**Python code**

import datetime

import hashlib

import json

from flask import Flask, jsonify, request

import requests

from uuid import uuid4

from urllib.parse import urlparse

import pyodbc

# Part 1 - Building a Blockchain

class Project:

def \_\_init\_\_(self):

self.projectId=0

self.project\_name=None

self.address=None

self.owner=None

def set(self,projectId,project\_name,address):

self.projectId=projectId

self.project\_name=project\_name

self.address=address

def get(self):

project={'projectId': self.projectId,

'project\_name':self.project\_name,

'address': self.address,

'owner':self.owner}

return project

class Activity:

def \_\_init\_\_(self):

self.activity\_no=0

self.project=None

self.activity\_name=None

self.duration=None

self.price=0

self.start\_date=None

self.finish\_date=None

self.status=None

self.approved=None

# self.deal=None

self.delays=0

self.contract\_name=None

def set(self,activity\_no,project,activity\_name,duration,price,start\_date,finish\_date,status,approved,delays,contract\_name):

self.activity\_no=activity\_no

self.project=project

self.activity\_name=activity\_name

self.duration=duration

self.price=price

self.start\_date=start\_date

self.finish\_date=finish\_date

self.status=status

self.approved=approved

self.delays=delays

self.contract\_name=contract\_name

def get(self):

act={'activity\_no': self.activity\_no,

'project':self.project,

'activity\_name': self.activity\_name,

'duration':self.duration,

'price':self.price,

'start\_date':self.start\_date,

'finish\_date':self.finish\_date,

'status':self.status,

'approved':self.approved,

'delays':self.delays,

'contract\_name':self.contract\_name}

return act

class Contract:

def \_\_init\_\_(self):

self.contract\_no=0

self.contract\_name=None

self.project=None

self.amount=0

self.date=None

self.Type=None

self.parties=None

self.penalty=0

self.status=None

def set(self,contract\_no,contract\_name,project,amount,date,Type,parties,penalty,status):

self.contract\_no=contract\_no

self.contract\_name=contract\_name

self.project=project

self.amount=amount

self.date=date

self.Type=Type

self.parties=parties

self.penalty=penalty

self.status=status

def get(self):

contract={'contract\_no':self.contract\_no,

'contract\_name':self.contract\_name,

'project': self.project,

'amount': self.amount,

'date': self.date,

'Type': self.Type,

'parties':self.parties,

'penalty':self.penalty,

'status':self.status}

return contract

#class user

class User:

def \_\_init\_\_(self):

self.userId=None

self.username=None

self.password=None

self.companyName=None

self.credit=0

self.identity=None

self.email=None

def set(self,userId,username,password,companyName,credit,identity,email):

self.userId=userId

self.username=username

self.password=password

self.companyName=companyName

self.credit=credit

self.identity=identity

self.email=email

def get(self):

user={'userId':self.userId,

'username':self.username,

'password': self.password,

'companyName': self.companyName,

'credit':self.credit,

'identity':self.identity,

'email':self.email}

return user

class Notification:

def \_\_init\_\_(self):

self.notifyId=0

self.notification=None

self.contract\_name=None

self.timestamp=None

self.identity=None

self.username=None

def set(self,notifyId,notification,contract\_name,timestamp,identity,username):

self.notifyId=notifyId

self.notification=notification

self.contract\_name=contract\_name

self.timestamp=timestamp

self.identity=identity

self.username=username

def get(self):

notification={'notifyId':self.notifyId,

'notification':self.notification,

'contract\_name':self.contract\_name,

'timestamp':self.timestamp,

'identity':self.identity,

'username':self.username}

return notification

class Claim:

def \_\_init\_\_(self):

self.claimId=0

self.claim=None

self.status=None

self.username=None

self.timestamp=None

def set(self,claimId,claim,status,username,timestamp):

self.claimId=claimId

self.claim=claim

self.status=status

self.username=username

self.timestamp=timestamp

def get(self):

claim={'claimId':self.claimId,

'claim':self.claim,

'status':self.status,

'username':self.username,

'timestamp':self.timestamp}

return claim

"""

class auth:

def encryption(self,message):

encrypted\_message=public\_key.encrypt(

message,

padding.OAEP(

mgf=padding.MGF1(algorithm=hashes.SHA256()),

algorithm=hashes.SHA256(),

label=None

)

)

return encrypted\_message

def decryption(sellf,message):

original\_message = private\_key.decrypt(

encrypted,

padding.OAEP(

mgf=padding.MGF1(algorithm=hashes.SHA256()),

algorithm=hashes.SHA256(),

label=None

)

)

return original\_message

"""

class Blockchain:

def \_\_init\_\_(self):

self.user=[]

self.chain = []

self.transactions = []

self.description=[]

self.project=[]

self.activity=[]

self.contract=[]

self.create\_block(proof = 1, previous\_hash = '0',contract\_current=[],activity\_current=[])

self.nodes = set()

#first blockchain only

def create\_block(self, proof, previous\_hash,contract\_current,activity\_current):

block = {'index': len(self.chain) + 1,

'timestamp': str(datetime.datetime.now()),

'proof': proof,

'previous\_hash': previous\_hash,

'transactions': self.transactions,

'project':self.project,

'contract': contract\_current,

'activity':activity\_current,

'party':self.user,

'description':self.description}

self.transactions = []

self.user=[]

self.claim=[]

self.project=[]

self.activity=[]

self.contract=[]

self.description=[]

self.chain.append(block)

return block

def get\_previous\_block(self):

return self.chain[-1]

def add\_user(self,name):

self.user.append(name)

def proof\_of\_work(self, previous\_proof):

new\_proof = 1

check\_proof = False

while check\_proof is False:

hash\_operation = hashlib.sha256(str(new\_proof\*\*2 - previous\_proof\*\*2).encode()).hexdigest()

if hash\_operation[:4] == '0000':

check\_proof = True

else:

new\_proof += 1

return new\_proof

def hash(self, block):

encoded\_block = json.dumps(block,indent=4, sort\_keys=True, default=str).encode()

return hashlib.sha256(encoded\_block).hexdigest()

def is\_chain\_valid(self, chain):

previous\_block = chain[0]

block\_index = 1

while block\_index < len(chain):

block = chain[block\_index]

if block['previous\_hash'] != self.hash(previous\_block):

return False

previous\_proof = previous\_block['proof']

proof = block['proof']

hash\_operation = hashlib.sha256(str(proof\*\*2 - previous\_proof\*\*2).encode()).hexdigest()

if hash\_operation[:4] != '0000':

return False

previous\_block = block

block\_index += 1

return True

def add\_transaction(self, sender, receiver, amount):

self.transactions.append({'sender': sender,

'receiver': receiver,

'amount': amount})

previous\_block = self.get\_previous\_block()

return previous\_block['index'] + 1

def add\_project(self,project):

self.project.append(project)

previous\_block = self.get\_previous\_block()

return previous\_block['index'] + 1

def add\_activity(self,activity):

self.activity.append(activity)

previous\_block = self.get\_previous\_block()

return previous\_block['index'] + 1

def add\_contract(self,contract):

self.contract.append(contract)

previous\_block = self.get\_previous\_block()

return previous\_block['index'] + 1

def add\_description(self,message):

self.description.append(message)

previous\_block = self.get\_previous\_block()

return previous\_block['index'] + 1

def add\_node(self, address):

parsed\_url = urlparse(address)

self.nodes.add(parsed\_url.netloc)

def replace\_chain(self,longest\_chain):

self.chain=longest\_chain

return True

# Part 2 - Mining our Blockchain

own\_notification=[]

# Creating a Web App

app = Flask(\_\_name\_\_)

# Creating an address for the node on Port 5001

node\_address = str(uuid4()).replace('-', '')

block\_cont=[]

block\_act=[]

# Creating a Blockchain

blockchain = Blockchain()

connection\_string='Driver={Microsoft Access Driver (\*.mdb, \*.accdb)};DBQ=.\Smart\_Contract.accdb'

user=User()

# Mining a new block

@app.route('/mine\_block', methods = ['GET'])

def mine\_block():

previous\_block = blockchain.get\_previous\_block()

previous\_proof = previous\_block['proof']

proof = blockchain.proof\_of\_work(previous\_proof)

previous\_hash = blockchain.hash(previous\_block)

contract\_currunt=blockchain.contract

block\_cont.append(contract\_currunt)

activity\_currunt=blockchain.activity

block\_act.append(activity\_currunt)

blockchain.add\_transaction(sender = node\_address, receiver = 'Owner', amount = 1)

block = blockchain.create\_block(proof, previous\_hash,contract\_currunt,activity\_currunt)

response = {'message': 'Congratulations, you just mined a block!',

'index': block['index'],

'timestamp': block['timestamp'],

'proof': block['proof'],

'previous\_hash': block['previous\_hash'],

'transactions': block['transactions'],

'project':block['project'],

'contract': contract\_currunt,

'activity':activity\_currunt,

'party':block['party'],

'description':block['description']}

return jsonify(response), 200

# Getting the full Blockchain

@app.route('/get\_chain', methods = ['GET'])

def get\_chain():

response = {'chain': blockchain.chain,

'length': len(blockchain.chain)}

return jsonify(response), 200

# Getting ALL contracts

@app.route('/get\_contracts', methods = ['GET'])

def get\_contracts():

list\_contract=[]

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("select \* from contract ")

for row in cursor.fetchall():

contract=Contract()

contract.set(row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8])

list\_contract.append(contract.get())

cursor.close()

conn.close()

response = {'contracts': list\_contract}

return jsonify(response), 200

# Getting ALL projects

@app.route('/get\_projects', methods = ['GET'])

def get\_projects():

list\_project=[]

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("select \* from project ")

for row in cursor.fetchall():

project=Project()

project.set(row[0],row[1],row[2])

list\_project.append(project.get())

cursor.close()

conn.close()

response = {'projects': list\_project}

return jsonify(response), 200

# Getting ALL admins

@app.route('/get\_admins', methods = ['GET'])

def get\_users():

users=[]

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("select \* from user")

for row in cursor.fetchall():

user=User()

user.set(row[1],row[3],row[4],row[5],row[6])

users.append(user.get())

cursor.close()

conn.close()

response = {'users': users}

return jsonify(response), 200

@app.route('/remove\_admin', methods = ['POST'])

def remove\_admin():

json = request.get\_json()

contract\_keys = ['username']

if not all(key in json for key in contract\_keys):

return 'Some elements of the transaction are missing', 400

user=json['username']

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("delete from user where username=?",user)

conn.commit()

cursor.execute("insert into notification([notification],[timestamp],[identity]) VALUES(?,?)",(f'admin username:{user} has been removed',datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.close()

conn.close()

response = {'message': f'admin username:{user} has been removed'}

return jsonify(response), 201

@app.route('/get\_activities', methods = ['GET'])

def get\_act():

list\_activity=[]

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("select \* from activity ")

for row in cursor.fetchall():

activity=Activity()

activity.set(row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8],row[9],row[10])

list\_activity.append(activity.get())

cursor.close()

conn.close()

response = {'activities': list\_activity}

return jsonify(response), 200

# Getting ALL info for the node

@app.route('/get\_info', methods = ['GET'])

def get\_info():

response = user.get()

return jsonify(response), 200

# Getting ALL claims for the node

@app.route('/get\_claims', methods = ['GET'])

def get\_claims():

claims=[]

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("select \* from claim ")

for row in cursor.fetchall():

claim=Claim()

claim.set(row[0],row[1],row[2],row[3],row[4])

claims.append(claim.get())

cursor.close()

conn.close()

response = {'claims': claims}

return jsonify(response), 200

#contractor method for getting claim

@app.route('/get\_claim', methods = ['GET'])

def get\_claim():

claims=[]

username = request.args.get('username')

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("select \* from claim where username=?",username)

for row in cursor.fetchall():

claim=Claim()

claim.set(row[0],row[1],row[2],row[3],row[4])

claims.append(claim.get())

cursor.close()

conn.close()

response = {'claims': claims}

return jsonify(response), 200

# Checking if the Blockchain is valid

@app.route('/is\_valid', methods = ['GET'])

def is\_valid():

is\_valid = blockchain.is\_chain\_valid(blockchain.chain)

if is\_valid:

response = {'message': 'All good. The Blockchain is valid.'}

else:

response = {'message': 'The Blockchain is not valid.'}

return jsonify(response), 200

# authorization part

@app.route('/login', methods = ['POST'])

def login():

json = request.get\_json()

username=(json['username'],)

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("select username,password,identity,credit,companyName,email from user where username=?",username)

result = cursor.fetchone()

cursor.close()

conn.close()

if not result:

return 'wrong username', 400

if result[1]==json['password']:

user.username=result[0]

user.identity=result[2]

user.credit=result[3]

user.companyName=result[4]

user.email=result[5]

response = {'message': f'You are logged in as {user.identity}'}

else:

return 'wrong password', 400

return jsonify(response), 201

# notifying owner

@app.route('/get\_all\_notifications', methods = ['GET'])

def notify():

notify\_list=[]

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute(f"select \* from notification where identity='{user.identity}'")

for row in cursor.fetchall():

notification=Notification()

notification.set(row[0],row[1],row[2],row[3],row[4],row[5])

notify\_list.append(notification.get())

cursor.close()

conn.close()

response = {'notfications': notify\_list}

return jsonify(response), 200

# Adding a new admin

@app.route('/add\_admin', methods = ['POST'])

def add\_admin():

json = request.get\_json()

if user.identity!="admin":

return 'unauthorized party',400

contract\_keys = ['username','password', 'identity']

if not all(key in json for key in contract\_keys):

return 'Some elements of the transaction are missing', 400

username=json['username']

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("insert into user([username],[password],[identity]) VALUES(?,?,?)",(username,json['password'],json['identity']))

conn.commit()

cursor.execute("insert into notification([notification],[timestamp],[identity],[username]) VALUES(?,?,?,?)",(f'admin: {username} is added.',datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.close()

conn.close()

response = {'message': f'admin: {username} is added.'}

return jsonify(response), 201

#adding project

@app.route('/add\_project', methods = ['POST'])

def add\_project():

json = request.get\_json()

contract\_keys = ['project\_name','address']

if not all(key in json for key in contract\_keys):

return 'Some elements of the transaction are missing', 400

project\_name=json['project\_name']

project=Project()

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("insert into project([project\_name],[address]) VALUES(?,?)",(project\_name,json['address']))

conn.commit()

cursor.execute("insert into notification([notification],[timestamp],[identity],[username]) VALUES(?,?,?,?)",(f'project: {project\_name} is added.',datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.execute("select \* from project where project\_name=?",project\_name)

for row in cursor.fetchall():

project.set(row[0],row[1],row[2])

cursor.close()

conn.close()

index=blockchain.add\_project(project.get())

blockchain.add\_description({'message': f'project: {project\_name} is added'})

response = {'message': f'project: {project\_name} is added to block {index}'}

return jsonify(response), 201

#adding user(contractor,owner,consultant)

@app.route('/add\_user', methods = ['POST'])

def add\_user():

json = request.get\_json()

contract\_keys = ['username','password', 'companyName','identity','email']

if not all(key in json for key in contract\_keys):

return 'Some elements of the transaction are missing', 400

username=json['username']

identity=json['identity']

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("insert into user([username],[password],[companyName],[credit],[identity],[email]) VALUES(?,?,?,?,?,?)",(json['username'],json['password'],json['companyName'],0,json['identity'],json['email']))

conn.commit()

cursor.execute("insert into notification([notification],[timestamp],[identity],[username]) VALUES(?,?,?,?)",(f'contractor: {username} is added',datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.close()

conn.close()

index=blockchain.add\_description({'message': f'{identity}: {username} is added'})

response = {'message': f'contactor: {username} is added to block {index}.'}

return jsonify(response), 201

@app.route('/status\_update', methods = ['POST'])

def status\_update():

json = request.get\_json()

if user.identity!="contractor":

return 'unauthorized party',400

transaction\_keys = ['activity\_name','contract\_name','status']

if not all(key in json for key in transaction\_keys):

return 'Some elements of the transaction are missing', 400

activity=Activity()

contract\_name=json['contract\_name']

activity\_name=json['activity\_name']

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("update activity set status=? where activity\_name=? and contract\_name=?",(json['status'],json['activity\_name'],json['contract\_name']))

conn.commit()

cursor.execute("insert into notification([notification],[timestamp],[identity],[username]) VALUES(?,?,?,?)",(f'contractor updated status of activity: {activity\_name} in contract: {contract\_name}',datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.execute("select \* from activity where activity\_name=? and contract\_name=?",(activity\_name,contract\_name))

for row in cursor.fetchall():

activity.set(row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8],row[9],row[10])

cursor.close()

conn.close()

index=blockchain.add\_activity(activity.get())

blockchain.add\_user(user.username)

blockchain.add\_description({'message': f'contractor updated status of activity: {activity\_name} in contract: {contract\_name}'})

response = {'message': f'contractor updated status of activity: {activity\_name} in contract: {contract\_name} to block {index}'}

return jsonify(response), 201

#activity amount is paid out by owner

@app.route('/pay', methods = ['POST'])

def pay():

json = request.get\_json()

transaction\_keys = ['contract\_name','activity\_name','owner\_username','contractor\_username','retention']

owner\_username=json['owner\_username']

contract\_name=json['contract\_name']

contractor\_username=json['contractor\_username']

retention=json['retention']

if not all(key in json for key in transaction\_keys):

return 'Some elements of the transaction are missing', 400

activity\_name=json['activity\_name']

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("select price from activity where activity\_name=?",activity\_name)

amount=cursor.fetchone()

print(amount[0])

cursor.execute("select credit from user where username = ?",owner\_username)

credit=cursor.fetchone()

cal=amount[0]-(amount[0]\*retention/100)

credit[0]=credit[0]-cal# retention

cursor.execute("update user set credit = 0 where username = ?",contractor\_username)

conn.commit()

cursor.execute("update user set credit = ? where username =? ",(credit[0],owner\_username))

conn.commit()

cursor.execute("update user set credit = ? where username = ? ",(cal,contractor\_username))

conn.commit()

cursor.execute("insert into notification([notification],[contract\_name],[timestamp],[identity],[username]) VALUES(?,?,?,?,?)",( f'contractor:{contractor\_username} has received the activity amount {cal} of activity {activity\_name}',contract\_name,datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.close()

conn.close()

index=blockchain.add\_description({'message': f'contractor:{contractor\_username} has received the activity amount of contract {activity\_name}'})

blockchain.add\_user(user.username)

response = {'message': f'contractor:{contractor\_username} has received the activity amount {amount[0]} of contract {activity\_name} to block {index}'}

return jsonify(response), 201

#contract amount is paid out by owner

@app.route('/paycontract', methods = ['POST'])

def paycontract():

json = request.get\_json()

transaction\_keys = ['contract\_name','activity\_name','owner\_username','contractor\_username','retention']

owner\_username=json['owner\_username']

contract\_name=json['contract\_name']

contractor\_username=json['contractor\_username']

retention=json['retention']

if not all(key in json for key in transaction\_keys):

return 'Some elements of the transaction are missing', 400

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("select amount from contract where contract\_name=?",contract\_name)

amount=cursor.fetchone()

cursor.execute("select credit from user where username = ?",owner\_username)

credit=cursor.fetchone()

cal=amount[0]\*(retention\*0.01)

print(cal)

credit[0]=credit[0]-cal# retention

cursor.execute("update user set credit = 0 where username = ?",contractor\_username)

conn.commit()

cursor.execute("update user set credit = ? where username =? ",(credit[0],owner\_username))

conn.commit()

cursor.execute("update user set credit = ? where username = ? ",(cal,contractor\_username))

conn.commit()

cursor.execute("insert into notification([notification],[contract\_name],[timestamp],[identity],[username]) VALUES(?,?,?,?,?)",( f'contractor:{contractor\_username} has received the retention amount {cal} of contract {contract\_name}',contract\_name,datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.close()

conn.close()

index=blockchain.add\_description({'message': f'contractor:{contractor\_username} has received the retention amount of contract {contract\_name}'})

blockchain.add\_user(user.username)

response = {'message': f'contractor:{contractor\_username} has received the retention amount {retention} of contract {contract\_name} to block {index}'}

return jsonify(response), 201

#approving the contract by consultant

@app.route('/approve', methods = ['POST'])

def approve():

json = request.get\_json()

transaction\_keys = ['activity\_name','contract\_name','approved']

activity=Activity()

activity\_name=json['activity\_name']

contract\_name=json['contract\_name']

if not all(key in json for key in transaction\_keys):

return 'Some elements of the transaction are missing ', 400

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("update activity set approved=? where activity\_name=? and contract\_name=?",(json['approved'],json['activity\_name'],json['contract\_name']))

conn.commit()

cursor.execute("insert into notification([notification],[contract\_name],[timestamp],[identity],[username]) VALUES(?,?,?,?,?)",(f'Consultant name: {user.username} approved activity:{activity\_name} in contract no.{contract\_name}',contract\_name,datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.execute("select \* from activity where activity\_name=? and contract\_name=?",(activity\_name,contract\_name))

for row in cursor.fetchall():

activity.set(row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8],row[9],row[10])

cursor.close()

conn.close()

index=blockchain.add\_activity(activity.get())

blockchain.add\_user(user.username)

blockchain.add\_description({'message': f'Consultant {user.username} approved activity:{activity\_name} in contract: {contract\_name} '})

response = {'message':f'Consultant {user.username} approved activity:{activity\_name} in contract: {contract\_name} to block {index}'}

return jsonify(response), 201

#approving the contract by consultant

@app.route('/not\_approve', methods = ['POST'])

def not\_approve():

json = request.get\_json()

transaction\_keys = ['activity\_name','contract\_name','approved']

activity=Activity()

activity\_name=json['activity\_name']

contract\_name=json['contract\_name']

if not all(key in json for key in transaction\_keys):

return 'Some elements of the transaction are missing ', 400

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("update activity set approved=? where activity\_name=? and contract\_name=?",(json['approved'],json['activity\_name'],json['contract\_name']))

conn.commit()

cursor.execute("insert into notification([notification],[contract\_name],[timestamp],[identity],[username]) VALUES(?,?,?,?,?)",(f'Consultant name: {user.username} did not approve activity:{activity\_name} in contract no.{contract\_name}',contract\_name,datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.execute("select \* from activity where activity\_name=? and contract\_name=?",(activity\_name,contract\_name))

for row in cursor.fetchall():

activity.set(row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8],row[9],row[10])

cursor.close()

conn.close()

index=blockchain.add\_activity(activity.get())

blockchain.add\_user(user.username)

blockchain.add\_description({'message': f'Consultant {user.username} did not approved activity:{activity\_name} in contract: {contract\_name} '})

response = {'message':f'Consultant {user.username} did not approved activity:{activity\_name} in contract: {contract\_name} to block {index}'}

return jsonify(response), 201

# Adding a new transaction to the Blockchain

@app.route('/add\_transaction', methods = ['POST'])

def add\_transaction():

json = request.get\_json()

transaction\_keys = ['sender', 'receiver', 'amount']

if not all(key in json for key in transaction\_keys):

return 'Some elements of the transaction are missing', 400

index = blockchain.add\_transaction(json['sender'], json['receiver'], json['amount'])

response = {'message': f'This transaction will be added to Block {index}'}

return jsonify(response), 201

# Adding a new transaction to the Blockchain

@app.route('/add\_contract', methods = ['POST'])

def add\_contract():

json = request.get\_json()

"""if user.identity!="admin" or user.identity!="owner":

return 'unauthorized party',400"""

contract=Contract()

penalty=0

contract\_keys = ['contract\_no','contract\_name','project','amount', 'date', 'type','parties','status','penalty']

if not all(key in json for key in contract\_keys):

return 'Some elements of the transaction are missing', 400

contract\_name=json['contract\_name']

project\_name=json['project']

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("insert into contract([contract\_name],[project],[amount],[date],[type],[parties],[penalty],[status]) VALUES(?,?,?,?,?,?,?,?)",(contract\_name,json['project'],json['amount'],json['date'] , json['type'],json['parties'],penalty,"active"))

conn.commit()

cursor.execute("insert into notification([notification],[contract\_name],[timestamp],[identity],[username]) VALUES(?,?,?,?,?)",(f'Contract\_name:{contract\_name} is added to project: {project\_name}.',contract\_name,datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.execute("select \* from contract where contract\_name=?",contract\_name)

for row in cursor.fetchall():

contract.set(row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8])

cursor.close()

conn.close()

index=blockchain.add\_contract(contract.get())

blockchain.add\_user(user.username)

blockchain.add\_description({'message': f'contract:{contract\_name} is added to project: {project\_name}'})

response = {'message': f'contract:{contract\_name} is added to project: {project\_name} to block {index}'}

return jsonify(response), 201

#updating contract

@app.route('/update\_contract', methods = ['POST'])

def update\_contract():

json = request.get\_json()

if user.identity!="admin":

return 'unauthorized party',400

if not json['contract\_name']:

return 'Project name is missing', 400

contract=Contract()

contract\_name=json['contract\_name']

s=""

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

if json['amount']:

cursor.execute("update contract set amount=? where contract\_name=?",(json['amount'],contract\_name))

conn.commit()

s+="amount"

change=str(json['amount'])

if json['date']:

cursor.execute("update contract set [date]=? where contract\_name=?",(json['date'],contract\_name))

conn.commit()

s+=str(',')+" date"

change+=str(', ')+json['date']

if json['Type']:

cursor.execute("update contract set type=? where contract\_name=?",(json['Type'],contract\_name))

conn.commit()

s+=str(',')+" Type"

change+=str(', ')+str(json['Type'])

if json['parties']:

cursor.execute("update contract set parties=? where contract\_name=?",(json['parties'],contract\_name))

conn.commit()

s+=str(',')+" parties"

change+=str(',')+str(json['parties'])

cursor.execute("insert into notification([notification],[contract\_name],[timestamp],[identity],[username]) VALUES(?,?,?,?,?)",(f'Contract\_name:{contract\_name} ({s}) is changed to ({change}).',contract\_name,datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.execute("select \* from contract where contract\_name=?",contract\_name)

for row in cursor.fetchall():

contract.set(row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8])

cursor.close()

conn.close()

index=blockchain.add\_contract(contract.get())

blockchain.add\_user(user.username)

blockchain.add\_description({'message': f'Contract:contract\_name ({s}) is changed to block to ({change})'})

response = {'message': f'Contract:contract\_name ({s}) is changed to block to ({change}) to block {index}.'}

return jsonify(response), 201

#adding penalty

@app.route('/update\_penalty', methods = ['POST'])

def update\_penalty():

json = request.get\_json()

if user.identity!="admin":

return 'unauthorized party',400

contract\_keys = ['contract\_name','penalty']

if not all(key in json for key in contract\_keys):

return 'Some elements of the transaction are missing', 400

contract=Contract()

contract\_name=json['contract\_name']

penalty=json['penalty']

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

if json['penalty']:

cursor.execute("update contract set penalty=? where contract\_name=?",(json['penalty'],contract\_name))

conn.commit()

cursor.execute("insert into notification([notification],[contract\_name],[timestamp],[identity],[username]) VALUES(?,?,?,?,?)",(f'{penalty} penalty is added to contract:{contract\_name}',contract\_name,datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.execute("select \* from contract where contract\_name=?",contract\_name)

for row in cursor.fetchall():

contract.set(row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8])

cursor.close()

conn.close()

index=blockchain.add\_contract(contract.get())

blockchain.add\_user(user.username)

blockchain.add\_description({'message': f'{penalty} penalty is added to contract:{contract\_name}'})

response = {'message': f'{penalty} penalty is added to contract:{contract\_name} to block {index}'}

return jsonify(response), 201

#adding delays

@app.route('/add\_delay', methods = ['POST'])

def add\_delay():

json = request.get\_json()

if user.identity!="consultant":

return 'unauthorized party',400

transaction\_keys = ['contract\_name','activity\_name','delays']

if not all(key in json for key in transaction\_keys):

return 'Some elements of the transaction are missing ', 400

activity=Activity()

contract\_name=json['contract\_name']

activity\_name=json['activity\_name']

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("update activity set delays=? where contract\_name=?",(json['delays'],contract\_name,activity\_name))

conn.commit()

cursor.execute("insert into notification([notification],[contract\_name],[timestamp],[identity],[username]) VALUES(?,?,?,?,?)",(f'Consultant has added delays to activity:{activity\_name} and penalty to contract:{contract\_name}',contract\_name,datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.execute("select \* from activity where activity\_name=? and contract\_name=?",(activity\_name,contract\_name))

for row in cursor.fetchall():

activity.set(row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8],row[9],row[10])

cursor.close()

conn.close()

blockchain.add\_activity(activity.get())

blockchain.add\_user(user.username)

blockchain.add\_description({'message': f'Consultant has added delays to activity:{activity\_name} and penalty to contract:{contract\_name}'})

response = {'message': f'Consultant has added delays to activity:{activity\_name} and penalty to contract:{contract\_name}'}

return jsonify(response), 201

#removing contract

@app.route('/remove\_contract', methods = ['POST'])

def remove\_contract():

json = request.get\_json()

if user.identity!="admin":

return 'unauthorized party',400

contract\_keys = ['contract\_name']

if not all(key in json for key in contract\_keys):

return 'Some elements of the transaction are missing', 400

contract=Contract()

contract\_name=json['contract\_name']

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("select \* from contract where contract\_name=?",contract\_name)

for row in cursor.fetchall():

contract.set(row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8])

cursor.execute("delete from contract where contract\_name=?",contract\_name)

conn.commit()

cursor.execute("insert into notification([notification],[contract\_name],[timestamp],[identity],[username]) VALUES(?,?,?,?,?)",(f'contract:{contract\_name} is removed .',contract\_name,datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.close()

conn.close()

index=blockchain.add\_contract(contract.get())

blockchain.add\_user(user.username)

blockchain.add\_description({'message': f'contract:{contract\_name} is removed.'})

response = {'message': f'contract:{contract\_name} is removed to block {index}'}

return jsonify(response), 201

# adding activity to contract

@app.route('/add\_activity', methods = ['POST'])

def add\_activity():

json = request.get\_json()

act\_keys = ['project', 'activity\_name', 'duration','price','start\_date','finish\_date','status','approved','delays','contract\_name']

if not all(key in json for key in act\_keys):

return 'Some elements of the transaction are missing', 400

contract\_name=json['contract\_name']

activity\_name=json['activity\_name']

activity=Activity()

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("insert into activity([project],[activity\_name],[duration],[price],[start\_date],[finish\_date],[status],[approved],[delays],[contract\_name]) VALUES(?,?,?,?,?,?,?,?,?,?)",(json['project'],json['activity\_name'] ,json['duration'],json['price'],json['start\_date'],json['finish\_date'],json['status'],json['approved'],json['delays'],contract\_name))

conn.commit()

cursor.execute("insert into notification([notification],[contract\_name],[timestamp],[identity],[username]) VALUES(?,?,?,?,?)",(f'Activity:{activity\_name} is added to contract:{contract\_name}',contract\_name,datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.execute("select \* from activity where activity\_name=? and contract\_name=?",(activity\_name,contract\_name))

for row in cursor.fetchall():

activity.set(row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8],row[9],row[10])

cursor.close()

conn.close()

index=blockchain.add\_activity(activity.get())

blockchain.add\_user(user.username)

blockchain.add\_description({'message': f'Activity: {activity\_name} is added to contract: {contract\_name}'})

response = {'message': f'Activity: {activity\_name} is added to contract: {contract\_name} in block {index}'}

return jsonify(response), 201

#updating activity terms

@app.route('/update\_activity', methods = ['POST'])

def update\_activity():

json = request.get\_json()

if user.identity!="admin":

return 'unauthorized party',400

if not json['activity\_name']:

return 'Contract name is missing', 400

if not json['contract\_name']:

return 'activity name is missing', 400

contract\_name=json['contract\_name']

activity\_name=json['activity\_name']

activity=Activity()

s=""

change=''

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

if json['duration']:

cursor.execute("update activity set duration=? where contract\_name=? and activity\_name=?",(json['duration'],contract\_name,activity\_name))

conn.commit()

s+=" duration"

change=str(json['duration'])

if json['price']:

cursor.execute("update activity set price=? where contract\_name=? and activity\_name=?",(json['price'],contract\_name,activity\_name))

conn.commit()

s+=str(',')+" price"

change=str(',')+str(json['price'])

if json['start\_date']:

cursor.execute("update activity set [start\_date]=? where contract\_name=? and activity\_name=?",(json['start\_date'],contract\_name,activity\_name))

conn.commit()

s+=str(',')+" start\_date"

change+=str(',')+json['start\_date']

if json['finish\_date']:

cursor.execute("update activity set [finish\_date]=? where contract\_name=? and activity\_name=?",(json['finish\_date'],contract\_name,activity\_name))

conn.commit()

s+=str(',')+" finish\_date"

change+=str(',')+json['finish\_date']

cursor.execute("insert into notification([notification],[contract\_name],[timestamp],[identity],[username]) VALUES(?,?,?,?,?)",( f'Activity: {activity\_name} in Contract:{contract\_name} ({s}) is changed to ({change}).',contract\_name,datetime.datetime.now(),user.identity,user.username))

conn.commit()

cursor.execute("select \* from activity where activity\_name=? and contract\_name=?",(activity\_name,contract\_name))

for row in cursor.fetchall():

activity.set(row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8],row[9],row[10])

cursor.close()

conn.close()

index=blockchain.add\_activity(activity.get())

blockchain.add\_user(user.username)

blockchain.add\_description({'message': f'Activity: {activity\_name} in Contract:{contract\_name} ({s}) is changed to ({change})'})

response = {'message': f'Activity: {activity\_name} in Contract:{contract\_name} ({s}) is changed to ({change}) to block {index}.'}

return jsonify(response), 201

#removing act

@app.route('/remove\_activity', methods = ['POST'])

def remove\_activity():

json = request.get\_json()

if user.identity!="admin":

return 'unauthorized party',400

contract\_keys = ['activity\_name','contract\_name']

if not all(key in json for key in contract\_keys):

return 'Some elements of the transaction are missing', 400

activity=Activity()

contract\_name=json['contract\_name']

activity\_name=json['activity\_name']

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("select \* from activity where activity\_name=? and contract\_name=?",(activity\_name,contract\_name))

for row in cursor.fetchall():

activity.set(row[0],row[1],row[2],row[3],row[4],row[5],row[6],row[7],row[8],row[9],row[10])

cursor.execute("delete from activity where contract\_name=? and activity\_name=?",contract\_name,activity\_name)

conn.commit()

cursor.close()

conn.close()

index=blockchain.add\_activity(activity.get())

blockchain.add\_user(user.username)

blockchain.add\_description({'message': f'Activity: {activity\_name} is removed from contract: {contract\_name}'})

response = {'message': f'Activity:{activity\_name} is removed from contract: {contract\_name} to block {index}.'}

return jsonify(response), 201

# Connecting new nodes

@app.route('/connect\_node', methods = ['POST'])

def connect\_node():

json = request.get\_json()

nodes = json.get('nodes')

if nodes is None:

return "No node", 400

for node in nodes:

blockchain.add\_node(node)

response = {'message': 'All the nodes are now connected. The Unicoin Blockchain now contains the following nodes:',

'total\_nodes': list(blockchain.nodes)}

return jsonify(response), 201

# claim request

@app.route('/claim', methods = ['POST'])

def claim():

json = request.get\_json()

if not json['claim']:

return 'Claim is missing', 400

conn = pyodbc.connect(connection\_string)

conn.autocommit=True

cursor = conn.cursor()

cursor.execute("insert into claim([claim],[status],[username],[timestamp]) VALUES(?,?,?,?)",(json['claim'],"sent",user.username,datetime.datetime.now()))

conn.commit()

cursor.close()

conn.close()

blockchain.add\_user(user.username)

index=blockchain.add\_description({'message': f'Claim has been sent by contractor:{user.username}'})

response = {'message': f'Claim has been added by contractor:{user.username} to block {index}.'}

return jsonify(response), 201

# Replacing the chain by the longest chain if needed

@app.route('/replace\_chain', methods = ['GET'])

def replace\_chain():

response=None

network = blockchain.nodes

longest\_chain = None

max\_length = len(blockchain.chain)

is\_chain\_replaced=None

for node in network:

response0 = requests.get(f'http://{node}/get\_chain')

if response0.status\_code == 200:

length = response0.json()['length']

chain = response0.json()['chain']

if length >max\_length and blockchain.is\_chain\_valid(chain):

max\_length = length

longest\_chain = chain

if longest\_chain:

is\_chain\_replaced=blockchain.replace\_chain( longest\_chain)

if is\_chain\_replaced:

response = {'message': 'The nodes had different chains so the chain was replaced by the longest one.',

'new\_chain': blockchain.chain}

else:

response = {'message': 'All good. The chain is the largest one.',

'actual\_chain': blockchain.chain}

return jsonify(response), 200

# Running the app

app.run(host = '0.0.0.0', port = 5001)

**C# code**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class activities : Form

{

Httprequest request = new Httprequest();

DataTable act = new DataTable();

int n = 0;

public activities()

{

InitializeComponent();

this.CenterToScreen();

dynamic r = request.get(@"" + Program.url + "replace\_chain");

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

private void toolStripButton4\_Click(object sender, EventArgs e)

{

PrintDialog printDialog = new PrintDialog();

printDialog.Document = printDocument1;

printDialog.UseEXDialog = true;

//Get the document

if (DialogResult.OK == printDialog.ShowDialog())

{

printDocument1.DocumentName = "Test Page Print";

printDocument1.Print();

}

}

private void printDocument1\_PrintPage(object sender, System.Drawing.Printing.PrintPageEventArgs e)

{

Bitmap bm = new Bitmap(this.dataGridView1.Width, this.dataGridView1.Height);

dataGridView1.DrawToBitmap(bm, new Rectangle(0, 0, this.dataGridView1.Width, this.dataGridView1.Height));

e.Graphics.DrawImage(bm, 0, 0);

}

private void activities\_Load(object sender, EventArgs e)

{

dataGridView1.BorderStyle = BorderStyle.None;

dataGridView1.AlternatingRowsDefaultCellStyle.BackColor = Color.FromArgb(238, 239, 249);

dataGridView1.CellBorderStyle = DataGridViewCellBorderStyle.SingleHorizontal;

dataGridView1.DefaultCellStyle.SelectionBackColor = Color.DarkTurquoise;

dataGridView1.DefaultCellStyle.SelectionForeColor = Color.WhiteSmoke;

dataGridView1.BackgroundColor = Color.White;

dataGridView1.EnableHeadersVisualStyles = false;

dataGridView1.ColumnHeadersBorderStyle = DataGridViewHeaderBorderStyle.None;

dataGridView1.ColumnHeadersDefaultCellStyle.BackColor = Color.FromArgb(20, 25, 72);

dataGridView1.ColumnHeadersDefaultCellStyle.ForeColor = Color.White;

dynamic jsonResponse = request.get(@"" + Program.url + "get\_activities");

act.Columns.AddRange(

new DataColumn[11] { new DataColumn("activity\_name", typeof(string)),

new DataColumn("activity\_no", typeof(string)),

new DataColumn("project", typeof(string)),

new DataColumn("duration", typeof(string)),

new DataColumn("price", typeof(string)),

new DataColumn("start\_date", typeof(string)),

new DataColumn("finish\_date", typeof(string)),

new DataColumn("status", typeof(string)),

new DataColumn("approved", typeof(string)),

new DataColumn("delays", typeof(string)),

new DataColumn("contract\_name", typeof(string))

});

foreach (dynamic c in jsonResponse.activities)

{

act.Rows.Add(c.activity\_name,c.activity\_no, c.project, c.duration, c.price, c.start\_date, c.finish\_date, c.status, c.approved, c.delays, c.contract\_name);

n++;

}

dataGridView1.DataSource = act;

dataGridView1.Rows[n].Cells[0].Value = "Total:-";

dataGridView1.Rows[n].Cells[1].Value = dataGridView1.Rows.Count - 1;

dataGridView1.AutoResizeColumns(

DataGridViewAutoSizeColumnsMode.AllCells);

dataGridView1.Rows[n].DefaultCellStyle.BackColor = Color.FromArgb(20, 25, 72);

dataGridView1.Rows[n].DefaultCellStyle.ForeColor = Color.White;

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class users\_view : Form

{

public users\_view()

{

InitializeComponent();

}

private void button2\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Newtonsoft.Json;

using System.Net.Http;

using System.Net;

using System.IO;

namespace blockchain

{

public partial class Form1 : Form

{

Httprequest request = new Httprequest();

public Form1()

{

InitializeComponent();

this.CenterToScreen();

InitializeMyControl();

}

public dynamic get\_info()

{

dynamic jsonResponse = request.get(@""+Program.url+"get\_info");

return jsonResponse;

}

private void InitializeMyControl()

{

textBox2.PasswordChar = '\*';

textBox2.MaxLength = 14;

}

private void button1\_Click(object sender, EventArgs e)

{

Program.url = textBox3.Text;

label3.Text="";

label4.Text = "";

dynamic info;

dynamic r = request.get(@"" + Program.url + "replace\_chain");

Admin\_menu a = new Admin\_menu();

owner\_Menu o = new owner\_Menu();

contractor\_menu c = new contractor\_menu();

consultant\_menu cl = new consultant\_menu();

owner\_Menu u = new owner\_Menu();

if(textBox1.Text=="")

label3.Text = "\*username required";

if (textBox2.Text == "")

label4.Text = "\*password required";

Program.user.username = textBox1.Text;

Program.user.password = textBox2.Text;

if (textBox1.Text != "" &&textBox2.Text != "")

try

{

string response = request.post(@""+Program.url+"login", Program.user);

if (response.Contains("wrong username"))

label3.Text = "wrong username";

if (response.Contains("wrong password"))

label4.Text = "wrong password";

if (response.Contains("admin"))

{

info = get\_info();

Program.user.identity = info.identity;

Program.user.email = info.email;

a.ShowDialog();

}

else if (response.Contains("owner"))

{

info = get\_info();

Program.user.identity = "owner";

Program.user.companyName = info.companyName;

Program.user.credit = info.credit;

Program.user.email = info.email;

o.ShowDialog();

}

else if (response.Contains("consultant"))

{

info = get\_info();

Program.user.identity = "consultant";

Program.user.companyName = info.companyName;

Program.user.credit = info.credit;

Program.user.email = info.email;

cl.ShowDialog();

}

else if (response.Contains("contractor"))

{

info = get\_info();

Program.user.identity = "contractor";

Program.user.companyName = info.companyName;

Program.user.credit = info.credit;

Program.user.email = info.email;

c.ShowDialog();

}

}catch(Exception exception)

{

MessageBox.Show("Connection Error.",

"Important Note",

MessageBoxButtons.OK,

MessageBoxIcon.Exclamation,

MessageBoxDefaultButton.Button1);

}

}

private void button2\_Click(object sender, EventArgs e)

{

Users sign\_up = new Users();

sign\_up.ShowDialog();

}

private void button2\_Click\_1(object sender, EventArgs e)

{

this.Close();

}

private void linkLabel1\_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)

{

Users u = new Users();

u.ShowDialog();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class users\_view : Form

{

public users\_view()

{

InitializeComponent();

}

private void button2\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class owner\_Menu : Form

{

public owner\_Menu()

{

InitializeComponent();

this.CenterToScreen();

}

private void informationToolStripMenuItem\_Click(object sender, EventArgs e)

{

Information\_user user\_info = new Information\_user();

user\_info.ShowDialog();

}

private void projectToolStripMenuItem\_Click(object sender, EventArgs e)

{

Project\_Menu pm = new Project\_Menu();

pm.ShowDialog();

}

private void contractToolStripMenuItem\_Click(object sender, EventArgs e)

{

contract\_menu cm = new contract\_menu();

cm.ShowDialog();

}

private void notificationsToolStripMenuItem\_Click(object sender, EventArgs e)

{

Notification n = new Notification();

n.ShowDialog();

}

private void pToolStripMenuItem\_Click(object sender, EventArgs e)

{

payment p = new payment();

p.ShowDialog();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace blockchain

{

public class pay

{

public string activity\_name { get; set; }

public string owner\_username { get; set; }

public string contractor\_username { get; set; }

public float retention { get; set; }

public string contract\_name { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class payment : Form

{

Httprequest request = new Httprequest();

List<Activity> plist= new List<Activity>();

string contract\_n;

int count=0;

int flagc=0;

public payment()

{

InitializeComponent();

this.CenterToScreen();

dynamic jsonResponse = request.get(@"" + Program.url + "get\_projects");

dynamic r = request.get(@"" + Program.url + "replace\_chain");

foreach (dynamic c in jsonResponse.projects)

{

comboBox1.Items.Add(c.project\_name);

}

jsonResponse = request.get(@"" + Program.url + "get\_activities");

foreach (dynamic c in jsonResponse.activities)

{

count++;

Activity ac = new Activity();

comboBox2.Items.Add(c.activity\_name);

ac.contract\_name=c.contract\_name;

ac.activity\_name = c.activity\_name;

ac.status = c.status;

ac.approved = c.approved;

plist.Add(ac);

}

}

private void comboBox2\_SelectedIndexChanged(object sender, EventArgs e)

{

foreach (Activity f in plist)

if (f.activity\_name == (comboBox2.SelectedItem).ToString())

{

label3.Text = f.approved;

contract\_n = f.contract\_name;

}

}

private void button1\_Click(object sender, EventArgs e)

{

pay p = new pay();

p.owner\_username=Program.user.username;

p.retention = float.Parse(textBox1.Text);

p.activity\_name=comboBox2.Text;

p.contract\_name = contract\_n;

p.contractor\_username = "contractor";

foreach (Activity f in plist)

{

if (f.approved == "approved")

flagc++;

}

string response = request.post(@"" + Program.url + "pay", p);

dynamic r = request.get(@"" + Program.url + "mine\_block");

r = request.get(@"" + Program.url + "replace\_chain");

if (flagc == count)

{

string res = request.post(@"" + Program.url + "paycontract", p);

dynamic ro = request.get(@"" + Program.url + "mine\_block");

ro = request.get(@"" + Program.url + "replace\_chain");

}

MessageBox.Show("Payment sent");

}

private void label2\_Click(object sender, EventArgs e)

{

}

private void button2\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace blockchain

{

class Project

{

public int projectId { get; set; }

public string project\_name { get; set;}

public string address { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class users\_view : Form

{

public users\_view()

{

InitializeComponent();

}

private void button2\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace blockchain

{

public class status\_

{

public string activity\_name { get; set; }

public string contract\_name{get;set;}

public string status { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class status\_update : Form

{

Httprequest request = new Httprequest();

List<Activity> plist = new List<Activity>();

Activity act = new Activity();

public status\_update()

{

InitializeComponent();

this.CenterToScreen();

label3.Text = "not done";

dynamic jsonResponse = request.get(@"" + Program.url + "get\_projects");

dynamic r = request.get(@"" + Program.url + "replace\_chain");

foreach (dynamic c in jsonResponse.projects)

{

comboBox1.Items.Add(c.project\_name);

}

jsonResponse = request.get(@"" + Program.url + "get\_contracts");

foreach (dynamic c in jsonResponse.contracts)

{

comboBox2.Items.Add(c.contract\_name);

}

jsonResponse = request.get(@"" + Program.url + "get\_activities");

foreach (dynamic c in jsonResponse.activities)

{

comboBox3.Items.Add(c.activity\_name);

act.contract\_name = c.contract\_name;

act.activity\_name = c.activity\_name;

plist.Add(act);

}

}

private void button1\_Click(object sender, EventArgs e)

{

status\_ s = new status\_();

s.status = "done";

s.activity\_name = comboBox3.Text;

s.contract\_name = comboBox2.Text;

dynamic response = request.post(@"" + Program.url + "status\_update", s);

dynamic r = request.get(@"" + Program.url + "mine\_block");

r = request.get(@"" + Program.url + "replace\_chain");

MessageBox.Show("status update");

label3.Text = "done";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace blockchain

{

public class User

{

public int userId { get; set; }

public string username { get; set;}

public string password { get; set;}

public string companyName { get; set; }

public float credit { get; set;}

public string identity { get; set;}

public string email { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class user\_choice : Form

{

public user\_choice()

{

InitializeComponent();

this.CenterToScreen();

}

private void button3\_Click(object sender, EventArgs e)

{

Users users = new Users();

users.ShowDialog();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class Users : Form

{

Httprequest request = new Httprequest();

public Users()

{

InitializeComponent();

this.CenterToScreen();

dynamic r = request.get(@"" + Program.url + "replace\_chain");

}

private void button2\_Click(object sender, EventArgs e)

{

this.Close();

}

void clear()

{

label5.Text = "";

label6.Text = "";

label7.Text = "";

label8.Text = "";

label9.Text = "";

}

private void button1\_Click(object sender, EventArgs e)

{

clear();

User user= new User();

if (textBox1.Text != "")

user.username = textBox1.Text;

else

label5.Text = "\*username required";

if (textBox2.Text != "")

user.password = textBox2.Text;

else

label6.Text = "\*password required";

if (textBox3.Text != "")

user.companyName= textBox3.Text;

else

label7.Text = "\*company required";

if (comboBox1.Text != "")

user.identity = comboBox1.Text;

else

label8.Text = "\*identity required";

if (textBox4.Text != "")

user.email = textBox4.Text;

else

label9.Text = "\*email required";

if(textBox1.Text!=""&&textBox2.Text!=""&&textBox3.Text!=""&&comboBox1.Text!=""&&textBox4.Text!="")

{

string response = request.post(@""+Program.url+"add\_user", user);

dynamic jsonResponse = request.get(@""+Program.url+"mine\_block");

dynamic r = request.get(@"" + Program.url + "replace\_chain");

textBox5.Text = jsonResponse.ToString();

}

}

private void label10\_Click(object sender, EventArgs e)

{

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class information\_admin : Form

{

Form1 login = new Form1();

public information\_admin()

{

InitializeComponent();

this.CenterToScreen();

label3.Text = Program.user.username;

label4.Text = Program.user.identity;

label6.Text = (Program.user.credit).ToString();

label8.Text = Program.user.email;

}

private void button1\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Newtonsoft.Json;

using System.Net.Http;

using System.Net;

using System.IO;

namespace blockchain

{

public class Httprequest

{

public string post(string URL, Object item)

{

string result;

var client = new HttpClient();

string json = JsonConvert.SerializeObject(item);

var content = new StringContent(json, Encoding.UTF8, "application/json");

var response = client.PostAsync(URL, content).Result;

if (response.IsSuccessStatusCode)

{

//result = response.StatusCode.ToString();

var s = response.Content.ReadAsStringAsync();

string r = s.Result.ToString();

result = r;

}

else

{

var s = response.Content.ReadAsStringAsync();

string r = s.Result.ToString();

result = r;

}

return result;

}

public dynamic get(string URL)

{

JsonSerializer serializer = new JsonSerializer();

var client = new HttpClient();

var content = client.GetStringAsync(URL);

string response = content.Result.ToString();

dynamic jsonResponse = JsonConvert.DeserializeObject(response);

return jsonResponse;

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class contracts : Form

{

Httprequest request = new Httprequest();

DataTable cont = new DataTable();

int n = 0;

public contracts()

{

InitializeComponent();

this.CenterToScreen();

}

private void printDocument1\_PrintPage(object sender, System.Drawing.Printing.PrintPageEventArgs e)

{

Bitmap bm = new Bitmap(this.dataGridView1.Width, this.dataGridView1.Height);

dataGridView1.DrawToBitmap(bm, new Rectangle(0, 0, this.dataGridView1.Width, this.dataGridView1.Height));

e.Graphics.DrawImage(bm, 0, 0);

}

private void contracts\_Load(object sender, EventArgs e)

{

dataGridView1.BorderStyle = BorderStyle.None;

dataGridView1.AlternatingRowsDefaultCellStyle.BackColor = Color.FromArgb(238, 239, 249);

dataGridView1.CellBorderStyle = DataGridViewCellBorderStyle.SingleHorizontal;

dataGridView1.DefaultCellStyle.SelectionBackColor = Color.DarkTurquoise;

dataGridView1.DefaultCellStyle.SelectionForeColor = Color.WhiteSmoke;

dataGridView1.BackgroundColor = Color.White;

dataGridView1.EnableHeadersVisualStyles = false;

dataGridView1.ColumnHeadersBorderStyle = DataGridViewHeaderBorderStyle.None;

dataGridView1.ColumnHeadersDefaultCellStyle.BackColor = Color.FromArgb(20, 25, 72);

dataGridView1.ColumnHeadersDefaultCellStyle.ForeColor = Color.White;

dynamic jsonResponse = request.get(@""+Program.url+"get\_contracts");

cont.Columns.AddRange(

new DataColumn[9] { new DataColumn("Type", typeof(string)),

new DataColumn("amount", typeof(string)),

new DataColumn("contract\_name", typeof(string)),

new DataColumn("contract\_no", typeof(string)),

new DataColumn("date", typeof(string)),

new DataColumn("parties", typeof(string)),

new DataColumn("penlty", typeof(string)),

new DataColumn("project", typeof(string)),

new DataColumn("status", typeof(string)),

});

foreach (dynamic c in jsonResponse.contracts)

{

cont.Rows.Add(c.Type, c.amount, c.contract\_name, c.contract\_no, c.date, c.parties, c.penlty, c.project, c.status);

n++;

}

dataGridView1.DataSource = cont;

dataGridView1.Rows[n].Cells[0].Value = "Total:-";

dataGridView1.Rows[n].Cells[1].Value = dataGridView1.Rows.Count - 1;

dataGridView1.AutoResizeColumns(

DataGridViewAutoSizeColumnsMode.AllCells);

dataGridView1.Rows[n].DefaultCellStyle.BackColor = Color.FromArgb(20, 25, 72);

dataGridView1.Rows[n].DefaultCellStyle.ForeColor = Color.White;

}

private void toolStripButton4\_Click(object sender, EventArgs e)

{

PrintDialog printDialog = new PrintDialog();

printDialog.Document = printDocument1;

printDialog.UseEXDialog = true;

//Get the document

if (DialogResult.OK == printDialog.ShowDialog())

{

printDocument1.DocumentName = "Test Page Print";

printDocument1.Print();

}

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class contractor\_menu : Form

{

public contractor\_menu()

{

InitializeComponent();

this.CenterToScreen();

}

private void informationToolStripMenuItem\_Click(object sender, EventArgs e)

{

Information\_user iu= new Information\_user();

iu.ShowDialog();

}

private void projectToolStripMenuItem\_Click(object sender, EventArgs e)

{

Claim\_Menu cm = new Claim\_Menu();

cm.ShowDialog();

}

private void activitiesToolStripMenuItem\_Click(object sender, EventArgs e)

{

Activity\_Menu am = new Activity\_Menu();

am.ShowDialog();

}

private void notificationsToolStripMenuItem\_Click(object sender, EventArgs e)

{

Notification n = new Notification();

n.ShowDialog();

}

private void projectStatusToolStripMenuItem\_Click(object sender, EventArgs e)

{

status\_update f = new status\_update();

f.ShowDialog();

}

private void contractor\_menu\_Load(object sender, EventArgs e)

{

}

}

}

namespace blockchain

{

partial class contract\_menu

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.textBox3 = new System.Windows.Forms.TextBox();

this.textBox1 = new System.Windows.Forms.TextBox();

this.label3 = new System.Windows.Forms.Label();

this.label2 = new System.Windows.Forms.Label();

this.label1 = new System.Windows.Forms.Label();

this.groupBox1 = new System.Windows.Forms.GroupBox();

this.comboBox1 = new System.Windows.Forms.ComboBox();

this.textBox5 = new System.Windows.Forms.TextBox();

this.label6 = new System.Windows.Forms.Label();

this.textBox4 = new System.Windows.Forms.TextBox();

this.label5 = new System.Windows.Forms.Label();

this.label4 = new System.Windows.Forms.Label();

this.dateTimePicker1 = new System.Windows.Forms.DateTimePicker();

this.button2 = new System.Windows.Forms.Button();

this.button1 = new System.Windows.Forms.Button();

this.groupBox1.SuspendLayout();

this.SuspendLayout();

//

// textBox3

//

this.textBox3.Location = new System.Drawing.Point(162, 125);

this.textBox3.Margin = new System.Windows.Forms.Padding(4);

this.textBox3.Name = "textBox3";

this.textBox3.Size = new System.Drawing.Size(296, 22);

this.textBox3.TabIndex = 8;

//

// textBox1

//

this.textBox1.Location = new System.Drawing.Point(162, 40);

this.textBox1.Margin = new System.Windows.Forms.Padding(4);

this.textBox1.Name = "textBox1";

this.textBox1.Size = new System.Drawing.Size(296, 22);

this.textBox1.TabIndex = 6;

//

// label3

//

this.label3.AutoSize = true;

this.label3.Location = new System.Drawing.Point(59, 130);

this.label3.Margin = new System.Windows.Forms.Padding(4, 0, 4, 0);

this.label3.Name = "label3";

this.label3.Size = new System.Drawing.Size(60, 17);

this.label3.TabIndex = 2;

this.label3.Text = "Amount:";

//

// label2

//

this.label2.AutoSize = true;

this.label2.Location = new System.Drawing.Point(59, 86);

this.label2.Margin = new System.Windows.Forms.Padding(4, 0, 4, 0);

this.label2.Name = "label2";

this.label2.Size = new System.Drawing.Size(56, 17);

this.label2.TabIndex = 1;

this.label2.Text = "Project:";

//

// label1

//

this.label1.AutoSize = true;

this.label1.Location = new System.Drawing.Point(59, 43);

this.label1.Margin = new System.Windows.Forms.Padding(4, 0, 4, 0);

this.label1.Name = "label1";

this.label1.Size = new System.Drawing.Size(104, 17);

this.label1.TabIndex = 0;

this.label1.Text = "Contract name:";

this.label1.Click += new System.EventHandler(this.label1\_Click);

//

// groupBox1

//

this.groupBox1.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(192)))), ((int)(((byte)(192)))));

this.groupBox1.Controls.Add(this.comboBox1);

this.groupBox1.Controls.Add(this.textBox5);

this.groupBox1.Controls.Add(this.label6);

this.groupBox1.Controls.Add(this.textBox4);

this.groupBox1.Controls.Add(this.label5);

this.groupBox1.Controls.Add(this.label4);

this.groupBox1.Controls.Add(this.dateTimePicker1);

this.groupBox1.Controls.Add(this.textBox3);

this.groupBox1.Controls.Add(this.textBox1);

this.groupBox1.Controls.Add(this.label3);

this.groupBox1.Controls.Add(this.label2);

this.groupBox1.Controls.Add(this.label1);

this.groupBox1.Location = new System.Drawing.Point(45, 18);

this.groupBox1.Margin = new System.Windows.Forms.Padding(4);

this.groupBox1.Name = "groupBox1";

this.groupBox1.Padding = new System.Windows.Forms.Padding(4);

this.groupBox1.Size = new System.Drawing.Size(523, 322);

this.groupBox1.TabIndex = 21;

this.groupBox1.TabStop = false;

this.groupBox1.Text = "Information";

//

// comboBox1

//

this.comboBox1.FormattingEnabled = true;

this.comboBox1.Location = new System.Drawing.Point(162, 86);

this.comboBox1.Name = "comboBox1";

this.comboBox1.Size = new System.Drawing.Size(296, 24);

this.comboBox1.TabIndex = 20;

//

// textBox5

//

this.textBox5.Location = new System.Drawing.Point(162, 266);

this.textBox5.Margin = new System.Windows.Forms.Padding(4);

this.textBox5.Name = "textBox5";

this.textBox5.Size = new System.Drawing.Size(296, 22);

this.textBox5.TabIndex = 15;

//

// label6

//

this.label6.AutoSize = true;

this.label6.Location = new System.Drawing.Point(59, 271);

this.label6.Margin = new System.Windows.Forms.Padding(4, 0, 4, 0);

this.label6.Name = "label6";

this.label6.Size = new System.Drawing.Size(56, 17);

this.label6.TabIndex = 14;

this.label6.Text = "Parties:";

//

// textBox4

//

this.textBox4.Location = new System.Drawing.Point(162, 222);

this.textBox4.Margin = new System.Windows.Forms.Padding(4);

this.textBox4.Name = "textBox4";

this.textBox4.Size = new System.Drawing.Size(296, 22);

this.textBox4.TabIndex = 13;

//

// label5

//

this.label5.AutoSize = true;

this.label5.Location = new System.Drawing.Point(59, 222);

this.label5.Margin = new System.Windows.Forms.Padding(4, 0, 4, 0);

this.label5.Name = "label5";

this.label5.Size = new System.Drawing.Size(44, 17);

this.label5.TabIndex = 12;

this.label5.Text = "Type:";

//

// label4

//

this.label4.AutoSize = true;

this.label4.Location = new System.Drawing.Point(59, 178);

this.label4.Margin = new System.Windows.Forms.Padding(4, 0, 4, 0);

this.label4.Name = "label4";

this.label4.Size = new System.Drawing.Size(42, 17);

this.label4.TabIndex = 11;

this.label4.Text = "Date:";

this.label4.Click += new System.EventHandler(this.label4\_Click);

//

// dateTimePicker1

//

this.dateTimePicker1.Location = new System.Drawing.Point(162, 173);

this.dateTimePicker1.Margin = new System.Windows.Forms.Padding(4);

this.dateTimePicker1.Name = "dateTimePicker1";

this.dateTimePicker1.Size = new System.Drawing.Size(296, 22);

this.dateTimePicker1.TabIndex = 10;

//

// button2

//

this.button2.Location = new System.Drawing.Point(403, 369);

this.button2.Margin = new System.Windows.Forms.Padding(4);

this.button2.Name = "button2";

this.button2.Size = new System.Drawing.Size(100, 28);

this.button2.TabIndex = 23;

this.button2.Text = "Cancel";

this.button2.UseVisualStyleBackColor = true;

this.button2.Click += new System.EventHandler(this.button2\_Click);

//

// button1

//

this.button1.Location = new System.Drawing.Point(107, 369);

this.button1.Margin = new System.Windows.Forms.Padding(4);

this.button1.Name = "button1";

this.button1.Size = new System.Drawing.Size(100, 28);

this.button1.TabIndex = 22;

this.button1.Text = "Save";

this.button1.UseVisualStyleBackColor = true;

this.button1.Click += new System.EventHandler(this.button1\_Click);

//

// contract\_menu

//

this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(613, 413);

this.Controls.Add(this.button2);

this.Controls.Add(this.button1);

this.Controls.Add(this.groupBox1);

this.Margin = new System.Windows.Forms.Padding(4);

this.Name = "contract\_menu";

this.Text = "contract\_menu";

this.groupBox1.ResumeLayout(false);

this.groupBox1.PerformLayout();

this.ResumeLayout(false);

}

#endregion

private System.Windows.Forms.TextBox textBox3;

private System.Windows.Forms.TextBox textBox1;

private System.Windows.Forms.Label label3;

private System.Windows.Forms.Label label2;

private System.Windows.Forms.Label label1;

private System.Windows.Forms.GroupBox groupBox1;

private System.Windows.Forms.Label label4;

private System.Windows.Forms.DateTimePicker dateTimePicker1;

private System.Windows.Forms.TextBox textBox5;

private System.Windows.Forms.Label label6;

private System.Windows.Forms.TextBox textBox4;

private System.Windows.Forms.Label label5;

private System.Windows.Forms.Button button2;

private System.Windows.Forms.Button button1;

private System.Windows.Forms.ComboBox comboBox1;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace blockchain

{

class Contract

{

public int contract\_no { get; set; }

public string contract\_name { get; set; }

public string project { get; set; }

public float amount { get; set; }

public string date { get; set; }

public string type { get; set; }

public string parties { get; set; }

public int penalty { get; set; }

public string status { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class consultant\_menu : Form

{

public consultant\_menu()

{

InitializeComponent();

this.CenterToScreen();

}

private void informationToolStripMenuItem\_Click(object sender, EventArgs e)

{

Information\_user iu = new Information\_user();

iu.ShowDialog();

}

private void projectToolStripMenuItem\_Click(object sender, EventArgs e)

{

Project\_Menu pm = new Project\_Menu();

pm.ShowDialog();

}

private void projectToolStripMenuItem1\_Click(object sender, EventArgs e)

{

approval f = new approval();

f.ShowDialog();

}

private void notificationsToolStripMenuItem\_Click(object sender, EventArgs e)

{

Notification n = new Notification();

n.ShowDialog();

}

private void contractToolStripMenuItem\_Click(object sender, EventArgs e)

{

contract\_menu x = new contract\_menu();

x.ShowDialog();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace blockchain

{

class Claim

{

public int claimId { get; set; }

public string claim { get; set; }

public string status { get; set; }

public string username { get; set; }

public DateTime timestamp { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class Claim\_Menu : Form

{

Admin\_menu admin\_menu = new Admin\_menu();

Httprequest request = new Httprequest();

public Claim\_Menu()

{

InitializeComponent();

this.CenterToScreen();

}

private void button1\_Click(object sender, EventArgs e)

{

Claim claim = new Claim();

if (textBox1.Text != "")

{

claim.claim = textBox1.Text;

//claim.username = admin\_menu.user.username;

claim.status = "received";

claim.timestamp = DateTime.Now;

dynamic response = request.post(@"" + Program.url + "claim", claim);

dynamic r = request.get(@"" + Program.url + "mine\_block");

r = request.get(@"" + Program.url + "replace\_chain");

MessageBox.Show("Claim has been sent to admins for review.");

}

else

label2.Text = "Claim Required.";

}

private void button2\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class Chain : Form

{

Httprequest request = new Httprequest();

public Chain()

{

InitializeComponent();

this.CenterToScreen();

dynamic r = request.get(@"" + Program.url + "replace\_chain");

}

private void button2\_Click(object sender, EventArgs e)

{

this.Close();

}

private void button1\_Click(object sender, EventArgs e)

{

dynamic r = request.get(@"" + Program.url + "replace\_chain");

dynamic jsonResponse = request.get(@""+Program.url+"get\_chain");

textBox1.Text = jsonResponse.chain.ToString();

}

private void Chain\_Load(object sender, EventArgs e)

{

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class approval : Form

{

Httprequest request = new Httprequest();

List<Activity> plist = new List<Activity>();

public void activity\_update()

{

comboBox3.Items.Clear();

dynamic jsonResponse = request.get(@"" + Program.url + "get\_activities");

foreach (dynamic c in jsonResponse.activities)

{

Activity act = new Activity();

comboBox3.Items.Add(c.activity\_name);

act.contract\_name = c.contract\_name;

act.activity\_name = c.activity\_name;

act.approved = c.approved;

plist.Add(act);

}

label4.Text = "";

}

public approval()

{

InitializeComponent();

this.CenterToScreen();

dynamic r = request.get(@"" + Program.url + "replace\_chain");

dynamic jsonResponse = request.get(@"" + Program.url + "get\_projects");

foreach (dynamic c in jsonResponse.projects)

{

comboBox1.Items.Add(c.project\_name);

}

jsonResponse = request.get(@"" + Program.url + "get\_contracts");

foreach (dynamic c in jsonResponse.contracts)

{

comboBox2.Items.Add(c.contract\_name);

}

activity\_update();

}

private void button1\_Click(object sender, EventArgs e)

{

approve\_ p = new approve\_();

p.activity\_name = comboBox3.Text;

p.contract\_name = comboBox2.Text;

if (comboBox4.Text == "Approved")

{

p.approved = "approved";

dynamic response = request.post(@"" + Program.url + "approve", p);

dynamic r = request.get(@"" + Program.url + "mine\_block");

r = request.get(@"" + Program.url + "replace\_chain");

MessageBox.Show("approved");

}

else if (comboBox4.Text == "Not approved")

{

p.approved = "not approved";

dynamic response = request.post(@"" + Program.url + "not\_approve", p);

dynamic r = request.get(@"" + Program.url + "mine\_block");

r = request.get(@"" + Program.url + "replace\_chain");

MessageBox.Show("Not approved");

}

activity\_update();

}

private void button2\_Click(object sender, EventArgs e)

{

this.Close();

}

private void comboBox3\_SelectionChangeCommitted(object sender, EventArgs e)

{

}

private void comboBox3\_DisplayMemberChanged(object sender, EventArgs e)

{

}

private void comboBox3\_SelectedIndexChanged(object sender, EventArgs e)

{

foreach (Activity f in plist)

if (f.activity\_name == (comboBox3.SelectedItem).ToString())

{

label4.Text = f.approved;

}

}

private void comboBox3\_SelectedValueChanged(object sender, EventArgs e)

{

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class Admin\_menu : Form

{

Httprequest request = new Httprequest();

public Admin\_menu()

{

InitializeComponent();

this.CenterToScreen();

}

private void projectsToolStripMenuItem\_Click\_1(object sender, EventArgs e)

{

projects projects = new projects();

projects.ShowDialog();

}

private void contractsToolStripMenuItem\_Click\_1(object sender, EventArgs e)

{

contracts contracts = new contracts();

contracts.ShowDialog();

}

private void usersToolStripMenuItem\_Click\_1(object sender, EventArgs e)

{

users\_view uv = new users\_view();

uv.ShowDialog();

}

private void activitiesToolStripMenuItem\_Click\_1(object sender, EventArgs e)

{

activities activities = new activities();

activities.ShowDialog();

}

private void claimsToolStripMenuItem\_Click(object sender, EventArgs e)

{

claims claims = new claims();

claims.ShowDialog();

}

private void userToolStripMenuItem\_Click(object sender, EventArgs e)

{

user\_choice uc = new user\_choice();

uc.ShowDialog();

}

private void projectToolStripMenuItem\_Click(object sender, EventArgs e)

{

Project\_Menu project\_menu = new Project\_Menu();

project\_menu.ShowDialog();

}

private void activityToolStripMenuItem1\_Click(object sender, EventArgs e)

{

Activity\_Menu activity\_menu = new Activity\_Menu();

activity\_menu.ShowDialog();

}

private void informationToolStripMenuItem\_Click(object sender, EventArgs e)

{

information\_admin info\_admin = new information\_admin();

info\_admin.ShowDialog();

}

private void chainToolStripMenuItem\_Click(object sender, EventArgs e)

{

Chain chain = new Chain();

chain.ShowDialog();

}

private void Admin\_menu\_Load(object sender, EventArgs e)

{

}

private void contractToolStripMenuItem1\_Click(object sender, EventArgs e)

{

contract\_menu cm = new contract\_menu();

cm.ShowDialog();

}

private void notificationToolStripMenuItem\_Click(object sender, EventArgs e)

{

Notification n = new Notification();

n.ShowDialog();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace blockchain

{

class Activity

{

public int activity\_no { get; set; }

public string project { get; set; }

public string activity\_name { get; set; }

public string duration { get; set; }

public string start\_date { get; set; }

public string finish\_date { get; set; }

public string price { get; set; }

public string status { get; set; }

public string approved { get; set; }

public string delays { get; set; }

public string contract\_name { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace blockchain

{

public partial class Activity\_Menu : Form

{

Httprequest request = new Httprequest();

public Activity\_Menu()

{

InitializeComponent();

this.CenterToScreen();

dynamic r = request.get(@"" + Program.url + "replace\_chain");

dynamic jsonResponse = request.get(@"" + Program.url + "get\_projects");

foreach (dynamic c in jsonResponse.projects)

{

comboBox2.Items.Add(c.project\_name);

}

jsonResponse = request.get(@""+Program.url+"get\_contracts");

foreach (dynamic c in jsonResponse.contracts)

{

comboBox1.Items.Add(c.contract\_name);

}

}

private void button2\_Click(object sender, EventArgs e)

{

this.Close();

}

private void button1\_Click(object sender, EventArgs e)

{

Activity act= new Activity();

act.activity\_name = textBox1.Text;

act.project = comboBox2.Text;

act.duration = textBox3.Text;

act.start\_date = dateTimePicker1.Value.ToString() ;

act.finish\_date = dateTimePicker2.Value.ToString();

act.contract\_name = comboBox1.Text;

act.price = textBox4.Text;

act.status = textBox5.Text;

act.approved = textBox6.Text;

string response = request.post(@"" + Program.url + "add\_activity", act);

dynamic r = request.get(@"" + Program.url + "mine\_block");

r = request.get(@"" + Program.url + "replace\_chain");

MessageBox.Show("Activity added");

}

}

}