Please answer the questions to the best of your ability. Feel free to google, but please let us know on each question. Also please state how long time you spent on each question. Submit the answers and solutions as a downloadable link (not zip in email).

## Task 1

Two separate .NET systems are set up to validate MD5 keys (or SHA1 etc) from a database. The MD5 keys are generated from a list of employee names. The validation is matching 100% on one system, but only 95% on the other system. What is the most probable reason for this?

**Answer:** **It can also happen that the target system is 64-bit and the code has 64-bit portability issues.**

**The only way to solve the problem is to debug where exactly the two versions of your code behave differently.**

**Spend Time: 30 minutes**

## Task 2

The code below is used in an ASP.NET application where ”GreetingService.CurrentGreeting” is fetched at the following times: 08, 12 and 16. Which message is returned on each run?



**Answer: Good morning**

**Good evening**

**Good evening**

**Spend Time: 5 minutes**

## Task 3

Entity Framework is used to fetch data from a table in a database and display the result with a text header that shows the total number of rows returned in the result. Which of the following versions would you suggest, and why?

**Answer: Version 2, because using .ToList() will cause the query to pull back the entire list, and then count it. However, the .Count() extension method will translate it into a SQL COUNT statement on the database side.**

**Spend Time: 10 minutes**

## Task 4 (optional for backend position)

Take a look at the following badly written page. Please rewrite this with similar functionality but in a modern way. Focus is on a good and modern structure rather then replicating the functionality exactly.



**Answer:** **See another attached file (Task4.html, Task4.js, Task4.css)**

**Spend Time: 45-60 Minutes (**because some of functionality not worked, I resolved it**)**

## Task 5

Create a web page with a button. When you click on the button the application should retrieve the currency rates for USD and EUR and display them on the page. Please use *dotnet core* for backend and a modern JS framework for frontend. Please think of how you structure your code on both backend and frontend.

Currency rates feeds can be found on:

<http://www.floatrates.com/daily/usd.xml>

<https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml>

<https://www.tcmb.gov.tr/kurlar/today.xml>

Or other feed you can find would work.

Use: good practices and architecture, clean solution, dotnet backend, async calls, VueJS (or Angular/React to bind data)

**Answer: I have created CleanArchitecture in Dot Net Core 5.0 and Angular 12 using below command. Some of extra functionality are there just because of deafult architecture.**

1. Run dotnet new --install Clean.Architecture.Solution.Template to install the project template
2. Create a folder for your solution and cd into it (the template will use it as project name)
3. Run dotnet new ca-sln to create a new project
4. Navigate to src/WebUI/ClientApp and run npm install
5. Navigate to src/WebUI/ClientApp and run npm start to launch the front end (Angular)
6. Navigate to src/WebUI and run dotnet run to launch the back end (ASP.NET Core Web API)

**Task Requirement:**

1. **Run Application using below command**
   1. *Navigate to src/WebUI/ClientApp and run npm start to launch the front end*
   2. *Navigate to src/WebUI and run dotnet run to launch the back end*
2. **Click on ”Currency Data” nav list item.**
3. **Data populated in form of list (I used** <http://www.floatrates.com/daily/usd.xml>**).**

**Spend Time: Approximate 6-7 Hour**

* Analysis : 1 Hour
* Create Architecture : 30 minute
* npm install : 10 minute
* Restore dependencies: 10
* Remove Docker : 10 minute
* Add API Controller, Currency Model, Controller Action functionality : 1 Hour
* Add Query Model along with functionality : 10 Minute
* Analysis “floateates.com” xml result. : 30 Minute
* Implement functionality to get result and SerializeXmlNode then DeserializeObject into Json/model : 1 Hour
* Parse XML and Convert Json result into model using VS “Paste Special”. : 20 Minute
* Create new “Currency-Data” Component along with View to show the list : 30 minute
* Add component in app.module, Create routing in app-routing.module: 2 Minute
* Create interface model, Add CurrencyClient in web-api-client and implement its functionality.: 30 minute
* Fetch the result in component using CurrencyClient (httpClient) and Display list.: 10 minute