WEB EXERCISE

FOR BACKEND DEVELOPERS
V2.0

CONSID

INTRODUCTION

This exercise is both a challenge and a way to display knowledge of modern web technologies. **Consid** is using this to test and review candidates. The scenarios can be solved in several different ways, and there is no single correct solution. The goal is to evaluate how the candidates work to solve problems, with the help of existing frameworks.

There are **no restrictions,** but we expect you to write this as a .NET or java web application. You do not need to write vanilla JavaScript, or your own CSS, unless you're really **awesome** at that. It's 100% ok to use frameworks like jQuery and Bootstrap. We hope that we have included all information necessary, but if you have any questions, please contact us.

CONTENTS

1. PREREQUISITES

1.1 TOOLS

For this exercise, you need Visual Studio or similar IDE for .NET, and a database.
We can recommend the below.

Visual Studio Community Edition and Visual Studio Code is free, and can be downloaded here:
https://visualstudio.microsoft.com/

Microsoft SQL Server Express is free, and can be downloaded here:
https://www.microsoft.com/en-us/download/details.aspx?id=55994

For Java developers you can use:

https://www.eclipse.org/downloads/packages/release/kepler/sr1/eclipse-ide-java-developers

https://www.jetbrains.com/idea/

If the links above for some reason are out of date a quick search on the web will help you find these tools.
Also if you want to use another database go ahead, just make sure it's something we easily can access when reviewing your project. For example, if you use SQLite, include the db-file in the zip.

1.2 LANGUAGE

Unless anything else has been communicated, we want you to use C# or java. Don't hold back on the new features.

1.3 FRONTEND

The purpose of this assignment is not to test your frontend design skills. The meat and bone of the exercise lies in the backend.
With that in mind, tackle the frontend part of the exercise in a way that makes you happy and productive. It is possible to do the assignment without the use of any frontend frameworks.

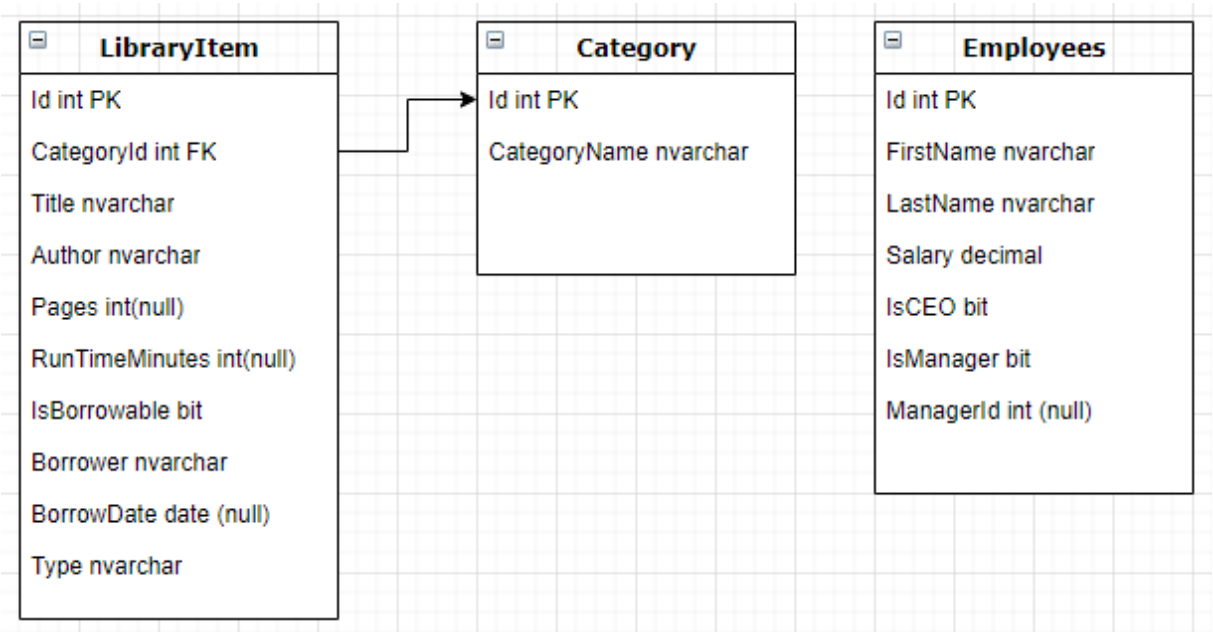
2. Scenario 1

Working as a developer at a company, you are tasked with creating a web application to manage library items and employees.
The application will store data in a database, and you are responsible for designing the applications infrastructure, models, business login and presentation from a set of criteria given to you by the customer.

For the purpose of this exercise, the application has no requirements on security and the presentation layout is of low priority. Follow the steps below to progress in the task at hand.

2.1 DATABASE

Create your database with three tables (se diagram below, Employees will be used in scenario 2). You can decide for yourself the length of the fields, pick something you feel appropriate or leave unset. A **LibraryItem** should have a required category tied with a foreign key of **CategoryId** mapped to **Id** in the table **Category** as shown.



Note:
Not all types of library items have the same behavior in the application but is stored in a common table in the database. The **LibraryItem** column **Type** is just a string describing the type of library item just to keep things simple, so we don't have to waste a lot of time creating tables for each type of library item, more information about that later. This goes for **Employees** as well but here we use the columns **IsManager** and **IsCEO** to determine if an **Employee** has a higher rank.

2.2 THE APPLICATION

2.2.1 CATEGORIES

As the application user I want to be able to create categories with unique names, no duplicates allowed. A user should also be able to edit and delete categories. **But** if a category is referenced in any library item it cannot be deleted until the reference is removed first.

2.2.2 LIBRARY ITEMS

As the library application user I want to be able to create update and delete library items. Library items must be assigned to a category.
When listing the library items in the view they must be sorted by **Category name** by default, but as a user I can change this to be sorted by **Type**.
Listing should also show if the item is borrowed by someone showing name of who borrowed it and the date when it was borrowed.
Listing should also show each items title acronym in parentheses after the title, for example if the title of the book is "The book title" it will be displayed in the listings view as "The book title (TBT)". An acronym as you can see is the first letter of each word in the title. "lol" is an acronym of "laugh out loud" for example.

When creating and editing a library item there are some different behavior to have in mind. A library item can be the following types: **book**, **reference book**, **dvd** and **audio book**. Below follows the general behavior and the behavior of each library item type and what fields/columns they require:

Book:

- Required Fields: Id, Title, Author, Pages, Type, IsBorrowable, CategoryId
- Fields to set when borrowed (Checked out): Borrower, BorrowDate
- A book can be borrowed e.g. IsBorrowable = true

DVD:

- Required Fields: Id, Title, RunTimeMinutes, Type, IsBorrowable, CategoryId
- Fields to set when borrowed (Checked out): Borrower, BorrowDate
- A dvd can be borrowed e.g. IsBorrowable = true

Audio Book:

- Required Fields: Id, Title, RunTimeMinutes, Type, IsBorrowable, CategoryId
- Fields to set when borrowed (Checked out): Borrower, BorrowDate
- An audio book can be borrowed e.g. IsBorrowable = true

Reference Book:

- Required Fields: Id, Title, Author, Pages, Type, CategoryId, IsBorrowable
- A reference book can NOT be borrowed e.g. IsBorrowable = false. This is a book you read at the library but can't be borrowed home 😊

General:
When editing a **library item,** if the item is borrowable (**IsBorrowable = true)**, the application user can **check out** the item for a customer. When lending an item to a customer the user enters the customer's name which will set the **Borrower** field in the database, along with the current date which will be set in the **BorrowDate** field.

A borrowed item cannot be borrowed by another customer.

However, a borrowed item can be returned by the customer. When this occurs the application user can **check in** the item. This means the item has been returned and can once again be borrowed. The operation unsets the values in the fields **Borrower** and **BorrowDate.**

Extra points: Creating a library item could be done in many ways. Nice user experience would be to only see input fields of the fields relevant to the type of library item the user is creating/editing. One way is to ask the user what type of library item they are creating before showing the form where they create the library item. Another is to show/hide fields when user is changing the type of item they want to create in a dropdown.

2.3 ACCEPTANCE CRITERIAS

- Create and edit Categories
- Delete Categories (if they are not referenced in any library item)
- Create library items of the types book, reference book, dvd and audio book.
- Edit and delete library items.
- Check in/Check out library items that is borrowable (book, dvd and audiobook)
- Listing library items should be sorted by Category Name. This can be changed to Type by the user. (This change need to persist in current session but not after application restart)
- Acronym after the title of library items (e.g. "The title (TT)")
- Validation on input fields.
- Data should be stored in the database

3. SCENARIO 2

As a user of the application I want to be able to manage all employees working at the library. I want to be able to create, update and delete employees.
When listing employees I want them to be grouped by role e.g. employees, managers and CEO.
The application should also calculate what salary the employee should have based on a rank multiplied with a constant coefficient (more about that later).
An employee can be a "regular" employee, or a manager. There can also be a CEO, but only one at a time. The database columns **IsManager** and **IsCEO** will determine if the employee is a manager or CEO. If false on both these the employee is a regular employee.
All employees including managers (except for the CEO) can have someone to report to, e.g. their manager. This is set with the **ManagerId** mapped against an **Employee Id** of the manager entity.

3.1 THE APPLICATION

General:
All fields are required for the employee. Except for the ManagerId on managers and the CEO. All regular employee must have a manager. All employee's salaries including the CEO's are **calculated when creating the entity**.

When creating the employee, the user inputs a rank (integer between 1-10). This rank is later used in conjunction with the salary coefficient to calculate the employee's salary. The salary is the product of the rank and the salary coefficient.

See below for the different salary coefficients based on the employee type.

Employee:
- Salary coefficient: 1.125
- IsManager = false
- Can be managed by managers but not the CEO.
- Cannot manage others

Manager:

- Can be managed by the CEO or by other Managers e.g. ManagerId set to another managers Id.
- Salary coefficient: 1.725
- IsManager = true

CEO:
- There can only be one at a time
- No one can be manager of the CEO
- IsCEO = true
- Salary coefficient: 2.725

3.2 ACCEPTANCE CRITERIAS

- Create, update and delete employees
- Say the employee should be a manager or even a CEO
- Only one CEO can be created at a time in the application database
- The salary should be calculated when creating the employee using the logic described in the task
- You should not be able to delete a manger or CEO that is managing another employee
- CEO can manage managers but not employees
- Managers can manage other managers and employees
- No one can manage the CEO
- Validation on input fields

4. THINGS WE LIKE

These are some subjects that we wanted to mention. It's no requirement that you use these techniques. We are just happy if you have better ways of solving problems or want to use some techniques you like. Feel free to go crazy. We always want to learn new things, improve our techniques and get better at what we do.

4.1 ASYNC PROGRAMMING

We are big fans of high performance, and async programming helps when it comes to high loads, and lots of integrations. Luckily for us, C# makes that easy compared to many languages.

4.2 IoC & DEPENDENCY INJECTION

Inversion of control and dependency injection is important when writing testable software. It also helps managing lifetime for objects, and structuring code. We'd love if you used it. You are free to use any framework you like, or write your own.

4.3 SERVICE LAYER PATTERN

What would the world look like without Domain Driven Design? Probably a lot worse. This is a design pattern that we frequently use. It simplifies structure, and you get reusable code. We like it. We use it.

4.4 SOLID OBJECT ORIENTED DESIGN

For example calculating the salary on employees. Will this be done in a service? A helper? Or method on the class itself (smart class).
Will we make use of the powerful interface segregation principle when creating our backend classes for library items and employees?
Think about how you structure your code so it can be easily maintained.

We absolutely **LOVE** comments in the code shortly describing what happens so someone else working in the same project easily can understand what you have written.
Another way (or preferably use both ways) is to use easy to understand names for variables, fields, classes and services etc.

5. HOW WILL YOU BE ASSESSED?

That depends on what role you are applying for. By having a fairly open assignment, we hope to be able to get to know you better, and see what your current level is. We believe that this exercise is a good way to do that, for both junior and senior developers. The scenarios above are both possible in our daily work, with the frameworks we are working with.

6. BEFORE SENDING YOUR WORK

To avoid large emails, please make you sure you do these steps first.

- Delete "bin" folders
- Delete "obj" folders
- Delete "packages"-folder
- Only include dll's if you use something that's not on Nuget.
- Zip remaining files.

WELCOME TO THE WORLD OF HIGH PERFORMERS.