Email OTP Module Implementation- DSO Assignment

Description

In this document, I will describe how the EMail Module is implemented.For this assignment, I developed C# Console Application to demonstrate how the Email OTP flow is implemented.

Technology Stack & Concepts

Technology	Version	Description
.NET	8	The framework used to develop
C#	12	The main language used to implement
XUnit		For Unit Testing
Moq		For mocking the services for Unit Testing

- Follow the TDD [Test Driven Development] practice to implement this module.
- No database integration is used, to construct the main flow within the given timeline. Thus mainly focused on implementing the main business flow.
- Monolithic architecture pattern mainly used, as this is sample C# module.
- Assumed the SendEmail which is used to send out the email, is already implemented.

Technical Concepts Used

1. Used followings from SOLID Principles
    a. Dependency Inversion
    b. Interface Segregation
    c. Single Responsibility
2. Use Lifetime of services as Transient [C# DI Lifetime]
3. Test Driven Development
4. Transaction CancellationToken

Function Flow

In this module there are 2 main functionalities.

1. Generate OTP and Send Email
2. Validate OTP

Flow Introduction - Generate Email

Flow Constraints

1. Email should be valid format
2. Email should be from **dso.org.sg**

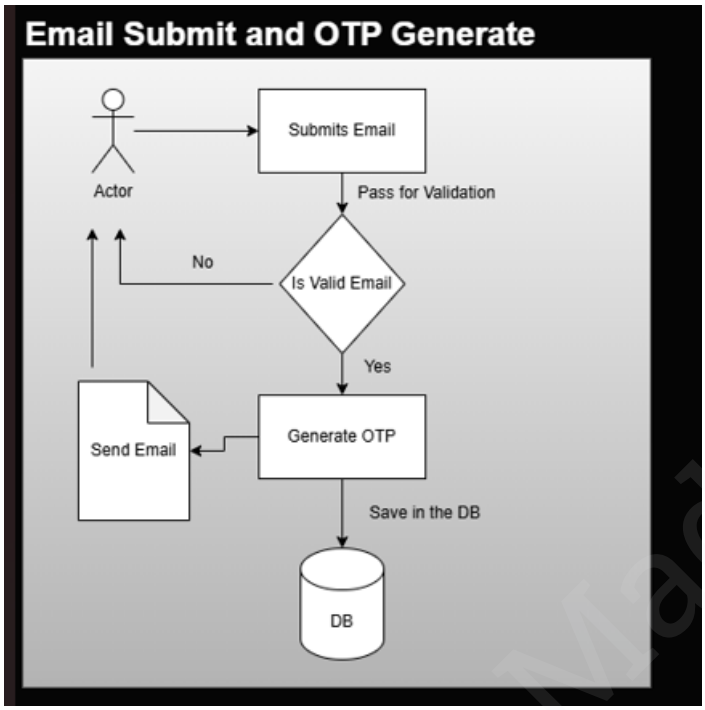

Image 1. Generate OTP and Send EMail

Flow Introduction - Validate OTP

Flow Constraints

1. OTP should be 6 Length
2. Users can do it 10 times within the 1 minute period.

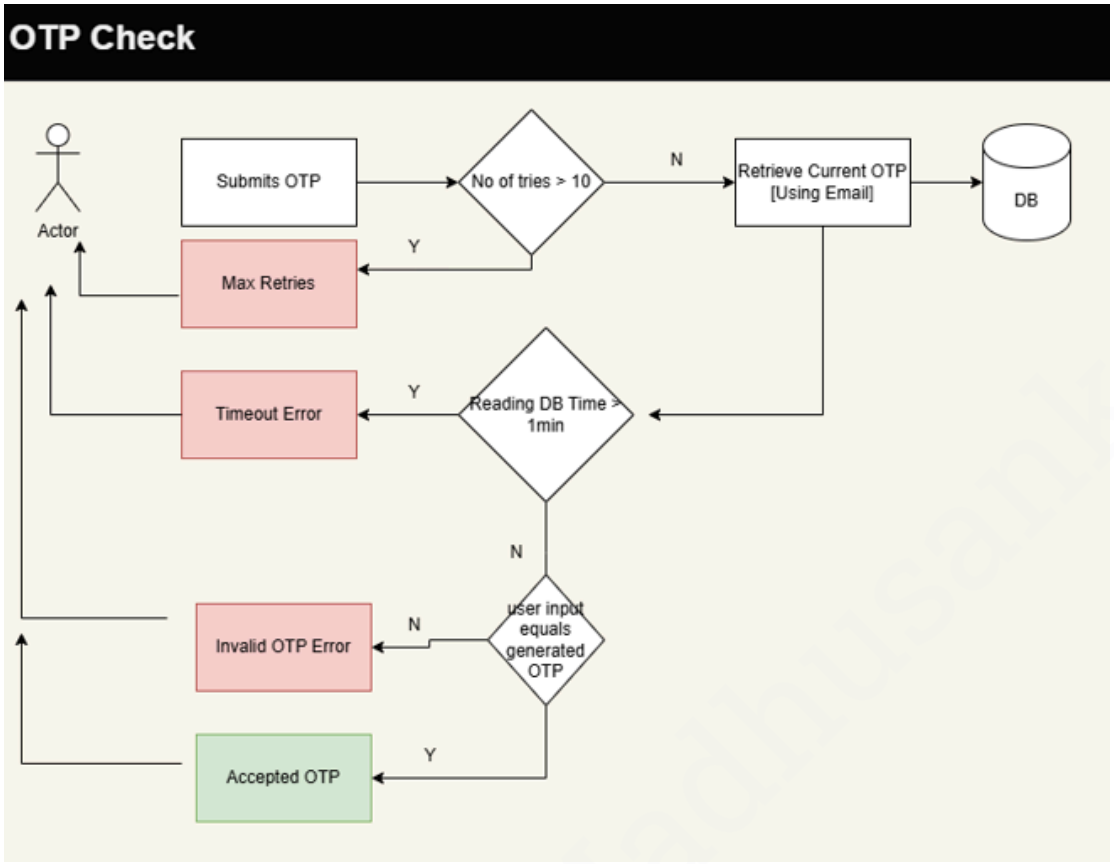

Image 2. OTP Validate Flow

Code Work

Module Structure

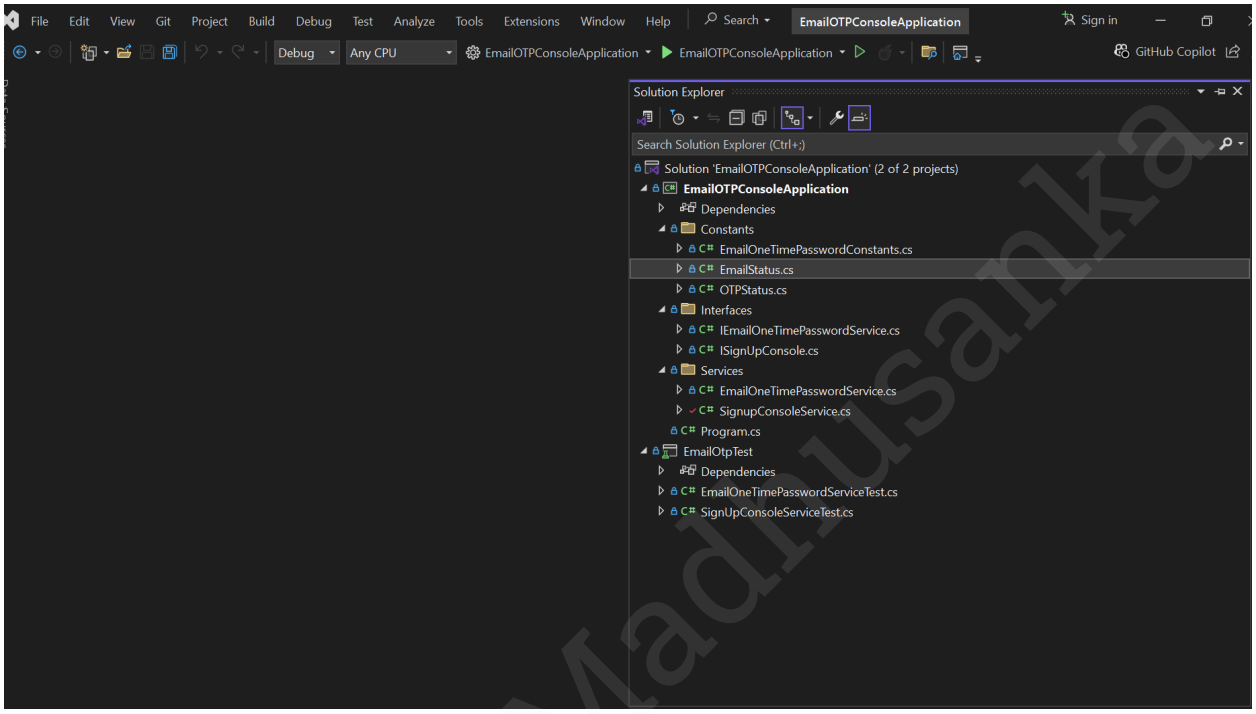

Image 3. Module Structure

Item	Description
EMailOTPConsoleApplication	This is the Startup Project. In here the Console Application Project and main interface where the main flow is initiated.
EmailOtpTest	XUnit Class library, where write all the Unit Tests.
Constants	Folder, which consists all the constants/ Enums used throughout the application
EmailOneTimePasswordConstant	Main constant file, which has all the Constants used throughout the application.
EmailStatus	Enum, which consists of the Email Send Status as Enum.

OTPStatus	Enum, which consists of the OTP Status.
Interfaces	Folder. Which consists of the all the interfaces used
IEmailOneTimePasswordService	An interface which is used for EmailOneTimePasswordService
ISignUpConsole	An interface which is used for the SignUpConsole.
Services	Folder, which consist of the Services used in the application
EmailOneTimePasswordService	The Service, which consists the core functional logics used to GenerateOTP and ValidateOTP
SignupConsoleService	The service which is used to handle core user interactions through Console.
Program	Bootstrap class of the Application, which has the core configuration, service injection logic.
EmailOneTimePasswordServiceTest	XUnit Tests for the EmailOneTimePasswordService
SignUpConsoleServiceTest	XUnit Tests for the SignUpConsoleService.

Code Work

1. EmailOneTimePasswordConstants

```
namespace EmailOTPConsoleApplication.Constants
{
    public static class EmailOneTimePasswordConstants
    {
        public static readonly string YourOTPCodeIs = "Your OTP Code is ";
        public static readonly string OtpCodeValid = ". The code is valid
for 1 minute.";
```

```
        public static readonly string OtpIsValidAndChecked = "OTP is valid
and checked";
        public static readonly string OtpIsWrongAfter10Tries = "OTP is
wrong after 10 tries";
        public static readonly string OtpTimeout = "Timeout after 1 min";

        public static readonly string EmailSendStatusOk = "Email containing
OTP has been sent successfully.";
        public static readonly string EmailSendFail = "Email address does
not exist or sending to the email has failed.";
        public static readonly string EmailInvalid = "Email address is
invalid.";

        public static readonly string InvalidOtp = "Please enter a valid
OTP.";
    }
}
```

2. EMail Status

```
namespace EmailOTPConsoleApplication.Constants
{
    internal enum EmailStatus
    {
        STATUS_EMAIL_OK,
        STATUS_EMAIL_FAIL,
        STATUS_EMAIL_INVALID
    }
}
```

3. OTPStatus

```
namespace EmailOTPConsoleApplication.Constants
{
    public enum OTPStatus
    {
        STATUS_OTP_OK,
        STATUS_OTP_FAIL,
        STATUS_OTP_TIMEOUT
    }
```

```
}
```

4. IEmailOneTimePasswordService

```
using EmailOTPConsoleApplication.Constants;

namespace EmailOTPConsoleApplication.Interfaces
{
    public interface IEmailOneTimePasswordService
    {
        public (bool isValidEmail, string? emailBody)
GenerateEmailOneTimePassword(string userEmail);
        public Task<OTPStatus> CheckOneTimePassword(string inputOtp);
    }
}
```

5. ISignUpConsole

```
namespace EmailOTPConsoleApplication.Interfaces
{
    internal interface ISignUpConsole
    {
        public Task Start();
    }
}
```

6. EmailOneTimePasswordService

```
using EmailOTPConsoleApplication.Constants;
using EmailOTPConsoleApplication.Interfaces;
using System.Text.RegularExpressions;

namespace EmailOTPConsoleApplication.Services
{
    public sealed class EmailOneTimePasswordService :
IEmailOneTimePasswordService
    {
        private string? Email { get; set; }
```

```
        // Use configuration file to read these
        private readonly string _otpDomain = "dso.org.sg";

        private readonly Regex _emailRegex =
new(@"^[^@\s]+@[^@\s]+\.[^@\s]+$", RegexOptions.IgnoreCase,
TimeSpan.FromMilliseconds(250));

        #region Generate Email OTP
        public (bool isValidEmail, string? emailBody)
GenerateEmailOneTimePassword(string userEmail)
        {
            var y = userEmail.EndsWith(_otpDomain);
            if (!IsValidEmail(userEmail) ||
!userEmail.EndsWith(_otpDomain))
            {
                return (false, null);
            }
            Email = userEmail;
            string otpCode = GenerateRandomOTP();
            string emailBody =
$"{EmailOneTimePasswordConstants.YourOTPCodeIs}{otpCode}{EmailOneTimePasswo
rdConstants.OtpCodeValid}";

            return (true, emailBody);
        }

        private bool IsValidEmail(string email)
        {
            var x = _emailRegex.IsMatch(email);
            return x;
        }

        private static string GenerateRandomOTP()
        {
            Random random = new();
            return random.Next(100000, 999999).ToString();
        }
        #endregion

        #region Validate OTP
        public async Task<OTPStatus> CheckOneTimePassword(string inputOtp)
        {
```

```
            string generatedUserOtp;
            try
            {
                generatedUserOtp = await RetrieveOneTimePassword();
            }
            catch (TimeoutException)
            {
                return OTPStatus.STATUS_OTP_TIMEOUT;
            }

            if (string.Equals(inputOtp, generatedUserOtp,
StringComparison.OrdinalIgnoreCase))
            {
                return OTPStatus.STATUS_OTP_OK;
            }

            return OTPStatus.STATUS_OTP_FAIL;

        }

        public async Task<string> RetrieveOneTimePassword()
        {
            // Retrieve the current OTP saved in DB
            // to get the OTP sent out
            await Task.Delay(2000);
            return "123456";
        }
        #endregion

    }
}
```

7. SignUpConsoleService

```
using EmailOTPConsoleApplication.Constants;
using EmailOTPConsoleApplication.Interfaces;

namespace EmailOTPConsoleApplication.Services
{
    public class SignupConsoleService(IEmailOneTimePasswordService
```

```
emailOneTimePasswordService) : ISignUpConsole
    {
        private readonly IEmailOneTimePasswordService
_emailOneTimePasswordService = emailOneTimePasswordService;
        private readonly int _maxRetries = 10;
        private readonly TimeSpan _otpTimeout = TimeSpan.FromMinutes(1);
        public int tries = 1;

        public async Task Start()
        {
            GenerateEmailOneTimePassword();

Console.WriteLine("************************************************");
            await CheckOneTimePassword();
        }

        #region Generate OTP and Send Email
        public void GenerateEmailOneTimePassword()
        {
            Console.WriteLine("Please enter your email");
            string? userEmail = Console.ReadLine();
            if (userEmail == null)
            {
                return;
            }
            userEmail = userEmail.Trim();
            var (isValidEmail, emailBody) =
_emailOneTimePasswordService.GenerateEmailOneTimePassword(userEmail);
            if (!isValidEmail)
            {

Console.WriteLine(EmailOneTimePasswordConstants.EmailInvalid);
                return;
            }
            var emailStatus = SendEmail(userEmail, emailBody!);
            Console.WriteLine(emailStatus);
        }

        public static string SendEmail(string emailAddress, string
emailBody)
        {
            throw new NotImplementedException();
        }
```

```
        #endregion

        #region Check and Validate OTP
        public async Task CheckOneTimePassword()
        {
            using var cts = new CancellationTokenSource(_otpTimeout);
            var token = cts.Token;

            while (tries <= _maxRetries && !token.IsCancellationRequested)
            {
                Console.Write("Enter OTP: ");

                string? otp = Console.ReadLine();

                if (string.IsNullOrEmpty(otp) || otp.Length != 6)
                {

Console.WriteLine(EmailOneTimePasswordConstants.InvalidOtp);
                    return;
                }

                var status = await
_emailOneTimePasswordService.CheckOneTimePassword(otp!);

                if (status == OTPStatus.STATUS_OTP_OK)
                {

Console.WriteLine(EmailOneTimePasswordConstants.OtpCodeValid);
                    return;
                }

                if (status == OTPStatus.STATUS_OTP_TIMEOUT)
                {

Console.WriteLine(EmailOneTimePasswordConstants.OtpTimeout);
                    return;
                }
                tries++;
            }


Console.WriteLine(EmailOneTimePasswordConstants.OtpIsWrongAfter10Tries);
        }
```

```
        #endregion
    }
}
```

8. EmailOneTimePasswordTest

```
using EmailOTPConsoleApplication.Constants;
using EmailOTPConsoleApplication.Services;

namespace EmailOtpTest
{
    public class EmailOneTimePasswordServiceTest
    {

        #region Constraint Check
        public static IEnumerable<object[]>
OneTimePasswordGenerateConstraintsCheckMemberData()
        {
            yield return new object[] { "InvalidEmail", "test" };
            yield return new object[] { "InvalidEmailWithNonAllowedOrg",
"dosorgtest@gmail.com" };
        }

        [Theory]

[MemberData(nameof(OneTimePasswordGenerateConstraintsCheckMemberData))]
        public void
EmailOneTimePasswordService_CheckValidEmail_Positive(string scenario,
string userInput)
        {
            switch (scenario)
            {
                case "InvalidEmail":
                    // Arrange
                    var emailService1 = new EmailOneTimePasswordService();

                    // Act
                    var (isValidEmail, _) =
emailService1.GenerateEmailOneTimePassword(userInput);
```

```
                    // Assert
                    Assert.False(isValidEmail);
                    break;
                case "InvalidEmailWithNonAllowedOrg":
                    // Arrange
                    var emailService2 = new EmailOneTimePasswordService();

                    // Act
                    var results2 =
emailService2.GenerateEmailOneTimePassword(userInput);

                    // Assert
                    Assert.False(results2.isValidEmail);
                    break;

            }


        }
        #endregion

        #region Constraint Check
        [Theory]
        [InlineData("testemail@dso.org.sg")]
        public void
EmailOneTimePasswordService_GenerateEmail_Positive(string userInput)
        {
            // Arrange
            var service = new EmailOneTimePasswordService();

            // Act
            var (isValidEmail, emailBody) =
service.GenerateEmailOneTimePassword(userInput);

            // Assert
            Assert.True(isValidEmail);
            Assert.Contains("Your OTP Code is ", emailBody!);
        }
        #endregion

        #region Check OTP
        [Fact]
        public async Task CheckOneTimePassword_ValidOtp_ReturnsSuccess()
```

```
        {
            // Arrange
            string validOtp = "123456";
            var service = new EmailOneTimePasswordService();

            // Act
            var result = await service.CheckOneTimePassword(validOtp);

            // Assert
            Assert.Equal(OTPStatus.STATUS_OTP_OK, result);
        }

        [Fact]
        public async Task CheckOneTimePassword_InvalidOtp_ReturnsFailure()
        {
            // Arrange
            string validOtp = "567567";
            var service = new EmailOneTimePasswordService();

            // Act
            var result = await service.CheckOneTimePassword(validOtp);

            // Assert
            Assert.Equal(OTPStatus.STATUS_OTP_FAIL, result);
        }
        #endregion
    }
}
```

9. SignUpConsoleServiceTest

```
using EmailOTPConsoleApplication.Constants;
using EmailOTPConsoleApplication.Interfaces;
using EmailOTPConsoleApplication.Services;
using Moq;

namespace EmailOtpTest
{
    public class SignUpConsoleServiceTest
    {
        private readonly Mock<IEmailOneTimePasswordService>
```

```
_mockEmailOneTimePasswordService;
        private readonly SignupConsoleService _service;
        public SignUpConsoleServiceTest()
        {
            _mockEmailOneTimePasswordService = new
Mock<IEmailOneTimePasswordService>();
            _service = new
SignupConsoleService(_mockEmailOneTimePasswordService.Object);
        }

        [Fact]
        public void
GenerateEmailOneTimePassword_ValidEmail_SendsEmail_Positive()
        {
            // Arrange
            string validEmail = "test@example.com";
            (bool isValid, string? emailBody) expectedResult = (false,
null);
            _mockEmailOneTimePasswordService.Setup(x =>
x.GenerateEmailOneTimePassword(validEmail)).Returns(expectedResult);

            var output = new StringWriter();
            Console.SetOut(output);

            var input = new StringReader(validEmail);
            Console.SetIn(input);

            // Act
            _service.GenerateEmailOneTimePassword();

            // Assert
            _mockEmailOneTimePasswordService.Verify(x =>
x.GenerateEmailOneTimePassword(validEmail), Times.Once);
            Assert.Contains(EmailOneTimePasswordConstants.EmailInvalid,
output.ToString());
            output.Dispose();
            input.Dispose();
        }

        [Fact]
        public async Task
CheckOneTimePassword_ValidOtp_ReturnsSuccess_Positive()
        {
```

```
            // Arrange
            string validOtp = "123456";

            var output = new StringWriter();
            Console.SetOut(output);

            _mockEmailOneTimePasswordService.Setup(x =>
x.CheckOneTimePassword(validOtp))
                .ReturnsAsync(OTPStatus.STATUS_OTP_OK);

            // Act
            Console.SetIn(new StringReader(validOtp +
Environment.NewLine));
            await _service.CheckOneTimePassword();

            // Assert
            Assert.Contains(EmailOneTimePasswordConstants.OtpCodeValid,
output.ToString());
        }

        [Fact]
        public async Task
CheckOneTimePassword_InvalidOtp_ReturnInvalid_Positive()
        {
            // Arrange
            string validOtp = "78978978";

            var output = new StringWriter();
            Console.SetOut(output);

            _mockEmailOneTimePasswordService.Setup(x =>
x.CheckOneTimePassword(validOtp))
                .ReturnsAsync(OTPStatus.STATUS_OTP_OK);

            // Act
            // Simulate console input (you can use a mocking framework or
directly set the input)
            Console.SetIn(new StringReader(validOtp +
Environment.NewLine));
            await _service.CheckOneTimePassword();

            // Assert
            Assert.Contains(EmailOneTimePasswordConstants.InvalidOtp,
```

```
output.ToString());
        }

        [Fact]
        public async Task CheckOneTimePassword_MaxTriesExceed_Positive()
        {
            // Arrange
            string validOtp = "7899789";

            var output = new StringWriter();
            Console.SetOut(output);

            _mockEmailOneTimePasswordService.Setup(x =>
x.CheckOneTimePassword(validOtp))
                .ReturnsAsync(OTPStatus.STATUS_OTP_OK);

            _service.tries = 11;

            // Act
            // Simulate console input (you can use a mocking framework or
directly set the input)
            Console.SetIn(new StringReader(validOtp +
Environment.NewLine));
            await _service.CheckOneTimePassword();

            // Assert

Assert.Contains(EmailOneTimePasswordConstants.OtpIsWrongAfter10Tries,
output.ToString());
        }
    }
}
```

Testing

1. Unit Test [DRY Test]

- The main testing approach used is Unit Tests.
- Each code unit is tested using XUnit Test
- For the Unit Tests, in this module as I have not used any Database integration, I didn't use InMemory DB.
- Used Test Patterns, Fact, Theory Inline Data & Theory Member Data.
- Used MOQ Library to mock services and to mock service results.

Ex: `_mockEmailOneTimePasswordService.Setup(x => x.GenerateEmailOneTimePassword(validEmail)).Returns(expectedResult);`

==========

GitHub Repository - [EmailOTP_NETCore]