# Wheezly Technical Assessment

**First Question:**

Please take a look for repository link here.

You will find: Asp.net project, sql commands, database diagram.

**Second Question:**

To optimize data storage and retrieval, a caching strategy can be implemented based on the size of the data:

- Large Data: Store large datasets in a database. This approach supports complex operations and large-scale access patterns.
- Small Data: For smaller datasets, consider using lightweight storage solutions such as JSON files or hardcoding the data directly within the application.

This approach balances performance and resource utilization, ensuring an efficient caching mechanism tailored to the size and complexity of the data.

**Third Question:**

```
public void UpdateCustomersBalanceByInvoices(List<Invoice> invoices)
{
        //List to store customers needing updates.
        List<customer> customers = new List<customer>();
        foreach (var invoice in invoices)
        {
                // Use FirstOrDefault to check if the customer exists.
                // SingleOrDefault is return new object of Customers class and that
                don't make me take a check befour update
                var customer = dbContext.Customers.FirstOrDefault(a => a.customerId ==
                invoice.CustomerId.Value);
                //if customer is null, throw an exception with status code 500 and an
                appropriate error message.
                if (customer == null)
                {
                        throw new ApplicationException($"Customer with id
                        {invoice.Customer.Value} is not found");
                }
                // After confirming the customer is found, apply changes and add the
                customer to the list for updates.
                customer.Balance -= invoice.Total;
                customers.add(customer);
        }

         // After verifying all customers are found and updated in the list, perform a
        range update to execute a single database query.
         dbContext.UpdateRange(customers);
        // Make the save changes operation awaitable to handle long-running processes
        efficiently.
         await dbContext.SaveChangesAsync();
}
```

**Forth Question:**

```
public async Task<List<OrderDTO>> GetOrders(DateTime dateFrom, DateTime dateTo,
List<int> customerIds, List<int> statusIds, bool? isActive)
{
      var orders = dbContext.orders.where(o =>
            //dateFrom & dateTo is mandatory in function, can't be null
            o.creationDate >= dateFrom && a.creationDate <= dateTo
             //Check if customerIds is not empty before check.
             && customerIds.any() ? customerIds.any(c => c == o.customerId) : true
             //Check if statusIds is not empty before check.
             && statusIds.any() ? statusIds.any(c => c == o.statusId) : true
             //Take isActive true if it is null.
             && isActive != null ? o.isActive : true
      );

      return orders;
}
```

**Fifth Question:**

- **Review the Bug Thoroughly**: Carefully read and analyze the bug to ensure a clear understanding of the issue.

- **Investigate the Code Area**: Examine the relevant section of the code. If the issue is within the area I was working on, I will make the necessary changes to fix the bug and conduct tests. If the issue pertains to another area, I will schedule a call with the team member who worked on that area to understand their changes and gather insights before proceeding with my fix.

- **Conduct Smoke Testing**: Perform smoke testing around the affected area to ensure that the changes did not unintentionally impact other parts of the system.

- **Consult with QA Team**: Schedule a call with Bill from the QA Department to verify that the bug has been properly addressed and resolved.

- **Submit the Pull Request**: Once the bug is fixed and verified, create the pull request for review.