**Fix Bugs**:

Here are the **identified bugs** and **fixes** in the code:

* **Incorrect if Condition for Sending "Comeback Mail" on Sundays**

**Bug:**if (DateTime.Now.DayOfWeek.Equals(DayOfWeek.Monday))The comment says "Every Sunday run Comeback mail", but the condition checks for Monday instead.  
**Fix:**Change DayOfWeek.Monday to DayOfWeek.Sunday

* **Incorrect Email Address in DoEmailWork2**

**Bug:**m.From = new System.Net.Mail.MailAddress("infor@EO.com");  
**Fix:**The email "infor@EO.com" is likely a typo. The correct format (matching the welcome mail) should be:  
m.From = new System.Net.Mail.MailAddress("info@EO.com");

* **Incorrect Boolean Logic in Main Method**

**Bug:**if (success == true)  
{  
Console.WriteLine("All mails are sent, I hope...");  
}  
if (success == false)  
{  
Console.WriteLine("Oops, something went wrong when sending mail (I think...)");  
}  
  
success is reassigned in the Comeback Mail block but should reflect the final outcome of both email types.  
**Fix:**  
Before, we used one variable to check if emails were sent, but it was replaced. Now, we **check each email result separately** to avoid losing the first result. This helps us know if **both emails** were sent successfully.

* **Incorrect Use of Try-catch for Exception Handling**

**Bug:**The catch (Exception) hides exceptions without logging any helpful information. It becomes difficult to debug errors when sending emails.

**Fix:**

Previously, exceptions were caught without showing any details, making it hard to find out what went wrong. Now, the fix logs a clear error message for each failed email with the recipient’s address and the exact error. This makes debugging easier and more helpful.  
  
catch (Exception ex)

{

Console.WriteLine($"Failed to send email to {recipient.Email}. Error: {ex.Message}");

result.AddFailure(recipient);

}

* **Missing using Statement for System.Net.Mail**

**Bug:**The code directly uses System.Net.Mail.MailMessage and System.Net.Mail.SmtpClient without using the System.Net.Mail; directive.  
This reduces readability and makes the code longer.  
**Fix:**Add using System.Net.Mail; at the top of the file:  
using System.Net.Mail;

* **Magic Strings Instead of Constants**

**Bug:**The program uses hardcoded strings like:  
  
m.From = new MailAddress("[info@EO.com](mailto:info@EO.com)");  
m.Subject = "Welcome as a new customer at EO!";  
  
Problem: If you need to change the email or subject in multiple places, you must update them manually.  
**Fix:**  
Replaced hardcoded strings with configuration values and constants. Now, email addresses, subjects, and templates are managed centrally, making updates easier and reducing the chance of errors.

* **SMTP Client Not Disposed Properly**

**Bug:**The code creates a SmtpClient but does not properly dispose of it, which may lead to memory issues. To avoid this, it’s best practice to use the 'using' statement, ensuring that it is disposed of correctly after use.

**Fix:**  
 using (var smtpClient = new SmtpClient(\_configuration.EmailHost))  
 {  
 smtpClient.Port = \_configuration.Port;   
smtpClient.Credentials = new NetworkCredential(\_configuration.UserName, \_configuration.Password);   
var mailMessage = new MailMessage {  
 From = new MailAddress(from),   
Subject = subject,  
 Body = body,  
 IsBodyHtml = true   
};  
 mailMessage.To.Add(to);   
  
await smtpClient.SendMailAsync(mailMessage);   
}

* **Incorrect Email Formatting (Missing IsBodyHtml)**

**Bug:**The email body contains HTML (<br>) but is not marked as HTML.  
Some email clients may display raw HTML tags instead of formatting them correctly.

**Fix:**  
Set IsBodyHtml = true; for all emails:  
IsBodyHtml = true

* **Using == true and == false in Boolean Comparisons**

**Bug:**

The code uses redundant comparisons like:  
if (success == true)  
  
Instead, just use:  
if (success)  
  
Similarly,  
if (Send == false)  
Should be:  
if (!Send) **Fix:**Remove unnecessary == true and == false comparisons for cleaner code.

* **Inefficient Customer Filtering for "Welcome Mail"**

**Bug:**if (e[i].CreatedDateTime > DateTime.Now.AddDays(-1))  
  
Problem: DateTime.Now.AddDays(-1) calculates a different value each time it's used inside a loop.  
This is inefficient because the function is called for each customer.

**Fix:**

Optimized date filtering by calculating DateTime.Now.AddDays(-1).Date once before filtering. This prevents repeated computation inside the loop, improving performance and readability.

DataLayer.ListCustomers() .Where(c => c.CreatedDateTime.Date == DateTime.Now.AddDays(-1).Date) .ToList();

### **Improved Readability:**

**Serilog Integration:**

I added a tool called Serilog to make it easier to log important information. This helps me see what's happening in the app, like if something goes wrong or if everything is working well. It also makes it easy to change how and where the logs are saved (like in a file or on the screen).

**Background Job Execution (Hangfire):**

I use Hangfire to schedule background tasks, which means some things (like sending emails) happen automatically at the right time without me having to do them manually. This helps the app run smoothly without getting stuck.

**Cleaner Dependency Injection (DI):**

I made sure all the important services, like sending emails or managing tasks, are organized properly and added in one place. This makes it easier to update or change things in the future.

**DateTimeProvider:**

I created a special tool to handle dates and times, like checking if today is Sunday. This makes the code cleaner, easier to test, and less likely to cause mistakes.

### **Refactored Structure:**

**Modularized Email Sending Logic:**

I broke down the email-sending process into smaller parts, like a special section for sending welcome emails. This makes it easier to add new types of emails later and keeps everything organized.

**Hangfire Jobs Refactor:**

Before, some tasks (like sending emails) were done in one big place, but now they are moved to Hangfire so they can run in the background without slowing down the main part of the app. This makes the app faster and more reliable.

**Simplified Job Execution:**

Now, I schedule tasks like sending emails to run automatically, instead of doing them manually. This keeps the app running smoothly and quickly.

**Improved Configurations and Constants Management:**

I moved all important settings, like where to save logs or what type of emails to send, into a special file. This makes it easier to change things later without breaking the app.

**Error Handling and Logging:**

Serilog helps me track any problems in the app. If something goes wrong, I can easily find out what happened and fix it without the app crashing.

**Job Execution and Scheduling Improvements:**

With Hangfire, I can set up jobs to run at certain times, like sending emails every day or every Sunday. This makes sure the app does everything it's supposed to do on time.

**Scheduled Task Management:**

I use Hangfire to make sure certain tasks happen at the right time, like sending emails or updating information. This makes the app work on schedule without me having to think about it.

**Enhanced Email Variables Handling:**

I set up a way to add information to emails (like a special code) so it's easy to update or change what’s in the email without messing with the main code.