Eco Hotel Pomelia

Sistema di tracciamento dei consumi e dell’energia prodotta

Requisiti

Costruire un sistema di tracciamento dei consumi e dell'energia prodotta tramite pannelli fotovoltaici tramite applicazione web.

Questo sistema dovrà ricevere, a un endpoint specificato, richieste POST in JSON nel seguente formato:

```
{
        'produced_energy_in_watt':  121293434,
        'consumed_energy_in_watt': 239293
}
```

Queste richieste andranno poi visualizzate in forma tabella nell'applicazione web e verrà fatta partire una transazione su Ethereum Goerli contenente i due valori.

L’applicazione web dovrà avere:

- Una pagina principale, accessibile soltanto dagli utenti loggati, dove mostrare la tabella contenente i valori in questione e l’hash della transazione.
- Una pagina, alla quale soltanto gli amministratori possono accedere, dove è possibile vedere il totale consumato e prodotto.
- Un sistema di logging per memorizzare l’ultimo IP che ha avuto accesso alla piattaforma per un certo utente amministratore, in modo da mostrare un messaggio di avvertimento quando questo è diverso dal precedente.

Link utili

Riporto di seguito i vari link del progetto:

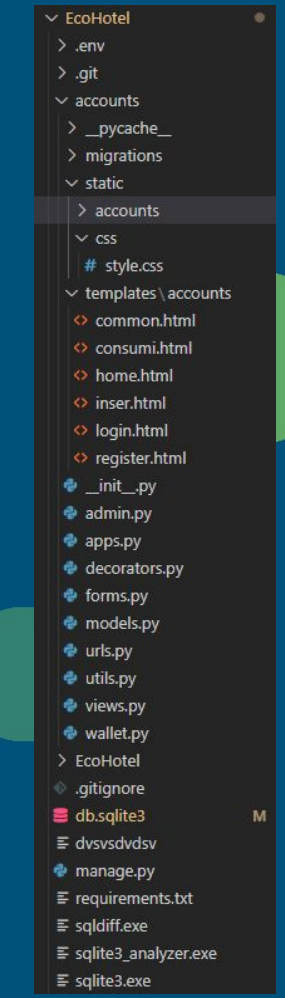
- Repository Github
- Link al sito web

Per l'accesso all'utente admin il nome utente e la password sono: ***admin***

Struttura progetto Django

L'applicazione Django all'interno del progetto EcoHotel si chiama *accounts* dentro questa applicazione sono presenti:

- la cartella *static* dove sono presenti tutti i file statici, come il logo del sito e il file *.css*
- la cartella *templates/accounts* dove sono presenti tutti i file *.html*
- i file *.py*



File models.py

```python
# Classe che attribuisce l'IP e la data di accesso
class IpAddress(models.Model):
    pub_date = models.DateTimeField('date published')
    ip_address = models.GenericIPAddressField()

# Classe per creare oggetti con relativi dati di consumo e energia prodotta,
# con il metodo writeOnChain si scrive nella chain i suddetti dati
class Consumi(models.Model):
    datetime = models.DateTimeField(auto_now_add=True)
    produced_energy_in_watt = models.TextField()
    consumed_energy_in_watt = models.TextField()
    hash = models.CharField(max_length=32, default=None, null=True)
    hash2 = models.CharField(max_length=32, default=None, null=True)
    txId=models.CharField(max_length=66, default=None, null=True)

    def writeOnChain(self):
        self.hash=hashlib.sha256(self.produced_energy_in_watt.encode('utf-8')).hexdigest()
        self.hash2=hashlib.sha256(self.consumed_energy_in_watt.encode('utf-8')).hexdigest()
        self.txId=sendTransaction(self.produced_energy_in_watt + '\n' + self.consumed_energy_in_watt)
        self.save()
```

Il file *models.py* contiene:

- la classe *IpAddress* che crea un oggetto nel db con l'Ip e la data attuale
- la classe *Consumi* crea un oggetto nel db con la data di inserimento, energia prodotta, consumata, l'hash dei due valori e l'id della transazione ed ha un metodo writeOnChain() che permette di scrivere nella chain di Goerli i suddetti dati.

File urls.py

```
urlpatterns = [
    path('accounts/login/', views.loginPage, name='login'),
    path('accounts/logout/', views.logoutUser, name='logout'),
    path('accounts/register/', views.registerPage, name='register'),
    path('consumi/', views.consumiPage, name='consumi'),
    path('home/', views.homePage, name='home'),
    path('inser/', views.inserData, name='inser'),
]
```

Il file *urls.py* contiene la lista *urlpatterns* dove son presenti tutti i collegamenti tra le varie views e il percorso del link per accederci.

File views.py

View loginPage

```python
# View pagina di login
@unauthenticated_user
@csrf_exempt
def loginPage(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(request, username=username, password=password)

        if user is not None:
            login(request, user)
            return redirect('home')
        else:
            messages.info(request, 'Username o Password sono sbagliati')

    return render(request, 'accounts/login.html')
```

Questa view prende le richieste POST della pagina di login e se le credenziali sono corrette indirizza l'utente alla Home.

Il decoratore *unauthenticated_user* fa in modo che se un utente loggato prova ad entrare nella pagina di login digitando il link nella barra di ricerca del browser questo venga reindirizzato alla pagina Home.

View registerPage

```python
# View pagina di registrazione
@unauthenticated_user
@csrf_exempt
def registerPage(request):

    if request.method == 'POST':
        form = CreateUserForm(request.POST)
        if form.is_valid():
            user = form.save()
            username = form.cleaned_data.get('username')

            group = Group.objects.get(name='customer')
            user.groups.add(group)

            messages.success(request, 'Account a nome ' +
                             username + ' è stato creato!')
            return redirect('login')

    context = {'form': form}
    return render(request, 'accounts/register.html', context)
```

Questa view prende le richieste POST della pagina di registrazione e crea un nuovo utente con i dati inseriti nel form.

Il decoratore unauthenticated_user fa in modo che se un utente loggato prova ad entrare nella pagina di registrazione digitando il link nella barra di ricerca del browser questo venga reindirizzato alla pagina Home.

View homePage

```python
# View home
@login_required(login_url='login')
@csrf_exempt
def homePage(request):

    # Memorizza l'ultimo IP che ha avuto accesso alla
    # piattaforma per un admin, mostra un messaggio di
    # avvertimento quando questo è diverso dal precedente
    checkIp={'checkIp': None}
    if request.user.is_staff:
        dbIp = IpAddress.objects.all().values().last()
        actualIp = getActualIP(request)

        if not dbIp:
            addIp(actualIp)
        else:
            if actualIp != dbIp['ip_address']:
                addIp(actualIp)
                checkIp={'checkIp': True}

    return render(request, 'accounts/home.html', checkIp)
```

Questa view mostra la HomePage, cioè la pagina che viene mostrata dopo essersi loggati.

In questa view si esegue il **controllo dell'IP**: viene mostrato un messaggio di avvertimento se l'IP di un utente admin che si logga è diverso dall'IP dell'utente admin che si è loggato precedentemente.

Il decoratore *login_required* impedisce agli utenti non loggati di accedere alla pagina.

La HomePage mostra un link testuale che porta alla tabella con i dati.
Se si è loggati con un profilo che possiede i privilegi di staff si vede, alla fine della tabella, anche il totale; si vede anche un altro link testuale nella HomePage che porta alla pagina dov'è possibile inserire i dati di energia prodotta e consumata e che verranno mandati al db e nella chain Goerli.

View consumiPage

```python
# View della pagina con la tabella dati
@csrf_exempt
@login_required(login_url='login')
def consumiPage(request):

    # Quando vede una request POST prende i dati e crea un oggetto nel
    # db e manda la transazione on chain (VIA POSTMAN)
    if request.method == "POST":
        body_unicode = request.body.decode('utf-8')
        body = json.loads(body_unicode)
        produced_energy_in_watt = body['produced_energy_in_watt']
        consumed_energy_in_watt = body['consumed_energy_in_watt']

        t = Consumi(produced_energy_in_watt=produced_energy_in_watt,
                    consumed_energy_in_watt=consumed_energy_in_watt)
        Consumi.writeOnChain(t)

    # Calcola il totale per mandarlo all'html
    data = Consumi.objects.all()
    totaleP = 0
    totaleC = 0
    for object in data:
        totaleP += int(object.produced_energy_in_watt)
        totaleC += int(object.consumed_energy_in_watt)

    return render(request, 'accounts/consumi.html',  {'data': data, 'totaleP': totaleP, 'totaleC': totaleC})
```

Questa view prende le richieste POST della pagina della tabella e crea un oggetto nel db con i dati energia prodotta e consumata e manda la transazione nella chain con i suddetti dati.

Il decoratore *login_required* impedisce agli utenti non loggati di entrare nella pagina.

View inserData

```python
# Inserisce nel database i dati che si inseriscono in inser.html nella textarea
def inserData(request):

    # Quando vede una request POST prende i dati e crea un oggetto
    # nel db e manda la transazione on chain
    if request.method == "POST":
        dati = request.POST.get('dati')
        dati = json.loads(dati)
        print(type(dati))
        produced_energy_in_watt = dati['produced_energy_in_watt']
        consumed_energy_in_watt = dati['consumed_energy_in_watt']

        t = Consumi(produced_energy_in_watt=produced_energy_in_watt,
                    consumed_energy_in_watt=consumed_energy_in_watt)
        Consumi.writeOnChain(t)

        messages.success(request, 'Dati inseriti nel database!')

        return redirect('home')

    context = {}
    return render(request, 'accounts/inser.html', context)
```

Questa view prende le richieste POST della pagina di inserimento dati e crea un oggetto nel db con i dati energia prodotta e consumata e manda la transazione nella chain con i suddetti dati.