How the azure email service works

In this file I will talk about how the azure email service works and how you can set it up for my project.

The setup

The azure side

Before you start you need to have:

- An Azure Email Communication Services Resource created and ready with a provisioned domain Create Email Communication Resource.
- An active Communication Services resource connected with Email Domain and a Connection String. Create and manage Email Communication Service resources.

If you don't have a communication service I recommend following the tutorial from Microsoft communication resource or even the Email Communication Service

Now if you have both of these I recommend looking if you setup the azure managed domain for the email service if not follow the Microsoft tutorial on Managed domains and look if the connection is made at domains.

The project side

Before you start on the project side you need to have:

- a console or API application
  - I go with the API application to show the implemented way
- a class library
  - To zone of the logic of the email and not have cross cutting concerns.
- An idea what you want to do with the email service

With the project where the actual sending of the mails will be executed you need to add the nuget for `Azure.Communication.Email` this can be done by using the command `dotnet add package Azure.Communication.Email` in the directory of the project.

There are now multiple ways that you can connect to the server with `connection string`, `Microsoft Entra ID` and `AzureKeyCredential`.

I would say there are no huge differences between them as you still need a way to store the actual values but differs on what that information is and how it's being accessed. So I first went with the connection string as this is a regular environment variable that you can set but I ended up using the `AzureKeyCredential` as this used more obscure information that is less

valuable for bad actors as they can not directly interact with the server if they get ahold of it somehow.

Following that the recomendation from Microsoft will be done that concerns how to send emails as they recommend sending it async. To actually send the message you do:

```
EmailSendOperation emailSendOperation = await emailClient.SendAsync(
    Azure.WaitUntil.Started,
    sender,
    recipient,
    subject,
    htmlContent);
```

Then you can also poll the status of sending it by doing `await emailSendOperation.UpdateStatusAsync()` this is done in a loop and you write the contents when it's done like `emailSendOperation.Value.Status`

Now that piece of logic is done you can use the API or console project (depending on which you will use) to connect to this project and use the methods from it.

How is it implemented in the project

In the project I have a class library that handles the email service only to have a single-responsibility principle on project basis and not have cross cutting concerns due to the nature of the dependencies on projects with C# applications.

So because it is on its own their is no need to worry about it effecting something without it being included. It's also easier to swap providers due to this due to making another project en implementing that details of the other provider.

Now how the code is structured I currently have the data api be the connection to this class library because I want to use this mainly to create 2FA-auth with emails by sending a simple code and checking if it's right. For that I also need to save the code that is generated and I didn't see the need to create a full new API service that should handle this and could be possibly used to send reports back based on the data.

Another feature that it should provide is to ban a player but that is something for later down the line. As banning is low priority right now and is considered a nice to have.

Now it starts at the frontend when the login is requested we go through the normal flow of the login if both user and password is correct then a mail will be generated/requested with the code that will be asked by the frontend to fill in before they are officially logged-in. So when the pop-up for the code is being shown the user puts in their code -> goes to Data API and is received by the email controller which executes the generation and verify of the email code.
For generation it will be generated code -> send to user -> enter in database.
While for verify it's email check -> code format check -> search for both email and code is correct together -> send either a positive result or a negative result back -> if positive send user to the home page.