
Monzo API

Release 0.3.0

Peter McDonald

Apr 15, 2023

CONTENTS:

1	Disclaimer	3
1.1	Tutorials	3
1.2	monzo	17
1.3	API Observations	19
1.4	Developer Guide	19
1.5	TODO	21
1.6	Change Log	22
1.7	LICENSE	24

Many banks now provide an API to allow you too obtain data from your account by utilising an API. This access can give you great insights into your financials.

The Monzo API package helps simplify usage of the API.

DISCLAIMER

As per Monzo's documentation:

The Monzo Developer API is not suitable for building public applications.

You may only connect to your own account or those of a small set of users you explicitly whitelist. Please read our [blog post](#) for more detail.

Please do not use this package on a public server. This should only be used on a private server.

1.1 Tutorials

Taking your first steps using any package can be daunting, here we will help you get started quickly.

1.1.1 Monzo Setup

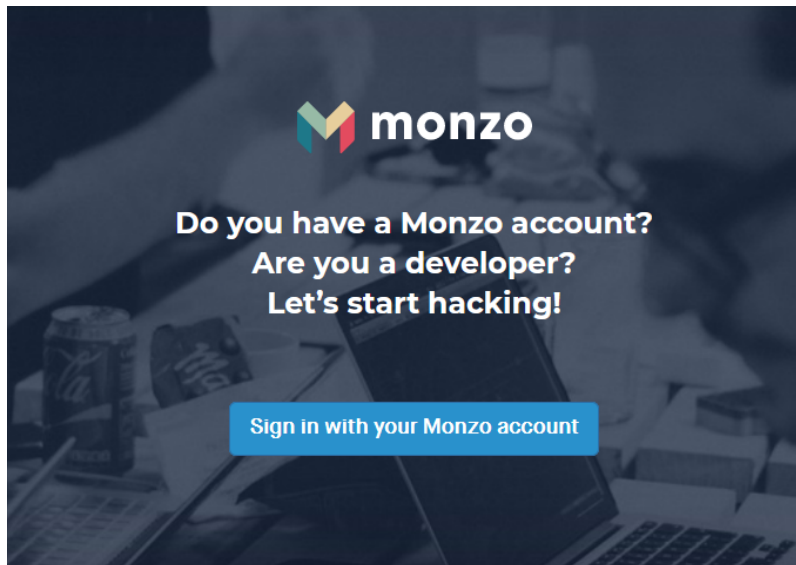
As with any restricted resource on the internet, you require credentials to login. The Monzo API is no different.

The following tutorial will guide you through creating credentials for your account.

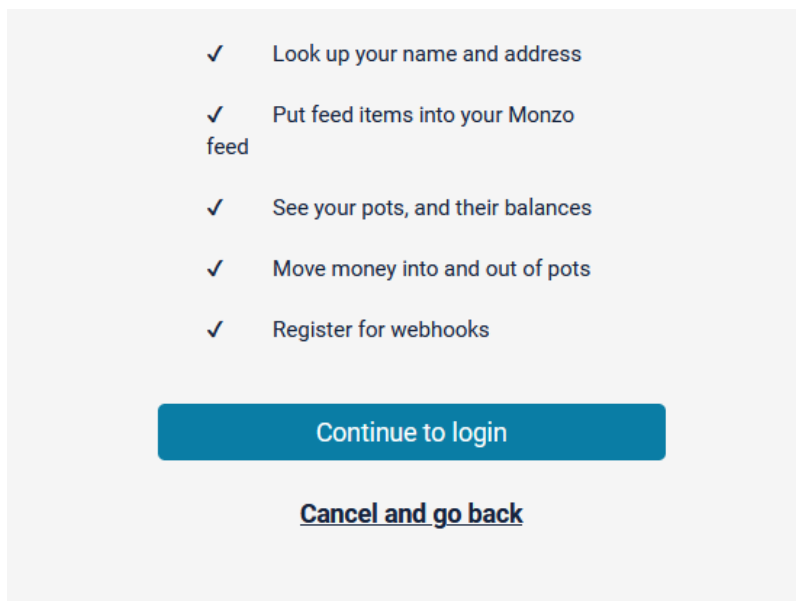
Logging In

To start with we have to log into the Monzo developer system, this can be found at <https://developers.monzo.com/>.

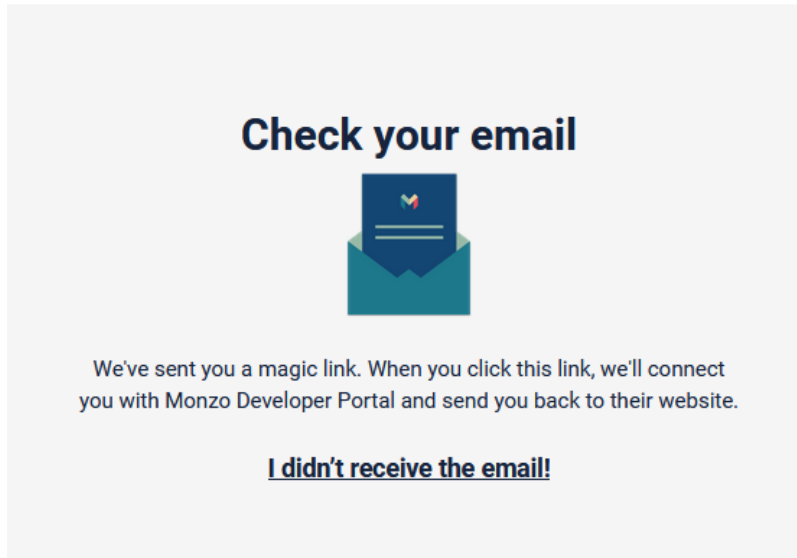
On visiting this page you will find a "Sign in with your Monzo Account"



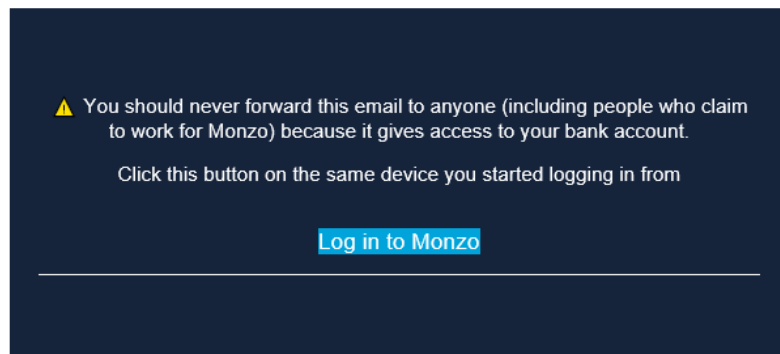
Once you have clicked on this link you will be presented with another page inviting you to log in. On this page click on “Continue to login”



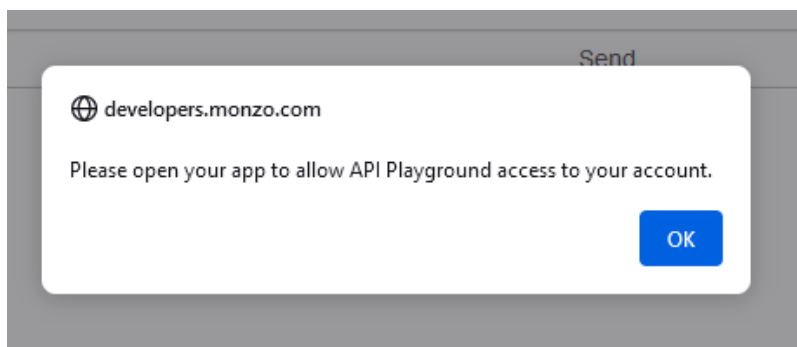
You will now be asked to enter your email address, you need to enter the email address associated with your account. Once entered click “Submit” and you will be presented with the following page:



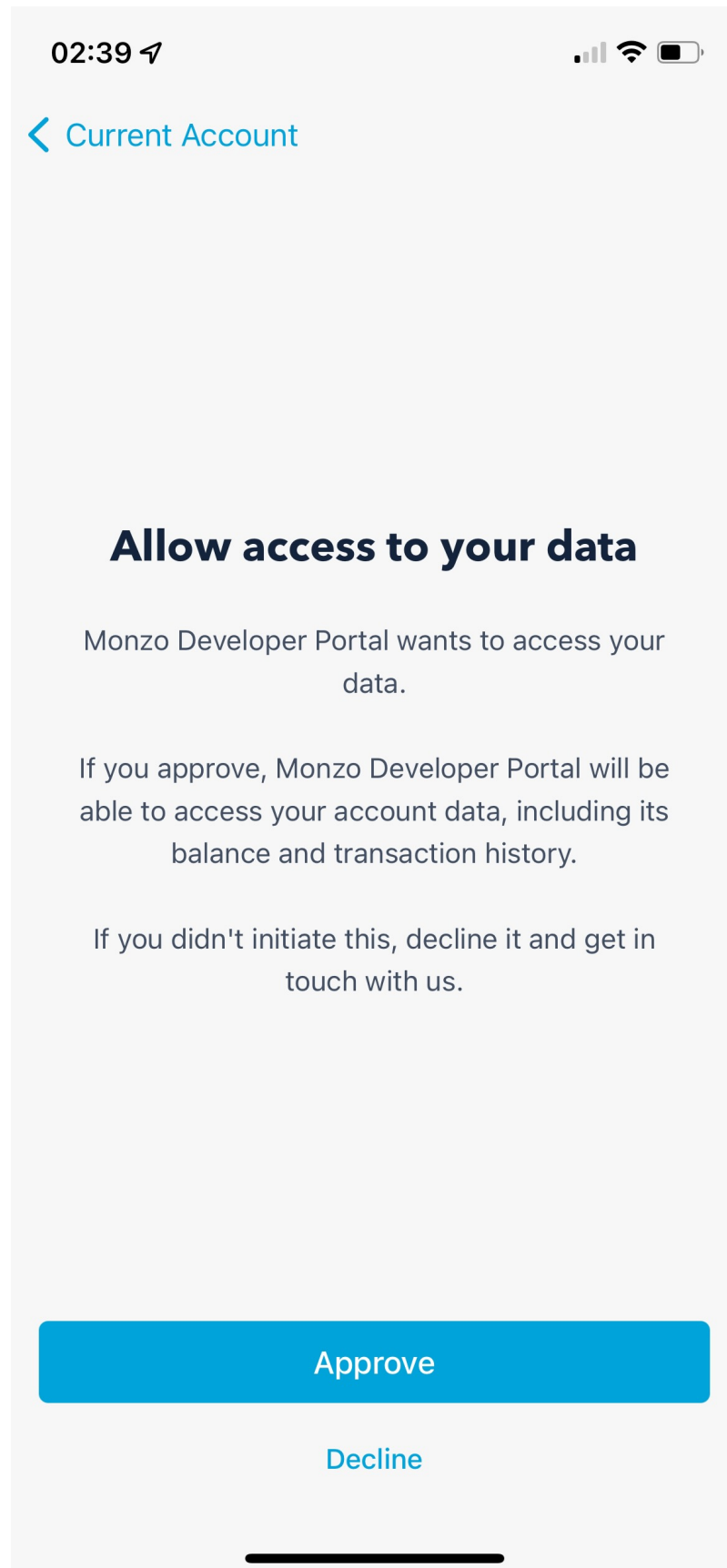
It is now time to check your email, you should have received an email containing:



You should click on the “Log Into Monzo” link. This will now take you back to <https://developers.monzo.com> but you will now be logged in. HOWEVER you will receive an alert on screen similar to the following:



Although we have successfully logged in, we have not authorised the token this process to have full access to your account. To complete authorisation we now need to go to the Monzo app where you will find an alert, clicking on the alert will display a message such as the following:

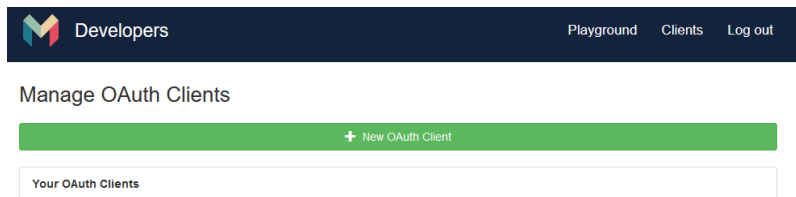


To be able to use the developer interface properly you will need to select approve.

You have now successfully logged into the Monzo developer forum and granted the relevant permissions. We can now move onto creating an API client.

Creating A Client

Now that we are fully logged in we can create the API client. To start with click the client link in the top right hand side. You should now see a list of existing clients, albeit it an empty list.



To get started you can click on “+ New OAuth Client”. This will open a form allowing us to create a new client.

Manage OAuth Clients

+ New OAuth Client

Your OAuth Clients

New OAuth Client

Details

Name

Logo URL

Redirect URLs

Description

Confidentiality

As you can see there are a number of fields that we can enter details into.

Name

You can enter any name here, it would be best to give it a name that identifies the project it is being used on as you can have multiple clients.

Logo URL

This can be left blank

Redirect URL's

The redirect URL's are used during the authorisation phase of using the API. You will need to have a webserver listening on the address to capture the get parameters (we can work around this if you don't have one).

Description

You can enter any description here.

Confidentiality

Unless you are only going to utilise the API for a quick task you should select “Confidential”, this will allow a refresh token to be provided. Under normal circumstances a token will expire, a refresh token will allow access to be renewed.

Now that we have completed the form you can click “Submit”. Once submitted you will be presented with a client list again, this time however there will be one entry:

Manage OAuth Clients


[+ New OAuth Client](#)

Your OAuth Clients
Test

If you now click on the new entry you will be presented with details such as:

Overview

Name	Test
Client ID	oauth2client_
Owner ID	user_
Created at	25/09/2021, 02:43:50
Confidential	true
Status	NEW

 Revoke Client

Client secret:

mnzconf,

From this page we need to take three pieces of information (the values in the above screenshot has been redacted, make sure you also keep yours secret):

- Client ID
- Owner ID
- Client Secret

You have now successfully created an OAuth client that you can use with the API. Keep the above details handy and you can now go and look at some of the other tutorials.

1.1.2 Generating An Access Token

If you have followed along with the tutorials you will already have access to Client ID, Owner ID and Client Secret. Although these help us access the Monzo API they are not actually credentials that will work when querying the API, rather they give us the ability to get the tokens we can use.

Obtaining An Access Token

Converting our Client ID and Client Secret to an access token is a two step process, the first step will feel familiar, it is similar to the process we'd use to obtain access to the developer area.

Creating A Request URL

Our first step is that we need to generate a URL that you can use to approve API access. Luckily we have thought of that and have a method to do this.

The code I am going to use here is available in the examples folder as `auth_step_01.py`.

```
"""Example code to authenticate against the Monzo API."""
from monzo.authentication import Authentication

client_id = '' # Client ID obtained when creating Monzo client
client_secret = '' # Client secret obtained when creating Monzo client
redirect_uri = 'http://127.0.0.1/monzo' # URL requests via Monzo will be redirected in,
↳ a browser

monzo = Authentication(client_id=client_id, client_secret=client_secret, redirect_
↳ url=redirect_uri)

# The user should visit this url
print(monzo.authentication_url)
```

In the above code you need to assign the Client ID and Client Secret and then execute the script. This will output a URL similar to:

```
https://auth.monzo.com?client_id=oauth2client_OBFESCATED&redirect_uri=http://127.0.0.1/monzo&response_
type=code&state=e7aF6mtU6MFNovkxUGfCsic6Kt7GUCIBQWi0KkZY1YJuULK2QVEEujcfkLnF2Jxh
```

So that we know what is going on here let's break this URL down a bit:

Monzo Authentication URL

Value: <https://auth.monzo.com> Description: This is the URI Monzo uses for authentication

Client ID

Value: `oauth2client_OBFESCATED` Description: This will match the Client ID that you received from Monzo when creating a client and entered into the above script.

redirect_uri

Value: <http://127.0.0.1/monzo> Description: As you will see shortly the authentication process redirects you to a URL, this is the URL it will redirect too (more on this later). This must match the URI you entered when creating the client in the Monzo developer site. If you need to change this you can login and edit the client.

response_type

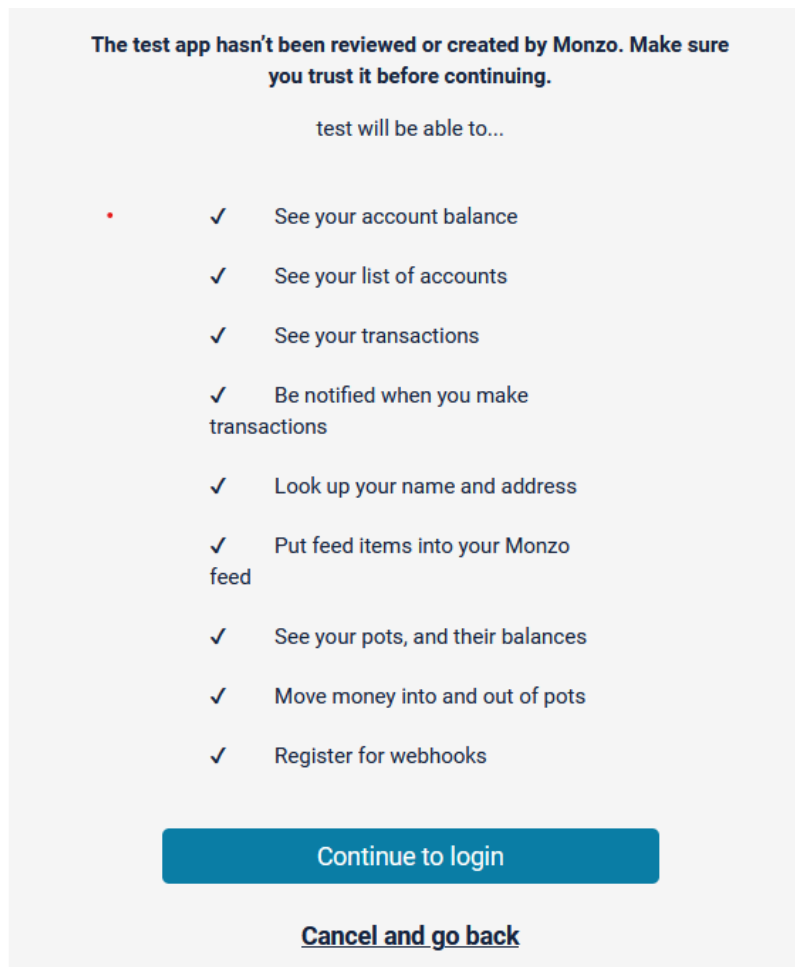
Value: `code` Description: This dictates the type of authentication that the API will use, Monzo only supports code.

state

Value: `e7aF6mtU6MFNovkxUGfCsic6Kt7GUCIBQWi0KkZY1YJuULK2QVEEujcfkLnF2Jxh` Description: This is a random string that is intended to ensure when Monzo redirects you that the request is valid. This is created automatically therefore you can ignore this.

Authenticating The API

Now that we have the URL you can safely click on it. The process here will look familiar:



The test app hasn't been reviewed or created by Monzo. Make sure you trust it before continuing.

test will be able to...

- ✓ See your account balance
- ✓ See your list of accounts
- ✓ See your transactions
- ✓ Be notified when you make transactions
- ✓ Look up your name and address
- ✓ Put feed items into your Monzo feed
- ✓ See your pots, and their balances
- ✓ Move money into and out of pots
- ✓ Register for webhooks

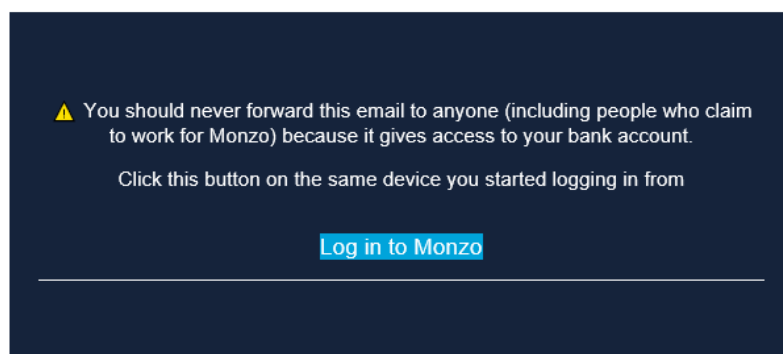
[Continue to login](#)

[Cancel and go back](#)

This will look identical to a page we visited when creating the client (the developer site uses the API just as we are). There is one thing to note, in the summary at the top it specifies the name of the client that you are granting access.

Once you are happy click on “Continue To Login”, enter your email address and click on “Submit”.

You will now receive an email like the following:



⚠ You should never forward this email to anyone (including people who claim to work for Monzo) because it gives access to your bank account.

Click this button on the same device you started logging in from

[Log in to Monzo](#)

Click on “Log In To Monzo”. Once your browser opens you will likely find that you go to a page that does not exist (the redirect URL that you specified above) but with some extra parameters. Take a copy of the URL in your browser.

Lets take a closer look at the URL:

Redirect URL

Value: <https://127.0.0.1/monzo> Description: This is the redirect URL that was specified when creating the client and entered in the script when creating the URL.

code

Value: REDACTED Description: This is an authorization code that our script can use to obtain an access token. I have redacted the value, this should be kept private.

state

Value: e7aF6mtU6MFNovkxUGfCsic6Kt7GUCIBQWi0KkZY1YJuULK2QVEEujcfkLnF2Jxh Description: This will match the random string that we saw in the URL that you clicked on a moment ago, this is single use and proves the redirect URL was in response to your request to authorise.

Retrieving An Access Token

We are almost there. We have all the details we need to create an access token. As you would expect, we have an example for that:

```
"""Code to handle the seconds stage of authentication."""
from monzo.authentication import Authentication
from monzo.exceptions import MonzoAuthenticationError, MonzoServerError

client_id = '' # Client ID obtained when creating Monzo client
client_secret = '' # Client secret obtained when creating Monzo client
redirect_uri = 'http://127.0.0.1/monzo' # URL requests via Monzo will be redirected in
↳ a browser
state = '' # State random string created when creating the Monzo URL (generated in step
↳ 1 and appended to the URL)
code = '' # Authorization code from Monzo (this will be in the redirected URL after
↳ clicking the link from step 1)

monzo = Authentication(client_id=client_id, client_secret=client_secret, redirect_
↳ url=redirect_uri)
try:
    monzo.authenticate(authorization_token=code, state_token=state)
except MonzoAuthenticationError:
    print('State code does not match')
    exit(1)
except MonzoServerError:
    print('Monzo Server Error')
    exit(1)

# The following 3 items should be stored for future requests
print(f"access_token = '{monzo.access_token}'")
print(f'expiry = {monzo.access_token_expiry}')
print(f"refresh_token = '{monzo.refresh_token}'")

# Now authorise access in the Monzo app
```

As you can see from the code above, there is a number of variables we need to populate:

client_id: The Client ID we obtained from the Monzo developer area client_secret: The rClient Secret we obtained from the Monzo developer area redirect_uri: The redirect URL we entered into the Monzo developer area state: This is the random string that we created in step one and should match the state value in the response code: The authorisation code in the redirected URL

Once you have populated these you can execute the python script. This will result in three print statements outputting text:

```
access_token = 'REDACTED' expiry = 1632680343 refresh_token = 'REDACTED'
```

You should take a copy of these values as you will need them for any API call.

Just a quick note on what these are:

access_token

This is the token that is used to make any API calls.

expiry

When creating an access token, to limit any damage that can be done from this being exposed, it has a limited lifespan before it expires. This lifetime is dictated in seconds.

To make things easier this package converts it into the unix timestamp that the token will expire.

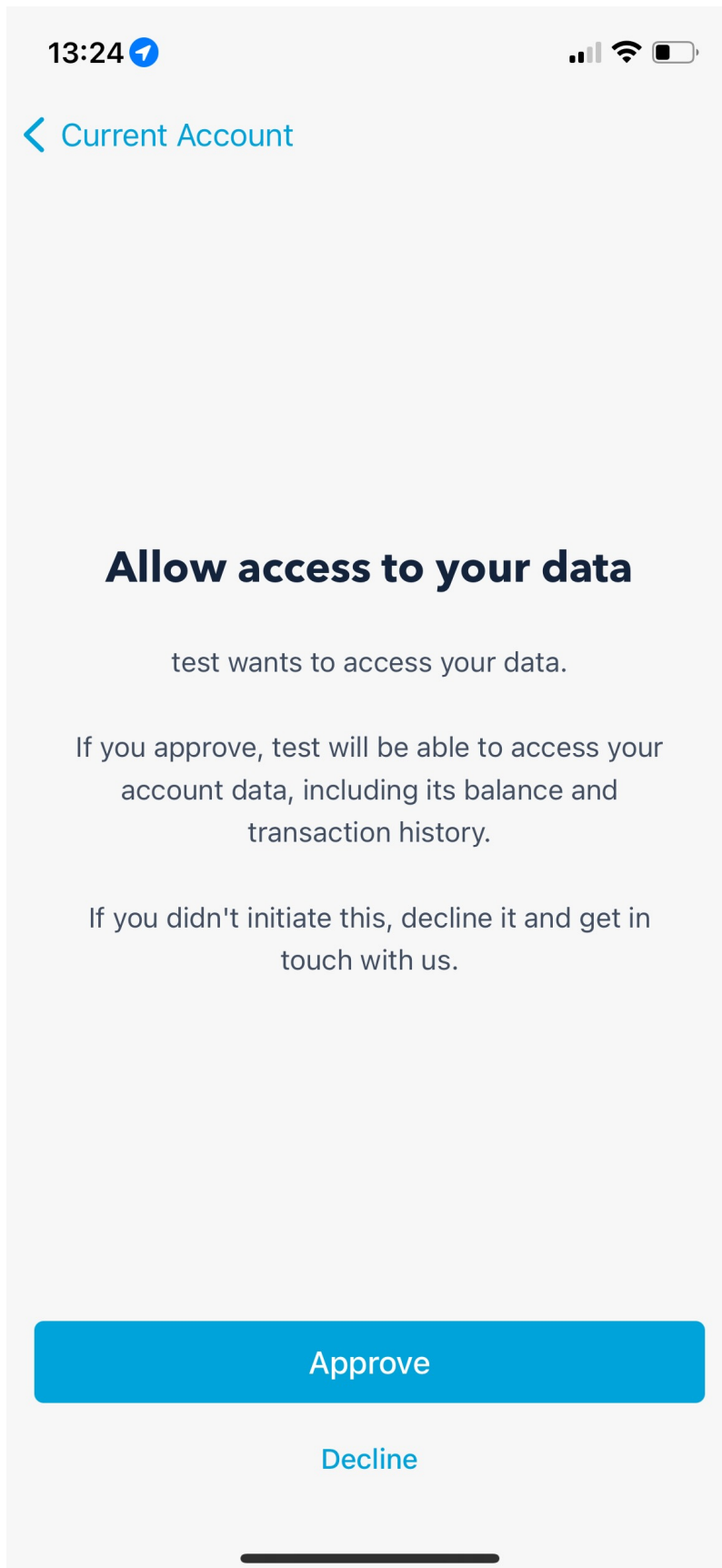
refresh_token

If you specified “Confidential” when creating the client in the Monzo developer area then you will have received a refresh token. This token can be used to generate a new access token without following this complete process again.

This does not extend the life of the token, instead a new token is created with a new expiry.

There is one further step that we need to take. Although the access token can be used to authenticate for your account and can login to the API, it is not currently authorised to carry out many requests. After using the second example your Monzo app will have triggered an alert.

Open the Monzo app and you will see something like:



Click on “Approve”

Congratulations, you now have credentials that you can use to query the API.

1.1.3 Basic API Calls

All API calls in the package take the same form, therefore following this tutorial should help you understand how the package works.

For this tutorial we are going to select all accounts that we have in Monzo.

Obtaining Our Account List

This tutorial will work through the following script:

```
"""Example code to fetch accounts."""
from monzo.authentication import Authentication
from monzo.endpoints.account import Account
from monzo.exceptions import MonzoError

client_id = '' # Client ID obtained when creating Monzo client
client_secret = '' # Client secret obtained when creating Monzo client
redirect_uri = 'http://127.0.0.1/monzo' # URL requests via Monzo will be redirected in,
↳ a browser
access_token = ''
expiry = 0
refresh_token = ''

monzo = Authentication(
    client_id=client_id,
    client_secret=client_secret,
    redirect_url=redirect_uri,
    access_token=access_token,
    access_token_expiry=expiry,
    refresh_token=refresh_token
)

try:
    accounts = Account.fetch(monzo)
    for account in accounts:
        _ = account.balance
    print(len(accounts))
except MonzoError:
    print('Failed to retrieve accounts')
```

As before the script requires several pieces of information to start with. We obtained each of these in the previous tutorials.

Client ID, Client Secret and Redirect URL are as we obtained from the Monzo developer site.

```
monzo = Authentication(
    client_id=client_id,
    client_secret=client_secret,
```

(continues on next page)

(continued from previous page)

```

    redirect_url=redirect_uri,
    access_token=access_token,
    access_token_expiry=expiry,
    refresh_token=refresh_token
)

```

Access Token, Expiry and Refresh Token were obtained in the “Generating An Access Token” (handily in a format you could copy and paste overwriting the variables in the above script). In a real world application these would be stored in a database or a state file.

We now create an Authentication object using the variables above. This object is responsible for authentication of the API, making requests and refreshing access. This object is passed about to almost any other object we create.

We can now carry out a fetch of the accounts that we have with Monzo:

```
accounts = Account.fetch(monzo)
```

As you can see we have called a class method, with this package you should not need to instantiate objects yourself (apart from the Authentication object).

Each endpoint that you may call has a class in the endpoints folder, each of these contain either a fetch or fetchone class method that will carry out a query and return a list of objects matching your query.

You now have a list of accounts that you can inspect and work with.

1.1.4 Refreshing Tokens

In a previous tutorial we spoke about the fact that tokens will expire.

Although the Authentication class has a refresh_token method you can use there should be no reason to do so. The package refreshes tokens when it is required automatically.

This raises the question how do you know if a token has been refreshed.

Identifying Refreshed Tokens

There are two ways that you can identify if a token has been refreshed.

Checking The Current Token

Unfortunately you cannot rely on a token not having expired when identifying if a token will be replaced or not, there can be instances when a valid token needs to be replaced.

Therefore the first method, although far from ideal, is to fetch the current token after you have carried out a request comparing it to the token you already have.

An Authentication object contains three properties to fetch the current details:

```

from monzo.authentication import Authentication

monzo = Authentication(
    client_id=client_id,
    client_secret=client_secret,
    redirect_url=redirect_uri,

```

(continues on next page)

(continued from previous page)

```
        access_token=access_token,
        access_token_expiry=expiry,
        refresh_token=refresh_token
    )

    # Carry out some API calls

    print(monzo.access_token)
    print(monzo.access_token_expiry)
    print(monzo.access_token_expiry)
```

You should then store these for the next time you wish to carry out API calls.

Registering A Handler

As mentioned, the above method is not convenient at all, a more convenient method would be if for example the package just told you that those details changed.

This is where handlers come in.

```
from monzo.authentication import Authentication
from monzo.endpoints.echo import Echo

monzo = Authentication(
    client_id=client_id,
    client_secret=client_secret,
    redirect_url=redirect_uri,
    access_token=access_token,
    access_token_expiry=expiry,
    refresh_token=refresh_token
)

# Instantiate our handler
handler = Echo()

#Register the handler
monzo.register_callback_handler(handler)

# Carry out API calls
```

Now whenever the package needs to fetch new tokens the store method of the handler will be called.

This package currently contains two handlers:

Echo

This handler simply prints the new token details.

FileSystem

This handler stores the details on the file system in a location you specify when instantiating the handler.

The file system handler also implements a Fetch method allowing you to retrieve the details from the file.

Implementing A Handler

It is not beyond the realm of reality that you may wish to store these details in a database or using some other mechanism.

To achieve this you should create (or modify) a class that extends from the Storage abstract base class. At present this simply dictates that you must have a store method and the parameters that must be present.

Reading the Echo handler code should clarify how this works

```

"""Class to echo credentials."""
from monzo.handlers.storage import Storage

class Echo(Storage):
    """Class that will echo out credentials."""

    def store(
        self,
        access_token: str,
        client_id: str,
        client_secret: str,
        expiry: int,
        refresh_token: str = ''
    ) -> None:
        """
        Echo the Monzo credentials.

        Args:
            access_token: New access token
            client_id: Monzo client ID
            client_secret: Monzo client secret
            expiry: Access token expiry as a unix timestamp
            refresh_token: Refresh token that can be used to renew an access token
        """
        print(f"client_id = '{client_id}'")
        print(f"client_secret = '{client_secret}'")
        print(f"access_token = '{access_token}'")
        print(f"expiry = {expiry}")
        print(f"refresh_token = '{refresh_token}'")

```

1.2 monzo

1.2.1 monzo package

Subpackages

monzo.endpoints package

Submodules

`monzo.endpoints.account` module

`monzo.endpoints.attachment` module

`monzo.endpoints.balance` module

`monzo.endpoints.feed_item` module

`monzo.endpoints.monzo` module

`monzo.endpoints.pot` module

`monzo.endpoints.receipt` module

`monzo.endpoints.transaction` module

`monzo.endpoints.webhooks` module

`monzo.endpoints.whoami` module

Module contents

`monzo.handlers` package

Submodules

`monzo.handlers.echo` module

`monzo.handlers.filesystem` module

`monzo.handlers.storage` module

Module contents

Submodules

`monzo.authentication` module

`monzo.exceptions` module

`monzo.helpers` module

`monzo.httpio` module

Module contents

1.3 API Observations

While working with the API I often come across peculiarities with the API. These will be posted here.

1.3.1 Transactions

Although the accounts provide all the different types including:

- Current Account
- Loan
- Flex
- Flex Backing Loan

Only current account and Flex actually have the ability to return transactions.

The API call to fetch transactions will work but return no transactions.

The API call for Flex Backing Loan will result in a 403 Forbidden HTTP response from the API.

Monzo support have confirmed these account types are not supported by the transaction endpoint.

1.3.2 Transaction Merchants

As part of the returned data you are provided with the merchant ID. This can be expanded to include further information. Merchant information is not present on the following type of transactions:

- Bank transfers
- Transfers between pots and the main account
- Loan payments to Monzo
- Interest payments to Monzo

There may be others.

1.4 Developer Guide

Many banks now provide an API to allow you too obtain data from your account by utilising an API. This access can give you great insights into your financials.

The Monzo API package helps simplify usage of the API.

1.4.1 CICD and Code Standards

We have attempted to reduce the work required to ensure the code conforms to our coding standards. You can help ensure that any code changes will pass CICD by making use of pre-commit. The following steps will set this up for you:

```
pip install pre-commit
pre-commit install
pre-commit run --all-file
```

The above commands will create pre-commit hooks, this will test the code prior to code being committed by Git. Some of the tasks will even correct the data instead of throwing an error.

If you would like to run the checks without committing code you can run the following command:

```
pre-commit run --all-file
```

1.4.2 Building Documentation

Unless testing you should have no need to build the docs, ReadTheDocs does this automatically, however, if you find you need to build the documentation from source the following steps can be taken:

```
cd docs
pip install -e .[docs]
sphinx-build -b html source/ build/html
```

1.4.3 Tagging

Tagging should only take place once a feature branch has been merged. The tag should match the version that can be found in setup.cfg

To create and push a tag the following steps should be taken replacing x.x.x with the version in setup.cfg:

```
git checkout main
git pull
git tag -a vx.x.x -m "vx.x.x SHORT MESSAGE"
git push origin vx.x.x
```

1.4.4 Distributing Package

Distributing the package should no longer be required. Github actions automatically upload the generated .tar.gz and .whl files.

Prior to being able to upload a package to Pypi you first need to create an API key, once obtained create a file called .pyirc in %homepath% with the following details, replacing API_TOKEN with the real API token.

```
[pypi]
username = __token__
password = API_TOKEN
```

You can now run the following to upload the package.


```
git checkout main
git pull
pip install -e .[build]
twine upload dist/*
```

1.5 TODO

1.5.1 Implemented End Points

Table 1: Monzo Endpoints

End Point	Implemented	Version	Tests Written
Authentication	yes	0.0.1	No
Refresh Access	yes	0.0.1	No
Whoami	yes	0.0.1	Yes
Logout	yes	0.0.1	Yes
List Accounts	yes	0.0.2	Yes
Read Balance	yes	0.0.2	Yes
List Pots	yes	0.0.3	No
Deposit Into Pot	yes	0.0.3	No
Withdraw From Pot	yes	0.0.3	No
Get Transaction	yes	0.0.4	No (Testing Fails)
Get Transaction List	yes	0.0.4	Yes
Annotate Transaction	yes	0.1.1 *	Yes
Create Feed Item	yes	0.0.2	Yes
Upload Attachment			No
Register Attachment			No
Deregister Attachment			No
Fetch Receipt	yes	0.1.2	Yes
Create Receipt	yes	0.1.2	Yes
Delete Receipt	yes	0.1.2	No Functionality currently broken
Register Webhook	yes	0.0.3	Yes
List Webhooks	yes	0.0.3	Yes
Delete Webhooks	yes	0.0.3	Yes
Open Banking API		**	No

- * Annotations only appear to work for existing keys such as Notes [view on the forum](#).
- ** It is unlikely that this package will implement usage of the Open Banking API due to restrictions accessing it.

1.5.2 Miscellaneous

- Tidy exceptions and ensure scenarios are captured correctly
- Implement testing
- Facilitate receiving webhook calls
- Enhancing the test server for development
- Improve documentation

1.6 Change Log

0.3.0

- Added the ability to continually fetch a new balance rather than only the cached value (fetch_balance())
- Fixed a bug in which the closed attribute of Account was incorrectly set.

0.2.1

- Resolved bug checking pot balance instead of account balance during transfer to pot (Thanks to @eliasbenaddou for the report and PR)

0.2.0

- Removed Viewer from the main repository and is now its own project

0.1.6

- Resolved issue with missing HTML files for the viewer when not installing in edit mode

0.1.5

- Removed secrets from log files for security reasons

0.1.4

- Fixed receipt fetch issue
- Improved testing

0.1.3

- Fixed logout method using incorrect http verb
- Implemented pytest for most endpoints and logout
- Fixed various typing issues

0.1.2

- Implemented receipts
- Implemented a fetch single method for pots
- Fixed bug in pots checking incorrect balance
- Various minor fixes in httpio for differing body types
- Removed the superfluous Viewer2 directory.

0.1.1

- Added the ability to add annotations to transactions. The Monzo API is broken for custom keys

- Various spelling issues in notes

0.1.0

- Fixed viewer when authentication is not already setup

0.0.10

- Fixed incorrect key for spend_today in balance (thanks @psleep)
- Various bug fixes to Viewer
- Added ability to fetch transactions using viewer
- Added MonzoPermissionsError exception to Account
- Increased debug logging
- Now passing client id and client secret to handlers
- Various documentation fixes

0.0.9

- Minor release resolving github actions issue

0.0.8

- Minor release resolving github actions issue

0.0.7

- Resolved a packaging issue. Previous versions now obsolete

0.0.6

- Started to implement an API viewer, currently can aid in getting an access token
- Documentation improvements

0.0.5

- Improved error handling
- Created documentation (<https://monzo-api.readthedocs.io>)

0.0.4

- Fixed missing return in style property of pot class
- Updated auth_step_02.py to output token details that can be copied and pasted easily
- Implemented listeners process for when new tokens are retrieved including 2 listeners (echo, FileSystem)
- Implemented fetching a single translation
- Implemented fetching transaction list

0.0.3

- Fixed assignment of type rather than variable for AUth in 2 endpoints
- Fixed docstring for balance fetch method
- Implemented creating, listing and deleting webhooks
- Implemented listing, depositing into and withdrawing from pots

0.0.2

- Fixed error in API URL causing extra forward slash

- Implemented listing accounts
- Implemented reading balances
- Implemented posting feed items

0.0.1

- Initial minimal release

1.7 LICENSE

MIT License

Copyright (c) 2023 (Peter McDonald)

Permission **is** hereby granted, free of charge, to **any** person obtaining a copy of this software **and** associated documentation files (the "**Software**"), to deal **in** the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, **and/or** sell copies of the Software, **and** to permit persons to whom the Software **is** furnished to do so, subject to the following conditions:

The above copyright notice **and** this permission notice shall be included **in all** copies **or** substantial portions of the Software.

THE SOFTWARE IS PROVIDED "**AS IS**", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.