Technical Test – Route to Market Team

The purpose of this test is to show us how you solve a given task and what thoughts you have along the way. We take this test very seriously and it plays a large part in our evaluation of you as a developer.

It is not a requirement that the solution is polished to the point where it can be put into production. It is however important that you have considered, and through examples in the code, shows how the solution would look if it was made production ready. This includes but is not limited to:

- Automated tests (including different test scopes)
- Deployment strategies
- Error handling strategies
- Patterns
- Logging

You may pick whatever technologies and frameworks you want, but whenever relevant you should develop the solution according to the guidelines outlined in the "Architectural Guidelines" (See page 2).

TASK 1

You have some salespersons who sell their products to several stores.
The stores are located in a district. Each district has a name e.g. "North Denmark", "Southern Denmark" etc.
Each salesperson is associated with one or more districts. Each district ALWAYS has a SINGLE primary salesperson.
Sometimes a district also has one or more secondary salespersons.
During transition periods, if there is a shortage of salespersons, the same salesperson can be the primary salesperson for multiple districts.
Create tables that fit the scenario described above. As much as possible the business rules should be enforced by constraints in the database.

TASK 2

Add some test data to the database (districts and salespersons)
Create a WPF desktop or Angular application that shows all the districts in a list. When you click a district an overview of the salespersons associated with the district should be shown along with the stores belonging to the district.

TASK 3

Add the possibility of adding and removing salespersons from a district. It must be possible to specify whether the salesperson should be primary or secondary.
Some of the subtasks are:
Create a server-side data layer which accesses the database. This must be created in native SQL. You are not allowed to use an object-relational-mapper like Entity Framework or LINQ to SQL.
Create a RESTful ASP.NET Core service that uses the data layer to expose and update the database.
Update the WPF or Angular application, so that it accesses the web service and shows the data to the user and allows him / her to update the data.

Route to Market – Architectural Guidelines

Design

- **Avoid monoliths**
  Small services / apps / systems
- **Single system data ownership**
  Only one system owns data
- **Don't user other systems databases**
  User their web services

Technologies

Web:	Typescript, Angular
Desktop:	C#, WPF
Web Services:	Asp.Net Core
Logging:	Serilog + Structured logging
Deploy:	Argo CI/CD
Database:	MSSQL + Postgress + MongoDB
IOC Container:	.Net Core standard
Data Access:	Entity Framework Core, Dapper

- **All code and associated artifacts must be committed to a GIT repository**

Logging

Fatal:	Unrecoverable crash
Error:	Recoverable + Affects users
Warning:	Recoverable
Info:	Significant event
Debug:	Developer information
Information:	Detailed debugging information

Communications

- **Systems expose / share data via web services**
  Data is available via HTTP GET
- **Systems receive data via web services**
  Data is sent via HTTP POST / PUT
- **Systems send event notification via**
  - RabbitMQ (guaranteed delivery)
  - SignalR

Web Services

- **Protocol:** REST via HTTP (GET, POST, PUT, DELETE)
- **Data format:** JSON
  Other formats are allowed, JSON is required
  Use Protobuf for high performance needs
- **Must have Swagger page**
- **Must be backward compatible**
- **Use UTC for data / time values**
- **Authorization and authentication are required**
  User JWT for access outside company network
  User Kerberos for access inside company network

2024-01-17