C# developer test

Practical

Note

Searching online is allowed and even encouraged for this part.

DateTime

Write an extension method for the DateTime class that returns the next business day (Monday to Friday). Bonus points if you

1. Allow the caller to specify the number of business days instead of simply the next business day.
2. Allow the caller to specify holidays.
3. Account for time zone differences.
4. Allow the caller to specify the work week schedule to account for the business week not being the same in every country, for example Israel's workweek is Sunday to Thursday.

Asynchronous programming

The code snippet below retrieves a product's price and stock asynchronously. How can you optimize this code to execute both methods concurrently?

```
public async Task<ProductInfo> GetProductInfoAsync(int productId)
{
  var price = await GetProductPriceAsync(productId);
  var stock = await GetProductStockAsync(productId);

  return new ProductInfo
  {
    ProductId = productId,
    Price = price,
    Stock = stock,
  };
}
```

Querying efficiency

The code snippet below adds product information to every order line during order checkout. We noticed that the checkout process is rather slow for customers with a lot of items in their cart. How would you solve this performance challenge?

```
var order = new Order();
foreach (var cartItem in cartItems)
{
  var matchingProduct = await dbContext
    .Products
    .SingleOrDefaultAsync(product => product.Number == cartItem.Number);

  order.Lines.Add(new OrderLine
  {
    ProductId = matchingProduct.Id,
    ProductNumber = cartItem.ProductNumber,
    Quantity = cartItem.Quantity,
    UnitPrice = await priceQuerier.GetCustomerPriceAsync(
      customerNumber,
```

```
        cartItem.ProductNumber),
    });
}

await dbContext.Orders.AddAsync(order);
await dbContext.SaveChangesAsync();

return order;
```

Blazor web application

Objective

Develop a small web application, using C# in ASP.NET Core with Entity Framework Core and Blazor, to manage a list of tasks. The application should allow users to view, add, edit, delete, and mark tasks as complete.

Requirements

1. Create a simple database schema for tasks, with fields like "Title", "Description", "Due Date", "Priority", and "Status".
2. Use Entity Framework Core for database connectivity and CRUD (Create, Read, Update, and Delete) operations.
3. Implement a simple UI using Blazor to display the task list and allow the user to interact with it. The UI should include:
    a. A view of all tasks.
    b. A form to add new tasks.
    c. A way to edit and delete tasks.
    d. A way to mark tasks as complete or incomplete.
4. Implement proper exception handling and logging throughout the application.

Evaluation Criteria

1. Functionality
    a. Does the application work as expected?
    b. Does the application meet all the requirements?
2. Scalability and performance
    a. Is the application designed to handle increased load and avoid bottlenecks?
3. Error handling and validation
    a. Are input validations implemented correctly?
    b. Are error handling mechanisms implemented correctly?
4. Code quality:
    a. Is the code well-organized, modular, and readable?
    b. Does the code follow best practices and naming conventions?
5. Bonus
    a. Include unit tests and integration tests to ensure the application's reliability.
    b. Utilize MudBlazor.