```csharp
using Corp.Services.DataContracts;
using Grpc.Net.Client;
using Newtonsoft.Json;
using ProtoBuf.Grpc.Client;
using System;
using System.Linq;
using System.Net;
using System.Text;
using System.Threading.Tasks;
using static Corp.Resources.Infrastructure.Endpoints.Services;

namespace Corp.Services.Contracts
{

    public class FloodingAlertWorkflowService: IFloodingAlertWorkflowService
    {
        public async Task<FloodingAlertWorkflowResponse> StartWorkflow()
        {
            FloodingAlertWorkflowResponse response = new();
            try
            {
                DownloadDataResponse downloadDataResponse = await GetCurrentWaterLevelData();
                string filterredCsv = await FilterWaterLevelData(downloadDataResponse);
                string windSpeedUrl = GenereateDmiUrl(DmiParameter.WindSpeed);
                string windSpeed = await GetWindSpeed(windSpeedUrl);

                response.WindSpeed = windSpeed;
                response.WaterLevel = Int32.Parse(filterredCsv.Split('\n').Last());
                response.MessageInfo = "Request succeeded.";
            }
            catch(Exception e)
            {
                response.MessageInfo = "Request failed.";
            }
            return response;
        }

        private async Task<DownloadDataResponse> GetCurrentWaterLevelData()
        {
            string url = GenerateCoastDirectorateUrl();
            Uri uri = new Uri(url);
            DownloadDataRequest request = new DownloadDataRequest() { Uri = uri };
            string localHostAddress = $"http://localhost:{DataAccessServicePort}";
            GrpcChannel channel = GrpcChannel.ForAddress(localHostAddress);
            GrpcClientFactory.AllowUnencryptedHttp2 = true;
            DownloadDataResponse response;
            using(channel)
            {
                IDownloadDataService downloadDataService = 
                  channel.CreateGrpcService<IDownloadDataService>();
                response = await downloadDataService.DownloadWith(request);
            }
            return response;
        }

        private async Task<string> FilterWaterLevelData(DownloadDataResponse dataResponse)
        {
            string localHostAddress = $"http://localhost:{FilterServicePort}";
            GrpcChannel channel = GrpcChannel.ForAddress(localHostAddress);
            GrpcClientFactory.AllowUnencryptedHttp2 = true;
            string csv = Encoding.Default.GetString(dataResponse.Data);
            int[] keepColumns = new int[] { 1 };
            CsvFilterRequest filterRequest = new() { Csv = csv, KeepColumns = keepColumns, 
              RemoveHeader = true };
            string response;
            using(channel)
            {
                ITextFilterService textFilter = channel.CreateGrpcService<ITextFilterService>();
                response = await textFilter.FilterCsvColumns(filterRequest);
            }
            return response;
        }

        private async Task<string> GetWindSpeed(string url)
        {
            string windSpeed, response;
            using(WebClient client = new())
            {
                response = await client.DownloadStringTaskAsync(url);
            }
            response = response.Substring(1, response.Length - 2);
            WindSpeedResponse deserializedJson = JsonConvert.DeserializeObject<WindSpeedResponse>
              (response);
            windSpeed = deserializedJson.Value.ToString();
            return windSpeed;
        }

        private string GenerateCoastDirectorateUrl()
        {
            string baseUrl = "https://kystatlas.kyst.dk/public2/data/vandstand/response.aspx?";
            string station = "6701"; // Vester Vedsted
            string startDate = DateTime.Today.ToString("yyyyMMdd");
            string endDate = DateTime.Today.AddDays(1).ToString("yyyyMMdd");
            string format = "csv";
            string stationAndDates = $"ident={station}&startdate={startDate}&enddate={endDate}
              &format={format}";
            return $"{baseUrl}{stationAndDates}";
        }

        private string GenereateDmiUrl(DmiParameter parameter)
        {
            string url = "https://dmigw.govcloud.dk/metObs/v1/observation?latest=&parameterId=";
            switch(parameter)
            {
                case DmiParameter.WindSpeed:
                    url += "wind_speed";
                    break;
                case DmiParameter.WindDirection:
                    url += "wind_dir";
                    break;
                default:
                    break;
            }
            url += "&stationId=06093&api-key=5910e131-7fe5-43eb-9a29-bfe480b5f7b8";
            return url;
        }

        private enum DmiParameter
        {
            WindSpeed,
            WindDirection
        }

        public class WindSpeedResponse
        {
            [JsonProperty("_id")]
            public string Id { get; set; }

            [JsonProperty("parameterId")]
            public string ParameterId { get; set; }

            [JsonProperty("stationId")]
            public string StationId { get; set; }

            [JsonProperty("timeCreated")]
            public double TimeCreated { get; set; }

            [JsonProperty("timeObserved")]
            public double TimeObserved { get; set; }

            [JsonProperty("value")]
            public double Value { get; set; }
        }
    }
}
```