Technical Assessment

1) Define the database tables using SQL Server syntax for the following case:
   A customer wants to sell their car so we need the car information (car year, make, model, and submodel) and location (zip code). Note: make sure to structure the tables so we don't repeat data.
   For each zip code, there's a list of buyers that are willing to buy the cars and each buyer has a quote (amount). For example, Buyer ABC will cover 10 zip codes and it will pay $500 for each car.
   Once the data is saved, the system will pick the best quote and mark it as the current one. Only one quote can be marked as current and it's not necessarily the highest quote.
   We also want to track the progress of the case using different statuses (Pending Acceptance, Accepted, Picked Up, etc.) and considering that we care about the current status and the status history (previous statuses indicate when it happened, who changed it, etc.). The status "Picked Up" has a mandatory status date but the rest of the statuses don't.
   Write a SQL query to show the car information, current buyer name with its quote and current status name with its status date. Do the same thing using Entity Framework. Make sure your queries don't have any unnecessary data.

2) What would you do if you had data that doesn't change often but it's used pretty much all the time? Would it make a difference if you have more than one instance of your application?

3) Analyze the following method and make changes to make it better. Explain your changes.

```
public void UpdateCustomersBalanceByInvoices(List<Invoice> invoices)
{
        foreach (var invoice in invoices)
        {
                var customer =
dbContext.Customers.SingleOrDefault(invoice.CustomerId.Value);
                customer.Balance -= invoice.Total;
                dbContext.SaveChanges();
        }
}
```

4) Implement the following method using Entity Framework, making sure your query is efficient in all the cases (when all the parameters are set, when some of them are or

when none of them are). If a “filter” is not set it means that it will not apply any filtering over that field (no ids provided for customer ids it means we don’t want to filter by customer).

```
public async Task<List<OrderDTO>> GetOrders(DateTime? dateFrom, DateTime? dateTo, List<int> customerIds, List<int> statusIds, bool? isActive)
{
    // your implementation
}
```

5) Create a C# method that would read all the .cs files from a folder and subfolders and apply actions. These are the possible actions:
    a) Search for async methods where the name of the method doesn’t have Async as a suffix. When that case is found, modify the name of the method (no need to update references).
    b) Search for any word that ends in Vm, Vms, Dto or Dtos and rename them to VM, VMs, DTO, and DTOs respectively
    c) Find methods that don’t have a blank line between them and add the blank line.

    Add unit tests to validate these cases are working as expected.