

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 chiesta 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'elettricità 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 è 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 identità'.

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 così l'efficienza 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 scalabilità esponenziale e per il supporto alla distribuzione in più 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 già 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 può 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.

```

1 robert@workstation$ ssh root@electrocorp.com
2 root@www.electrocorp.com# iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --
  to 172.22.1.5:80
3 root@www.electrocorp.com# iptables -t nat -A PREROUTING -p tcp --dport 443 -j DNAT --
  to 172.22.1.5:443
4 root@www.electrocorp.com# logout
5 robert@workstation$ ssh root@admin.electrocorp.com
6 root@admin.electrocorp.com# iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT
  --to 172.22.3.5:80
7 root@admin.electrocorp.com# iptables -t nat -A PREROUTING -p tcp --dport 443 -j DNAT
  --to 172.22.3.5:443
8 root@admin.electrocorp.com# logout
9 robert@workstation$ ssh api@admin.electrocorp.com
10 root@api.electrocorp.com# iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --
  to 172.22.2.5:80
11 root@api.electrocorp.com# iptables -t nat -A PREROUTING -p tcp --dport 443 -j DNAT --
  to 172.22.2.5:443
12 root@api.electrocorp.com# # Facciamo lo static routing per avere la comunicazione fra
  il leader API e il DBMS.
13 root@api.electrocorp.com# ssh root@172.22.2.5
14 root@leader-172.22.2.5% ip route add 172.22.4.0/24 ia 172.22.2.1
15 root@leader-172.22.2.5% logout
16 root@api.electrocorp.com# ssh root@172.22.4.5
17 root@dbms-leader-172.22.4.5% ip route add 172.22.2.0/24 ia 172.22.4.1
18 root@dbms-leader-172.22.4.5% logout
19 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.

```
1 version: '3.4'
2
3
4 services:
5   api:
6     hostname: 'api.electrocorp.com'
7     build: './api/'
8     links:
9       - 'database'
10
11   volumes:
12     - './api/code:/usr/src/app:rw'
13
14   networks:
15     api_net:
16       ipv4_address: 172.22.2.5
17
18     database_net:
19       ipv4_address: 172.22.4.2
20
21   environment:
22     SECRET_SALT: ^&?cY,t}pE>6rQu] $C#pRK~dns3B({}.e~*BzSgb5d7h*F;&FB/jD(>N\
NSMX.b
23     DATABASE_URL: postgresql://leader:mysecretpassword@172.22.4.5:5432/
electrocorp
24
25   ports:
26     - '8555:80'
27
28   admin:
29     hostname: 'admin.electrocorp.com'
30     build: './admin/'
31     volumes:
32       - './admin/code:/usr/src/app:ro'
33
34     links:
35       - 'api'
36
37     networks:
38       admin_net:
39         ipv4_address: 172.22.3.5
40
41   www:
42     hostname: 'electrocorp.com'
43     build: './www/'
44     volumes:
45       - './www/code:/usr/src/app:rw'
46
47     links:
48       - 'api'
49
50     networks:
51       www_net:
52         ipv4_address: 172.22.1.5
53
54     environment:
55       FLASK_APP: run.py
56
57   database:
58     image: 'postgres:latest'
59     environment:
60       POSTGRES_PASSWORD: mysecretpassword
```



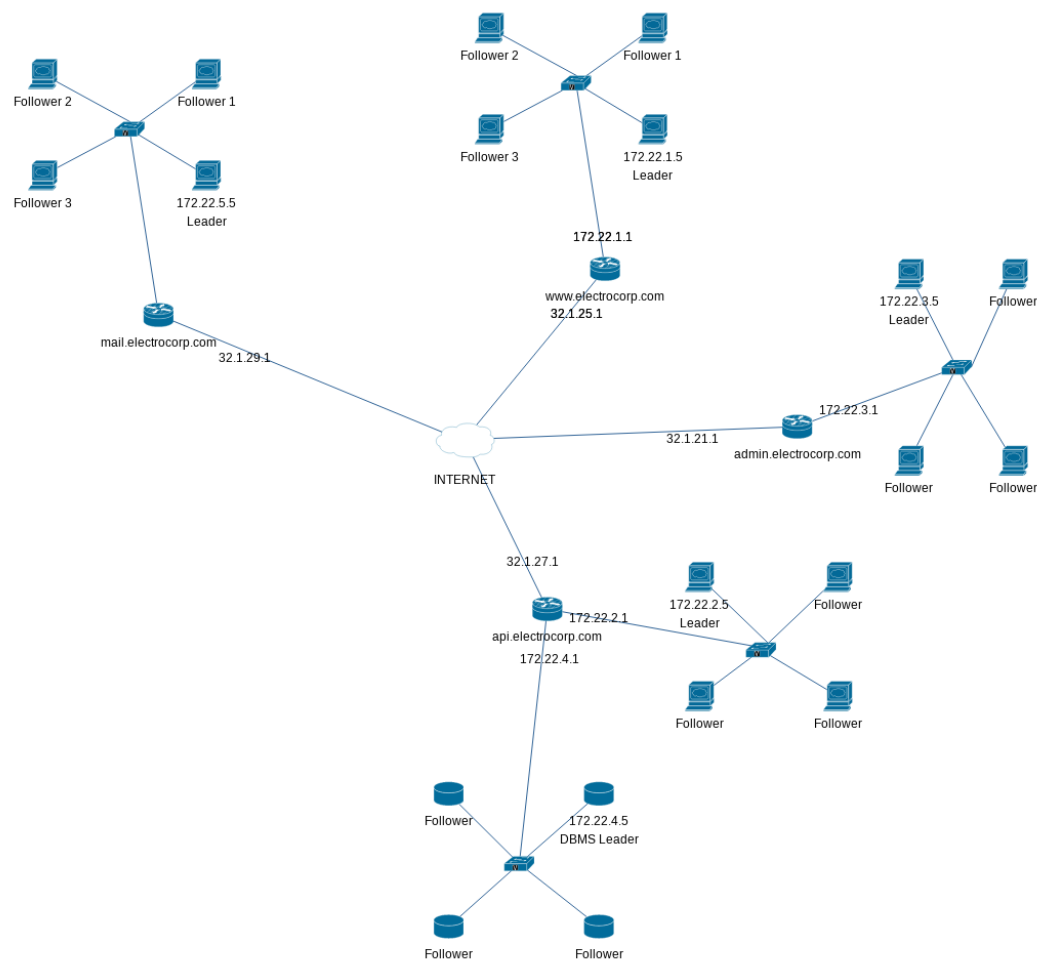
```
61     POSTGRES_DB: electrocorp
62     POSTGRES_USER: leader
63
64     volumes:
65     - psql_database:/var/lib/postgresql/data
66
67     networks:
68         database_net:
69             ipv4_address: 172.22.4.5
70
71     mail:
72         hostname: 'mail.electrocorp.com'
73         build: ./mail/
74         volumes:
75         - mail:/var/mail
76
77     networks:
78         mail_net:
79             ipv4_address: 172.22.5.5
80
81
82
83 volumes:
84     psql_database:
85     mail:
86
87
88
89 networks:
90     www_net:
91         external:
92             name: www
93
94     api_net:
95         external:
96             name: api
97
98     admin_net:
99         external:
100             name: admin
101
102     database_net:
103         external:
104             name: database
105
106     mail_net:
107         external:
108             name: mail
```

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

```
1 127.0.0.1 localhost
2 127.0.1.1 PwnStation
3
4 # The following lines are desirable for IPv6 capable hosts
5 ::1      ip6-localhost ip6-loopback
6 fe00::0  ip6-localnet
7 ff00::0  ip6-mcastprefix
8 ff02::1  ip6-allnodes
9 ff02::2  ip6-allrouters
10
11
```

```
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 navi-gare, 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.

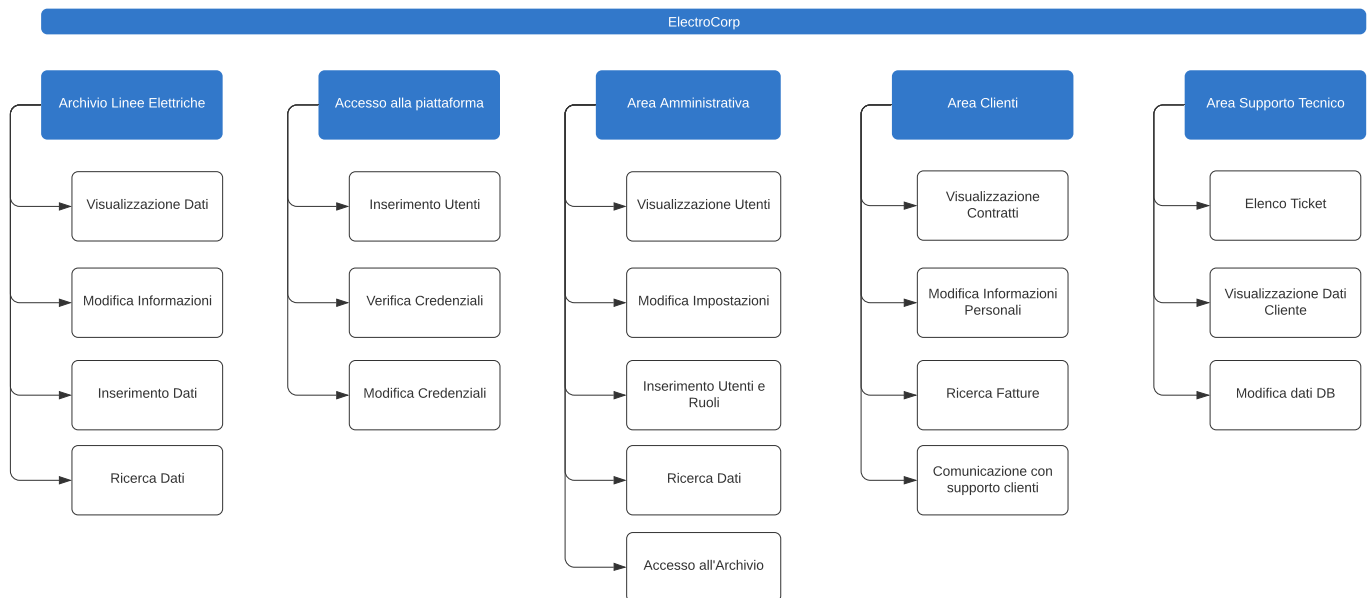
Access Control (admin.electrocorp.com)					
	Archive	Control Panel	Login	Finances	Logs
Administrator	ALL	ALL	AUTHENTICATE	ALL	READ
Finances			AUTHENTICATE	ALL	
not auth			AUTHENTICATE		

Accessibile solo dagli impiegati.

Questo e' il diagramma a blocchi.

Archivio

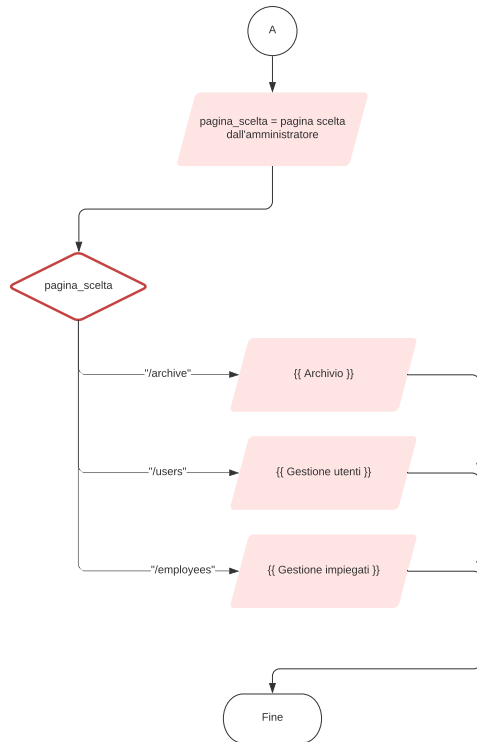
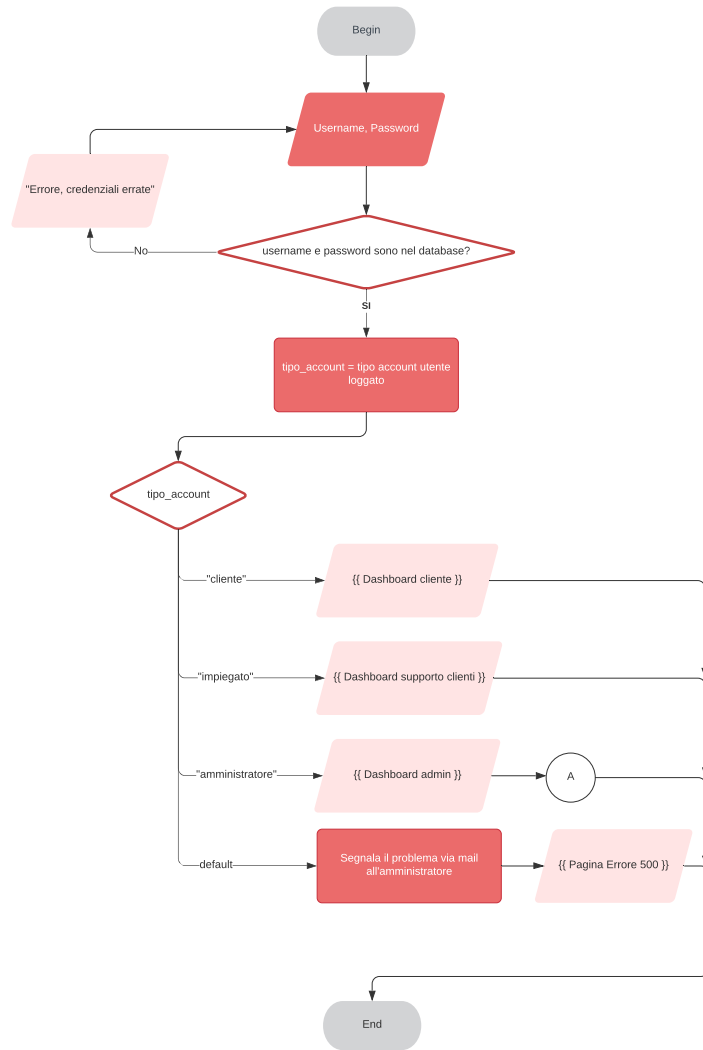
Zmlyc3Q bGFzdA | May 15, 2021



E questo e' il flow chart.

Diagramma di flusso dei moduli

Zmilyc3Q 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.

```
1 import json
2
3 from app import app, db
4 from flask import abort, jsonify, request, g
5 import models
6 import jwt
7 import traceback
8
9
10 # Simple endpoint for testing stuff
11 @app.route('/test', methods=['POST'])
12 def test():
13     return jsonify({
14         'context': g.data,
15         'full_context': dir(g),
16     })
17
18 # Authentication endpoint.
19 @app.route('/login/employee', methods=['POST'])
20 def login():
21     error = None
22     data = None
23     try:
24         l33t = request.get_json(force=True)
25         user = models.Employee.query.filter(
26             models.Employee.email == l33t.get('username', None)
27         ).first()
28
29         if user == None:
30             error = 'User does not exist'
31
32         elif user.password == l33t.get('password', None):
33             data = jwt.encode({'id': user.id, 'role': user.role}, app.config['
34 SECRET_KEY'], algorithm='HS256')
35             foo = jsonify({
36                 'error': None,
37                 'data': data
38             })
39             foo.set_cookie('token', data)
40             return foo
41
42         else:
43             error = 'Authentication Failed'
44
45     except Exception as e:
46         data = None
47         error = str(e)
48
49     finally:
50         return jsonify({
51             'error': error,
```

```
51         'data': data
52     })
53
54
55 # Power Plant endpoints.
56 @app.route('/power_plant/all', methods=['POST'])
57 def read_all_power_plants():
58     data = []
59     checked = check_authorization([1])
60     if checked is not None:
61         return checked
62
63     for x in models.PowerPlant.query.all():
64         data.append(x.toApiData())
65
66     return jsonify({
67         'error': None,
68         'data': data,
69     }), 200
70
71
72
73 @app.route('/power_plant/categories', methods=['GET'])
74 def get_power_plant_categories():
75     # It does NOT require privileges to access this data.
76     data = []
77     for x in models.PowerPlantCategory.query.all():
78         data.append(x.toApiData())
79
80     return jsonify({
81         'data': data
82     })
83
84
85 @app.route('/power_plant/create', methods=['POST'])
86 def create_power_plant():
87     error = None
88     status = 200
89     checked = check_authorization([1])
90     if checked is not None:
91         return checked
92
93     data = request.get_json(force=True)
94     try:
95         foobar = models.PowerPlant(
96             name=data['name'],
97             description=data['desc'],
98             category=int(data['category']),
99             latitude=float(data['lat']),
100             longitude=float(data['lng'])
101         )
102
103     except Exception as e:
104         status = 400
105         error = str(e)
106
107     finally:
108         return jsonify({
109             error: error
110         }), status
111
112
113 @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:
```

```
177     #     return checked
178
179     data = []
180     for x in models.HighVoltagePowerCabin.query.all():
181         data.append(x.toApiData())
182
183     for x in models.LowVoltagePowerCabin.query.all():
184         data.append(x.toApiData())
185
186     return jsonify({
187         'error': None,
188         'data': data,
189     }), 200
190
191
192 # @app.route('/power_cabin/categories', methods=['GET'])
193 # def get_power_cabin_categories():
194 #     # It does NOT require privileges to access this data.
195 #     data = []
196 #     for x in models.HighVoltagePowerCabinCategory.query.all():
197 #         data.append(x.toApiData())
198
199 #     return jsonify({
200 #         'data': data
201 #     })
202
203
204 @app.route('/power_cabin/<int:id>/read', methods=['POST'])
205 def read_power_cabin_by(id: int):
206     checked = check_authorization([1])
207     if checked is not None:
208         return checked
209
210     try:
211         return jsonify({
212             'error': None,
213             'data': models.HighVoltagePowerCabin.query.get(id).toApiData()
214         }), 200
215
216     except Exception as e:
217         return jsonify({
218             'error': str(e),
219             'data': None
220         }), 404
221
222
223 @app.route('/power_cabin/<int:id>/update', methods=['POST'])
224 def update_power_cabin_by(id: int):
225     checked = check_authorization([1])
226     if checked is not None:
227         return checked
228
229     try:
230         return jsonify({
231             'error': None,
232             'data': models.HighVoltagePowerCabin.query.get(id).toApiData()
233         }), 200
234
235     except Exception as e:
236         return jsonify({
237             'error': str(e),
238             'data': None
239         }), 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():

```

```

366         source = models.HighVoltagePowerCabin.query.get(x.power_cabin)
367         if source is None: continue
368         data.append({
369             'coords': [
370                 [
371                     source.latitude, source.longitude
372                 ],
373                 [
374                     x.latitude, x.longitude
375                 ]
376             ],
377             'properties': {
378                 'id': '#',
379                 'name': 'Distribution Line',
380                 'color': '#8B008B',
381                 'type': 'Distribution'
382             }
383         })
384     except Exception as e:
385         error = str(traceback.print_exc())
386         data = []
387     finally:
388         return jsonify({
389             'error': error,
390             'data': data,
391         }), 200
392
393 @app.route('/power_line/categories', methods=['GET'])
394 def get_power_line_categories():
395     data = []
396     for x in models.PowerLineCategory.query.all():
397         data.append(x.toApiData())
398
399     return jsonify({
400         'data': data
401     }), 200
402
403 # Users endpoint
404 @app.route('/user/all', methods=['POST'])
405 def get_users():
406     checked = check_authorization([1])
407     if checked is not None:
408         return checked
409
410     data = []
411     for x in models.Employee.query.all():
412         data.append(x.toApiData())
413
414     return jsonify({
415         'error': None,
416         'data': data
417     })
418
419 # Useful functions
420 def check_authorization(user_roles):
421     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:

```
1 import logo from './logo.svg';
2 import './App.css';
3 import {
4   BrowserRouter as Router,
5   Switch,
6   Route,
7   Link
8 } from "react-router-dom";
9 import {useState} from 'react'
10
11 // Import the views.
12 import LoginScreen from './views/Login'
13 import ArchiveScreen from './views/Archive'
14 import DashboardScreen from './views/Dashboard'
15 import TestScreen from './views/Test'
16 import LogoutScreen from './views/Logout'
17
18
19 // Import the error views.
20 import Error404 from './views/Error404'
21
22
23 // The code
24 function App() {
25   return (
26     <Router>
27       <Switch>
28         <Route exact path="/">
29           <LoginScreen />
30         </Route>
31
32         <Route exact path="/login">
33           <LoginScreen />
34         </Route>
35
36         <Route exact path="/dashboard">
37           <DashboardScreen />
38         </Route>
39
40         <Route exact path="/archive">
41           <ArchiveScreen />
42         </Route>
43
44         <Route exact path="/test">
45           <TestScreen />
46         </Route>
47
48         <Route exact path="/logout">
49           <LogoutScreen />
50         </Route>
51
52         <Route path="*">
53           <Error404 />
54         </Route>
55       </Switch>
56     </Router>
```

```
57   );  
58 }  
59  
60 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 `CLledocker-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 `command` 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.

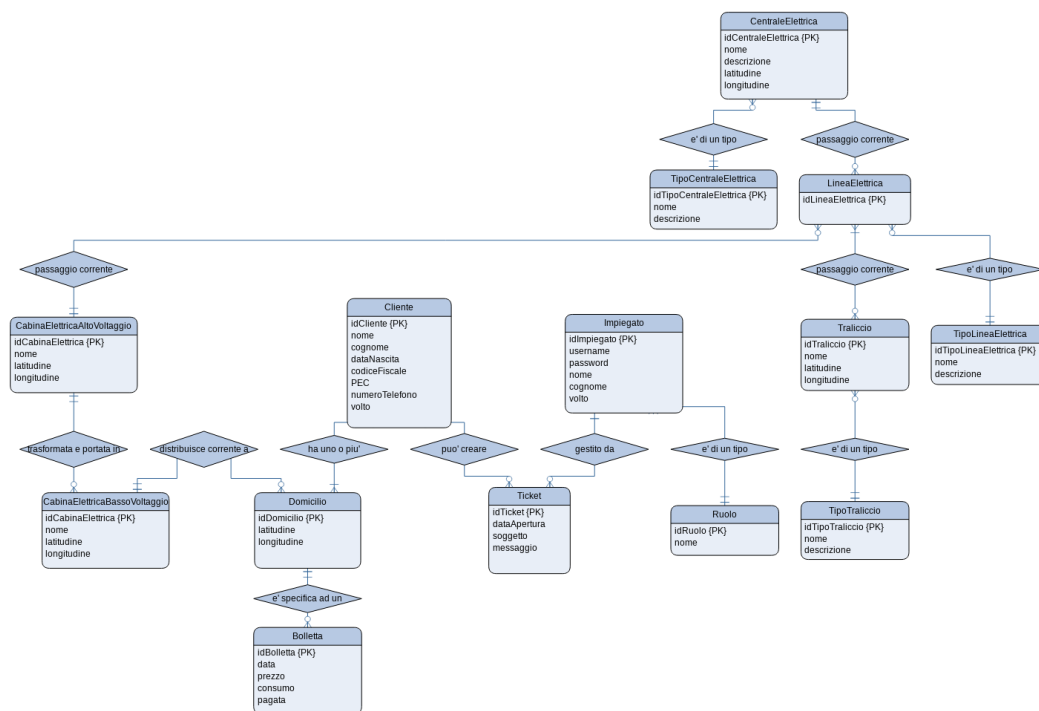
`iptables` richiede privilegi elevati per funzionare e deve essere eseguito dall'utente `root`, altrimenti non funziona. Sulla maggior parte dei sistemi Linux, `iptables` è installato come `/usr/sbin/iptables` e documentato nelle sue pagine `man`, che

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		

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

CabinaElettricaBassoVtaggio						
Name	Type	Size	Key	Null	Default	Extra
idCabina	INTEGER		PRIMARY	NO		AUTO INCREMENT
nome	VARCHAR	128		YES		
latitudine	FLOAT			NO		
longitudine	FLOAT			NO		
idCabinaAltoVtaggio	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	64	PRIMARY	NO		AUTO INCREMENT
nome	VARCHAR			SI		
latitude	FLOAT			NO		
longitude	FLOAT			NO		
idTipoTraliccio	INTEGER		FOREIGN	NO		

Cliente						
Name	Type	Size	Key	Null	Default	Extra
idCliente	INTEGER	64	PRIMARY	NO		AUTO INCREMENT
nome	VARCHAR			NO		
cognome	VARCHAR			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 INCREMENT
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		
dataBolletta	DATE			NO		
pagata	BOOLEAN			NO		

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

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.

```

1 CREATE DATABASE IF NOT EXISTS ARCHIVE;
2 USE ARCHIVE;
3
4 DROP TABLE IF EXISTS roles;
5 DROP TABLE IF EXISTS employees;
6
7 DROP TABLE IF EXISTS power_plant_categories;
8 DROP TABLE IF EXISTS pylon_categories;
9 DROP TABLE IF EXISTS power_line_categories;
10
11 DROP TABLE IF EXISTS low_voltage_power_cabins;
12 DROP TABLE IF EXISTS high_voltage_power_cabins;
13 DROP TABLE IF EXISTS power_plants;
14 DROP TABLE IF EXISTS power_lines;
15 DROP TABLE IF EXISTS pylons;
16
17 DROP TABLE IF EXISTS customers;
18 DROP TABLE IF EXISTS residences;
19 DROP TABLE IF EXISTS bills;
20 DROP TABLE IF EXISTS tickets;
21
22
23 CREATE TABLE roles (
24     id INTEGER NOT NULL,
25     date_created DATETIME,
26     date_modified DATETIME,
27     name VARCHAR(128) NOT NULL,
28     description VARCHAR(409) NOT NULL,
29     PRIMARY KEY (id),
30     UNIQUE (name)
31 );
32
33
34 CREATE TABLE high_voltage_power_cabins (
35     id INTEGER NOT NULL,
36     date_created DATETIME,
37     date_modified DATETIME,
38     name VARCHAR(128),
39     latitude FLOAT NOT NULL,
40     longitude FLOAT NOT NULL,
41     PRIMARY KEY (id)

```



```
42 );
43
44
45 CREATE TABLE customers (
46     id INTEGER NOT NULL,
47     date_created DATETIME,
48     date_modified DATETIME,
49     name VARCHAR(64) NOT NULL,
50     surname VARCHAR(128) NOT NULL,
51     birthdate DATETIME NOT NULL,
52     fiscal_code VARCHAR(16) NOT NULL,
53     pec VARCHAR(300) NOT NULL,
54     phone_number VARCHAR(20) NOT NULL,
55     PRIMARY KEY (id)
56 );
57
58
59 CREATE TABLE power_plant_categories (
60     id INTEGER NOT NULL,
61     date_created DATETIME,
62     date_modified DATETIME,
63     name VARCHAR(127) NOT NULL,
64     description VARCHAR(4096) NOT NULL,
65     PRIMARY KEY (id)
66 );
67
68
69 CREATE TABLE pylon_categories (
70     id INTEGER NOT NULL,
71     date_created DATETIME,
72     date_modified DATETIME,
73     name VARCHAR(127) NOT NULL,
74     description VARCHAR(4096) NOT NULL,
75     PRIMARY KEY (id)
76 );
77
78
79 CREATE TABLE power_line_categories (
80     id INTEGER NOT NULL,
81     date_created DATETIME,
82     date_modified DATETIME,
83     name VARCHAR(127) NOT NULL,
84     description VARCHAR(4096) NOT NULL,
85     PRIMARY KEY (id)
86 );
87
88
89 CREATE TABLE employees (
90     id INTEGER NOT NULL,
91     date_created DATETIME,
92     date_modified DATETIME,
93     name VARCHAR(128) NOT NULL,
94     surname VARCHAR(128) NOT NULL,
95     email VARCHAR(128) NOT NULL,
96     password VARCHAR(192) NOT NULL,
97     role INTEGER NOT NULL,
98     PRIMARY KEY (id),
99     UNIQUE (email),
100     FOREIGN KEY(role) REFERENCES roles (id)
101 );
102
103
104 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),

```

```

168     FOREIGN KEY(source) REFERENCES power_plants (id),
169     FOREIGN KEY(destination) REFERENCES high_voltage_power_cabins (id)
170 );
171
172
173 CREATE TABLE residences (
174     id INTEGER NOT NULL,
175     date_created DATETIME,
176     date_modified DATETIME,
177     latitude FLOAT NOT NULL,
178     longitude FLOAT NOT NULL,
179     owner INTEGER NOT NULL,
180     power_cabin INTEGER NOT NULL,
181     PRIMARY KEY (id),
182     FOREIGN KEY(owner) REFERENCES customers (id),
183     FOREIGN KEY(power_cabin) REFERENCES low_voltage_power_cabins (id)
184 );
185
186
187 CREATE TABLE tickets (
188     id INTEGER NOT NULL,
189     date_created DATETIME,
190     date_modified DATETIME,
191     subject VARCHAR(128) NOT NULL,
192     message VARCHAR(8192) NOT NULL,
193     solved BOOLEAN,
194     customer INTEGER NOT NULL,
195     staff INTEGER,
196     PRIMARY KEY (id),
197     FOREIGN KEY(customer) REFERENCES customers (id),
198     FOREIGN KEY(staff) REFERENCES employees (id)
199 );
200
201
202 CREATE TABLE bills (
203     id INTEGER NOT NULL,
204     date_created DATETIME,
205     date_modified DATETIME,
206     consumption FLOAT NOT NULL,
207     paid BOOLEAN,
208     residence INTEGER,
209     PRIMARY KEY (id),
210     FOREIGN KEY(residence) REFERENCES residences (id)
211 );

```

Il Metodo di SQLAlchemy

api/code/models.py

```

1 # https://www.submarinecablemap.com/
2 # This website is an archive of the electric lines.
3
4 from app import db
5
6
7 # Define a base model for other database tables to inherit
8 class Base(db.Model):
9     __abstract__ = True
10
11     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(),
14                               onupdate=db.func.current_timestamp())
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'),
72                          nullable=False)

```

```
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 {
```

```

136         'id': self.id,
137         'name': self.name,
138         'type': 'Power Cabin',
139         'color': '#3b82f6',
140         'lat': self.latitude,
141         'lng': self.longitude,
142         'size': 0.0221152,
143         'radius': 0.04,
144         'voltage': 'HIGH',
145     }
146
147
148 class LowVoltagePowerCabin(Base):
149     __tablename__ = 'low_voltage_power_cabins'
150
151     # Information of the cabin
152     name = db.Column(db.String(128), nullable=True)
153
154     # Location of the electric cabin
155     latitude = db.Column(db.Float, nullable=False)
156     longitude = db.Column(db.Float, nullable=False)
157
158     # Source power cabin
159     power_cabin = db.Column(db.Integer, db.ForeignKey('high_voltage_power_cabins.id'),
160                             , nullable=False)
161
162     def __repr__(self):
163         return '<LowVoltagePowerCabin %f:%f>' % (self.latitude, self.longitude)
164
165     def toApiData(self):
166         return {
167             'id': self.id,
168             'name': self.name,
169             'type': 'Power Cabin',
170             'color': '#1d4ed8',
171             'lat': self.latitude,
172             'lng': self.longitude,
173             'size': 0.0221152,
174             'radius': 0.04,
175             'voltage': 'LOW',
176             # 'source': HighVoltagePowerCabin.query.get(self.power_cabin).
177         }
178
179
180 class PowerLine(Base):
181     __tablename__ = 'power_lines'
182
183     # Information of the power line
184     line_name = db.Column(db.String(128), nullable=True)
185     line_type = db.Column(db.Integer, db.ForeignKey('power_line_categories.id'),
186                           , nullable=False)
187
188     # Path of the electric line
189     source = db.Column(db.Integer, db.ForeignKey('power_plants.id'), nullable=
190                       False)
191     destination = db.Column(db.Integer, db.ForeignKey('high_voltage_power_cabins.id')
192                             , nullable=False)
193
194     # Method to calculate the path.
195     def calculateRoute(self):
196         routes = []
197         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
247             0)
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'),
286                             nullable=False)
287
288     def __repr__(self) -> str:
289         return '<Residence %d of %s at %f:%f>' % (self.id, repr(Customer.query.get(
290             self.owner)), self.latitude, self.longitude)
291
292     def toApiData(self):
293         return {
294             'id': self.id,
295             'owner': Customer.query.get(self.owner).toApiData(),
296             'location': [self.latitude, self.longitude],
297             'cabin': LowVoltagePowerCabin.query.get(self.power_cabin).toApiData()
298         }
299
300 class Bill(Base):
301     __tablename__ = 'bills'
302
303     # Information
304     consumption = db.Column(db.Float, nullable=False)
305     paid = db.Column(db.Boolean, default=False)
306
307     # Relationship
308     residence = db.Column(db.Integer, db.ForeignKey('residences.id'))
309
310     def __repr__(self) -> str:
311         return '<Bill %d to %s>' % (self.id, repr(Residence.query.get(self.residence)
312         ))
313
314 class Ticket(Base):
315     __tablename__ = 'tickets'
316
317     # Information

```



```

317     subject = db.Column(db.String(128), nullable=False)
318     message = db.Column(db.String(8192), nullable=False)
319     solved = db.Column(db.Boolean, default=False)
320
321     # Relationships
322     customer = db.Column(db.Integer, db.ForeignKey('customers.id'), nullable=False)
323     staff = db.Column(db.Integer, db.ForeignKey('employees.id'), nullable=True)
324
325     def __repr__(self) -> str:
326         return '<Ticket id=%d solved=%s>' % (self.id, self.solved)
327
328
329 ## Redundancy deletus
330 class PowerPlantCategory(Base):
331     __tablename__ = 'power_plant_categories'
332
333     # Information about the category
334     name = db.Column(db.String(127), nullable=False)
335     description = db.Column(db.String(4096), nullable=False)
336
337     def __repr__(self):
338         return '<PowerPlantCategory %s>' % self.name
339
340     def toApiData(self):
341         return {
342             'id': self.id,
343             'name': self.name
344         }
345
346
347 class PylonCategory(Base):
348     __tablename__ = 'pylon_categories'
349
350     # Information about the category
351     name = db.Column(db.String(127), nullable=False)
352     description = db.Column(db.String(4096), nullable=False)
353
354     def __repr__(self):
355         return '<PylonCategory %s>' % self.name
356
357     def toApiData(self):
358         return {
359             'id': self.id,
360             'name': self.name
361         }
362
363
364 class PowerLineCategory(Base):
365     __tablename__ = 'power_line_categories'
366
367     # Categories:
368     # - Bassa Tensione
369     # - Media Tensione
370     # Information about the category
371     name = db.Column(db.String(127), nullable=False)
372     description = db.Column(db.String(4096), nullable=False)
373
374     def __repr__(self):
375         return '<PowerLineCategory %s>' % self.name
376
377     def toApiData(self):
378         return {
379             'id': self.id,

```

```
380         'name': self.name
381     }
```

Per creare le tabelle del database:

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

4.4 Inserimento dei Dati

Metodo Tradizionale

```
1 -- Insert the Roles.
2 INSERT INTO Roles (
3     NAME,
4     DESCRIPTION
5 ) VALUES (
6     "Administrator",
7     "The system administrator"
8 );
9
10 INSERT INTO Roles (
11     NAME,
12     DESCRIPTION
13 ) VALUES (
14     "Finances",
15     "The finances team"
16 );
17
18
19 -- Add the users.
20 INSERT INTO Users (
21     NAME,
22     SURNAME,
23     EMAIL,
24     PASSWORD,
25     ROLE
26 ) VALUES (
27     "Anderson",
28     "Smith",
29     "admin@localhost",
30     "admin",
31     (SELECT ID FROM Roles WHERE NAME = "Administrator" LIMIT 1)
32 );
33
34 INSERT INTO Users (
35     NAME,
36     SURNAME,
37     EMAIL,
38     PASSWORD,
39     ROLE
40 ) VALUES (
41     "William",
42     "Blake",
43     "william.blake@electrocorp.com",
44     "stonks",
45     (SELECT ID FROM Roles WHERE NAME = "Finances" LIMIT 1)
46 );
```

```
47
48 INSERT INTO Users (
49     NAME,
50     SURNAME,
51     EMAIL,
52     PASSWORD,
53     ROLE
54 ) VALUES (
55     "Mario",
56     "Rossi",
57     "mario.rossi@electrocorp.com",
58     "soldi-123",
59     (SELECT ID FROM Roles WHERE NAME = "Finances" LIMIT 1)
60 );
```

Il Metodo di SQLAlchemy

```
1  #!/usr/bin/env python3
2
3  from app import app, db
4  from models import *
5
6
7  role_admin = Role(
8      name='Administrator',
9      description='The system administrator'
10 )
11
12 role_finances = Role(
13     name='Finances',
14     description='The finances team'
15 )
16
17 db.session.add(role_admin)
18 db.session.add(role_finances)
19 db.session.commit()
20
21
22
23 ## Create the administrator user.
24 user_admin = Employee(
25     surname='Anderson',
26     name='Smith',
27     email='admin@localhost',
28     password='admin',
29     role=role_admin.id
30 )
31
32 db.session.add(user_admin)
33 db.session.commit()
34
35
36 ## Create some generic financial users.
37 role_finances = Role.query.get(2)
38 user_william = Employee(
39     name='William',
40     surname='Blake',
41     email='william.blake@electrocorp.com',
42     password='stonks',
43     role=role_finances.id
```

```
44 )
45
46 user_mario      = Employee(
47     name='Mario',
48     surname='Rossi',
49     email='mario.rossi@electrocorp.com',
50     password='soldi-123',
51     role=role_finances.id
52 )
53
54 db.session.add(user_william)
55 db.session.add(user_mario)
56 db.session.commit()
```

4.5 Alcune query

4.5.1 Membri del team "Financing"

Metodo Tradizionale

```
1 -- Get all the employees that are in the << Finances >> role.
2 SELECT
3     NAME,
4     SURNAME,
5     EMAIL,
6     ROLE
7
8 FROM
9     Users
10
11
12 WHERE
13     ROLE = (SELECT ID FROM Roles WHERE NAME = "Finances");
```

Il Metodo di SQLAlchemy

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

4.5.2 Centrali Elettriche rinnovabili

Il Metodo Tradizionale

```
1 SELECT
2     PowerPlant.ID,
3     PowerPlant.NAME,
4     PowerPlant.DESCRPTION,
5     PowerPlant.LATITUDE,
6     PowerPlant.LONGITUDE,
7     PowerPlant.DATE_CREATED,
8     PowerPlant.DATE_MODIFIED,
9     PowerPlantCategory.NAME as CATEGORY
10
11 FROM
12     PowerPlant
```

```
13 LEFT JOIN PowerPlantCategory
14 ON PowerPlant.CATEGORY = PowerPlantCategory.ID
15
16 WHERE
17 CATEGORY IN (
18     "Hydro",
19     "Solar",
20     "Wind",
21     "Nuclear",
22     "Geothermal"
23 );
```

Il Metodo di SQLAlchemy

```
1 PowerPlant.query.filter(
2     PowerPlant.category.in_(
3         PowerPlantCategory.query.filter_by(name='Hydro')[0].id,
4         PowerPlantCategory.query.filter_by(name='Solar')[0].id,
5         PowerPlantCategory.query.filter_by(name='Wind')[0].id,
6         PowerPlantCategory.query.filter_by(name='Nuclear')[0].id,
7         PowerPlantCategory.query.filter_by(name='Geothermal')[0].id,
8     )
9 )
```