TECHNICAL TEST

The purpose of this test is to show us how you solve a given task and what thoughts you make along the way. We take this test very seriously and it plays a big 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** you showing how the solution is auto-tested.

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

TASK 1

You have some salespersons who sell their products to several stores.
The stores are located in 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 on 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 desktop application in WPF or a web application in AngularJs/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.

Create a WPF or AngularJs/Angular client that accesses the web service and shows the data to the user and allows him/her to update the data.

NEAS ARCHITECTURE COMMANDMENTS

Design

- **Avoid monoliths**
  Small services / apps / systems
- **Single system data ownership**
  Only one system owns wind prognosis
- **DON'T use other systems databases**
  Use their WebServices

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 PUT
- **Systems send event notifications via**
  - **RabbitMQ** (*guaranteed delivery*)
  - **SignalR**

Technologies

- **Web:** Angular
- **Desktop:** C#, WPF
- **Web Services:** ASP.NET WebAPI + ASP.NET Core
- **Logging:** Serilog + ElasticSearch + Kibana
- **Deploy:** Jenkins + Octopus Deploy
- **Alerting:** ElastAlert, site24x7.com + PRTG
- **Database:** Microsoft SQL Server
- **IOC container:** Autofac, SimpleInjector
- **DB Access:** Dapper, Entity Framework
- **Shared DLLs:** Available from NuGet
- **All code (incl. stored procedures and views from the database) MUST be committed to a GIT repository**

Web Services

- **Protocol: REST via HTTP** (GET, POST and PUT)
- **Data format: JSON**
  Other formats are allowed, JSON is **required**
  Use Protobuf for high performance needs
- **URL must be** api.neasenergy.com/...
- **Must be available via the Data Portal**
- **Must be backwards compatible**
- **Automated End2End tests are required**
- **Use UTC for date/time values**
- **Use NEAS Market Model for 'key' values**
- **Windows authentication is required**
  Use the access management system for access outside Neas
  Use AD credentials for authentication inside Neas

Logging

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

Critical Systems

- **Must have a failover strategy**
- **Must have continuous integration**
- **Must have a test system (w/o dependencies)**
- **Must have an external integration test system**
- **Must have a release procedure**