DNP1

You have 3 hours to implement the following - PLEASE read everything before you start.

Question 1 (10%)

Given a Gas Pump that holds 1100 Liters of Gas, at 9.82,- DKK pr. Liter.

GAS PUMP **GAS TANK**

Create a new folder for your entire exam project named DNP1exam. In this folder, create a class library project named GasPump.

In the project, implement a class Pump with the Three methods:

- CostOfFullTank
  - Takes the size of a given gas tank and returns the price to fill it.
- FillTank
  - Takes how many liters to tank and subtracts the tanked liters from the Pumps capacity, then returns the price.
- FillPump
  - Fills the Pump back up and returns a DateTime.Now of when the pump was filled.

Note: Gas Tank is part of a car, you do not need to implement this car, just pass values corresponding to some gas tank size (fx a tank could be 42 liters big) and think about what happens if someone tries to fill 30 liters into their car, if there is only 20 liters left in the pump.

Question 2 (15%)

A. Within the DNP1exam folder you must create a new console application named PumpTest.

B. Add Reference to the GasPump class library in your PumpTest console applications Dependencies.

C. Implement a method to test the prices returned by CostOfFullTank and FillTank using the given tank sizes:

```
        double[] tankSize = { 42, 23, 112, 12, 4 };
```

D. Implement a method to test the DateTime returned by FillPump.

Note: DateTime might be unique (because it is called, then returned, which takes time), so can you do something to see if the DateTime was a time that is later than when you called the method, but earlier than once the method finished?

Question 3 (15%)

Within the DNP1exam folder you must create a Web API project named GasPumpWebService, that uses the GasPump to implement the three HTTP endpoints

- `/gaspump/CostOfFullTank?size=x (GET)`
- `/gaspump/FillTank?amount=x (GET)`
- `/gaspump/FillPump (POST)`

… Where x is a decimal. Each endpoint should return a double (the result of the calculation).

Question 4 (30%)

Use Entity Framework Core to create an SQLite database with a table that can contain a history of how often the Gas Pump has been filled (FillPump), and another table that tracks how much Gas people have taken out of the pump (FillTank).

Hint: You will need these two packages:

- Microsoft.EntityFrameworkCore.Design
- Microsoft.EntityFrameworkCore.Sqlite

Inject your database context into your Web API Controller, and whenever a request is made (using the same URLs as above), create and store the tanked amount in the database, and store the time, when the Pump is filled.

Question 5 (30%)

For this part, keep the GasPumpWebService running in the background.

Within the DNP1Exam folder create a new Web App project, call it GasPumpWebApp. It's up to you, whether you use MVC, Razor pages, or Blazor.

The Web App must have two pages:

1. A page/form to call your Web API.
    a. There must be a way to input a liter value for the gas to do the calculations
    b. There must be a button to fill the Gas Pump back up.
2. A page to view all Gas Pump refills. Retrieve these from your Web API, and put them in a list/table with the headline "Pump Maintenance"

The pages must use the endpoints of GasPumpWebService (i.e. consume the web service).

NB! To avoid binding the two programs to the same address, you can change the port of the "applicationUrl" value in launchSettings.json (located in the Properties folder). Remember to change the port both for HTTP and HTTPS.