**Multifactor Authentication**

Multifactor authentication (MFA) is an added layer of security used to verify an end user's identity when they sign in to an application.

An Okta admin can configure MFA at the organization or application level. If both levels are enabled, end users are prompted to confirm their credentials with factors when signing in to Okta and when accessing an application.

* **Enable MFA in Okta Org**

We need to enable MFA from the Admin Console of Okta org before we can use it with the Okta API.

* In the Admin Console, go to **Security** > **MultiFactor**.
* Under **Factor Types** we can see ([Okta Verify](https://help.okta.com/en-us/Content/Topics/Mobile/okta-verify-overview.htm), [SMS](https://help.okta.com/en-us/Content/Topics/Security/mfa/sms.htm), [Google Authenticator](https://help.okta.com/en-us/Content/Topics/Security/mfa/google-authenticator.htm), [Duo Security](https://help.okta.com/en-us/Content/Topics/Security/Security_Duo.htm), [WebAuthn](https://help.okta.com/en-us/Content/Topics/Security/mfa-webauthn.htm), [Symantec VIP](https://help.okta.com/en-us/Content/Topics/Security/mfa/symantec-vip.htm), [On-Prem agent (including RSA)](https://help.okta.com/en-us/Content/Topics/Security/MFA_OnPrem.htm), [Email](https://help.okta.com/en-us/Content/Topics/Security/mfa/email.htm)), click on Google Authenticator.
* Click **Inactive** in the upper right and then select **Activate.**
* Then go to **Factor Enrollment** tab.
* To Enroll Google Authenticator in a multifactor policy, Click on **Add Multifactor Policy**
  + Enter the Policy name, Policy Description
  + Assign to groups.
  + Set Google Authenticator to **Optional** or **Required**.
  + Click **Create Policy.**
* Then Click **Add Rule**
  + Enter Rule name, excluded users (if needed)
  + If User’s IP is **Anywhere** Then Enroll in Multi-factor **the first time a user signs in**.
  + Click **Update rule.**
* **Create Okta API token.**

Okta API tokens are used to authenticate requests to Okta APIs. When calling an Okta API endpoint, we need to supply a valid API token in the HTTP Authorization header, with a valid token specified as the header value. We need to prefix the value with the SSWS identifier, which specifies the proprietary authentication scheme that Okta uses. For example:

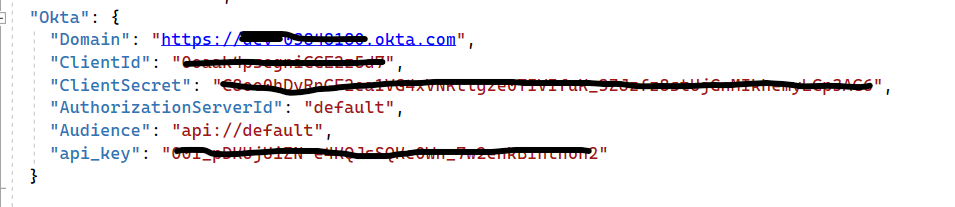
Authorization: SSWS 00QCjAl4MlV-WPXM...0HmjFx-vbGua

The steps to create Okta API Token:

* In the Admin Console, select **Security** > **API** from the menu and then select the **Tokens** tab.
* Click **Create Token**.
* Name your token and click **Create Token**.
* Record the token value. This is the only opportunity to see it and record it.

**API for MFA implementation :**

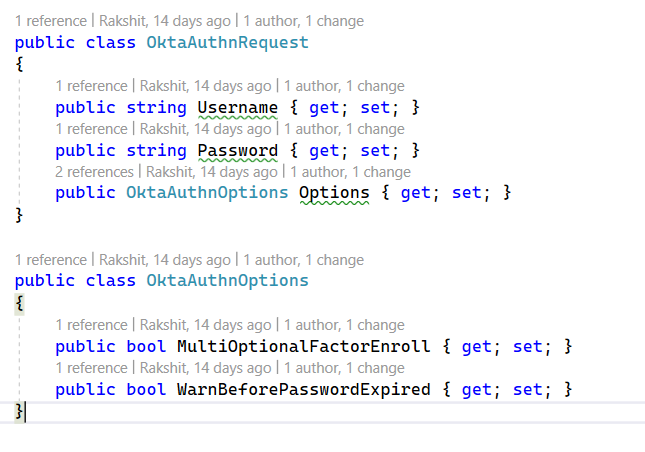
Before creating api’s we need to make some settings in appsetting.json file .



Make settings in appsettings**.**json file like shown in above screenshot clientId , domain and clientSecret is available in okta portal after you make an application , use those values here. **Api\_key** is okta **api token**. After making these configuration you can start creating api .

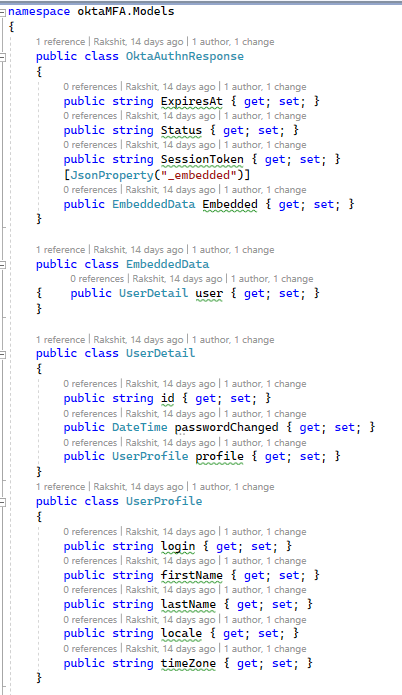
1. **AuthController :**

Firstly create **OktaAuthnRequest.cs** in model folder to make request :

****

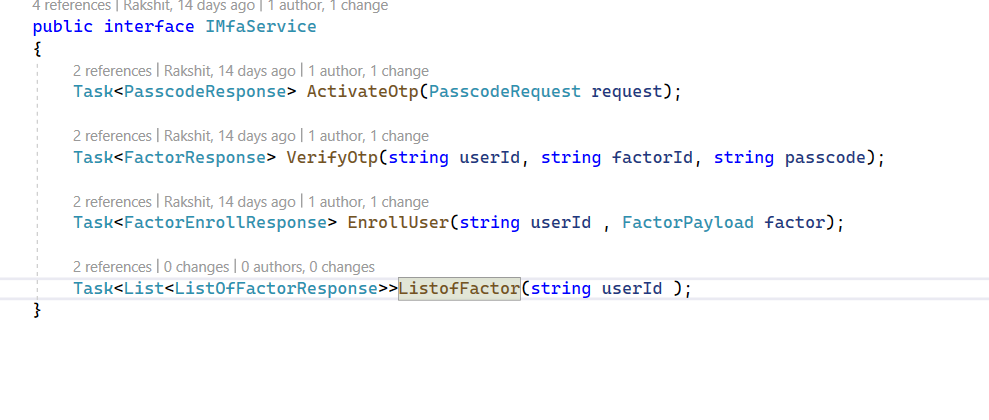
**oktaAuthnResponse.cs :**

Create another class in model folder and name it **oktaAuthnResponse**, we receive response in this format when we perform **PrimaryLogin**.

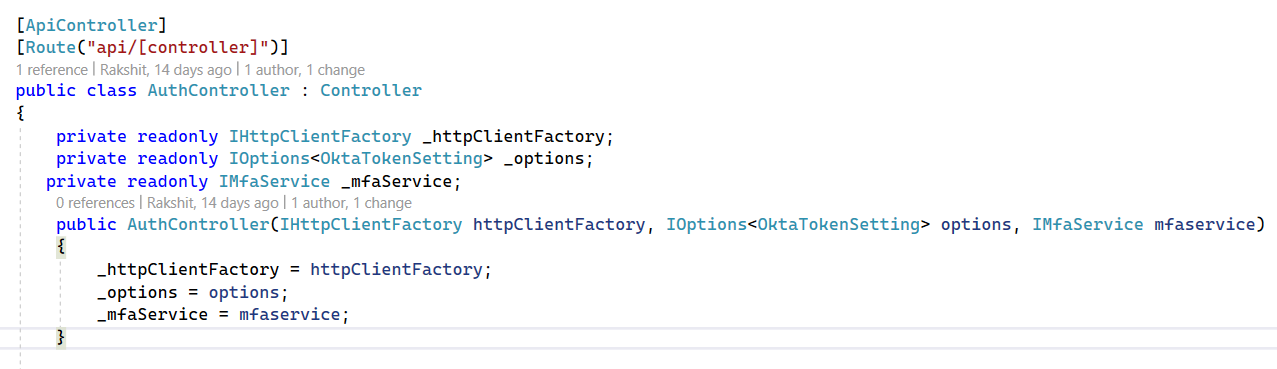
****

Create a controller name it as **AuthController** , initially declare services which we are going to use , for this we have created a service folder and interface

**IMfaService :**



Also create a class and name it as **MfaService** . Declare your services in AuthController.



Now in AuthController, create **PrimaryLogin** API it takes **OktaAuthnRequest** as requestand process it.

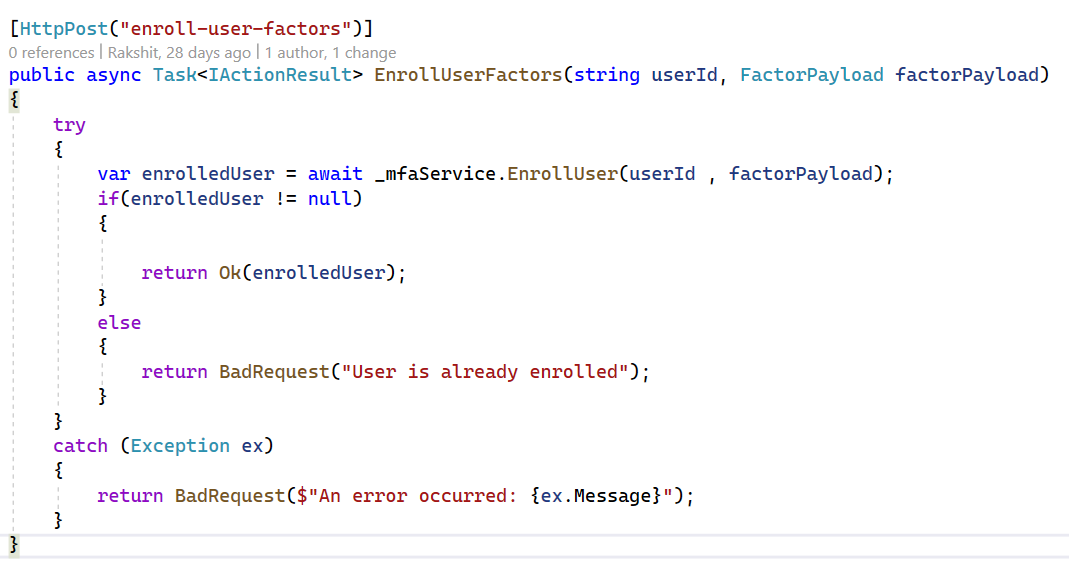


As per above code we need to make **requestBody** which contains username , password and options . We need to make post request using requestBody and **url** . If we get response from okta servers we Deserialize the **responseContent** and return it. Else we deserialize response and sends error.

**Enrolment User Factor :**

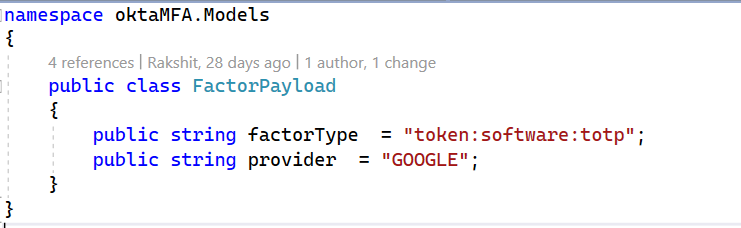
EnrolUserFactor is used to enroll a user in a 2FA factor when they first create an account, or when they want to add a new 2FA factor to their account

Make this changes in Auth controller to create this API.

****

Make **EnrollUserFactors** Api which takes userId and **FactorPayload** .

FactorPayLoad is a models class , which is define as below ,



Now we need to make httpPost Request to EnrollUser method of MfaSaervice,

We calls **EnrollUser()** method which takes this two parameters i.e. userId and factorPayload ,

**EnrollUser() :**

****

The EnrollUser method is used to enroll a user in a two-factor authentication (2FA) factor. It takes two parameters:

* **userId**: The ID of the user to enroll.
* **factorPayload**: A FactorPayload object containing the information about the factor to enroll.

The method first validates the user ID. If it is empty or null, an ArgumentException is thrown.

Next, the method creates a HttpClient object and sets the **Authorization** header with the API key.

The method then constructs the URL for the user factors endpoint. This URL is a combination of the domain name and the API path for the endpoint.

The method then serializes the **factorPayload** object to JSON and creates a **StringContent** object with the serialized JSON.

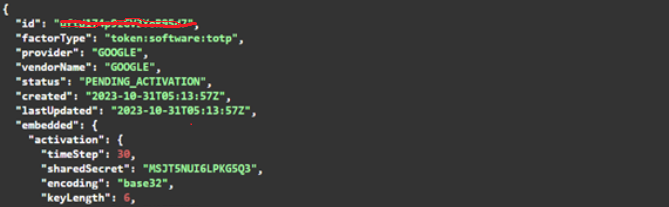
The method then sets the Content property of the **HttpClient** object to the **StringContent** object.

The method then makes a POST request to the user factors endpoint.

If the POST request is successful, the method deserializes the response content to a **FactorEnrollResponse** object and returns it.

If the POST request is not successful, the method deserializes the error response content to an **ErrorResponse** object and throws an **Exception** with the error message.

Output :

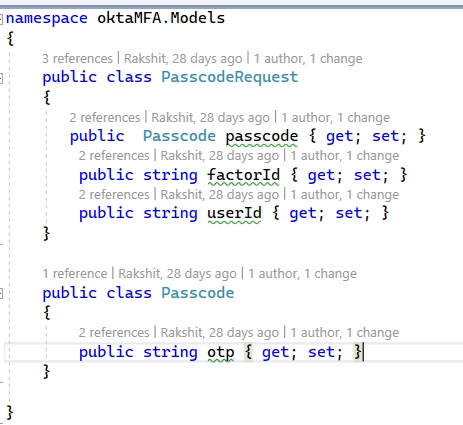


**ActivateFactor :**

The **ActivateFactor** method is a POST endpoint that can be used to activate a one-time password (OTP) factor for a user. The method takes one parameter:

* passcodeRequest: A **PasscodeRequest** object containing the passcode from the user's authenticator app.

PasscodeRequest is class of model :



The method first validates the **passcodeRequest** object. If it is null, a NullReferenceException is thrown.

Next, the method calls the ActivateOtp method on the \_**mfaService** object to activate the OTP factor.

The \_**mfaService** object is responsible for managing the user's MFA factors.

**ActivateOtp :**

The ActivateOtp method is an asynchronous method that activates a one-time password (OTP) factor for a user. It takes a PasscodeRequest object as input and returns a PasscodeResponse object as output.

The ActivateOtp method first validates the PasscodeRequest object to ensure that it is not null and that all of the required parameters are present. If the PasscodeRequest object is invalid, an ArgumentException is thrown.

Next, the ActivateOtp method creates an HttpClient object and sets the Authorization header to the API key. It also sets the Accept header to application/json.

The ActivateOtp method then constructs the URL for the API endpoint that is used to activate OTP factors. The URL is a combination of the domain name, the API path, the user ID, and the factor ID.

The ActivateOtp method then creates a HttpRequestMessage object with the HTTP method POST and the constructed URL. It also sets the Content property of the HttpRequestMessage object to a StringContent object that contains the serialized passcode.

The ActivateOtp method then sends the HttpRequestMessage object to the API service and awaits the response. If the response is successful, the ActivateOtp method deserializes the response content to a PasscodeResponse object and returns it.

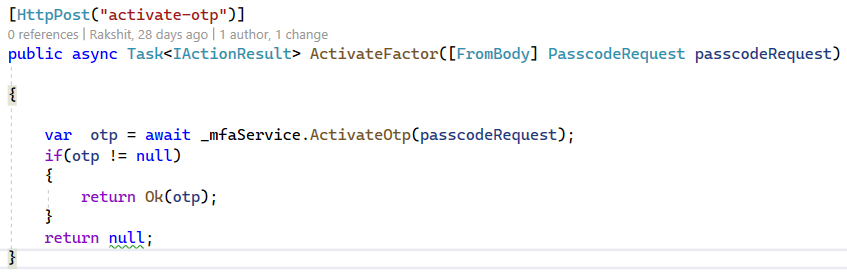
If the response is not successful, the ActivateOtp method deserializes the response content to an ErrorResponse object and throws an ApplicationException with the error message.

The ActivateOtp method also handles the following errors:

* E0000001: The OTP factor is already active.
* Any other error: An ApplicationException is thrown with the error message.



If the **ActivateOtp** method is successful, the ActivateFactor method returns the OTP to the user in an Ok response. Otherwise, the method returns a null response.



**Output :**

A computer screen shot of text

Description automatically generated

Once otp is activated ,If user tried to login primary authentication will happen first then till will navigate user to verify otp page , where user just need to enter otp given in google authenticator app.

**VerifyFactor :**

A screen shot of a computer program

Description automatically generated

1. This method is an HTTP POST endpoint defined by the [HttpPost("verify")] attribute.
2. It calls the **VerifyOtp** method from the **MfaService** to perform the OTP verification.

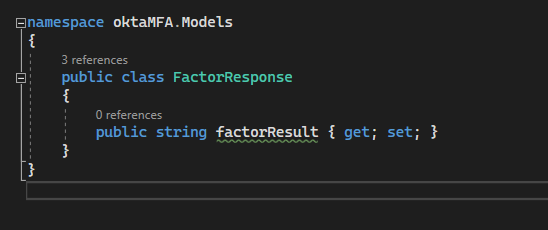
3.If the verification is successful (the result is not null), it returns an OK response with the verification result.

4.If the verification is not successful (result is null), it returns null (this might need further handling in your specific use case).

5.It catches HttpRequestException specifically, which might occur if there's an issue with the HTTP request (e.g., network issues).

For any other exception, a generic exception catch block is included, returning a BadRequest response with the error message.

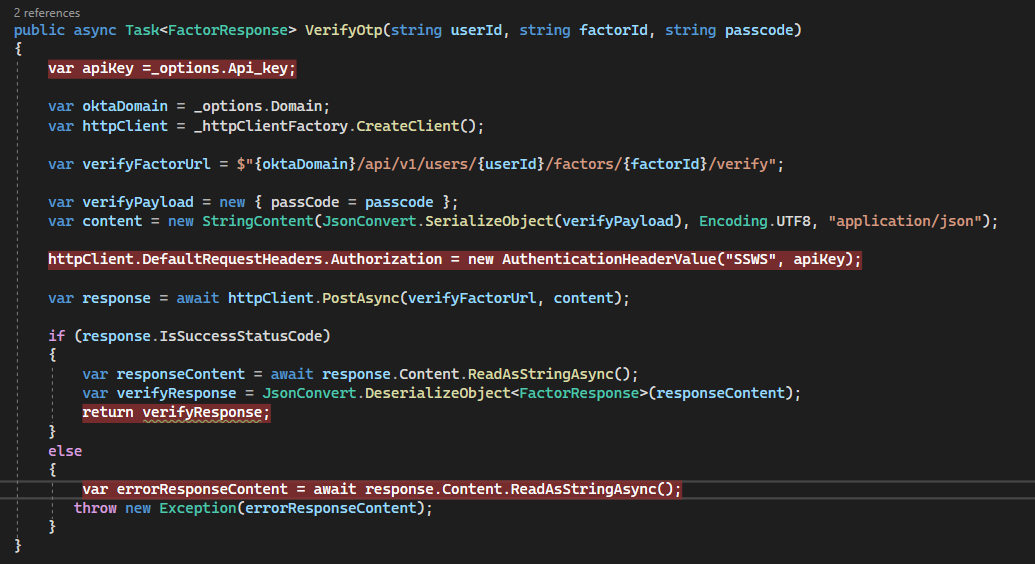
Create a other class in model folder and name it as **FactorResponse** :



This class represents the expected structure of the response from the Okta API.

It has a single property factorResult, which likely contains the result of the OTP verification.

**VerifyOtp :**



1.The method starts by retrieving the Okta API key and domain from the \_options object.

It creates an HttpClient to make HTTP requests.

2.The URL for verifying the OTP is constructed using the provided userId and factorId.

3.The passcode is encapsulated in a payload (verifyPayload), which is then converted to JSON and wrapped in a StringContent object.

4.The Okta API key is set in the authorization header of the HTTP client.

5.POST Request:

A POST request is made to the Okta endpoint for verifying the OTP (**verifyFactorUrl**), with the passcode payload.

Response Handling:

If the request is successful (status code 200), the response content is deserialized into a **FactorResponse** object using JSON.NET (JsonConvert).

If the request is unsuccessful, an exception is thrown, including the error message from the response content.

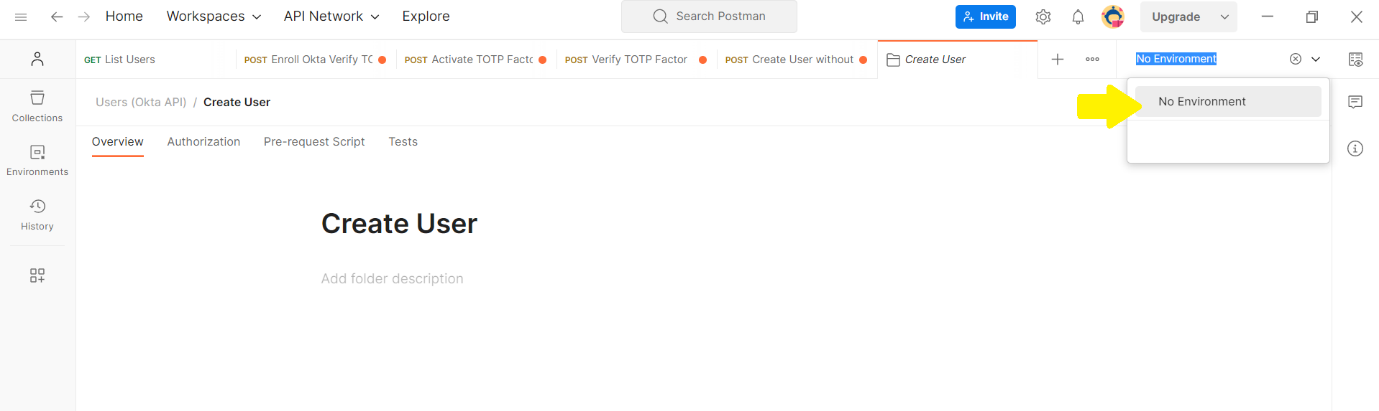
**Final Output :**

A screenshot of a computer

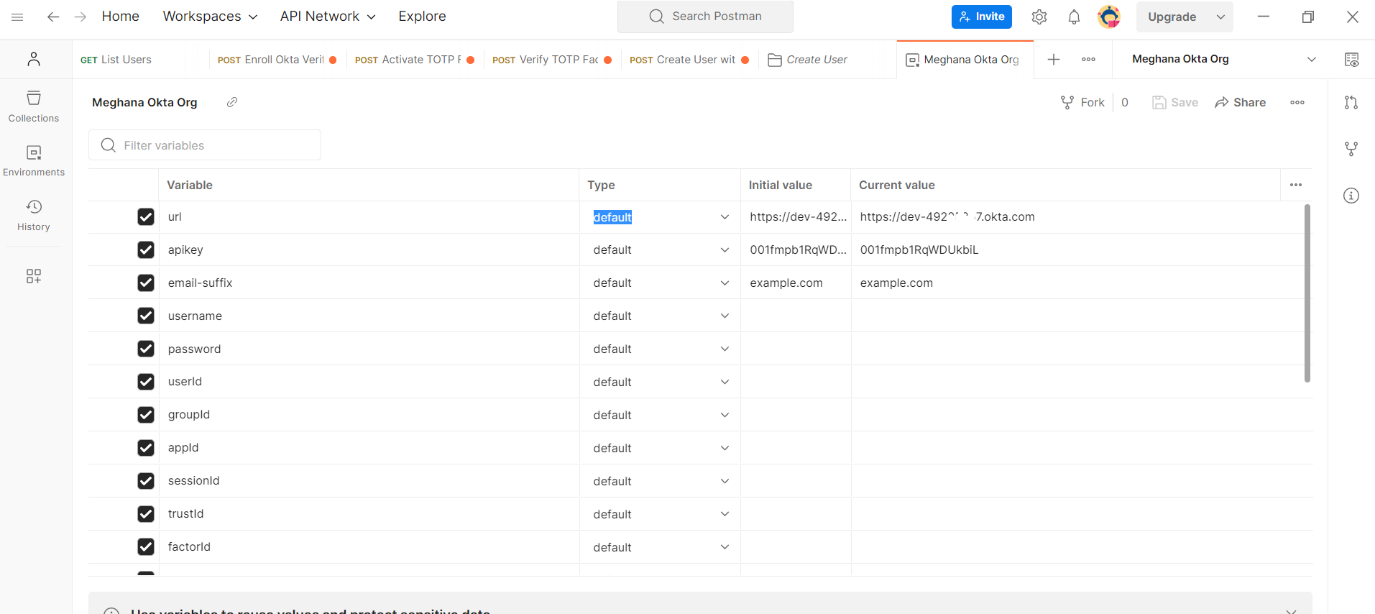
Description automatically generated

The code you see above was generated based on the response we received when we configured Postman to make an API request. The steps for configuring Postman are guided below

* **Configuring Postman to make API request.**
* Start the Postman if it's not open already.
* In the upper-left corner, click **Import**.
* In the **Import** dialog box, click **Link** and then paste the following link into the **Enter the URL** box: <https://developer.okta.com/docs/api/postman/example.oktapreview.com.environment>.
* Click **Continue** and then **Import** to confirm your environment import.
* In the upper-right corner, click the box that lists the environments and then select ${yourOktaDomain} from the list. The initial value is No Environment if you just installed Postman.



* In the upper-right corner, next to ${yourOktaDomain}, click **Environment quick look** Postman environment quick look button.
* In the upper-right corner of the ${yourOktaDomain} dialog box, click **Edit**.
* In the **Manage Environments** dialog box, do the following:
  + In the **Environment Name** box, delete the placeholder text and name your environment, for example: Person1 Okta Org.
  + For the url variable, in the **Initial Value** and **Current Value** columns, replace the placeholder text with your org's full URL, for example: https://dev-1234567.okta.com.
  + Remember to remove the -admin part of your subdomain.
  + For the apikey variable, in the **Initial Value** and **Current Value** columns, enter your OKTA API token that you created earlier, for example: 00LzMWxMq\_0sdErHy9Jf1sijEGexYZlsdGr9a4QjkS.



* Scroll to the bottom of the dialog box and click **Update**.
* To close the dialog box, click the **X** in the upper-right corner.
* **Test the Okta REST APIs using Postman.**
* List Users
  + Create a new request and select **Get** method, type the Url : {{url}}/api/v1/users?limit=25
  + Under Params tab add:

|  |  |
| --- | --- |
| Key | Value |
| limit | 25 |

* + Under Headers tab

|  |  |
| --- | --- |
| KEY | VALUE |
| Accept | application/json |
| Content-Type | application/json |
| Authorization | SSWS{{apikey}} |

* + Click on **Send** and you will be able to see list of users as response.

|  |
| --- |
| [      {          "id": "00uak4uad9sHWKPq55d7",          "status": "ACTIVE",          "created": "2023-07-28T10:06:30.000Z",          "activated": "2023-07-28T10:06:43.000Z",          "statusChanged": "2023-07-28T10:07:51.000Z",          "lastLogin": "2023-07-28T10:07:51.000Z",          "lastUpdated": "2023-07-28T10:07:51.000Z",          "passwordChanged": "2023-07-28T10:07:51.000Z",          "type": {              "id": "otyag7qui0t7J4F0j5d7"          },          "profile": {              "firstName": "Sandhya",              "lastName": "R",              "mobilePhone": **null**,              "secondEmail": **null**,              "login": "sandhyarakshit222@gmail.com",              "email": "sandhyarakshit222@gmail.com"          },          "credentials": {              "password": {},              "emails": [                  {                      "value": "sandhyarakshit222@gmail.com",                      "status": "VERIFIED",                      "type": "PRIMARY"                  }              ],              "provider": {                  "type": "OKTA",                  "name": "OKTA"              }          },          "\_links": {              "self": {                  "href": "https://dev-49225257.okta.com/api/v1/users/00uak4uad9sHWKPq55d7"              }          }      },      {          "id": "00uag7qumtKc794A45d7",          "status": "RECOVERY",          "created": "2023-07-20T05:40:19.000Z",          "activated": **null**,          "statusChanged": "2023-07-20T07:31:28.000Z",          "lastLogin": "2023-08-10T12:03:35.000Z",          "lastUpdated": "2023-07-20T13:09:54.000Z",          "passwordChanged": "2023-07-20T07:31:31.000Z",          "type": {              "id": "otyag7qui0t7J4F0j5d7"          },          "profile": {              "firstName": "meghana\_g",              "lastName": "megha46",              "mobilePhone": **null**,              "secondEmail": **null**,              "login": "megha46@github.oktaidp",              "email": "meghana4698@gmail.com"          },          "credentials": {              "password": {},              "emails": [                  {                      "value": "meghana4698@gmail.com",                      "status": "VERIFIED",                      "type": "PRIMARY"                  }              ],              "recovery\_question": {                  "question": "What is the toy/stuffed animal you liked the most as a kid?"              },              "provider": {                  "type": "OKTA",                  "name": "OKTA"              }          },          "\_links": {              "self": {                  "href": "https://dev-49225257.okta.com/api/v1/users/00uag7qumtKc794A45d7"              }          }      }  ] |

* **Create a test user**
  + Create a new request and select a **POST** method add the Url: {{url}}/api/v1/users?activate=false
  + In Params tab add

|  |  |
| --- | --- |
| Key | Value |
| activate | False |

* + In Body tab add the below code:

{

"profile": {

"firstName": "name",

"lastName": "lastanme",

"email": "email-prefix@*{{email-suffix}}*",

"login": "email-prefix@*{{email-suffix}}*"

}

}

* + Click **Send**. A successful request results in an HTTP status code of 200 and a JSON payload response.
  + Save the value of the User id that is returned in the response.
* **Enroll a factor:**
  + Create a new request Select a **POST** method and add URl: {{url}}/api/v1/users/userId/factors
  + Under Headers tab

|  |  |
| --- | --- |
| KEY | VALUE |
| Accept | application/json |
| Content-Type | application/json |
| Authorization | SSWS{{apikey}} |

* + In **Body** tab add the below code:

{

  "factorType": "token:software:totp",

  "provider": "OKTA"

}

* + Click **Send** to add the Google Authenticator Factor for your user. A successful request results in an HTTP status code of 200 and a JSON payload response.
  + Save the Factor ID value (id) from the response.
  + In the \\_embedded object located at the bottom of the response, locate the \_links object and then the href value of the qrcode property.
  + Copy this URL and open it in a new tab of your browser. A QR code appears.
  + Download Google Authenticator app in mobile and scan that QR code in that app.
  + After scanning the QR code in Google Authenticator app we will get a six-digit passcode.
* **Activate the factor**
  + Create a new request select a **POST** method and add the Url: {{url}}/api/v1/users/userID/factors/factorId/lifecycle/activate
  + Replace the ${userId} and ${factorId} variables with the User ID and Factor ID values that you copied previously.
  + Under Headers tab

|  |  |
| --- | --- |
| Key | Value |
| Accept | application/json |
| Content-Type | application/json |
| Authorization | SSWS{{apikey}} |

* + In **Body** tab add the below code:

{

  "passCode": "six-digits code in Goggle authenticator app"

}

* + Click **Send**. A successful request results in an HTTP status code of 200 and a JSON payload response.
* **Verify the factor**
  + Create a new request select a **POST** method and add the Url: {{url}}/api/v1/users/userID/factors/factorId/lifecycle/activate
  + Replace the ${userId} and ${factorId} variables with the User ID and Factor ID values that you copied previously.
  + Select the **Body** tab, and in the JSON body of the request, replace the passCode value with the passcode shown in the Google Authenticator app.
  + Click **Send**. A successful verification of a token results in an HTTP status code of 200 with a JSON payload that contains the key factorResult with the value of SUCCESS.
  + Unsuccessful verification attempts result in an HTTP status code of 403 with a JSON payload that contains the key errorCode.