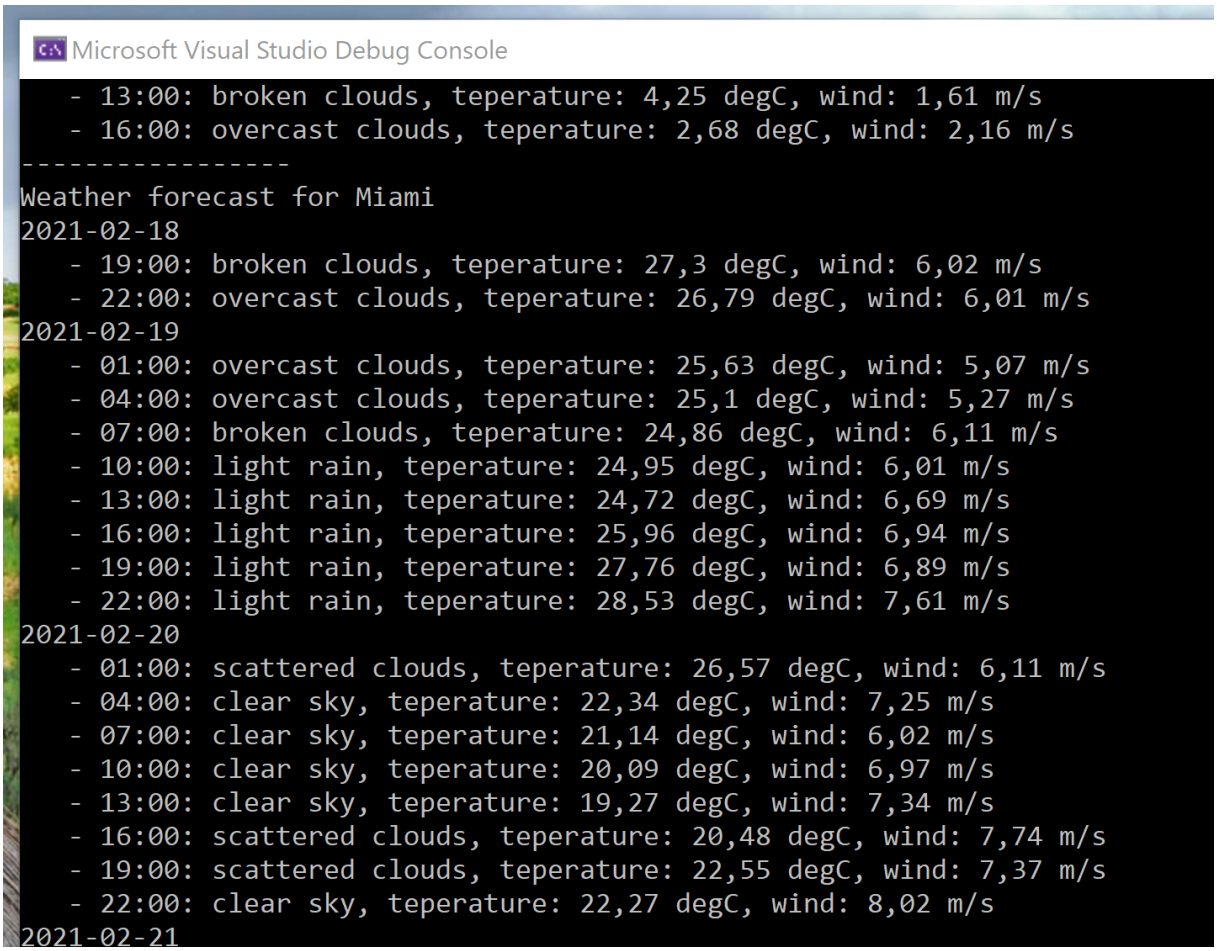
Hint:
Look in the theory paper how to create an event based on .NET *EventHandler<>* methodology and exception handling for tasks.

An output from *Main* on the Console could be something like:



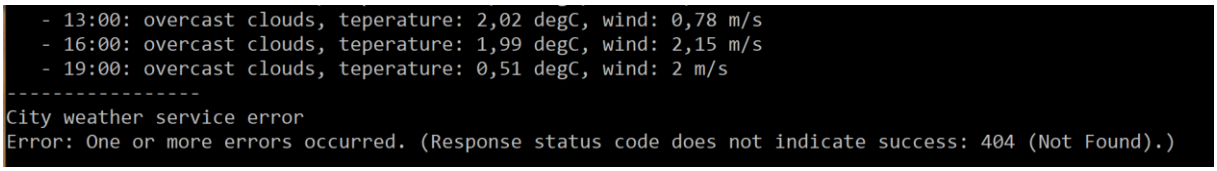
Output from Assignment A1.2

and further down on same printout for the city:



Output from Assignment A1.2

Finally, in case of a non-existing city and example of error message:



Output from Assignment A1.2

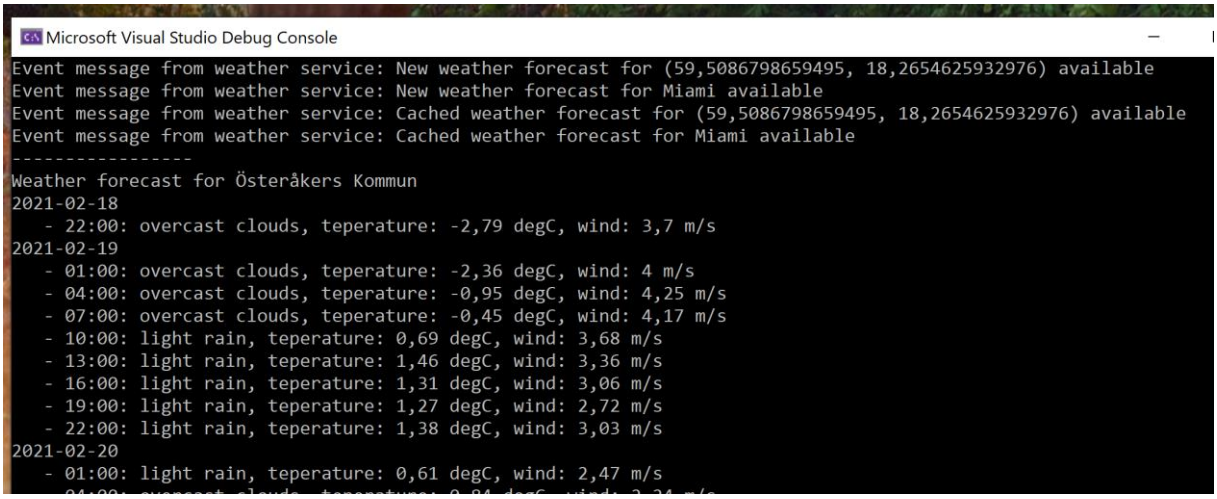
Assignment A1.3

Modify *OpenWeatherService* so a *Forecast* downloaded from the web-api is cached and returned for an identical request made within 1 minute. The message from the fired event should differ in case of cached or downloaded forecast.

Hint:
This can be done elegantly using the thread safe version of Dictionary<>. Why thread safe? Tuples are excellent for Keys and to get a string of current date/time without seconds can be done by *DateTime.Now.ToString("yyyy-MM-dd HH:mm")* .

Modify the caller to make multiple requests to show that cached data is received.

An output from *Main* on the Console could be something like:



Output from Assignment A1.3

Project part A: Assignment A2

Use the Console Solution Project Part A template.

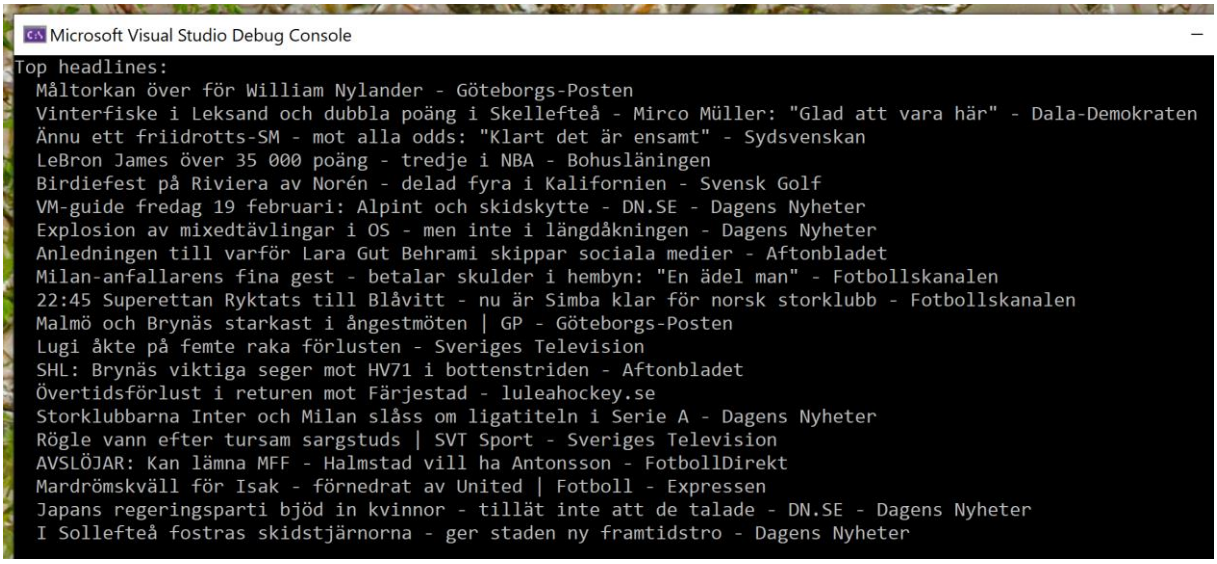
Note: For the News api you have “only” 100 calls per day. Therefore, I provide in the template directory *ModelsSampleData* sample data and a Method to read the data asynchronously. With this I am showing a test design pattern that you can use.

In the Service itself I use the #define structure so you can easily switch to compile for test (one line of code) or live data (your code). You could complete all 4 exercises without reading live data, so pls choose when to go live.

Assignment A2.1

Make a Service, *NewsService*, with a method *GetNewsAsync*() that returns a *NewsApiData object* to the caller. Implement a caller that prints out the top headlines.

Output should be similar to:



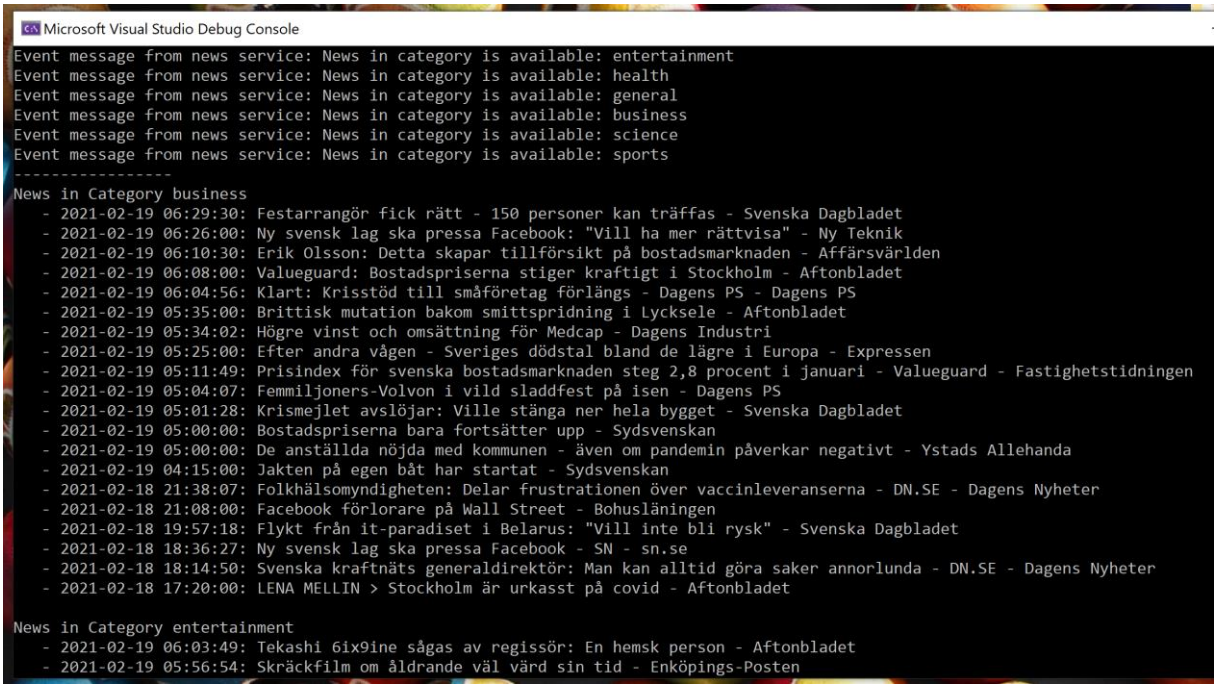
Output from Assignment A2.1

Assignment A2.2

Modify *GetNewsAsync(…)* to take a news category as an input parameter and return an object of type *News.* Implement an event that fires when News data for a category is loaded. The event should take a message as a parameter.

Modify caller to iterate over all the news categories and printout result. Implement exception handling in the caller. The caller should subscribe on the event and printout the event message.

Output should be similar to:

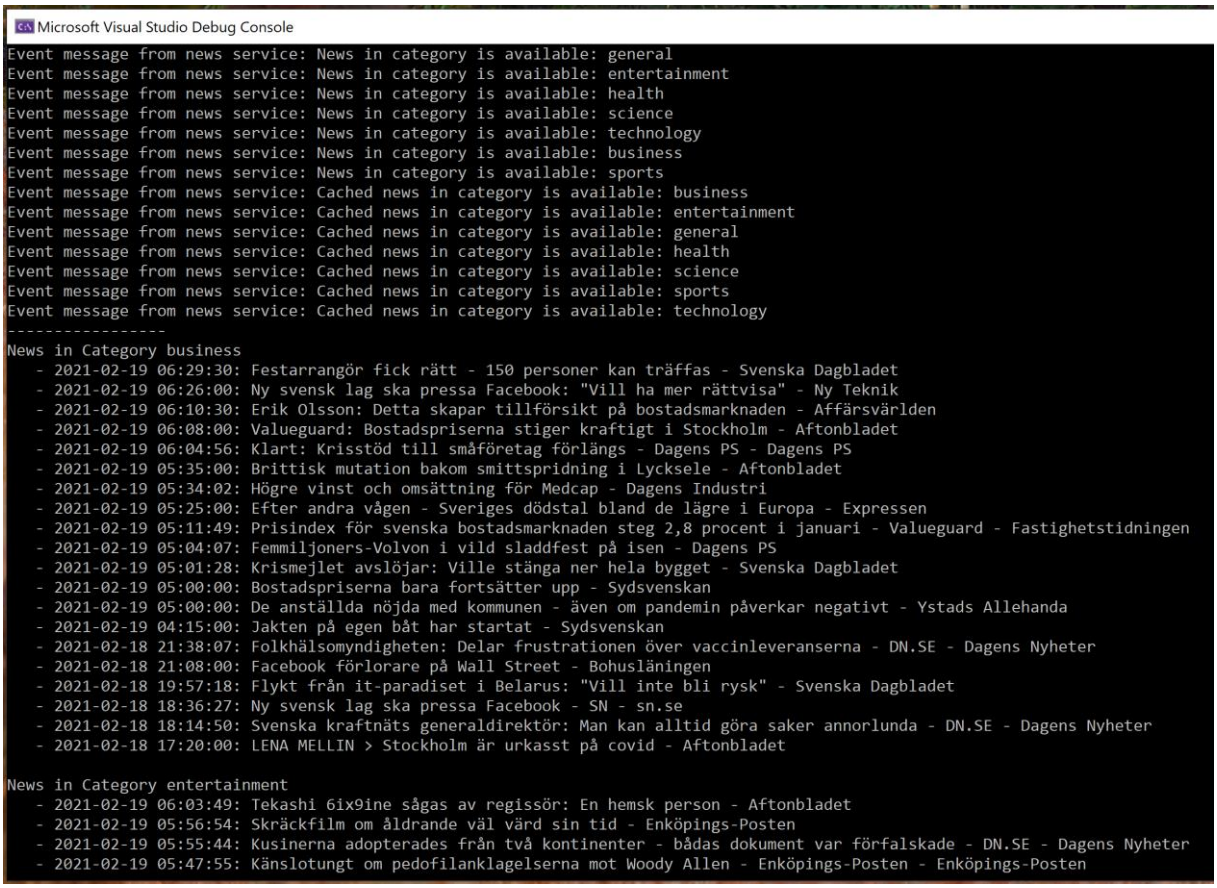


Output from Assignment A2.2

Assignment A2.3

Cache the news data. When a request comes in return the cached data if it is an identical request within 1 minute. Fire the event with a different message when cached data is returned.

Output should be similar to:



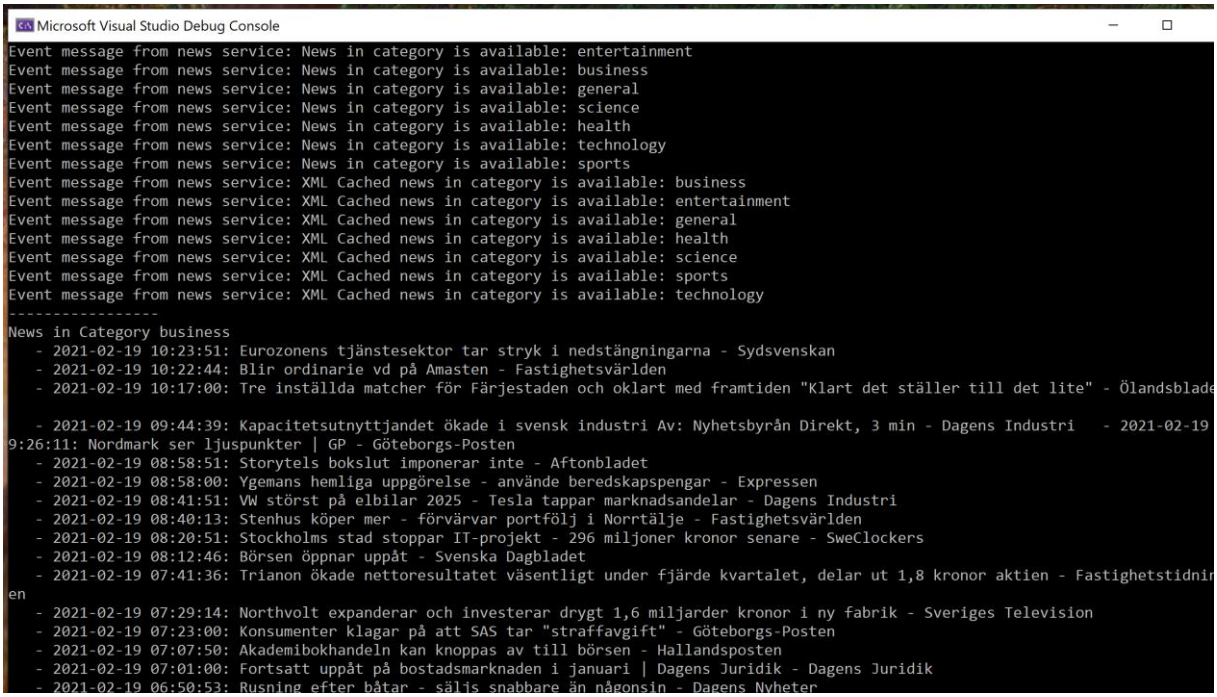
Output from Assignment A2.3

Assignment A2.4

Move the cache from an instantiated object into an XML file stream using Serialization. When a request comes in return the cached data from the XML stream if it is an identical request within 1 minute. Fire the event with a different message when cached data is returned.

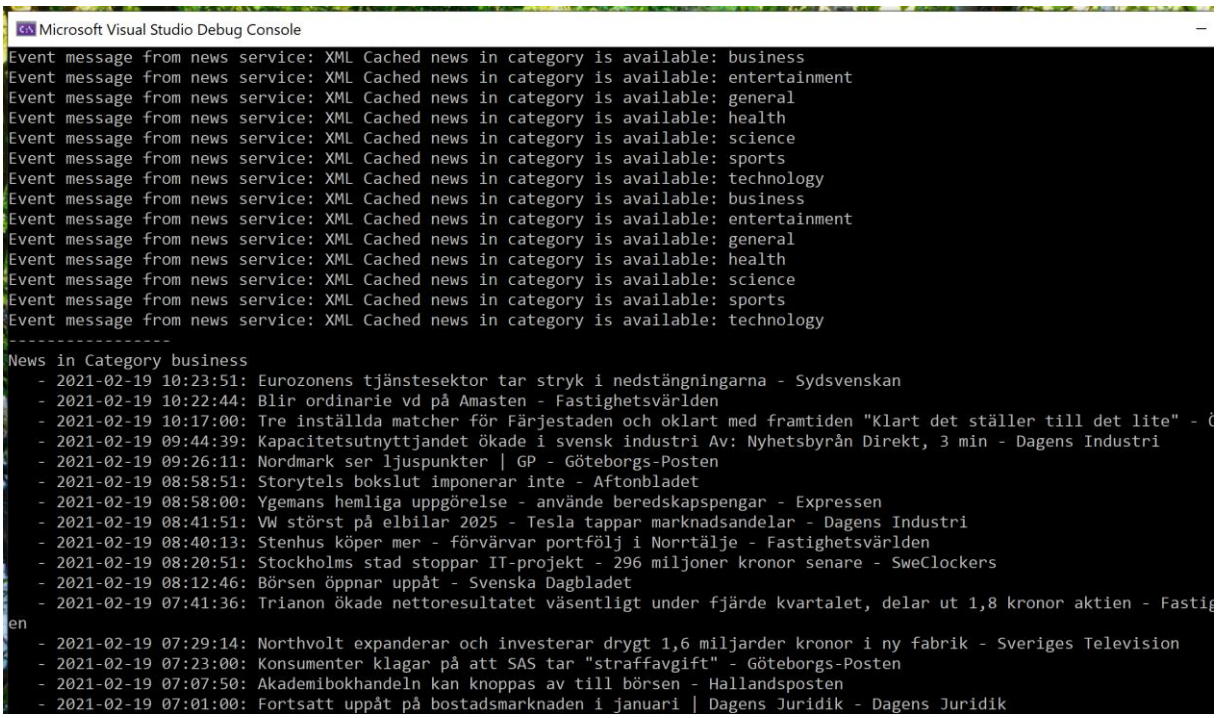
Hint:
Break out the cash key into a separate class which can give you a filename based on the key you used to index the Dictionary. Remember to use *lock* when you Serialize and Deserialize to make it thread safe.

Output should be similar to:



Output from Assignment A2.4 from first run

Finally, as the cache is now on disk, the second run within the minute should produce output similar to (all news collected from the chache):



Output from Assignment A2.4 the second run for as long as the XML cache is valid