EINSTEIN - DE LORENZO

Esame di Stato 2020-2021

Autore:
ROBERT-GEORGIAN NITU

Tutor:
Prof. LUIGI MENCHISE

May 31, 2021

Traccia

La società di distribuzione dell'energia elettrica intende sviluppare un archivio digitale di tutte le linee elettriche che gestisce. Il sistema deve permettere di mappe sul territorio la collocazione delle cabine di distribuzione e dei tralicci delle linee elettriche. Il candidato, formulate le opportune ipotesi aggiuntive sulle caratteristiche e la natura del problema in oggetto, sviluppi un'analisi della realtà di riferimento individuando quali devono essere le specifiche che il sistema deve soddisfare sia dal punto di vista software che dell'infrastruttura di rete. Sulla base delle specifiche individuate, illustri quali possono essere le soluzioni possibili e scegli a quella chiesto motivato giudizio è la più idonea a rispondere alle specifiche indicate.

Il candidato, quindi, sviluppi l'intero progetto del sistema informatico, in particolare riportando: lo schema a blocchi dei moduli del prodotto software da realizzare; il progetto del database completo dello schema concettuale, dello schema logico, delle istruzioni DDL necessarie per l'implementazione fisica del database e di alcune query necessarie per sviluppare dei moduli software individuati; il codice di una parte significativa del software in un linguaggio di programmazione a scelta del candidato.

Analisi della Realtà

1.1 Analisi del Problema

Molti Paesi hanno un modo diverso per distribuire l'energia elettrica. In Italia, per esempio, la Centrale Elettrica produce energia elettrica, che è distribuita ad una Cabina Elettrica ad alta tensione attraverso le Linee Elettriche che passano per i Tralicci. Gli Stati Uniti, però, hanno un approccio diverso:

1. L'elettricità è prodotta in una centrale composta da generatori i quali possono utilizzare il vento, il carbone, il gas naturale o l'acqua.
2. La corrente viene inviata ai trasformatori i quali alzano la tensione permettendo il trasferimento su lunghe distanze.
3. L'elettricitá passa attraverso le linee di trasmissione ad alta tensione che si estendono in tutto il paese.
4. Il flusso di corrente raggiunge una sottostazione, dove la tensione viene abbassata in modo da poter essere distribuita su linee elettriche più piccole.
5. Queste linee si estendono nel centro cittadino dove trasformatori più piccoli riducono nuovamente la tensione per renderla utilizzabile nelle abitazioni. Questi trasformatori più piccoli possono essere montati sui tralicci o posizionati a terra.
6. Si collega alle case e passa attraverso un contatore che misura il consumo di ogni cliente.
7. L'elettricità arriva al pannello di servizio il quale é fornito dell'interruttore magneto-termico che apre la rete in caso di cortocircuito, chiusura del circuito a terra o sovraccarico.
8. L'elettricità viaggia attraverso i fili all'interno dei muri fino alle prese e passa in tutta la casa.

Date queste differenze, solo le entità comuni a tutti i Paesi verranno memorizzate:

- Centrale Elettrica
- Traliccio
- Cabina Elettrica (ad alta tensione)
- Linea Elettrica

Il Software Permetterà agli amministratori di sistema di effettuare operazioni CRUD (Create, Read, Update e Delete) sulle entità.

1.2 L'azienda

L'unica cosa che sappiamo della società è che è una società di distribuzione di Energia Elettrica. Dato che la traccia la descrive come "La Società" e non ci sta una specificazione riguardanti la nazione, considero che l'azienda ha un monopolio sul mercato dell'energia elettrica nel mondo intero, semplicemente per lavorare sul worst-case ovvero la "peggior situazione" possibile.

Dato che non viene specificata come deve essere memorizzata la posizione delle entità, userò il sistema di coordinate GPS.

Siccome non viene specificato chi può accedere l'archivio, considero che solo gli amministratori di sistema possono accederci. Gli amministratori possono inserire ed eliminare i dati direttamente dalla pagina web dell'archivio, accessibile solo a loro.

Per fare l'autenticazione alla piattaforma, gli impiegati devono inserire le loro credenziali, e poi fare il riconoscimento facciale per confermare la loro identita'.
L'archivio sarà popolato originariamente con dati ufficiali da NASA, data.europa.eu, etc., e verranno citati nel codice sorgente.

1.3 L'infrastruttura

L'azienda ha varie sedi sparse nel mondo, i server sono collocati nel mare (come fa microsoft) aumentando cosí l'efficenza e riducendo i costi, essi sono anche divisi in diversi sotto domini tutti connessi allo stesso database.
Il DBMS scelto é PostgreSQL, per la sua scalabilitá esponenziale e per il supporto alla distribuzione in piú cluster.

Nell'analisi del problema sono presenti diversi sottodomini, quelli sviluppati in questo progetto sono:

- https://www.electrocorp.com/
- https://admin.electrocorp.com/
- https://api.electrocorp.com/
- https://mail.electrocorp.com/

(www.electrocorp.com risulta giá registrato da un'azienda non affiliata con questo progetto. Per questo il dominio deve essere sovrascritto in /etc/hosts per modificare il reindirizzamento al progetto, non puó essere applicato ad una macchina esterna alla rete).

I server utilizzano come OS SUSE Linux, una distribuzione Enterprise di GNU/Linux pensata per i server.
La rete si compone di diverse sottoreti, una per ogni cluster, tutte gestite da Kubernetes.
Ogni sottorete é protetta da alcune regole firewall, di seguito la lista:

Firewall on www.electrocorp.com					
Number	Protocol	Source IP	Destination IP	Destination Port	Action
4	ALL	0.0.0.0/0	0.0.0.0/0	0-65535	ACCEPT

Firewall on admin.electrocorp.com					
Number	Protocol	Source IP	Destination IP	Destination Port	Action
1	TCP	172.22.3.0/24	172.22.4.0/24	5432	ACCEPT
2	TCP	32.1.0.0/16	172.22.3.0/24	443	ACCEPT
3	ALL	0.0.0.0/0	0.0.0.0/0	0-65535	DROP

Firewall on api.electrocorp.com					
Number	Protocol	Source IP	Destination IP	Destination Port	Action
4	ALL	0.0.0.0/0	172.22.2.5	0-65535	ACCEPT

La sottorete 32.1.0.0/16 é gestita dalla compagnia ed é usata per permettere agli impiegati di connettersi all'area ristretta attraverso un RDP.

Firewall on the DBMS cluster					
Number	Protocol	Source IP	Destination IP	Destination Port	Action
1	TCP	172.22.2.0/24	172.22.4.0/24	5432	ACCEPT
2	TCP	172.22.4.0/24	172.22.4.0/24	5432	ACCEPT
3	ALL	0.0.0.0/0	0.0.0.0/0	0-65535	DROP

Di seguito la configurazione dei router (firewall, port forwarding) che girano Linux.

```
robert@workstation$ ssh root@electrocorp.com
root@www.electrocorp.com# iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --
    to 172.22.1.5:80
root@www.electrocorp.com# iptables -t nat -A PREROUTING -p tcp --dport 443 -j DNAT --
    to 172.22.1.5:443
root@www.electrocorp.com# logout
robert@workstation$ ssh root@admin.electrocorp.com
root@admin.electrocorp.com# iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT
    --to 172.22.3.5:80
root@admin.electrocorp.com# iptables -t nat -A PREROUTING -p tcp --dport 443 -j DNAT
    --to 172.22.3.5:443
root@admin.electrocorp.com# logout
robert@workstation$ ssh api@admin.electrocorp.com
root@api.electrocorp.com# iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --
    to 172.22.2.5:80
root@api.electrocorp.com# iptables -t nat -A PREROUTING -p tcp --dport 443 -j DNAT --
    to 172.22.2.5:443
root@api.electrocorp.com# # Facciamo lo static routing per avere la comunicazione fra
     il leader API e il DBMS.
root@api.electrocorp.com# ssh root@172.22.2.5
root@leader-172.22.2.5% ip route add 172.22.4.0/24 ia 172.22.2.1
root@leader-172.22.2.5% logout
root@api.electrocorp.com# ssh root@172.22.4.5
root@dbms-leader-172.22.4.5% ip route add 172.22.2.0/24 ia 172.22.4.1
root@dbms-leader-172.22.4.5% logout
robert@workstation$ logout
```

Gli host nella stessa sottorete possono solo interagire fra loro e con il DBMS cluster.

Questo é il docker-compose.yml, file di configurazione, che identifica anche le varie sottoreti.

```
version: '3.4'


services:
    api:
        hostname: 'api.electrocorp.com'
        build: './api/'
        links:
            - 'database'

        volumes:
            - './api/code:/usr/src/app:rw'

        networks:
            api_net:
                ipv4_address: 172.22.2.5

            database_net:
                ipv4_address: 172.22.4.2

        environment:
            SECRET_SALT: ^&?cY,t}pE>6rQu]$C#pRK^dns3B((}.e~*BzSgb5d7h*F;&FB/jD(>N\
    NSMX.b
            DATABASE_URL: postgresql://leader:mysecretpassword@172.22.4.5:5432/
    electrocorp

        ports:
            - '8555:80'

    admin:
        hostname: 'admin.electrocorp.com'
        build: './admin/'
        volumes:
            - './admin/code:/usr/src/app:ro'

        links:
            - 'api'

        networks:
            admin_net:
              ipv4_address: 172.22.3.5

    www:
        hostname: 'electrocorp.com'
        build: './www/'
        volumes:
            - './www/code:/usr/src/app:rw'

        links:
            - 'api'

        networks:
            www_net:
              ipv4_address: 172.22.1.5

        environment:
            FLASK_APP: run.py

    database:
        image: 'postgres:latest'
        environment:
            POSTGRES_PASSWORD: mysecretpassword
```

```
            POSTGRES_DB: electrocorp
            POSTGRES_USER: leader

        volumes:
            - psql_database:/var/lib/postgresql/data

        networks:
            database_net:
              ipv4_address: 172.22.4.5

    mail:
        hostname: 'mail.electrocorp.com'
        build: ./mail/
        volumes:
            - mail:/var/mail

        networks:
            mail_net:
                ipv4_address: 172.22.5.5



volumes:
    psql_database:
    mail:



networks:
    www_net:
        external:
            name: www

    api_net:
        external:
            name: api

    admin_net:
        external:
            name: admin

    database_net:
        external:
            name: database

    mail_net:
        external:
            name: mail
```

... e il file /etc/hosts (C: \Windows \system32 \etc \hosts)

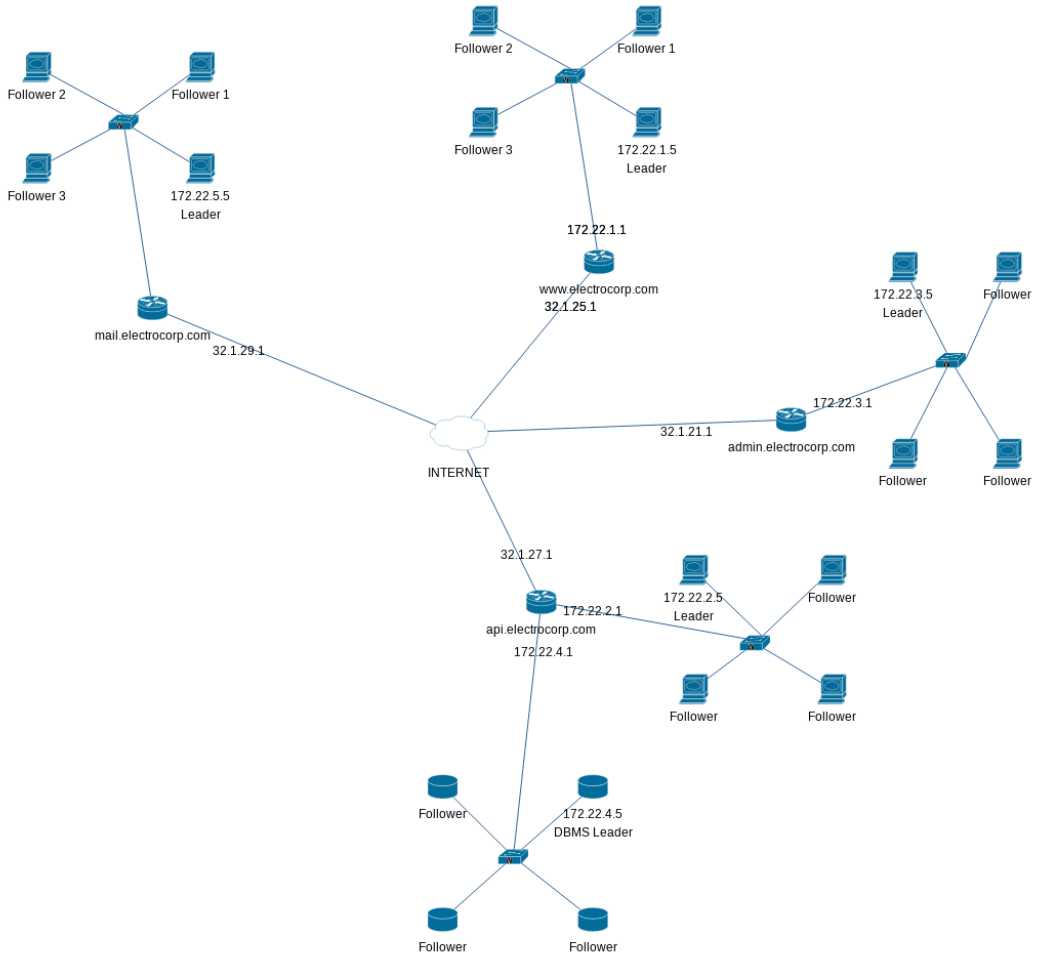
```
127.0.0.1 localhost
127.0.1.1 PwnStation

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters


```

```
12
13 # Progetto
14 172.22.2.5  api.electrocorp.com
15 172.22.1.5  electrocorp.com www.electrocorp.com
16 172.22.3.5  admin.electrocorp.com
17 172.22.5.5  mail.electrocorp.com
```

Ed ecco lo schema di rete:



1.4 Il sito

Il database é accessibile solo agli amministratori di sistema, i quali possono navigare, aggiungere, eliminare e modificare i dati dallo stesso.

Gli amministratori di sistema possono anche gestire i ruoli e gli utenti.

Queste sono le regole di accesso per gli utenti su ogni sito.

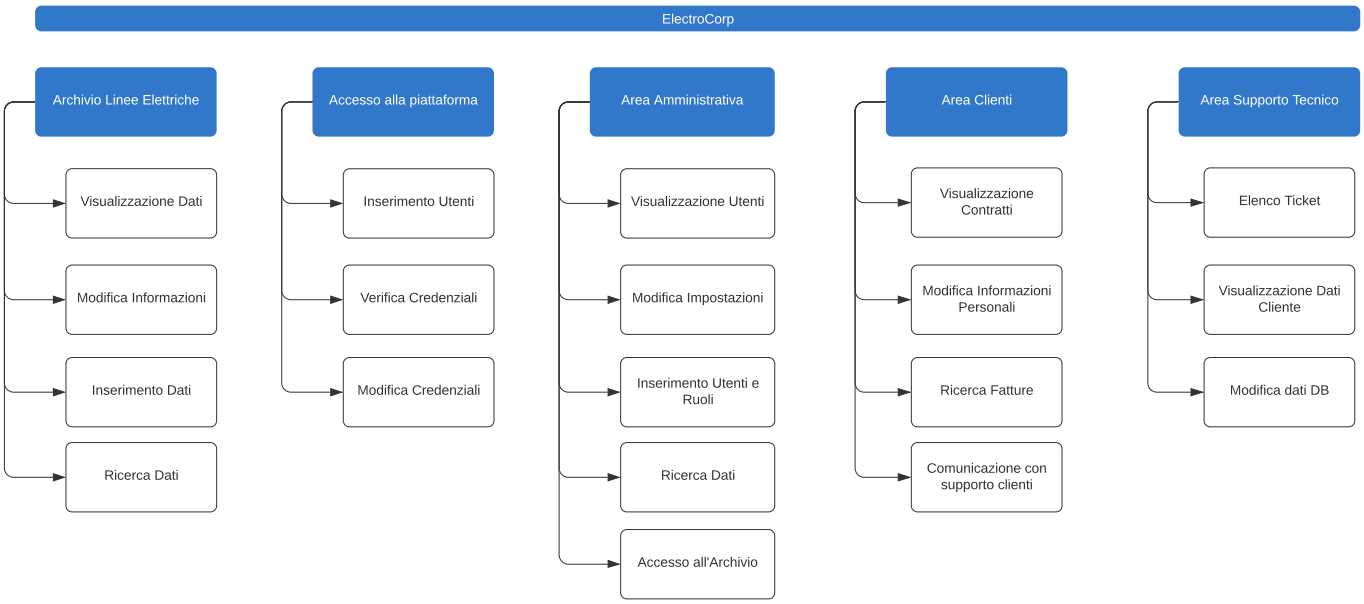
Access Control (admin.electrocorp.com)					
	Archive	Control Panel	Login	Finances	Logs
Administrator	ALL	ALL	AUTHENTICATE	ALL	READ
Finances			AUTHENTICATE	ALL	
not auth			AUTHENTICATE		

Accessibile solo dagli impiegati.

Questo e' il diagramma a blocchi.

Archivio

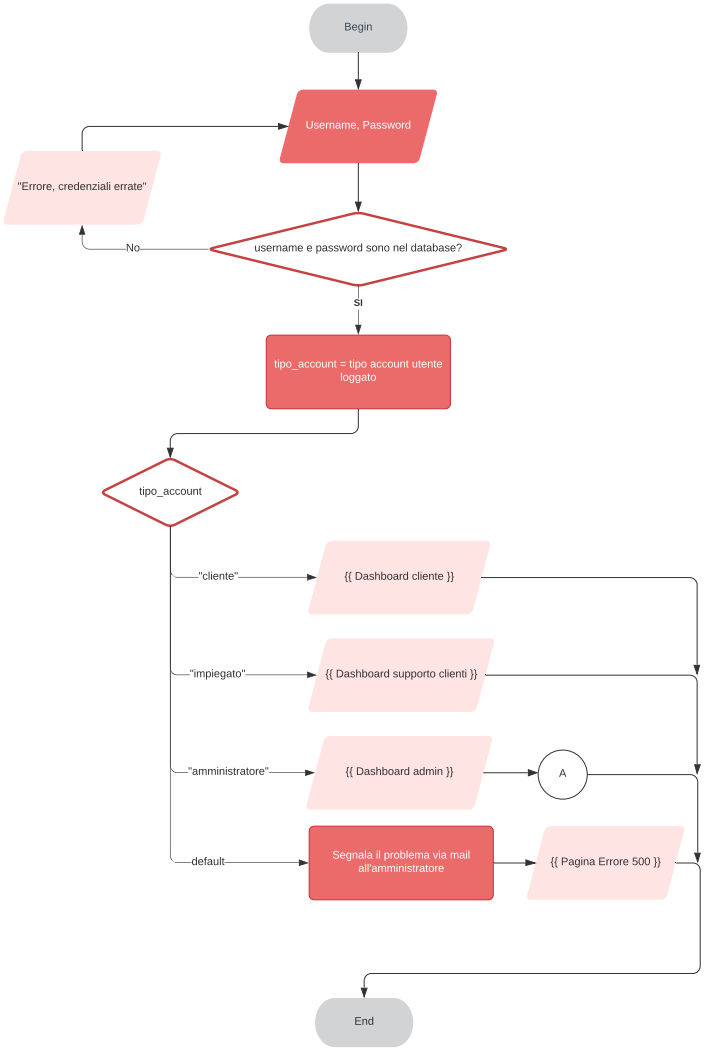
Zmlyc3Q bGFzdA | May 15, 2021

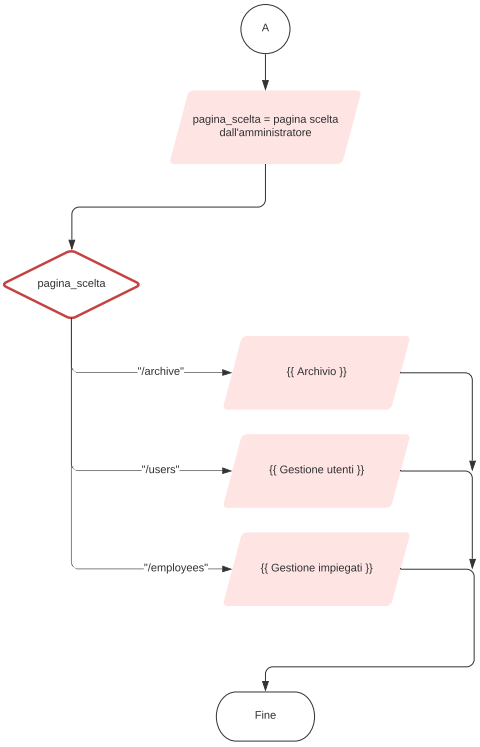


E questo e' il flow chart.

Diagramma di flusso dei moduli

Zmlyc3Q bGFzdA | May 15, 2021





Full Stack Development

2.1 Lo Stack

2.1.1 Docker

Docker è un insieme di prodotti PaaS (Platform as a Service) che utilizzano la virtualizzazione a livello di sistema operativo per fornire software in pacchetti chiamati contenitori. I contenitori sono isolati l'uno dall'altro e raggruppano i propri software, librerie e file di configurazione; possono comunicare tra loro attraverso canali ben definiti. Poiché tutti i contenitori condividono i servizi di un singolo kernel del sistema operativo, utilizzano meno risorse rispetto alle macchine virtuali.

Il servizio ha livelli sia gratuito che premium. Il software che ospita i contenitori si chiama Docker Engine. È stato avviato per la prima volta nel 2013 ed è sviluppato da Docker, Inc.

2.1.2 Flask su PyPy3

Flask è un micro framework web scritto in Python. È classificato come microframework perché non richiede strumenti o librerie particolari. Non ha un livello di astrazione del database, convalida di moduli o altri componenti in cui le librerie di terze parti preesistenti forniscono funzioni comuni. Tuttavia, Flask supporta estensioni che possono aggiungere funzionalità dell'applicazione come se fossero implementate in Flask stesso. Esistono estensioni per mappatori relazionali a oggetti, convalida di moduli, gestione del caricamento, varie tecnologie di autenticazione aperta e diversi strumenti comuni relativi al framework.

Le applicazioni che utilizzano il framework Flask includono Pinterest e LinkedIn.

Flask è composto da diversi componenti, fra cui:

- **Werkzeug**: Werkzeug (termine tedesco per "strumento") è una libreria per il linguaggio di programmazione Python, in altre parole un toolkit per applicazioni WSGI (Web Server Gateway Interface), ed è concesso in licenza con licenza BSD. Werkzeug può realizzare oggetti software per funzioni di richiesta, risposta e utilità. Può essere utilizzato per creare un framework software personalizzato su di esso e supporta Python 2.7 e 3.5 e versioni successive;
- **Jinja**: Jinja è un motore di modelli per il linguaggio di programmazione Python ed è concesso in licenza con licenza BSD. Simile al framework web Django, gestisce i modelli in una sandbox;
- **MarkupSafe**: MarkupSafe è una libreria per la gestione delle stringhe per il linguaggio di programmazione Python, con licenza BSD. L'omonimo tipo MarkupSafe estende il tipo di stringa Python e contrassegna il suo contenuto

come "sicuro"; la combinazione di MarkupSafe con stringhe regolari fa automaticamente l'escape delle stringhe non contrassegnate, evitando il doppio escape delle stringhe già contrassegnate;

- **ItsDangerous**: ItsDangerous è una libreria di serializzazione dei dati sicura per il linguaggio di programmazione Python, con licenza BSD. Viene utilizzato per memorizzare la sessione di un'applicazione Flask in un cookie senza consentire agli utenti di manomettere il contenuto della sessione.

2.1.3 SQLAlchemy

SQLAlchemy è un toolkit SQL open source e ORM (object-relational mapper) per il linguaggio di programmazione Python rilasciato sotto la licenza MIT.

La filosofia di SQLAlchemy è che i database relazionali si comportano meno come le raccolte di oggetti man mano che la scala diventa più grande e le prestazioni iniziano a essere un problema, mentre le raccolte di oggetti si comportano meno come tabelle e righe poiché viene progettata una maggiore astrazione. Per questo motivo ha adottato il modello di mappatura dati (simile a Hibernate per Java) piuttosto che il modello di registrazione attivo utilizzato da una serie di altri mappatori relazionali di oggetti. Tuttavia, i plugin opzionali consentono agli utenti di sviluppare utilizzando la sintassi dichiarativa.

SQLAlchemy è stato rilasciato per la prima volta nel febbraio 2006 ed è diventato rapidamente uno degli strumenti di mappatura relazionale a oggetti più utilizzati nella comunità Python, insieme a ORM di Django.

2.1.4 PostgreSQL

PostgreSQL, noto anche come Postgres, è un sistema di gestione del database relazionale (RDBMS) gratuito e open source che enfatizza l'estensibilità e la conformità SQL. Originariamente era chiamato POSTGRES, in riferimento alle sue origini come successore del database Ingres sviluppato presso l'Università della California, Berkeley. Nel 1996, il progetto è stato rinominato PostgreSQL per riflettere il suo supporto per SQL. Dopo una revisione nel 2007, il team di sviluppo ha deciso di mantenere il nome PostgreSQL e l'alias Postgres.

PostgreSQL offre transazioni con proprietà Atomicity, Consistency, Isolation, Durability (ACID), viste aggiornabili automaticamente, viste materializzate, trigger, chiavi esterne e stored procedure. È progettato per gestire una vasta gamma di carichi di lavoro, da singole macchine a data warehouse o servizi Web con molti utenti simultanei. È il database predefinito per macOS Server ed è disponibile anche per Windows, Linux, FreeBSD e OpenBSD.

2.1.5 React

React (noto anche come React.js o ReactJS) è una libreria JavaScript front-end open source per la creazione di interfacce utente o componenti dell'interfaccia utente. È gestito da Facebook e da una comunità di singoli sviluppatori e aziende. React può essere utilizzato come base per lo sviluppo di applicazioni a pagina singola o mobili. Tuttavia, React si occupa solo della gestione dello stato e del rendering di tale stato nel DOM, quindi la creazione di applicazioni React di solito richiede l'uso di librerie

aggiuntive per il routing, oltre a determinate funzionalità lato client.

2.2 Il Sito

2.2.1 Il server API

Questo è una parte del codice per il server API.

```
import json

from app import app, db
from flask import abort, jsonify, request, g
import models
import jwt
import traceback


# Simple endpoint for testing stuff
@app.route('/test', methods=['POST'])
def test():
    return jsonify({
        'context': g.data,
        'full_context': dir(g),
    })

# Authentication endpoint.
@app.route('/login/employee', methods=['POST'])
def login():
    error = None
    data = None
    try:
        l33t = request.get_json(force=True)
        user = models.Employee.query.filter(
            models.Employee.email == l33t.get('username', None)
        ).first()

        if user == None:
            error = 'User does not exist'

        elif user.password == l33t.get('password', None):
            data = jwt.encode({'id': user.id, 'role': user.role}, app.config['
    SECRET_KEY'], algorithm='HS256')
            foo = jsonify({
                'error': None,
                'data': data
            })
            foo.set_cookie('token', data)
            return foo

        else:
            error = 'Authentification Failed'

    except Exception as e:
        data = None
        error = str(e)

    finally:
        return jsonify({
            'error': error,
```

```
            'data': data
        })


# Power Plant endpoints.
@app.route('/power_plant/all', methods=['POST'])
def read_all_power_plants():
    data = []
    checked = check_authorization([1])
    if checked is not None:
        return checked

    for x in models.PowerPlant.query.all():
        data.append(x.toApiData())

    return jsonify({
        'error': None,
        'data': data,
    }), 200



@app.route('/power_plant/categories', methods=['GET'])
def get_power_plant_categories():
    # It does NOT require privileges to access this data.
    data = []
    for x in models.PowerPlantCategory.query.all():
        data.append(x.toApiData())

    return jsonify({
        'data': data
    })


@app.route('/power_plant/create', methods=['POST'])
def create_power_plant():
    error = None
    status = 200
    checked = check_authorization([1])
    if checked is not None:
        return checked

    data = request.get_json(force=True)
    try:
        foobar = models.PowerPlant(
            name=data['name'],
            description=data['desc'],
            category=int(data['category']),
            latitude=float(data['lat']),
            longitude=float(data['lng'])
        )

    except Exception as e:
        status = 400
        error = str(e)

    finally:
        return jsonify({
            error: error
        }), status


@app.route('/power_plant/<int:id>/read', methods=['POST'])
```

```
114 def read_power_plant_by(id: int):
115     checked = check_authorization([1])
116     if checked is not None:
117         return checked
118 
119     try:
120         return jsonify({
121             'error': None,
122             'data': models.PowerPlant.query.get(id).toApiData()
123         }), 200
124 
125     except Exception as e:
126         return jsonify({
127             'error': str(e),
128             'data': None
129         }), 404
130 
131 
132 @app.route('/power_plant/<int:id>/update', methods=['POST'])
133 def update_power_plant_by(id: int):
134     checked = check_authorization([1])
135     if checked is not None:
136         return checked
137 
138     try:
139         return jsonify({
140             'error': None,
141             'data': models.PowerPlant.query.get(id).toApiData()
142         }), 200
143 
144     except Exception as e:
145         return jsonify({
146             'error': str(e),
147             'data': None
148         }), 404
149 
150 
151 @app.route('/power_plant/<int:id>/delete', methods=['POST'])
152 def delete_power_plant_by(id: int):
153     checked = check_authorization([1])
154     if checked is not None:
155         return checked
156 
157     try:
158         foo = models.PowerPlant.query.get(id)
159         db.session.delete(foo)
160         db.session.commit()
161         return jsonify({
162             'error': None,
163         }), 200
164 
165     except Exception as e:
166         return jsonify({
167             'error': str(e)
168         }), 404
169 
170 
171 
172 # Power Cabin endpoints
173 @app.route('/power_cabin/all', methods=['POST'])
174 def read_all_power_cabins():
175     # checked = check_authorization([1])
176     # if checked is not None:
```

```
    #     return checked

    data = []
    for x in models.HighVoltagePowerCabin.query.all():
        data.append(x.toApiData())

    for x in models.LowVoltagePowerCabin.query.all():
        data.append(x.toApiData())

    return jsonify({
        'error': None,
        'data': data,
    }), 200


# @app.route('/power_cabin/categories', methods=['GET'])
# def get_power_cabin_categories():
#     # It does NOT require privileges to access this data.
#     data = []
#     for x in models.HighVoltagePowerCabinCategory.query.all():
#         data.append(x.toApiData())

#     return jsonify({
#         'data': data
#     })


@app.route('/power_cabin/<int:id>/read', methods=['POST'])
def read_power_cabin_by(id: int):
    checked = check_authorization([1])
    if checked is not None:
        return checked

    try:
        return jsonify({
            'error': None,
            'data': models.HighVoltagePowerCabin.query.get(id).toApiData()
        }), 200

    except Exception as e:
        return jsonify({
            'error': str(e),
            'data': None
        }), 404


@app.route('/power_cabin/<int:id>/update', methods=['POST'])
def update_power_cabin_by(id: int):
    checked = check_authorization([1])
    if checked is not None:
        return checked

    try:
        return jsonify({
            'error': None,
            'data': models.HighVoltagePowerCabin.query.get(id).toApiData()
        }), 200

    except Exception as e:
        return jsonify({
            'error': str(e),
            'data': None
        }), 404
```

```
240
241
242 @app.route('/power_cabin/<int:id>/delete', methods=['POST'])
243 def delete_power_cabin_by(id: int):
244     checked = check_authorization([1])
245     if checked is not None:
246         return checked
247
248     try:
249         foo = models.HighVoltagePowerCabin.query.get(id)
250         db.session.delete(foo)
251         db.session.commit()
252         return jsonify({
253             'error': None,
254         }), 200
255
256     except Exception as e:
257         return jsonify({
258             'error': str(e)
259         }), 404
260
261
262
263 # Pylon endpoints
264 @app.route('/pylon/all', methods=['POST'])
265 def read_all_pylons():
266     checked = check_authorization([1])
267     if checked is not None:
268         return checked
269
270     data = []
271     for x in models.Pylon.query.all():
272         print(x)
273         data.append(x.toApiData())
274
275     return jsonify({
276         'error': None,
277         'data': data,
278     }), 200
279
280
281 @app.route('/pylon/categories', methods=['GET'])
282 def get_pylon_categories():
283     # It does NOT require privileges to access this data.
284     data = []
285     for x in models.PylonCategory.query.all():
286         data.append(x.toApiData())
287
288     return jsonify({
289         'data': data
290     })
291
292
293 @app.route('/pylon/<int:id>/read', methods=['POST'])
294 def read_pylon_by(id: int):
295     checked = check_authorization([1])
296     if checked is not None:
297         return checked
298
299     try:
300         return jsonify({
301             'error': None,
302             'data': models.Pylon.query.get(id).toApiData()
```

```
303         }), 200
304
305     except Exception as e:
306         return jsonify({
307             'error': str(e),
308             'data': None
309         }), 404
310
311
312 @app.route('/pylon/<int:id>/update', methods=['POST'])
313 def update_pylon_by(id: int):
314     checked = check_authorization([1])
315     if checked is not None:
316         return checked
317
318     try:
319         return jsonify({
320             'error': None,
321             'data': models.Pylon.query.get(id).toApiData()
322         }), 200
323
324     except Exception as e:
325         return jsonify({
326             'error': str(e),
327             'data': None
328         }), 404
329
330
331 @app.route('/pylon/<int:id>/delete', methods=['POST'])
332 def delete_pylon_by(id: int):
333     checked = check_authorization([1])
334     if checked is not None:
335         return checked
336
337     try:
338         foo = models.Pylon.query.get(id)
339         db.session.delete(foo)
340         db.session.commit()
341         return jsonify({
342             'error': None,
343         }), 200
344
345     except Exception as e:
346         return jsonify({
347             'error': str(e)
348         }), 404
349
350
351 # Power Line endpoints.
352 @app.route('/power_line/all', methods=['POST'])
353 def read_all_power_lines():
354     error = None
355     data = []
356     # checked = check_authorization([1])
357     # if checked is not None:
358         # return checked
359
360     try:
361         data = []
362         for x in models.PowerLine.query.all():
363             data.append(x.toApiData())
364
365         for x in models.LowVoltagePowerCabin.query.all():
```

```
            source = models.HighVoltagePowerCabin.query.get(x.power_cabin)
            if source is None: continue
            data.append({
                'coords': [
                    [
                        source.latitude, source.longitude
                    ],

                    [
                        x.latitude, x.longitude
                    ]
                ],

                'properties': {
                    'id': '#',
                    'name': 'Distribution Line',
                    'color': '#8B008B',
                    'type': 'Distribution'
                }
            })

    except Exception as e:
        error = str(traceback.print_exc())
        data = []

    finally:
        return jsonify({
            'error': error,
            'data': data,
        }), 200


@app.route('/power_line/categories', methods=['GET'])
def get_power_line_categories():
    data = []
    for x in models.PowerLineCategory.query.all():
        data.append(x.toApiData())

    return jsonify({
        'data': data
    }), 200


# Users endpoint
@app.route('/user/all', methods=['POST'])
def get_users():
    checked = check_authorization([1])
    if checked is not None:
        return checked

    data = []
    for x in models.Employee.query.all():
        data.append(x.toApiData())

    return jsonify({
        'error': None,
        'data': data
    })


# Useful functions
def check_authorization(user_roles):
    if g.data == None or g.data.get('role', None) not in user_roles:
```

```
429         return jsonify({
430             'error': 'Unauthorized',
431             'data': str(g.data),
432         }), 401
433 
434     return None
```

Poiché abbiamo un server API (e un cluster di server), manterremo le informazioni in un token web. I token Web JSON (href https://jwt.io/ JWT) verranno utilizzati per archiviare i dati della sessione lato client in modo sicuro.
Questo è il middleware per l'utilizzo di JWT.

```
1  from app import app, db
2  from flask import redirect, request, session, jsonify, g
3  import models
4  import jwt
5  
6  
7  @app.before_request
8  def authorization_filter():
9      try:
10         token = request.get_json(force=True).get('token', None)
11         g.data = None
12 
13         if token is not None:
14             try:
15                 g.data = jwt.decode(token, app.config['SECRET_KEY'], algorithms=["
       HS256"])
16 
17             except Exception as e:
18                 return jsonify({'message': str(e)}), 401
19 
20         # else:
21         #     g.data = {
22         #         'id': 0,
23         #         'role': 0
24         #     }
25 
26     except Exception as e:
27         print('Error in authorization_filter')
28 
29 
30 @app.after_request
31 def set_headers_cors(response):
32     response.headers['Access-Control-Allow-Origin'] = '*'
33     # response.headers['Access-Control-Allow-Headers'] = '*'
34     return response
```

2.2.2 Il Frontend

Entrambi i frontend di admin.electrocorp.com e (www.)electrocorp.com sono sviluppati con React. Userò diverse librerie, fra cui:

- Tailwind.css: un framework web (tipo Bootstrap), che mi solleva dal dover scrivere un sacco di CSS e utilizzare Tailwind direttamente nella mia pagina HTML;

- React Router: una libreria di React.js per renderizzare più pagine nello stesso codice;
- Globe.gl: una libreria di React.js per renderizzare un globo.

Questo è il codice di App.js:

```
import logo from './logo.svg';
import './App.css';
import {
    BrowserRouter as Router,
    Switch,
    Route,
    Link
} from "react-router-dom";
import {useState} from 'react'

// Import the views.
import LoginScreen from './views/Login'
import ArchiveScreen from './views/Archive'
import DashboardScreen from './views/Dashboard'
import TestScreen from './views/Test'
import LogoutScreen from './views/Logout'


// Import the error views.
import Error404 from './views/Error404'


// The code
function App() {
    return (
        <Router>
            <Switch>
                <Route exact path='/'>
                    <LoginScreen />
                </Route>

                <Route exact path='/login'>
                    <LoginScreen />
                </Route>

                <Route exact path='/dashboard'>
                    <DashboardScreen />
                </Route>

                <Route exact path='/archive'>
                    <ArchiveScreen />
                </Route>

                <Route exact path='/test'>
                    <TestScreen />
                </Route>

                <Route exact path='/logout'>
                    <LogoutScreen />
                </Route>

                <Route path='*'>
                    <Error404 />
                </Route>
            </Switch>
        </Router>
```

```
    );
}

export default App;
```

Amministrazione di Sistema

3.1 Docker Compose

Docker Compose è uno strumento per la definizione e l'esecuzione di applicazioni Docker multi-container. Utilizza i file YAML per configurare i servizi dell'applicazione ed esegue il processo di creazione e avvio di tutti i contenitori con un unico comando. L'utilità CLIedocker-compose consente agli utenti di eseguire comandi su più contenitori contemporaneamente, ad esempio, la creazione di immagini, il ridimensionamento di contenitori, l'esecuzione di contenitori interrotti e altro ancora. I comandi relativi alla manipolazione delle immagini, o le opzioni interattive dell'utente, non sono rilevanti in Docker Compose perché indirizzano un contenitore. Il file docker-compose.yml viene utilizzato per definire i servizi di un'applicazione e include varie opzioni di configurazione. Ad esempio, l'opzione build definisce opzioni di configurazione come il percorso Dockerfile, l'opzione comando consente di sovrascrivere i comandi Docker predefiniti e altro ancora. La prima versione beta pubblica di Docker Com-pose (versione 0.0.1) è stata rilasciata il 21 dicembre 2013. La prima versione pronta per la produzione (1.0) è stata resa disponibile il 16 ottobre 2014.

3.2 Linux

Linux è una famiglia di sistemi operativi Unix open-source basati sul kernel Linux, un kernel del sistema operativo rilasciato per la prima volta il 17 settembre 1991 da Linus Torvalds. Linux è tipicamente impacchettato in una distribuzione Linux.

Le distribuzioni includono il kernel Linux e il software di sistema di supporto e le librerie, molte delle quali sono fornite dal progetto GNU. Molte distribuzioni Linux usano la parola "Linux" nel loro nome, ma la Free Software Foundation usa il nome "GNU / Linux" per enfatizzare l'importanza del software GNU, causando alcune controversie.

3.3 iptables

iptables è un programma di utilità per lo spazio utente che consente a un amministratore di sistema di configurare le regole del filtro dei pacchetti IP del firewall del kernel Linux, implementate come diversi moduli Netfilter. I filtri sono organizzati in diverse tabelle, che contengono catene di regole su come trattare i pacchetti di traffico di rete. Diversi moduli e programmi del kernel sono attualmente utilizzati per diversi protocolli; iptables si applica a IPv4, ip6tables a IPv6, arptables a ARP ed ebtables a frame Ethernet.

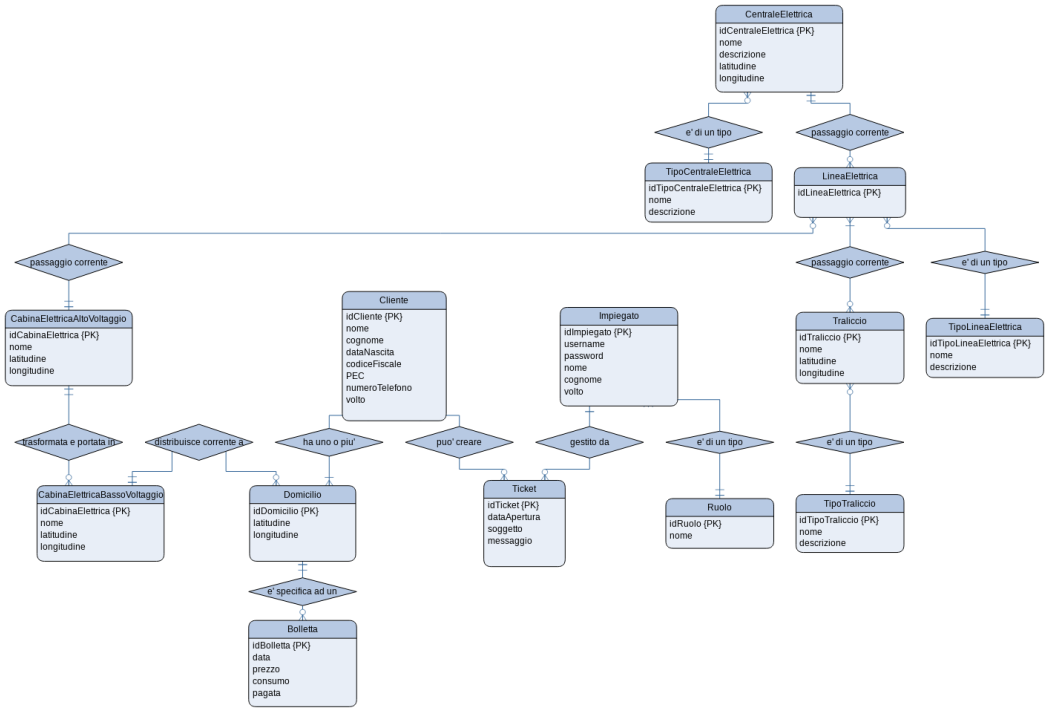
iptables richiede privilegi elevati per funzionare e deve essere eseguito dall'utente root, altrimenti non funziona. Sulla maggior parte dei sistemi Linux, iptables è installato come / usr / sbin / iptables e documentato nelle sue pagine man, che

possono essere aperte usando man iptables una volta installato. Può anche essere trovato in / sbin / iptables, ma poiché iptables è più simile a un servizio piuttosto che a un "binario essenziale", la posizione preferita rimane / usr / sbin.

Moduli del Software

4.1 Normalizzazione Database

4.1.1 Schema ER



4.1.2 1FN

Lo schema e' in 1a forma normale in quanto tutti gli attributi sono atomici e dello stesso tipo.
La prima forma normale è la più semplice, serve solo rispettare le regole di base per avere questa prima forma, le quali sono:

- Utilizzo di campi elementari e non composti
- Stesso numero di righe e colonne
- Record univoci

4.1.3 2FN

Dato che non abbiamo chiavi composte, non dobbiamo fare nulla per controllare la 2a forma normale.

4.1.3 3FN

Tutte le colonne dipendono direttamente dalla chiave primaria, e' ridotto in 3a forma normale.

4.2 Schema Logico

CentraleElettrica						
Name	Type	Size	Key	Null	Default	Extra
idCentrale	INTEGER		PRIMARY	NO		AUTO INCREMENT
nome	VARCHAR	128		NO		
descrizione	VARCHAR	4096		NO		
latitudine	FLOAT			NO		
longitudine	FLOAT			NO		
idTipoCentraleElettrica	INTEGER		FOREIGN	NO		

LineaElettrica						
Name	Type	Size	Key	Null	Default	Extra
idLineaElettrica	INTEGER		PRIMARY	NO		AUTO INCREMENT
origine	INTEGER		FOREIGN	NO		
destinazione	INTEGER		FOREIGN	NO		
idTipoLineaElettrica	INTEGER		FOREIGN	NO		

CabinaElettricaAltoVoltaggio						
Name	Type	Size	Key	Null	Default	Extra
idCabina	INTEGER		PRIMARY	NO		AUTO INCREMENT
nome	VARCHAR	128		YES		
latitudine	FLOAT			NO		
longitudine	FLOAT			NO		

CabinaElettricaBassoVoltaggio						
Name	Type	Size	Key	Null	Default	Extra
idCabina	INTEGER		PRIMARY	NO		AUTO INCREMENT
nome	VARCHAR	128		YES		
latitudine	FLOAT			NO		
longitudine	FLOAT			NO		
idCabinaAltoVoltaggio	INTEGER		FOREIGN	NO		

PassaggioLinea						
Name	Type	Size	Key	Null	Default	Extra
idPassaggio	INTEGER		PRIMARY	NO		AUTO INCREMENT
idLinea	INTEGER		FOREIGN	NO	1	
idTraliccioOrigine	INTEGER		FOREIGN	SI		
idTraliccioDestinazione	INTEGER		FOREIGN	SI		

Traliccio						
Name	Type	Size	Key	Null	Default	Extra
idTraliccio	INTEGER		PRIMARY	NO		AUTO INCREMENT
nome	VARCHAR	64		SI		
latitude	FLOAT			NO		
longitude	FLOAT			NO		
idTipoTraliccio	INTEGER		FOREIGN	NO		

Cliente						
Name	Type	Size	Key	Null	Default	Extra
idCliente	INTEGER		PRIMARY	NO		AUTO INCREMENT
nome	VARCHAR	64		NO		
cognome	VARCHAR	64		NO		
dataNascita	DATE			NO		
codiceFiscale	VARCHAR	16		NO		UNIQUE
numeroTelefono	VARCHAR	16		NO		UNIQUE
PEC	VARCHAR	300		NO		UNIQUE
password	VARCHAR	256		NO		
volto	BLOB			NO		UNIQUE

Domicilio						
Name	Type	Size	Key	Null	Default	Extra
idDomicilio	INTEGER		PRIMARY	NO		AUTO INCREMEN
idCliente	INTEGER		FOREIGN	NO		
idCabinaElettricaBassoVoltaggio	INTEGER		FOREIGN	NO		
latitudine	FLOAT			NO		
longitudine	FLOAT			NO		

Bolletta						
Name	Type	Size	Key	Null	Default	Extra
idBolletta	INTEGER		PRIMARY	NO		AUTO INCREMENT
idDomicilio	INTEGER		FOREIGN	NO		
consumoWatt	FLOAT			NO	0	
dataBolletta	DATE			NO	CURDATE	
pagata	BOOLEAN			NO	FALSE	

Impiegato						
Name	Type	Size	Key	Null	Default	Extra
idImpiegato	INTEGER		PRIMARY	NO		AUTO INCREMENT
username	VARCHAR	300		NO		UNIQUE
password	CHAR	128		NO		
nome	VARCHAR	64		NO		
cognome	VARCHAR	64		NO		
ruolo	INTEGER		FOREIGN	NO		
volto	BLOB			NO		UNIQUE

Ticket						
Name	Type	Size	Key	Null	Default	Extra
idTicket	INTEGER		PRIMARY	NO		AUTO INCREMENT
dataApertura	DATETIME			NO		
soggetto	VARCHAR	300		NO		
messaggio	VARCHAR	8192		NO		
idImpiegato	INTEGER		FOREIGN	SI		
idCliente	INTEGER		FOREIGN	NO		

4.3 DDL

Metodo Tradizionale

Questo progetto non usa il metodo "tradizionale" per creare le query SQL.
Di seguito, però, mostro le istruzioni DDL per creare il database.

```
CREATE DATABASE IF NOT EXISTS ARCHIVE;
USE ARCHIVE;

DROP TABLE IF EXISTS roles;
DROP TABLE IF EXISTS employees;

DROP TABLE IF EXISTS power_plant_categories;
DROP TABLE IF EXISTS pylon_categories;
DROP TABLE IF EXISTS power_line_categories;

DROP TABLE IF EXISTS low_voltage_power_cabins;
DROP TABLE IF EXISTS high_voltage_power_cabins;
DROP TABLE IF EXISTS power_plants;
DROP TABLE IF EXISTS power_lines;
DROP TABLE IF EXISTS pylons;

DROP TABLE IF EXISTS customers;
DROP TABLE IF EXISTS residences;
DROP TABLE IF EXISTS bills;
DROP TABLE IF EXISTS tickets;


CREATE TABLE roles (
        id INTEGER NOT NULL,
        date_created DATETIME,
        date_modified DATETIME,
        name VARCHAR(128) NOT NULL,
        description VARCHAR(409) NOT NULL,
        PRIMARY KEY (id),
        UNIQUE (name)
);


CREATE TABLE high_voltage_power_cabins (
        id INTEGER NOT NULL,
        date_created DATETIME,
        date_modified DATETIME,
        name VARCHAR(128),
        latitude FLOAT NOT NULL,
        longitude FLOAT NOT NULL,
        PRIMARY KEY (id)
```

```
);


CREATE TABLE customers (
        id INTEGER NOT NULL,
        date_created DATETIME,
        date_modified DATETIME,
        name VARCHAR(64) NOT NULL,
        surname VARCHAR(128) NOT NULL,
        birthdate DATETIME NOT NULL,
        fiscal_code VARCHAR(16) NOT NULL,
        pec VARCHAR(300) NOT NULL,
        phone_number VARCHAR(20) NOT NULL,
        PRIMARY KEY (id)
);


CREATE TABLE power_plant_categories (
        id INTEGER NOT NULL,
        date_created DATETIME,
        date_modified DATETIME,
        name VARCHAR(127) NOT NULL,
        description VARCHAR(4096) NOT NULL,
        PRIMARY KEY (id)
);


CREATE TABLE pylon_categories (
        id INTEGER NOT NULL,
        date_created DATETIME,
        date_modified DATETIME,
        name VARCHAR(127) NOT NULL,
        description VARCHAR(4096) NOT NULL,
        PRIMARY KEY (id)
);


CREATE TABLE power_line_categories (
        id INTEGER NOT NULL,
        date_created DATETIME,
        date_modified DATETIME,
        name VARCHAR(127) NOT NULL,
        description VARCHAR(4096) NOT NULL,
        PRIMARY KEY (id)
);


CREATE TABLE employees (
        id INTEGER NOT NULL,
        date_created DATETIME,
        date_modified DATETIME,
        name VARCHAR(128) NOT NULL,
        surname VARCHAR(128) NOT NULL,
        email VARCHAR(128) NOT NULL,
        password VARCHAR(192) NOT NULL,
        role INTEGER NOT NULL,
        PRIMARY KEY (id),
        UNIQUE (email),
        FOREIGN KEY(role) REFERENCES roles (id)
);


CREATE TABLE power_plants (
```

```
105         id INTEGER NOT NULL,
106         date_created DATETIME,
107         date_modified DATETIME,
108         name VARCHAR(128) NOT NULL,
109         description VARCHAR(4096),
110         category INTEGER NOT NULL,
111         latitude FLOAT NOT NULL,
112         longitude FLOAT NOT NULL,
113         PRIMARY KEY (id),
114         FOREIGN KEY(category) REFERENCES power_plant_categories (id)
115 );
116
117
118 CREATE TABLE pylons (
119         id INTEGER NOT NULL,
120         date_created DATETIME,
121         date_modified DATETIME,
122         name VARCHAR(128),
123         category INTEGER,
124         latitude FLOAT NOT NULL,
125         longitude FLOAT NOT NULL,
126         PRIMARY KEY (id),
127         FOREIGN KEY(category) REFERENCES pylon_categories (id)
128 );
129
130
131 CREATE TABLE pylon1_pylon2 (
132         id INTEGER NOT NULL,
133         date_created DATETIME,
134         date_modified DATETIME,
135         idLinea INTEGER NOT NULL,
136         idTraliccio1 INTEGER NOT NULL,
137         idTraliccio2 INTEGER NOT NULL,
138
139         PRIMARY KEY (id),
140         FOREIGN KEY(idLinea) REFERENCES power_lines(id),
141         FOREIGN KEY (idTraliccio1) REFERENCES pylons(id),
142         FOREIGN KEY (idTraliccio2) REFERENCES pylons(id)
143 );
144
145 CREATE TABLE low_voltage_power_cabins (
146         id INTEGER NOT NULL,
147         date_created DATETIME,
148         date_modified DATETIME,
149         name VARCHAR(128),
150         latitude FLOAT NOT NULL,
151         longitude FLOAT NOT NULL,
152         power_cabin INTEGER NOT NULL,
153         PRIMARY KEY (id),
154         FOREIGN KEY(power_cabin) REFERENCES high_voltage_power_cabins (id)
155 );
156
157
158 CREATE TABLE power_lines (
159         id INTEGER NOT NULL,
160         date_created DATETIME,
161         date_modified DATETIME,
162         line_name VARCHAR(128),
163         line_type INTEGER NOT NULL,
164         source INTEGER NOT NULL,
165         destination INTEGER NOT NULL,
166         PRIMARY KEY (id),
167         FOREIGN KEY(line_type) REFERENCES power_line_categories (id),
```

```
        FOREIGN KEY(source) REFERENCES power_plants (id),
        FOREIGN KEY(destination) REFERENCES high_voltage_power_cabins (id)
);


CREATE TABLE residences (
        id INTEGER NOT NULL,
        date_created DATETIME,
        date_modified DATETIME,
        latitude FLOAT NOT NULL,
        longitude FLOAT NOT NULL,
        owner INTEGER NOT NULL,
        power_cabin INTEGER NOT NULL,
        PRIMARY KEY (id),
        FOREIGN KEY(owner) REFERENCES customers (id),
        FOREIGN KEY(power_cabin) REFERENCES low_voltage_power_cabins (id)
);


CREATE TABLE tickets (
        id INTEGER NOT NULL,
        date_created DATETIME,
        date_modified DATETIME,
        subject VARCHAR(128) NOT NULL,
        message VARCHAR(8192) NOT NULL,
        solved BOOLEAN,
        customer INTEGER NOT NULL,
        staff INTEGER,
        PRIMARY KEY (id),
        FOREIGN KEY(customer) REFERENCES customers (id),
        FOREIGN KEY(staff) REFERENCES employees (id)
);


CREATE TABLE bills (
        id INTEGER NOT NULL,
        date_created DATETIME,
        date_modified DATETIME,
        consumption FLOAT NOT NULL,
        paid BOOLEAN,
        residence INTEGER,
        PRIMARY KEY (id),
        FOREIGN KEY(residence) REFERENCES residences (id)
);
```

Il Metodo di SQLAlchemy

`api/code/models.py`

```
# https://www.submarinecablemap.com/
# This website is an archive of the electric lines.

from app import db


# Define a base model for other database tables to inherit
class Base(db.Model):
    __abstract__  = True

    id            = db.Column(db.Integer, primary_key=True)
```

```
12      date_created  = db.Column(db.DateTime, default=db.func.current_timestamp())
13      date_modified = db.Column(db.DateTime, default=db.func.current_timestamp(),
        onupdate=db.func.current_timestamp())
14
15
16
17  # Define an Employee model
18  class Employee(Base):
19      __tablename__ = 'employees'
20
21      # User Name
22      name    = db.Column(db.String(128), nullable=False)
23      surname = db.Column(db.String(128), nullable=False)
24
25      # Identification Data: email & password
26      email    = db.Column(db.String(128), nullable=False, unique=True)
27      password = db.Column(db.String(192), nullable=False)
28
29      # Authorisation Data: role & status
30      role     = db.Column(db.Integer, db.ForeignKey('roles.id'), nullable=False)
31
32      def __repr__(self):
33          return '<User %r>' % (self.name)
34
35      def toApiData(self):
36          return {
37              'name': '%s %s' % (self.surname, self.name),
38              'email': '%s' % self.email,
39              'role': '%s' % Role.query.get(self.role).name
40          }
41
42
43  # Defining the Role model.
44  class Role(Base):
45      __tablename__ = 'roles'
46
47      # Role information
48      name            = db.Column(db.String(128), nullable=False, unique=True)
49      description     = db.Column(db.String(409), nullable=False)
50
51      # Relationship data
52      users           = db.relationship('Employee', backref='roles', lazy=False)
53
54      def __repr__(self):
55          return '<Role %r>' % (self.name)
56
57      def toApiData(self):
58          return {
59              'name': self.name,
60              'description': self.description
61          }
62
63
64  # Defining the PowerPlant model.
65  class PowerPlant(Base):
66      __tablename__ = 'power_plants'
67
68      # Information about the power plant
69      name        = db.Column(db.String(128), nullable=False)
70      description = db.Column(db.String(4096), nullable=True)
71      category    = db.Column(db.Integer, db.ForeignKey('power_plant_categories.id'),
        nullable=False)
72
```

```
73      # Location of the power plant
74      latitude    = db.Column(db.Float, nullable=False)
75      longitude   = db.Column(db.Float, nullable=False)
76
77      def __repr__(self):
78          return '<PowerPlant %s %f:%f>' % (self.name, self.latitude, self.longitude)
79
80      def toApiData(self):
81          return {
82              'id': self.id,
83              'name': self.name,
84              'type': 'Power Plant',
85              'description': self.description,
86              'category': PowerPlantCategory.query.get(self.category).name,
87              'location': [self.latitude, self.longitude],
88              'lat': self.latitude,
89              'lng': self.longitude,
90          }
91
92
93  class Pylon(Base):
94      __tablename__ = 'pylons'
95
96      # Information about the pylon
97      name        = db.Column(db.String(128), nullable=True)
98      category    = db.Column(db.Integer, db.ForeignKey('pylon_categories.id'))
99
100     # Location of the pylon
101     latitude    = db.Column(db.Float, nullable=False)
102     longitude   = db.Column(db.Float, nullable=False)
103
104     def __repr__(self):
105         return '<Pylon %f:%f>' % (self.latitude, self.longitude)
106
107     def toApiData(self):
108         return {
109             'id': self.id,
110             'name': self.name,
111             'type': 'Pylon',
112             'category': PylonCategory.query.get(self.category).name,
113             'color': '#00ff00',
114             'lat': self.latitude,
115             'lng': self.longitude,
116             'size': 0.0051215125,
117             'radius': 0.03,
118         }
119
120
121 class HighVoltagePowerCabin(Base):
122     __tablename__ = 'high_voltage_power_cabins'
123
124     # Information of the cabin
125     name        = db.Column(db.String(128), nullable=True)
126
127     # Location of the electric cabin
128     latitude    = db.Column(db.Float, nullable=False)
129     longitude   = db.Column(db.Float, nullable=False)
130
131     def __repr__(self):
132         return '<HighVoltagePowerCabin %f:%f>' % (self.latitude, self.longitude)
133
134     def toApiData(self):
135         return {
```

```
            'id': self.id,
            'name': self.name,
            'type': 'Power Cabin',
            'color': '#3b82f6',
            'lat': self.latitude,
            'lng': self.longitude,
            'size': 0.0221152,
            'radius': 0.04,
            'voltage': 'HIGH',
        }


class LowVoltagePowerCabin(Base):
    __tablename__ = 'low_voltage_power_cabins'

    # Information of the cabin
    name        = db.Column(db.String(128), nullable=True)

    # Location of the electric cabin
    latitude    = db.Column(db.Float, nullable=False)
    longitude   = db.Column(db.Float, nullable=False)

    # Source power cabin
    power_cabin = db.Column(db.Integer, db.ForeignKey('high_voltage_power_cabins.id')
    , nullable=False)


    def __repr__(self):
        return '<LowVoltagePowerCabin %f:%f>' % (self.latitude, self.longitude)

    def toApiData(self):
        return {
            'id': self.id,
            'name': self.name,
            'type': 'Power Cabin',
            'color': '#1d4ed8',
            'lat': self.latitude,
            'lng': self.longitude,
            'size': 0.0221152,
            'radius': 0.04,
            'voltage': 'LOW',
            # 'source': HighVoltagePowerCabin.query.get(self.power_cabin).
        }


class PowerLine(Base):
    __tablename__ = 'power_lines'

    # Information of the power line
    line_name   = db.Column(db.String(128), nullable=True)
    line_type   = db.Column(db.Integer, db.ForeignKey('power_line_categories.id'),
    nullable=False)

    # Path of the electric line
    source      = db.Column(db.Integer, db.ForeignKey('power_plants.id'), nullable=
    False)
    destination = db.Column(db.Integer, db.ForeignKey('high_voltage_power_cabins.id')
    , nullable=False)

    # Method to calculate the path.
    def calculateRoute(self):
        routes = []
        origin = PowerPlant.query.get(self.source)
```

```
195         destination = HighVoltagePowerCabin.query.get(self.destination)
196 
197         # This is the start (the power plant)
198         routes.append([origin.latitude, origin.longitude])
199 
200         first_item = None
201         while True:
202             foobar = Pylon1_Pylon2.query.filter_by(
203                 line = self.id,
204                 pylon1 = first_item
205             )
206 
207             if foobar.count() == 0:
208                 break
209 
210             pylon_data = Pylon.query.get(foobar.first().pylon2)
211             if pylon_data is None:
212                 break
213 
214             routes.append([pylon_data.latitude, pylon_data.longitude])
215             first_item = pylon_data.id
216 
217         # This is the end (the power cabin)
218         routes.append([destination.latitude, destination.longitude])
219 
220         # Return the calculated routes.
221         return routes
222 
223     def __repr__(self):
224         return '<PowerLine %f:%f>' % (self.source, self.destination)
225 
226     def toApiData(self):
227         return {
228             'coords': self.calculateRoute(),
229             'properties': {
230                 'id': self.id,
231                 'name': self.line_name,
232                 'color': 'DarkGray',
233                 'type': 'Power Line',
234             }
235         }
236 
237 
238 class Pylon1_Pylon2(Base):
239     __tablename__ = 'pylon1_pylon2'
240 
241     line   = db.Column(db.Integer, db.ForeignKey('power_lines.id'), nullable=False)
242     pylon1 = db.Column(db.Integer, db.ForeignKey('pylons.id'), nullable=True)
243     pylon2 = db.Column(db.Integer, db.ForeignKey('pylons.id'), nullable=True)
244 
245     def __repr__(self) -> str:
246         return '<PylonConnection from %d to %d>' % (self.pylon1 or 0, self.pylon2 or
    0)
247 
248 
249 # User informations
250 class Customer(Base):
251     __tablename__ = 'customers'
252 
253     # Informations about the customer.
254     name = db.Column(db.String(64), nullable=False)
255     surname = db.Column(db.String(128), nullable=False)
256     birthdate = db.Column(db.DateTime, nullable=False)
```

```
257     fiscal_code = db.Column(db.String(16), nullable=False)
258     pec = db.Column(db.String(300), nullable=False)
259     phone_number = db.Column(db.String(20), nullable=False)
260 
261     def __repr__(self) -> str:
262         return '<Customer@%d %s %s>' % (self.id, self.name, self.surname)
263 
264     def toApiData(self):
265         return {
266             'id': self.id,
267             'name': self.name,
268             'surname': self.surname,
269             'birthdate': self.birthdate,
270             'fiscal_code': self.fiscal_code,
271             'pec': self.pec,
272             'phone_number': self.phone_number
273         }
274 
275 
276 class Residence(Base):
277     __tablename__ = 'residences'
278 
279     # Information
280     latitude = db.Column(db.Float, nullable=False)
281     longitude = db.Column(db.Float, nullable=False)
282 
283     # Relationships
284     owner = db.Column(db.Integer, db.ForeignKey('customers.id'), nullable=False)
285     power_cabin = db.Column(db.Integer, db.ForeignKey('low_voltage_power_cabins.id'),
         nullable=False)
286 
287     def __repr__(self) -> str:
288         return '<Residence %d of %s at %f:%f>' % (self.id, repr(Customer.query.get(
        self.owner)), self.latitude, self.longitude)
289 
290     def toApiData(self):
291         return {
292             'id': self.id,
293             'owner': Customer.query.get(self.owner).toApiData(),
294             'location': [self.latitude, self.longitude],
295             'cabin': LowVoltagePowerCabin.query.get(self.power_cabin).toApiData()
296         }
297 
298 
299 class Bill(Base):
300     __tablename__ = 'bills'
301 
302     # Information
303     consumption = db.Column(db.Float, nullable=False)
304     paid = db.Column(db.Boolean, default=False)
305 
306     # Relationship
307     residence = db.Column(db.Integer, db.ForeignKey('residences.id'))
308 
309     def __repr__(self) -> str:
310         return '<Bill %d to %s>' % (self.id, repr(Residence.query.get(self.residence)
        ))
311 
312 
313 class Ticket(Base):
314     __tablename__ = 'tickets'
315 
316     # Information
```

```
    subject = db.Column(db.String(128), nullable=False)
    message = db.Column(db.String(8192), nullable=False)
    solved  = db.Column(db.Boolean, default=False)

    # Relationships
    customer = db.Column(db.Integer, db.ForeignKey('customers.id'), nullable=False)
    staff    = db.Column(db.Integer, db.ForeignKey('employees.id'), nullable=True)

    def __repr__(self) -> str:
        return '<Ticket id=%d solved=%s>' % (self.id, self.solved)


## Redundancy deleetus
class PowerPlantCategory(Base):
    __tablename__ = 'power_plant_categories'

    # Information about the category
    name        = db.Column(db.String(127), nullable=False)
    description = db.Column(db.String(4096), nullable=False)

    def __repr__(self):
        return '<PowerPlantCategory %s>' % self.name

    def toApiData(self):
        return {
            'id': self.id,
            'name': self.name
        }


class PylonCategory(Base):
    __tablename__ = 'pylon_categories'

    # Information about the category
    name        = db.Column(db.String(127), nullable=False)
    description = db.Column(db.String(4096), nullable=False)

    def __repr__(self):
        return '<PylonCategory %s>' % self.name

    def toApiData(self):
        return {
            'id': self.id,
            'name': self.name
        }


class PowerLineCategory(Base):
    __tablename__ = 'power_line_categories'

    # Categories:
    # - Bassa Tensione
    # - Media Tensione
    # Information about the category
    name        = db.Column(db.String(127), nullable=False)
    description = db.Column(db.String(4096), nullable=False)

    def __repr__(self):
        return '<PowerLineCategory %s>' % self.name

    def toApiData(self):
        return {
            'id': self.id,
```

```
            'name': self.name
        }
```

Per creare le tabelle del database:

```
# Da qualsiasi parte
db = SQLAlchemy(app)
db.create_db()
```

4.4 Inserimento dei Dati

Metodo Tradizionale

```
-- Insert the Roles.
INSERT INTO Roles (
    NAME,
    DESCRIPTION
) VALUES (
    "Administrator",
    "The system administrator"
);

INSERT INTO Roles (
    NAME,
    DESCRIPTION
) VALUES (
    "Finances",
    "The finances team"
);


-- Add the users.
INSERT INTO Users (
    NAME,
    SURNAME,
    EMAIL,
    PASSWORD,
    ROLE
) VALUES (
    "Anderson",
    "Smith",
    "admin@localhost",
    "admin",
    (SELECT ID FROM Roles WHERE NAME = "Administrator" LIMIT 1)
);

INSERT INTO Users (
    NAME,
    SURNAME,
    EMAIL,
    PASSWORD,
    ROLE
) VALUES (
    "William",
    "Blake",
    "william.blake@electrocorp.com",
    "stonks",
    (SELECT ID FROM Roles WHERE NAME = "Finances" LIMIT 1)
);
```

```

INSERT INTO Users (
    NAME,
    SURNAME,
    EMAIL,
    PASSWORD,
    ROLE
) VALUES (
    "Mario",
    "Rossi",
    "mario.rossi@electrocorp.com",
    "soldi-123",
    (SELECT ID FROM Roles WHERE NAME = "Finances" LIMIT 1)
);
```

Il Metodo di SQLAlchemy

```
#!/usr/bin/env python3

from app import app, db
from models import *


role_admin      = Role(
    name='Administrator',
    description='The system administrator'
)

role_finances   = Role(
    name='Finances',
    description='The finances team'
)

db.session.add(role_admin)
db.session.add(role_finances)
db.session.commit()



## Create the administrator user.
user_admin      = Employee(
    surname='Anderson',
    name='Smith',
    email='admin@localhost',
    password='admin',
    role=role_admin.id
)

db.session.add(user_admin)
db.session.commit()


## Create some generic financial users.
role_finances   = Role.query.get(2)
user_william    = Employee(
    name='William',
    surname='Blake',
    email='william.blake@electrocorp.com',
    password='stonks',
    role=role_finances.id
```

```
)

user_mario      = Employee(
    name='Mario',
    surname='Rossi',
    email='mario.rossi@electrocorp.com',
    password='soldi-123',
    role=role_finances.id
)

db.session.add(user_william)
db.session.add(user_mario)
db.session.commit()
```

4.5 Alcune query

4.5.1 Membri del team "Financing"

Metodo Tradizionale

```
-- Get all the employees that are in the << Finances >> role.
SELECT
    NAME,
    SURNAME,
    EMAIL,
    ROLE

FROM
    Users


WHERE
    ROLE = (SELECT ID FROM Roles WHERE NAME = "Finances");
```

Il Metodo di SQLAlchemy

```
User.query.filter_by(role=Role.query.filter_by(name='Finances')[0].id)
```

4.5.2 Centrali Elettriche rinnovabili

Il Metodo Tradizionale

```
SELECT
    PowerPlant.ID,
    PowerPlant.NAME,
    PowerPlant.DESCRIPTION,
    PowerPlant.LATITUDE,
    PowerPlant.LONGITUDE,
    PowerPlant.DATE_CREATED,
    PowerPlant.DATE_MODIFIED,
    PowerPlantCategory.NAME as CATEGORY

FROM
    PowerPlant
```

```
    LEFT JOIN PowerPlantCategory
    ON PowerPlant.CATEGORY = PowerPlantCategory.ID

WHERE
    CATEGORY IN (
        "Hydro",
        "Solar",
        "Wind",
        "Nuclear",
        "Geothermal"
    );
```

Il Metodo di SQLAlchemy

```
    PowerPlant.query.filter(
        PowerPlant.category.in_(
            PowerPlantCategory.query.filter_by(name='Hydro')[0].id,
            PowerPlantCategory.query.filter_by(name='Solar')[0].id,
            PowerPlantCategory.query.filter_by(name='Wind')[0].id,
            PowerPlantCategory.query.filter_by(name='Nuclear')[0].id,
            PowerPlantCategory.query.filter_by(name='Geothermal')[0].id,
        )
    )
```