# Complaint’s System – 3rd Parties – Part 1 - Wise

## Introduction

Our goal today would be to integrate Wise in our application. For this purpose, we will use “requests” library

We will integrate Wise to reimburse our complainers for their claim if the approver approves their complaint.

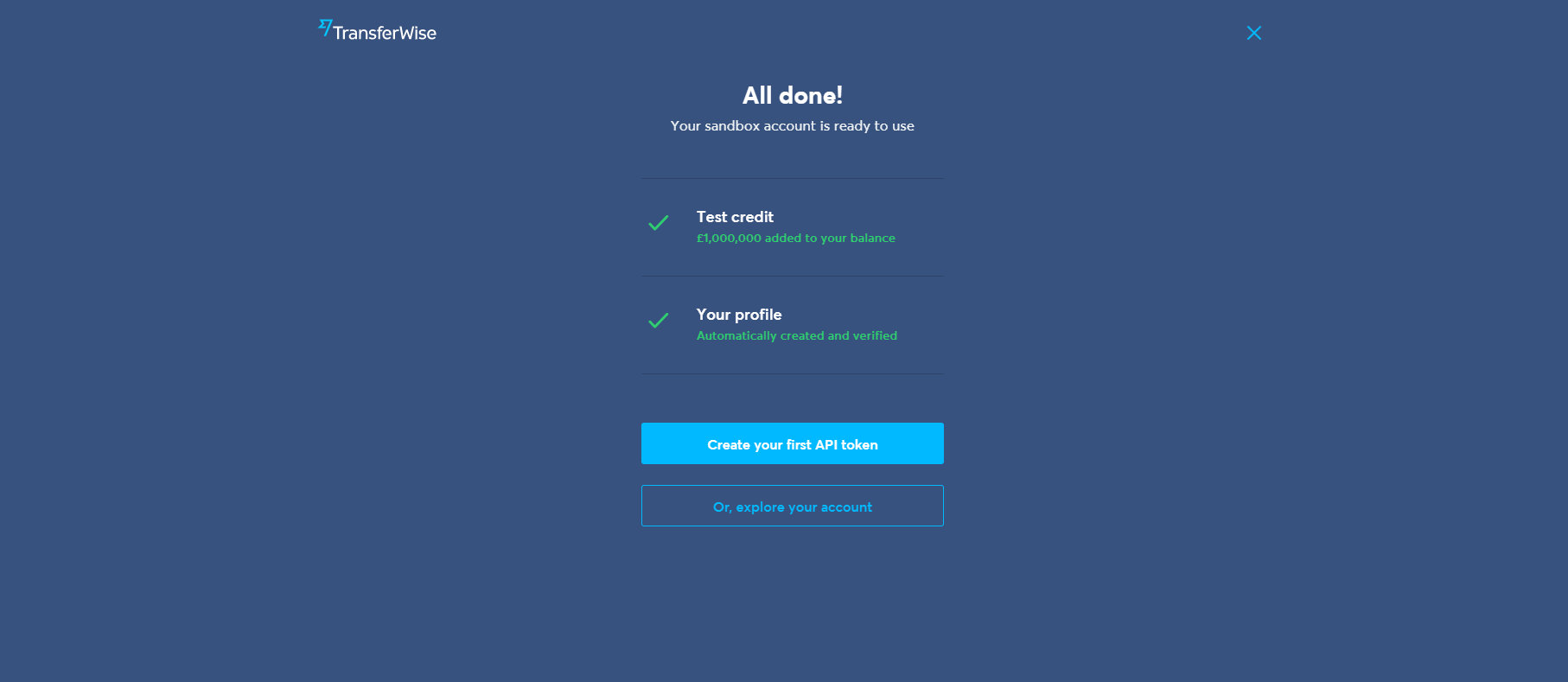
We will continue to develop our architecture and rely on clean code and structured files.

## Wise account setup

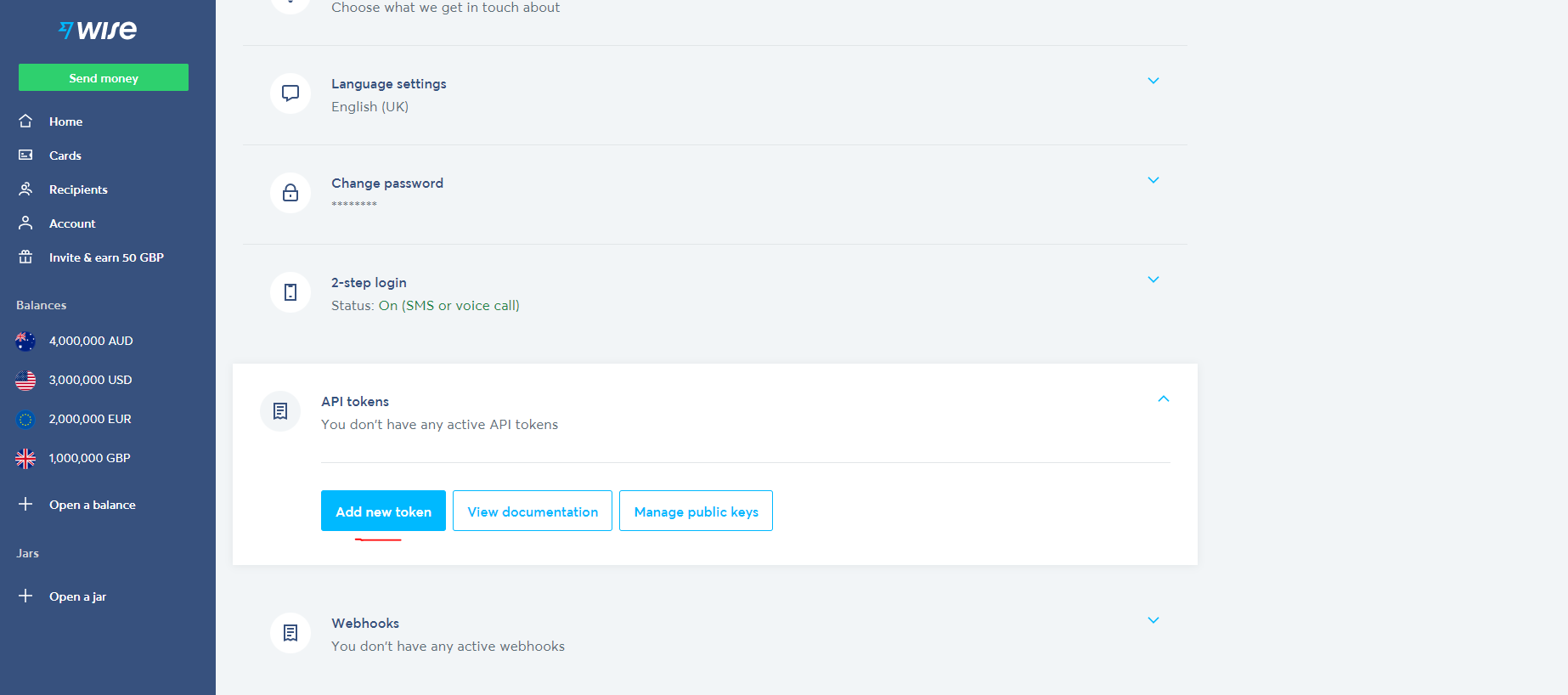
You can check the docs [here](https://api-docs.transferwise.com/#payouts-guide-api-access). I suggest you follow the steps described in the docs, if they have changed their docs without notification and there are some differences.

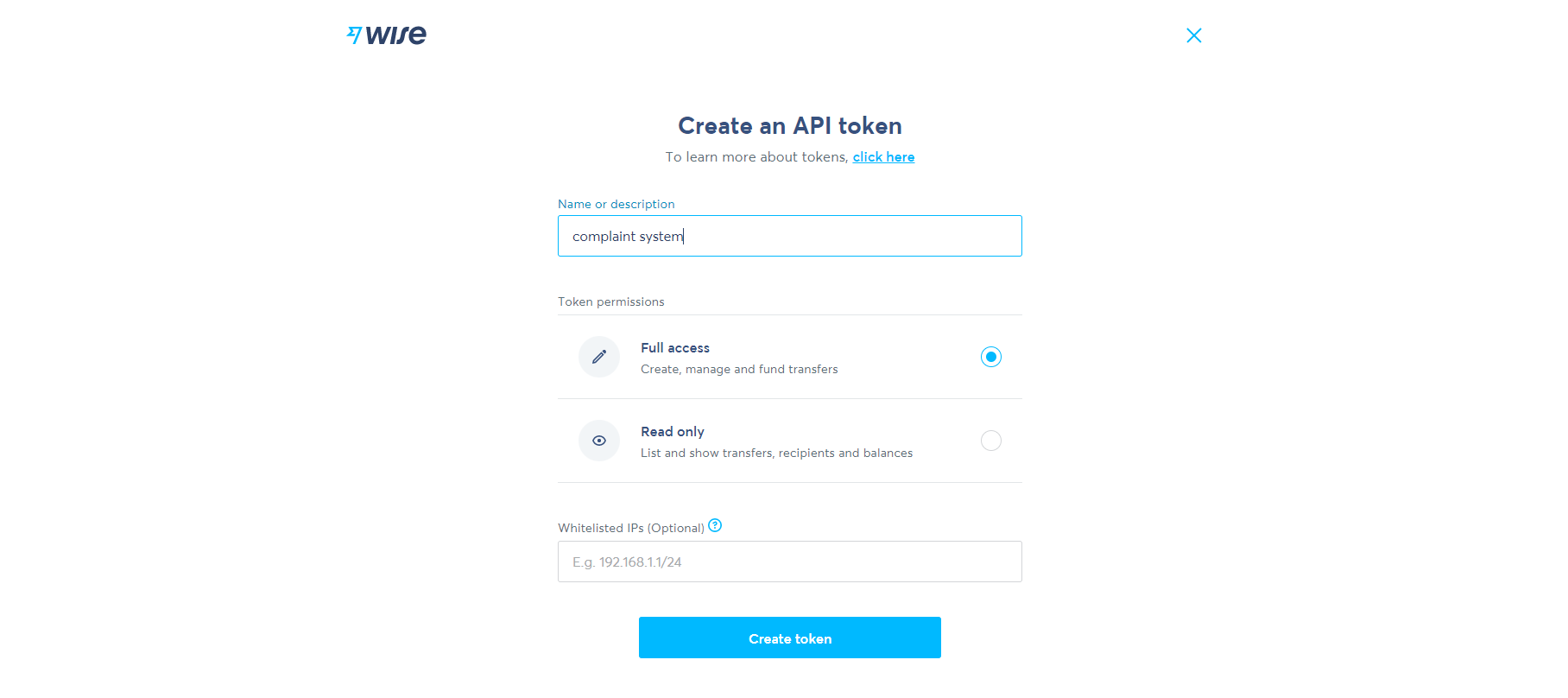
Create a **developer** account from [here](https://sandbox.transferwise.tech/register).

NB! Two factor authentication (2FA) code for sandbox login is **111111**.

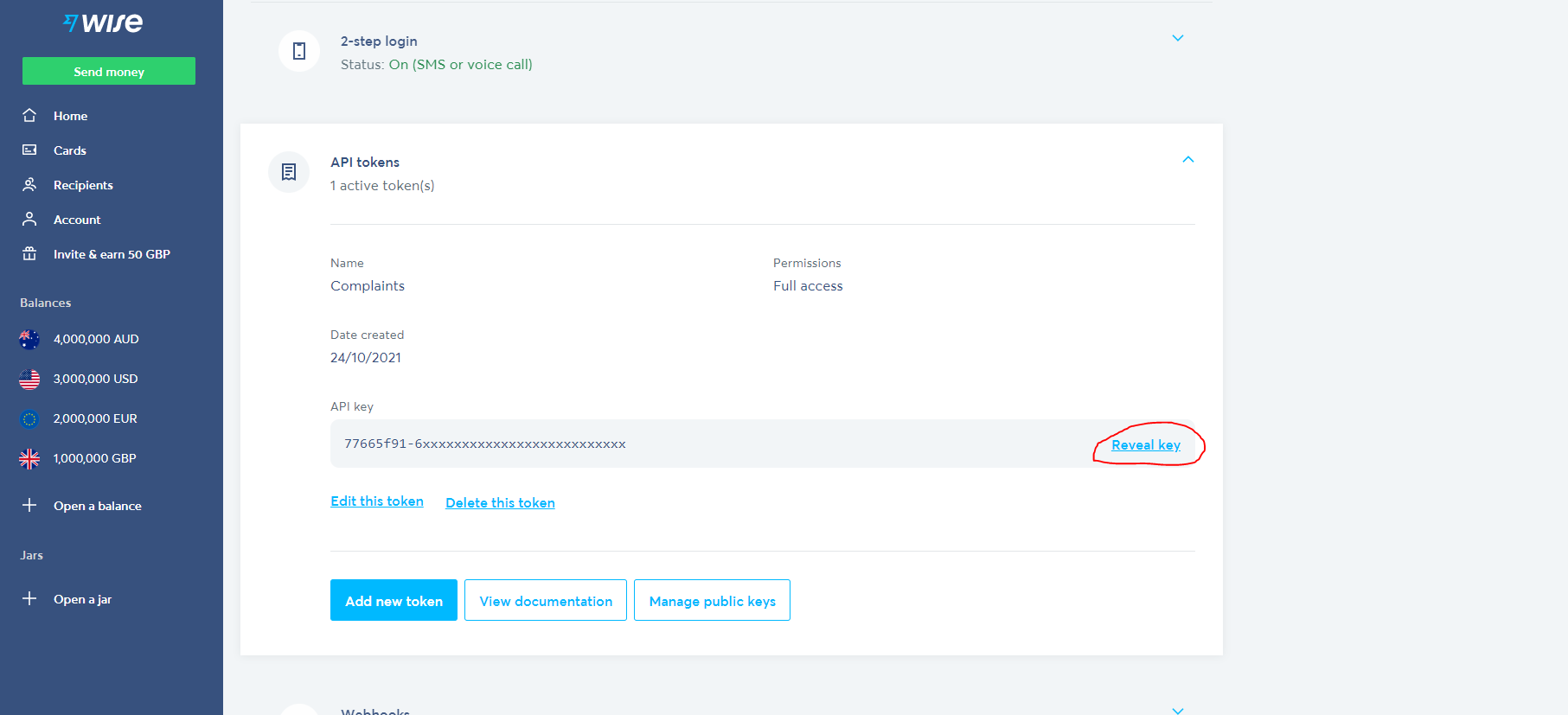


Click **Create your first API token**. Then choose personal account. Then you will see a couple multicurrency accounts with test money in it. We need to generate the token:





Then click **Reveal token** and store its value in the **.env** file under the name of **WISE\_TOKEN**.



**NB!** Always work with sandbox while developing, otherwise if you set up a real card number the money will be fetched for real!

## Wise integration in the app – issue transactions

There are a couple of steps to execute payouts according to the docs:

Step 0: Get your profile id

Step 1: Create a quote

Step 2: Create a recipient account

Step 3: Create a transfer

Step 4: Fund a transfer

Let’s start by creating a Wise service in our code, which will describe all of this steps:

pip install requests

Create a file called **wise.py** in **services** folder

import json  
import uuid  
  
import requests  
  
from werkzeug.exceptions import InternalServerError  
from decouple import config  
  
  
class WiseService:  
 def \_\_init\_\_(self):  
 self.main\_url = config(**"WISE\_URL"**)  
 self.headers = {  
 **"Content-Type"**: **"application/json"**,  
 **"Authorization"**: **f"Bearer** {config(**'WISE\_TOKEN'**)}**"**,  
 }  
 profile\_id = self.\_get\_profile\_id()  
 self.profile\_id = profile\_id  
  
 def \_get\_profile\_id(self):  
 url = self.main\_url + **"/v1/profiles"** resp = requests.get(url, headers=self.headers)  
  
 if resp.status\_code == 200:  
 resp = resp.json()  
 return [a[**"id"**] for a in resp if a[**"type"**] == **"personal"**][0]  
 else:  
 print(resp)  
 raise InternalServerError(**"Payment provider is not available at the moment"**)

Here we assign the headers and the profile id. We define a helper function \_get\_profile\_id to make the request and extract the personal profile id. Refer to [docs here](https://api-docs.transferwise.com/#payouts-guide-get-your-profile-id) to take a look at request fields and response format.

Then, we need to [create a quote](https://api-docs.transferwise.com/#payouts-guide-create-quote). We will extend our class by defining the following function:

def create\_quote(self, amount):  
 url = self.main\_url + **"/v2/quotes"** data = {  
 **"sourceCurrency"**: **"EUR"**,  
 **"targetCurrency"**: **"BGN"**,  
 **"targetAmount"**: amount,  
 **"profile"**: self.profile\_id,  
 }  
 resp = requests.post(url, headers=self.headers, data=json.dumps(data))  
  
 if resp.status\_code == 200:  
 resp = resp.json()  
 return resp[**"id"**]  
 else:  
 print(resp)  
 raise InternalServerError(**"Payment provider is not available at the moment"**)

We are defining fixed currencies, because our system will pay only in BGN. For the **sourceCurrency** you should pick one of the predefined currency accounts. If you wish to work with different currency, then you need to open a new currency account from your profile.

Now, we will [create a recipient account](https://api-docs.transferwise.com/#payouts-guide-create-recipient-account):

def create\_recipient\_account(self, full\_name, iban):  
 url = self.main\_url + **"/v1/accounts"** data = {  
 **"currency"**: **"BGN"**,  
 **"type"**: **"iban"**,  
 **"profile"**: self.profile\_id,  
 **"accountHolderName"**: full\_name,  
 **"legalType"**: **"PRIVATE"**,  
 **"details"**: {**"iban"**: iban},  
 }  
 resp = requests.post(url, headers=self.headers, data=json.dumps(data))  
  
 if resp.status\_code == 200:  
 resp = resp.json()  
 return resp[**"id"**]  
 else:  
 print(resp)  
 raise InternalServerError(**"Payment provider is not available at the moment"**)

Please note that we need only the iban here, because we will send in Bulgaria. If you wish to try it for different country, you need to refer to these [dynamic forms](https://sandbox.transferwise.tech/dynamic-forms-ui/v1) and see the required fields.

We need to [create a transfer](https://api-docs.transferwise.com/#payouts-guide-create-transfer):

def create\_transfer(self, target\_account\_id, quote\_id):  
 customer\_transaction\_id = str(uuid.uuid4())  
  
 url = self.main\_url + **"/v1/transfers"** data = {  
 **"targetAccount"**: target\_account\_id,  
 **"quoteUuid"**: quote\_id,  
 **"customerTransactionId"**: customer\_transaction\_id,  
 **"details"**: {},  
 }  
 resp = requests.post(url, headers=self.headers, data=json.dumps(data))  
  
 if resp.status\_code == 200:  
 resp = resp.json()  
 return resp[**"id"**]  
 else:  
 print(resp)  
 raise InternalServerError(**"Payment provider is not available at the moment"**)

Last, we will define the [fund transfer](https://api-docs.transferwise.com/#payouts-guide-fund-transfer) function:

def fund\_transfer(self, transfer\_id):  
 url = self.main\_url + **f"/v3/profiles/**{self.profile\_id}**/transfers/**{transfer\_id}**/payments"** resp = requests.post(url, headers=self.headers)  
  
 if resp.status\_code == 200:  
 resp = resp.json()  
 return resp[**"id"**]  
 else:  
 print(resp)  
 raise InternalServerError(**"Payment provider is not available at the moment"**)

Please, note that all POST requests require the data dictionary to be dumped (transformed to JSON format).

In this part we will adjust our code, so that it works properly. First we will change the Complainer request schema to accept iban:

class RequestRegisterComplainerSchema(RequestRegisterUserSchema):  
 iban = fields.String(min\_length=22, max\_length=22, required=True)

We will create a new class, because it is only applicable to complainers. Then we will change the ComplainerModel in models/user.py to have:

iban = db.Column(db.String(22), nullable=False)

We will make migrations. If you are having some issues try to set the FLASK\_APP and PYTHONPATH to the main.py file ad root project (export for unix).

SET PYTHONPATH=./

SET FLASK\_APP=./main.py

Then

flask db migrate -m "your message"

flask db upgrade

Last, we need to change the schema for the resources/auth.py:

class RegisterComplainer(Resource):  
 **@validate\_schema(RequestRegisterComplainerSchema)** def post(self):  
 data = request.get\_json()  
 token = ComplainerManager.register(data)  
 return {**"token"**: token}, 201

Now, we will define new model which will store our transaction information to the database. Create a file **transaction.py** in **models** folder:

from sqlalchemy import func  
  
from db import db  
  
  
class TransactionModel(db.Model):  
 \_\_tablename\_\_ = **'transactions'** id = db.Column(db.Integer, primary\_key=True)  
 quote\_id = db.Column(db.String(100), nullable=False)  
 transfer\_id = db.Column(db.String(100), nullable=False)  
 target\_account\_id = db.Column(db.String(100), nullable=False)  
 amount = db.Column(db.Float, nullable=False)  
 create\_on = db.Column(db.DateTime, server\_default=func.now())  
 complaint\_id = db.Column(db.Integer, db.ForeignKey(**'complaints.id'**))  
 complaint = db.relationship(**'ComplaintModel'**)

Do not forget to export it in the models/\_\_init\_\_.py

from .user import \*  
from .complaint import \*  
from .transaction import \*

We need to migrate and upgrade once more.

In ComplaintListCreate class we will change a little bit the post method by passing the complainer not the complainer id

complain = ComplaintManager.create(data, **complainer**)

Now in the ComplaintManager we will define a function responsible for creating a transaction:

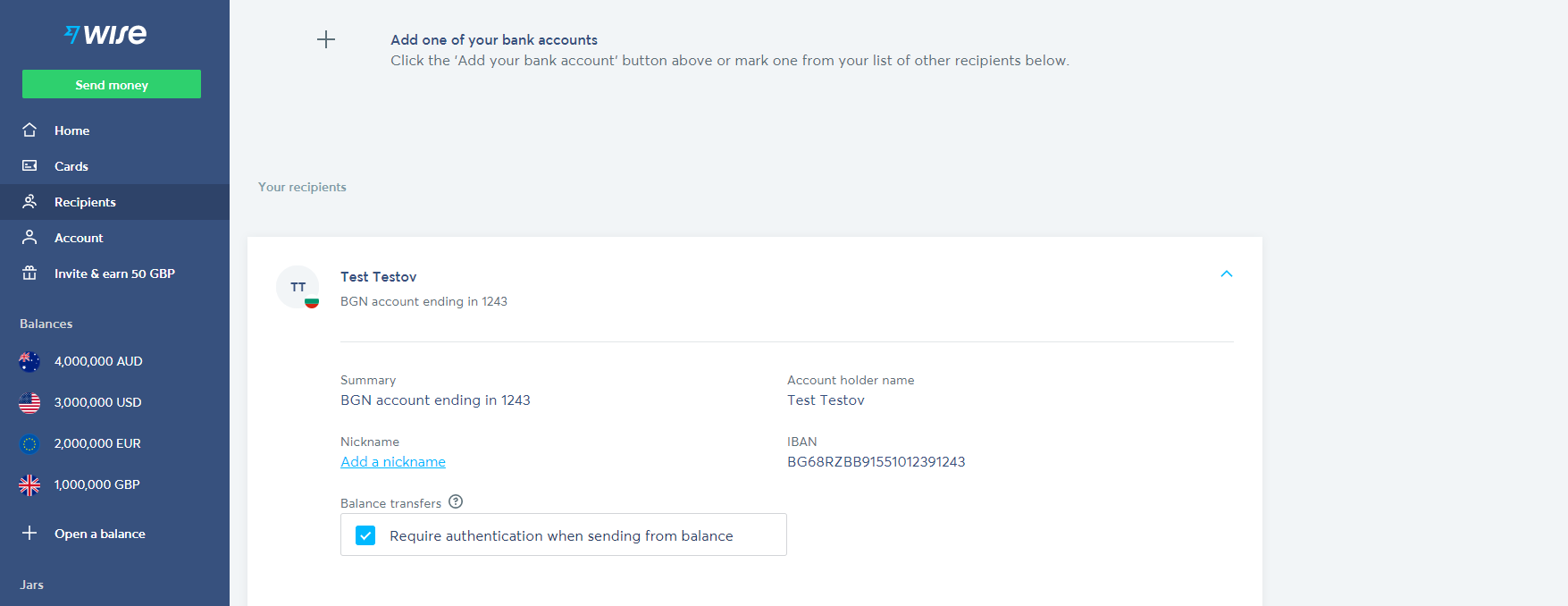
@staticmethod  
def issue\_transaction(amount, full\_name, iban, complaint\_id):  
 wise\_service = WiseService()  
 quote\_id = wise\_service.create\_quote(amount)  
 recipient\_id = wise\_service.create\_recipient\_account(full\_name, iban)  
 transfer\_id = wise\_service.create\_transfer(recipient\_id, quote\_id)  
 data = {  
 **"quote\_id"**: quote\_id,  
 **"transfer\_id"**: transfer\_id,  
 **"target\_account\_id"**: recipient\_id,  
 **"amount"**: amount,  
 **"complaint\_id"**: complaint\_id,  
 }  
 transaction = TransactionModel(\*\*data)  
 db.session.add(transaction)  
 db.session.flush()

We will change a bit the create method in the manager:

@staticmethod  
def create(data, complainer):  
*"""  
Decode the base64 encoded photo,  
uploads it to s3 and set the photo url to  
the s3 generated url.  
Creates a complaint and issues a transaction against it.  
Flushes the rows.  
"""* data[**"complainer\_id"**] = complainer.id  
encoded\_photo = data.pop(**"photo"**)  
extension = data.pop(**"photo\_extension"**)  
name = **f"**{str(uuid.uuid4())}**"**path = os.path.join(TEMP\_FILE\_FOLDER, **f"**{name}**.**{extension}**"**)  
decode\_photo(path, encoded\_photo)  
url = s3.upload\_photo(path, name, extension)  
os.remove(path)  
data[**"photo\_url"**] = url  
c = ComplaintModel(\*\*data)  
db.session.add(c)  
db.session.flush()  
ComplaintManager.issue\_transaction(data[**"amount"**], complainer.first\_name + **" "** + complainer.last\_name, complainer.iban, c.id)  
return c

Now we accept the complainer object, not just the idea and we are capable of getting the Iban and names from the object. Also, before we return the response, we issue a transaction in wise. You can see the recipient:

You can always refer to [this commit](https://github.com/InesIvanova/Flask-course-prep-materials/commit/79837c71b0a3f3c38970f399a7b6d2b207eb1014) if you have any difficulties.



## Wise integration in the app – release and cancel funds

Your home assignment will be to extend the functionality of the approvers. When they approve a transaction, you should release the funds by calling the already defined **fund\_transfer** in **WiseService** class (query the transaction to be able to fetch the transfer id). This logic should be done in manager class, just before the status update of the complaint. Note that there [is further implementation](https://api-docs.transferwise.com/#strong-customer-authentication-personal-token) if you really want to send the money in production environment.

If you want you can generate keys, upload the public one and extend this function fund\_transfer to authenticate properly. If it is too much hustle it is enough to just call the function. [Commit](https://github.com/InesIvanova/Flask-course-prep-materials/commit/95a151ed5355341a814df3a786410fcc31cc64eb).

## Small refactor

If you take a look at temp\_files folder you will see that we have saved all files, than have been decoded. This is not our purpose – to save them locally. We have s3 for that. Now we will add a code that will always delete the file locally after it uploads it to s3. In the coplaint manager you can add the following after the s3 upload:

…  
url = s3.upload\_photo(path, name)  
**os.remove(path)**

…

Another thing we would like to refactor is to remove temp\_files from gitignore and add .gitignore file in this folder with the following content:

*# Ignore everything in this directory*\*  
*# Except this file*!.gitignore

This is the [commit](https://github.com/InesIvanova/Flask-course-prep-materials/commit/2fe42423fbe8084ee7f657802e7b14f03119e38e#diff-6cb6149b04f4051e657f29a0213c5f9540586e3c9c33f6fb358824d1d98adf3c). Also you can refer to [this one](https://github.com/InesIvanova/Flask-course-prep-materials/commit/cf76708feed0342a079f045019299e0bacb7d164), where some refactor took place