|  |
| --- |
| from .payment import Card |
|  | from .external import ExternalPayment |

|  |
| --- |
| From application.services.payment import Card |
|  |  |
|  |  |
|  | class BasePaymentGateway: |
|  | def \_\_init\_\_(self, repeat=0): |
|  | self.repeat = repeat |
|  | self.gateway = None |
|  |  |
|  | def \_\_repr\_\_(self): |
|  | return "<{}>".format("BasePaymentGateway") |
|  |  |
|  | def connect(self, gateway=None, details=None): |
|  | if gateway != None: |
|  | if self.authenticate(details): |
|  | return True |
|  | return False |
|  |  |
|  | def authenticate(self, details=None): |
|  | if details != None: |
|  | return True |
|  | return False |
|  |  |
|  | def pay(self, amount, user\_details=None, gateway=None): |
|  | if gateway is None: |
|  | gateway = self.gateway |
|  | while self.repeat + 1 > 0: |
|  | if self.connect(gateway, user\_details): |
|  | print("payment of {} in gateway {} sucessful".format(amount, self.gateway)) |
|  | return True |
|  | self.repeat -= 1 |
|  | return False |
|  |  |
|  |  |
|  | class PremiumBasePaymentGateway(BasePaymentGateway): |
|  | def \_\_init\_\_(self, repeat=3): |
|  | super(PremiumBasePaymentGateway, self).\_\_init\_\_(repeat) |
|  | self.gateway = "PremiumBasePaymentGatway" |
|  |  |
|  | def \_\_repr\_\_(self): |
|  | return "<PremiumBasePaymentGateway>" |
|  |  |
|  |  |
|  | class ExpensiveBasePaymentGateway(BasePaymentGateway): |
|  | def \_\_init\_\_(self, repeat=1): |
|  | super(ExpensiveBasePaymentGateway, self).\_\_init\_\_(repeat) |
|  | self.gateway = "ExpensiveBasePaymentGateway" |
|  |  |
|  | def \_\_repr\_\_(self): |
|  | return "<ExpensiveBasePaymentGateway>" |
|  |  |
|  |  |
|  | class CheapBasePaymentGateway(BasePaymentGateway): |
|  | def \_\_init\_\_(self, repeat=0): |
|  | super(CheapBasePaymentGateway, self).\_\_init\_\_(repeat) |
|  | self.gateway = "CheapBasePaymentGateway" |
|  |  |
|  | def \_\_repr\_\_(self): |
|  | return "<CheapBasePaymentGateway>" |
|  |  |
|  |  |
|  | class ExternalPayment: |
|  | def \_\_init\_\_(self, amount, card\_details=None): |
|  | self.amount = amount |
|  | self.card\_details = card\_details |
|  |  |
|  | def make\_payment(self): |
|  | try: |
|  | payment\_mode = None |
|  | if self.amount <= 20: |
|  | payment\_mode = CheapBasePaymentGateway() |
|  | elif 20 < self.amount < 500: |
|  | payment\_mode = ExpensiveBasePaymentGateway() |
|  | elif self.amount >= 500: |
|  | payment\_mode = PremiumBasePaymentGateway() |
|  | else: |
|  | return False |
|  |  |
|  | status = payment\_mode.pay(self.amount, self.card\_details) |
|  | return status |
|  | except: |
|  | return False |

|  |
| --- |
| import re |
|  | import datetime |
|  | from decimal import Decimal |
|  |  |
|  |  |
|  | def validate\_card(card): |
|  | if not re.search(r"^[456]\d{3}(-?\d{4}){3}$", card) or re.search(r"(\d)\1{3}", re.sub("-", "", card)): |
|  | return False |
|  | return True |
|  |  |
|  |  |
|  | class BaseCardDetails: |
|  | def \_\_init\_\_(self): |
|  | self.CreditCardNumber = None |
|  | self.CardHolder = None |
|  | self.ExpirationDate = None |
|  | self.SecurityCode = None |
|  | self.Amount = None |
|  |  |
|  |  |
|  | class Card(BaseCardDetails): |
|  | def \_\_init\_\_(self): |
|  | super(Card, self).\_\_init\_\_() |
|  |  |
|  | def verify\_input(self, \*\*kwargs): |
|  | """ |
|  | CreditCardNumber(mandatory, string, it should be a valid credit card number) |
|  | -CardHolder: (mandatory, string) |
|  | -ExpirationDate (mandatory, DateTime, it cannot be in the past) |
|  | -SecurityCode (optional, string, 3 digits) |
|  | -Amount (mandatoy decimal, positive amount |
|  | :param kwargs: |
|  | :return: |
|  | """ |
|  | cards\_value = ["CreditCardNumber", "CardHolder", "ExpirationDate", "Amount"] |
|  | if set(cards\_value).intersection(kwargs.keys()) != set(cards\_value): |
|  | print("card details not found") |
|  | return False |
|  |  |
|  | if not type(kwargs['CreditCardNumber'] == str and validate\_card(kwargs['CreditCardNumber'])) or not len(kwargs['CreditCardNumber']) == 16: |
|  | print("invalid credit card number") |
|  | return False |
|  |  |
|  | if not type(kwargs['CardHolder']) == str: |
|  | print("card holder is not of string type") |
|  | return False |
|  |  |
|  | if kwargs.get('SecurityCode',None) : |
|  | if not (type(kwargs.get('SecurityCode', None)) == str and len(kwargs.get('SecurityCode', None)) == 3) or not kwargs.get('SecurityCode', None).isdigit(): |
|  | print("security code error") |
|  | return False |
|  |  |
|  | if not datetime.datetime.strptime(kwargs['ExpirationDate'], "%Y/%m/%d") > datetime.datetime.now(): |
|  | print("date time error") |
|  | return False |
|  |  |
|  | try: |
|  | if not Decimal(kwargs['Amount']) > 0: |
|  | print("amount is invalid") |
|  | return False |
|  | except: |
|  | return False |
|  |  |
|  | \_data\_to\_map = { |
|  | "CreditCardNumber": kwargs['CreditCardNumber'], |
|  | 'Amount': kwargs['Amount'], |
|  | 'CardHolder': kwargs['CardHolder'], |
|  |  |
|  | 'ExpirationDate': kwargs['ExpirationDate'] |
|  | } |
|  | if kwargs.get("SecurityCode", None): |
|  | \_data\_to\_map.update({"SecurityCode":kwargs.get("SecurityCode")}) |
|  | print("input is verified.") |
|  | self.\_\_map\_to\_card(\*\*kwargs) |
|  | return True |
|  |  |
|  | def \_\_map\_to\_card(self, \*\*kwargs): |
|  | self.CreditCardNumber = kwargs.get('CreditCardNumber', None) |
|  | self.Amount = kwargs.get('Amount', None) |
|  | self.CardHolder = kwargs.get('CardHolder', None) |
|  |  |
|  | self.SecurityCode = kwargs.get('SecurityCode', None) |
|  | self.ExpirationDate = kwargs.get('ExpirationDate', None) |
|  |  |
|  | print("mapping of user input is done sucessfully.") |
|  | return True |
|  |  |

|  |
| --- |
| import pytest |
|  | from webtest import TestApp |
|  |  |
|  | from application.app import create\_app |
|  |  |
|  |  |
|  |  |
|  | @pytest.fixture |
|  | def app(): |
|  | """Create application for the tests.""" |
|  | \_app = create\_app("test.settings") |
|  | ctx = \_app.test\_request\_context() |
|  | ctx.push() |
|  |  |
|  | yield \_app |
|  |  |
|  | ctx.pop() |
|  |  |
|  | @pytest.fixture |
|  | def testapp(app): |
|  | """Create Webtest app.""" |
|  | return TestApp(app) |

|  |
| --- |
| import pytest |
|  | import requests |
|  | import json |
|  |  |
|  | from application.settings import FLASK\_RUN\_HOST, FLASK\_RUN\_PORT |
|  |  |
|  | url = "http://{}:{}".format(FLASK\_RUN\_HOST, FLASK\_RUN\_PORT) |
|  |  |
|  |  |
|  | def test\_endpoint\_for\_invalid\_request\_type(): |
|  | # Sending the get request to the endpoint to check api behaviour for different request type |
|  | response = requests.get("{}/ProcessPayment".format(url)) |
|  |  |
|  | # checking error response, can be improve by ckecking what kind of error has occur |
|  | assert response.status\_code >= 400 |
|  |  |
|  |  |
|  | def test\_payment\_no\_data(): |
|  | # testing when request body has no data |
|  | card\_data = {} |
|  | response = requests.post("{}/ProcessPayment".format(url), data=json.dumps(card\_data), |
|  | headers={"Content-Type": "application/json"}) |
|  | assert response.status\_code == 400 |
|  |  |
|  |  |
|  | def test\_payment\_invalid\_argument(): |
|  | # testing api when request body has invalid argument, as compared to argument which is required |
|  | card\_data = dict(CreditCardNumbers="1234123412341234", CardHoldewr="prashant rana", SecurityCode="111", ExpirationDate="2020/1/1", Amount=222.2) |
|  | response = requests.post("{}/ProcessPayment".format(url), data=json.dumps(card\_data), headers={"Content-Type": "application/json"}) |
|  | assert response.status\_code == 400 |
|  |  |
|  |  |
|  | def test\_payment\_ext\_invalid\_credit\_card\_info(): |
|  | # testing for various cases when CreditCardNumber value is different, checking validity of credit |
|  | # card numbers |
|  | card\_data\_1 = {"CreditCardNumber": "qwer123456ijiojw","CardHolder":"prashant rana","SecurityCode": "111", |
|  | "ExpirationDate": "2020/1/1","Amount": 333.3} |
|  | card\_data\_2 = {"CreditCardNumber": "1234567890123456", "CardHolder": "prashant rana", "SecurityCode": "111", |
|  | "ExpirationDate": "2022/11/12", "Amount": 333.3} |
|  | response\_1 = requests.post("{}/ProcessPayment".format(url), data=json.dumps(card\_data\_1), headers={"Content-Type":"application/json"}) |
|  | response\_2 = requests.post("{}/ProcessPayment".format(url), data=json.dumps(card\_data\_2), headers={"Content-Type": "application/json"}) |
|  |  |
|  | assert response\_1.status\_code == 400 |
|  | assert response\_2.status\_code == 200 |
|  |  |
|  |  |
|  | def test\_valid\_input\_data\_for\_various\_amount(): |
|  | # testing the cases where for different amount input different external payment method invoke |
|  | card\_data\_1 = {"CreditCardNumber": "1234567890123456", "CardHolder": "prashant rana", "SecurityCode": "111", |
|  | "ExpirationDate": "2022/11/12", "Amount": 19} |
|  | card\_data\_2 = {"CreditCardNumber": "1234567890123456", "CardHolder": "prashant rana", "SecurityCode": "111", |
|  | "ExpirationDate": "2022/11/12", "Amount": 333} |
|  | card\_data\_3 = {"CreditCardNumber": "1234567890123456", "CardHolder": "prashant rana", "SecurityCode": "111", |
|  | "ExpirationDate": "2022/11/12", "Amount": 666} |
|  |  |
|  | response\_1 = requests.post("{}/ProcessPayment".format(url), data=json.dumps(card\_data\_1), headers={"Content-Type":"application/json"}) |
|  | response\_2 = requests.post("{}/ProcessPayment".format(url), data=json.dumps(card\_data\_2), headers={"Content-Type":"application/json"}) |
|  | response\_3 = requests.post("{}/ProcessPayment".format(url), data=json.dumps(card\_data\_3), headers={"Content-Type":"application/json"}) |
|  |  |
|  | assert response\_1.status\_code == 200 |
|  | assert response\_2.status\_code == 200 |
|  | assert response\_3.status\_code == 200 |
|  |  |
|  |  |
|  | def test\_payment\_ext\_exp\_date(): |
|  | # testing the ExpirationDate different case, when date is more then present and when date is past of present date |
|  | card\_data\_1 = {"CreditCardNumber": "qwer123456ijiojw", "CardHolder": "prashant rana", "SecurityCode": "111", |
|  | "ExpirationDate": "2022/1/1", "Amount": 333.3} |
|  | card\_data\_2 = {"CreditCardNumber": "1234567890123456", "CardHolder": "prashant rana", "SecurityCode": "111", |
|  | "ExpirationDate": "2019/11/12", "Amount": 333.3} |
|  |  |
|  | response\_1 = requests.post("{}/ProcessPayment".format(url), data=json.dumps(card\_data\_1), headers={"Content-Type": "application/json"}) |
|  | response\_2 = requests.post("{}/ProcessPayment".format(url), data=json.dumps(card\_data\_2), |
|  | headers={"Content-Type": "application/json"}) |
|  |  |
|  | assert response\_1.status\_code == 200 |
|  | assert response\_2.status\_code == 400 |
|  |  |
|  |  |
|  | def test\_security\_code(): |
|  | # testing for SecurityCode cases |
|  | card\_data\_1 = {"CreditCardNumber": "qwer123456ijiojw", "CardHolder": "prashant rana", "SecurityCode": "111", |
|  | "ExpirationDate": "2022/1/1", "Amount": 333.3} |
|  | card\_data\_2 = {"CreditCardNumber": "qwer123456ijiojw", "CardHolder": "prashant rana", "ExpirationDate": "2022/1/1", "Amount": 333.3} |
|  | card\_data\_3 = {"CreditCardNumber": "qwer123456ijiojw", "CardHolder": "prashant rana", "SecurityCode": 444, |
|  | "ExpirationDate": "2022/1/1", "Amount": 333.3} |
|  |  |
|  | response\_1 = requests.post("{}/ProcessPayment".format(url), data=json.dumps(card\_data\_1), |
|  | headers={"Content-Type": "application/json"}) |
|  |  |
|  | response\_2 = requests.post("{}/ProcessPayment".format(url), data=json.dumps(card\_data\_2), |
|  | headers={"Content-Type": "application/json"}) |
|  | response\_3 = requests.post("{}/ProcessPayment".format(url), data=json.dumps(card\_data\_3), |
|  | headers={"Content-Type": "application/json"}) |
|  | assert response\_1.status\_code == 200 |
|  | assert response\_2.status\_code == 200 |
|  | assert response\_3.status\_code == 400 |

from . import views

|  |
| --- |
| from flask import Blueprint, request, abort, jsonify |
|  | import json |
|  | from application.services.payment import Card, ExternalPayment |
|  |  |
|  | blueprint = Blueprint("views", \_\_name\_\_, url\_prefix="/") |
|  |  |
|  |  |
|  | @blueprint.route("/ProcessPayment", methods=['POST']) |
|  | def payment(): |
|  |  |
|  | if request.method == 'POST': |
|  | # getting the request data, which is send in request body as json |
|  | data = request.get\_data(as\_text=True) |
|  | # checking if there is any input data present or not, if not then throw bad request |
|  | if not data: |
|  | abort(400) |
|  | # loading the data into json format |
|  | request\_data = json.loads(data) |
|  |  |
|  | # loading the card object, to check the request data is same as the input data |
|  | card\_data = Card() |
|  | print("request data {}".format(request\_data)) |
|  | try: |
|  | # verifying all the request data input whether they follow all the criteria or not |
|  | # note assumption, date format is taken as %y:%m:%d as opposed to %m:%d in credit cards |
|  | if not card\_data.verify\_input(\*\*request\_data): |
|  | print("card data invalid") |
|  | abort(400) |
|  | except: |
|  | abort(400) |
|  | try: |
|  | # payment procedure begins |
|  | print("payment status started") |
|  | payment\_status = ExternalPayment(card\_data.Amount, card\_data) |
|  | print("payment process started") |
|  | # begin of payment process, starting from make connection, authenticate the user, made payment |
|  | # will provide result if the transaction is sucessfull or not. |
|  |  |
|  | payment\_sccessfull = payment\_status.make\_payment() |
|  | # checking the transaction status, if it is successful or not. |
|  | if payment\_sccessfull: |
|  | return {"status code": 200}, 200 |
|  | else: |
|  | abort(400) |
|  | except: |
|  | abort(500) |
|  | else: |
|  | abort(400) |

|  |
| --- |
| from . import app |
|  | from . import settings |

|  |
| --- |
| from typing import Any |
|  | from flask import Flask |
|  | from application import views |
|  |  |
|  | # see flask rest api for app initialization |
|  |  |
|  | def create\_app(config\_object="application.settings"): |
|  | app = Flask(\_\_name\_\_) |
|  | app.config.from\_object(config\_object) |
|  | register\_blueprints(app) |
|  | return app |
|  |  |
|  |  |
|  | def register\_blueprints(app): |
|  | """Register Flask blueprints.""" |
|  | app.register\_blueprint(views.views.blueprint) |
|  | return None |

|  |
| --- |
| from environs import Env |
|  |  |
|  | env = Env() |
|  | env.read\_env() |
|  |  |
|  | FLASK\_APP = env.str("FLASK\_APP") |
|  | SECRET = env.str("SECRET") |
|  | APP\_SETTINGS = env.str("APP\_SETTINGS") |
|  | FLASK\_RUN\_HOST = env.str("FLASK\_RUN\_HOST") |
|  | FLASK\_RUN\_PORT = env.int("FLASK\_RUN\_PORT") |