

EINSTEIN - DE LORENZO

Final Exam 2020-2021

Author:

ROBERT-GEORGIAN NITU

Supervisor:

Prof. LUIGI MENCHISE

May 31, 2021

Task

0.1 Translated Task

The electricity distribution company plans to develop a digital archive of all the power lines it manages. The digital archive must allow locating the power cabins and the pylons of the power lines. The candidate, after formulating the appropriate additional assumptions on the characteristics and nature of the problem in question, must identify the specifications that the system must satisfy both from the software point of view and of the network infrastructure. Based on these specifications, illustrate some of the possible solutions, choose one, and explain why.

The candidate, therefore, must develop the entire project of the IT system, in particular: the block diagram of the modules of the software product to be created; the conceptual scheme, the logical scheme, the required DDL instructions for the implementation of the database and some required queries for the development of the software modules identified; the source code of a significant part of the website in a programming language chosen by the candidate.

0.2 Italian Version

La società di distribuzione dell'energia elettrica intende sviluppare un archivio digitale di tutte le linee elettriche che gestisce. Il sistema deve permettere di mappare 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.

Assumptions

1.1 The Problem

Most countries have different methods to distribute electric power. In Italy, for example, a Power Plant produces electricity, which is sent to a High-Voltage Power Cabin via Power Lines travelling on Pylons.

The USA, however, has a different approach:

1. Electricity is made at a generating station by huge generators. Generating stations can use wind, coal, natural gas, or water.
2. The current is sent through transformers to increase the voltage to push the power long distances.
3. The electrical charge goes through high-voltage transmission lines that stretch across the country.
4. It reaches a substation, where the voltage is lowered so it can be sent on smaller power lines.
5. It travels through distribution lines to your neighborhood. Smaller transformers reduce the voltage again to make the power safe to use in our homes. These smaller transformers may be mounted on the poles, or sitting on the ground (they're the big green boxes, called pad mount transformers).
6. It connects to your house and passes through a meter that measures how much your family uses.
7. The electricity goes to the service panel in your basement or garage, where breakers or fuses protect the wires inside your house from being overloaded. (Never touch a service panel! It is only to be operated by your parents or a professional.)
8. The electricity travels through wires inside the walls to the outlets and switches all over your house.

Because this difference, only entities common to every country will be stored:

- Power Plants
- Pylons
- (High-Voltage) Electric Cabins
- Power Lines

The Software will allow the system administrators to perform CRUD operations (Create, Read, Update, Delete) operations on the entities.

1.2 The Company

The only thing we know about the company is that it is an electricity distribution company. Since the task describes it as "the company" and there is no known national constraint, I assume that the company has a monopoly on all the electricity market in the world, just to work on the worst-case-scenario.

I assume that the data to be added to the archive also includes the power plants and the power lines.

Since there is no constraint on how the location should be stored, I will use the GPS coordinates system.

Since it's not specified who can access the archive, I assume that only the system administrators should access it. The system administrators can insert and delete data directly from the archive's web page (<https://admin.electrocorp.com/archive>), which only allows them to access it.

The archive will be initialized with official data from sources like NASA, data.europa.eu, etc., and will be cited in the source code.

TODO: wasd

1.3 The Infrastructure

The Company has different workplaces around the world, the servers are underwater (like Microsoft does) for efficiency and cost reasons, and are divided in different subdomains, all connected to the same database.

The DBMS used is PostgreSQL, because it's super-scalable, and allows us to have a database cluster.

There are different subdomains, but the ones developed in this project are:

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

(www.electrocorp.com is already taken by a company, and is not affiliated with this project. the domain name is overwritten in my `/etc/hosts` file to point to my project, please do not connect to it externally).

The servers are running SUSE Linux, an Enterprise GNU/Linux distribution for Servers.

The network is composed of different subnetworks, one for each cluster managed by Kubernetes.

Each subnetwork has different firewall rules, listed here:

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

The subnet 32.1.0.0/16 is owned by the company and is used to allow employees to access the administrative page.

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

And here is the firewall configuration on a GNU/Linux router.

```

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

```

The hosts on the same subnetwork can only connect to each other on the DBMS cluster.

Here is the docker-compose.yml configuration file, that also specifies the networking configuration.

```

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
```

... and the /etc/hosts file (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

```

1.4 The Website

The archive is accessible at <https://admin.electrocorp.com/archive> by the system administrators only, and the page allows them to browse, add, remove, and update data from the archive.

The System Administrators also have the power to manage roles and users.

Here are the Access Control rules for the users on each website.

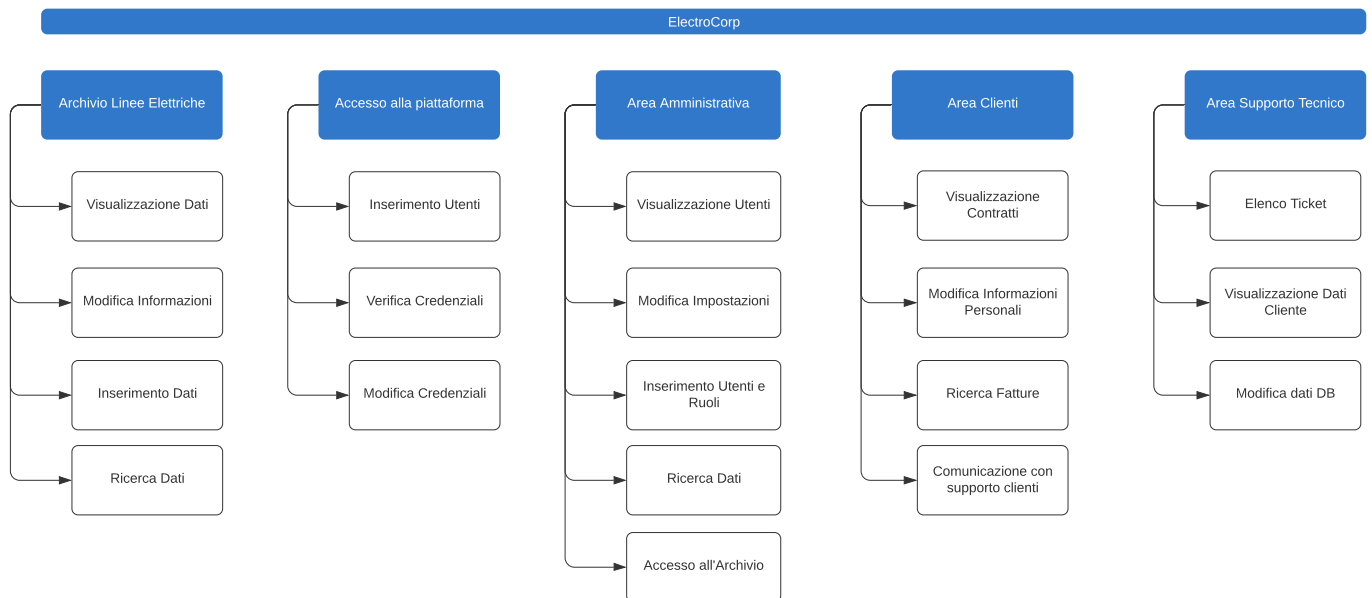
Access Control (admin.electrocorp.com)					
	Archive	Control Panel	Login	Finances	Logs
Administrator	ALL	ALL	AUTHENTICATE	ALL	READ
Finances			AUTHENTICATE	ALL	
not auth			AUTHENTICATE		

This is only accessible by the employees.

This is the block diagrams.

Archivio

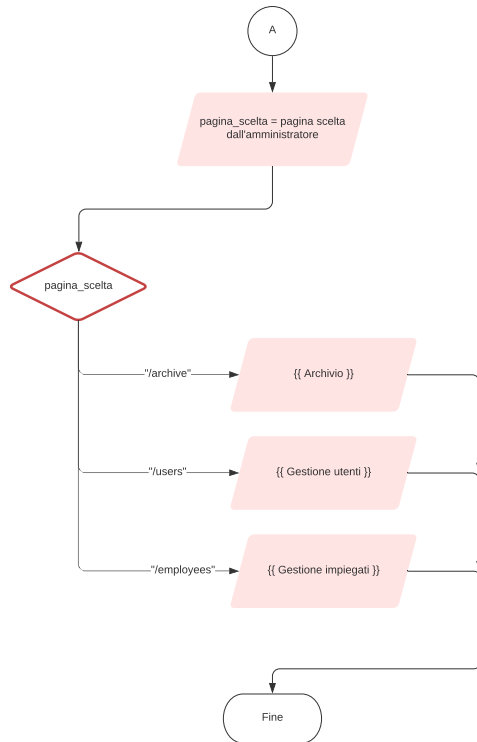
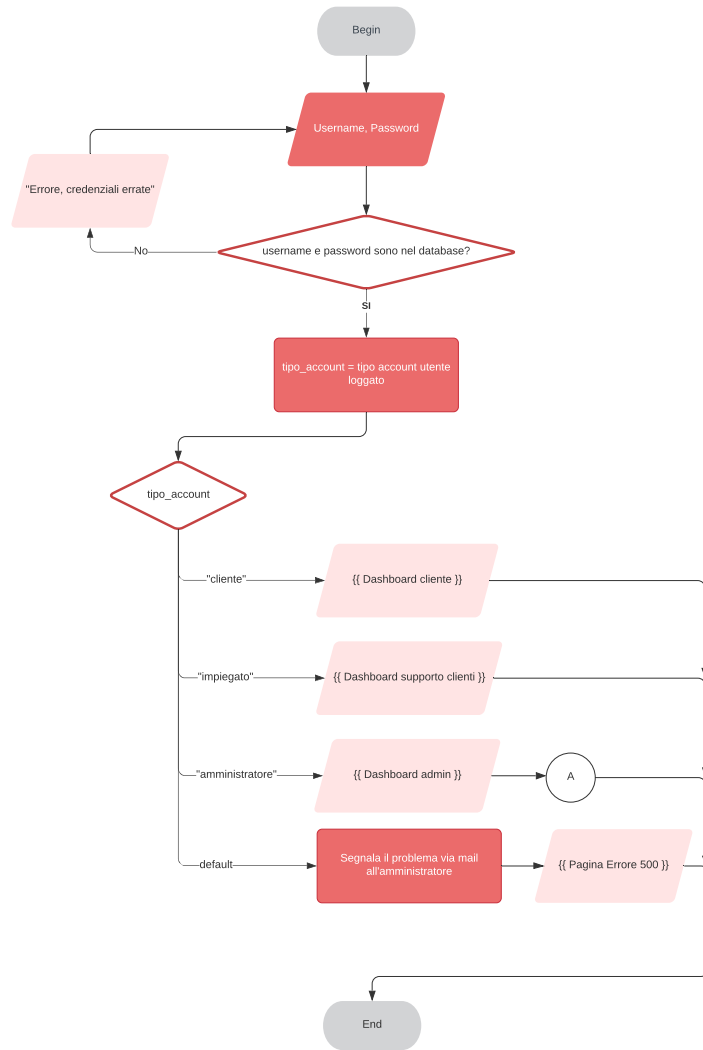
Zmlyc3Q bGFzdA | May 15, 2021



And this is a Flow Chart.

Diagramma di flusso dei moduli

Zmilyc3Q bGFZdA | May 15, 2021



Full Stack Development

2.1 The Stack

2.1.1 Docker

Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. Because all of the containers share the services of a single operating system kernel, they use fewer resources than virtual machines.

The service has both free and premium tiers. The software that hosts the containers is called Docker Engine. It was first started in 2013 and is developed by Docker, Inc.

2.1.2 Flask on PyPy3

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

Applications that use the Flask framework include Pinterest and LinkedIn.

Flask has several components, including:

- **Werkzeug:** Werkzeug (German for "tool") is a utility library for the Python programming language, in other words a toolkit for Web Server Gateway Interface (WSGI) applications, and is licensed under a BSD License. Werkzeug can realize software objects for request, response, and utility functions. It can be used to build a custom software framework on top of it and supports Python 2.7 and 3.5 and later;
- **Jinja:** Jinja is a template engine for the Python programming language and is licensed under a BSD License. Similar to the Django web framework, it handles templates in a sandbox;
- **MarkupSafe:** MarkupSafe is a string handling library for the Python programming language, licensed under a BSD license. The eponymous MarkupSafe type extends the Python string type and marks its contents as "safe"; combining MarkupSafe with regular strings automatically escapes the unmarked strings, while avoiding double escaping of already marked strings;

- **ItsDangerous:** ItsDangerous is a safe data serialization library for the Python programming language, licensed under a BSD license. It is used to store the session of a Flask application in a cookie without allowing users to tamper with the session contents.

2.1.3 SQLAlchemy

SQLAlchemy is an open-source SQL toolkit and object-relational mapper (ORM) for the Python programming language released under the MIT License.

SQLAlchemy's philosophy is that relational databases behave less like object collections as the scale gets larger and performance starts being a concern, while object collections behave less like tables and rows as more abstraction is designed into them. For this reason it has adopted the data mapper pattern (similar to Hibernate for Java) rather than the active record pattern used by a number of other object-relational mappers. However, optional plugins allow users to develop using declarative syntax.

SQLAlchemy was first released in February 2006 and has quickly become one of the most widely used object-relational mapping tools in the Python community, alongside Django's ORM.

2.1.4 PostgreSQL

PostgreSQL, also known as Postgres, is a free and open-source relational database management system (RDBMS) emphasizing extensibility and SQL compliance. It was originally named POSTGRES, referring to its origins as a successor to the Ingres database developed at the University of California, Berkeley. In 1996, the project was renamed to PostgreSQL to reflect its support for SQL. After a review in 2007, the development team decided to keep the name PostgreSQL and the alias Postgres.

PostgreSQL features transactions with Atomicity, Consistency, Isolation, Durability (ACID) properties, automatically updatable views, materialized views, triggers, foreign keys, and stored procedures. It is designed to handle a range of workloads, from single machines to data warehouses or Web services with many concurrent users. It is the default database for macOS Server and is also available for Windows, Linux, FreeBSD, and OpenBSD.

2.1.5 React

React (also known as React.js or ReactJS) is an open-source front-end JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

2.2 The Website

2.2.1 The API server

This is the code for the API server.

```
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 = 'Authentification Failed'
44
45     except Exception as e:
46         data = None
47         error = str(e)
48
49     finally:
50         return jsonify({
51             'error': error,
52             'data': data
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     [
375         x.latitude, x.longitude
376     ]
377 ],
378
379     'properties': {
380         'id': '#',
381         'name': 'Distribution Line',
382         'color': '#8B008B',
383         'type': 'Distribution'
384     }
385 })
386
387 except Exception as e:
388     error = str(traceback.print_exc())
389     data = []
390
391 finally:
392     return jsonify({
393         'error': error,
394         'data': data,
395     }), 200
396
397
398 @app.route('/power_line/categories', methods=['GET'])
399 def get_power_line_categories():
400     data = []
401     for x in models.PowerLineCategory.query.all():
402         data.append(x.toApiData())
403
404     return jsonify({
405         'data': data
406     }), 200
407
408
409 # Users endpoint
410 @app.route('/user/all', methods=['POST'])
411 def get_users():
412     checked = check_authorization([1])
413     if checked is not None:
414         return checked
415
416     data = []
417     for x in models.Employee.query.all():
418         data.append(x.toApiData())
419
420     return jsonify({
421         'error': None,
422         'data': data
423     })
424
425
426 # Useful functions
427 def check_authorization(user_roles):
428     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
```

Since we have an API server (and a cluster of servers) we will keep the information in a web token. JSON Web Tokens (JWT) will be used for storing session data client-side in a secure manner.

This is the middleware for using 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 The Frontend

The frontend of both admin.electrocorp.com and (www.)electrocorp.com are built on React. I will use several libraries, such as:

- **Tailwind.css**: a web framework, which relieves me from having to write loads of CSS, and use Tailwind directly in my HTML page;
- **React Router**: a React.js library to render multiple pages in the same code;
- **Globe.gl**: a React.js library to render a globe.

This is the App.js page:

```
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;
```


System Administration

3.1 Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. It uses YAML files to configure the application's services and performs the creation and start-up process of all the containers with a single command. The docker-compose CLI utility allows users to run commands on multiple containers at once, for example, building images, scaling containers, running containers that were stopped, and more. Commands related to image manipulation, or user-interactive options, are not relevant in Docker Compose because they address one container. The docker-compose.yml file is used to define an application's services and includes various configuration options. For example, the build option defines configuration options such as the Dockerfile path, the command option allows one to override default Docker commands, and more. The first public beta version of Docker Compose (version 0.0.1) was released on December 21, 2013. The first production-ready version (1.0) was made available on October 16, 2014.

3.2 Linux

Linux is a family of open-source Unix-like operating systems based on the Linux kernel, an operating system kernel first released on September 17, 1991, by Linus Torvalds. Linux is typically packaged in a Linux distribution.

Distributions include the Linux kernel and supporting system software and libraries, many of which are provided by the GNU Project. Many Linux distributions use the word "Linux" in their name, but the Free Software Foundation uses the name "GNU/Linux" to emphasize the importance of GNU software, causing some controversy.

3.3 iptables

iptables is a user-space utility program that allows a system administrator to configure the IP packet filter rules of the Linux kernel firewall, implemented as different Netfilter modules. The filters are organized in different tables, which contain chains of rules for how to treat network traffic packets. Different kernel modules and programs are currently used for different protocols; iptables applies to IPv4, ip6tables to IPv6, arptables to ARP, and ebtables to Ethernet frames.

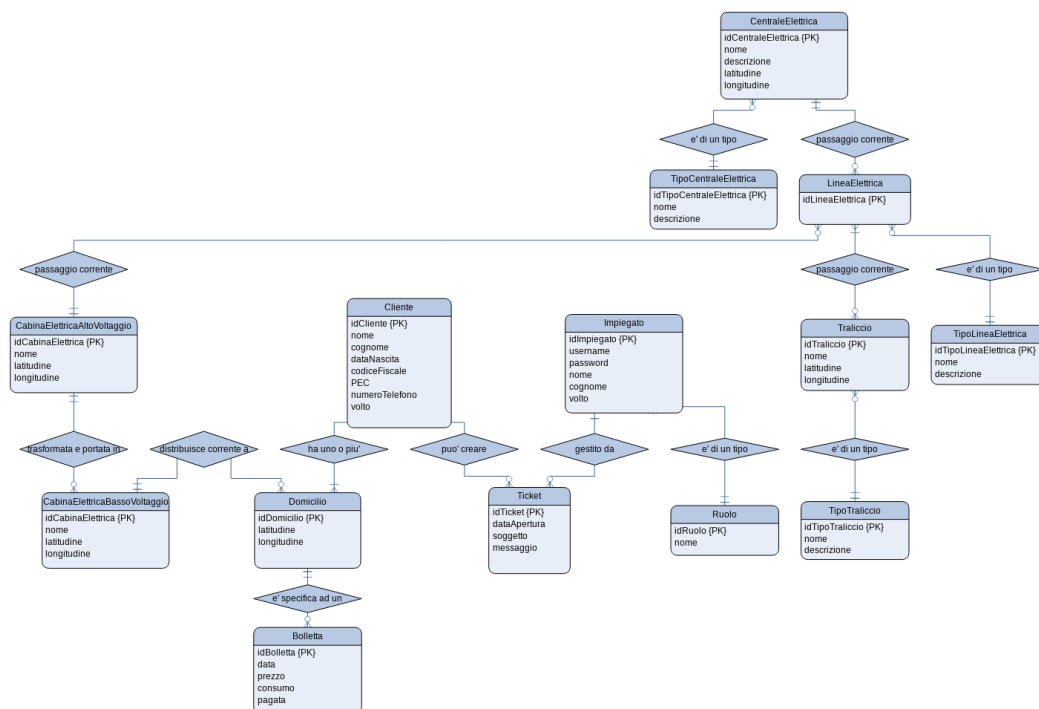
iptables requires elevated privileges to operate and must be executed by user root, otherwise it fails to function. On most Linux systems, iptables is installed as /usr/sbin/iptables and documented in its man pages, which can be opened using

man iptables when installed. It may also be found in /sbin/iptables, but since iptables is more like a service rather than an "essential binary", the preferred location remains /usr/sbin.

Software Modules

4.1 DB Normalization

4.1.1 ER



4.1.2 1FN

The scheme is in 1st normal form since all its attributes are atomic and have the same type.

The first normalization form is the easiest, you just need to respect the basic rules, which are:

- not using compound fields
- having the same number of rows and columns
- all records are unique

4.1.3 2FN

Since we don't have compound keys, we don't need to do anything to check 2nd normal form.

4.1.3 3FN

All columns directly depend on the primary key, it is reduced to 3rd normal form.

4.2 Logic Scheme

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

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

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

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

Traliccio						
Name	Type	Size	Key	Null	Default	Extra
idTraliccio	INTEGER		PRIMARY	NO		AUTO INCREMENT
latitude	FLOAT			NO		
longitude	FLOAT			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
email	VARCHAR	300		NO		UNIQUE

Domicilio						
Name	Type	Size	Key	Null	Default	Extra
idDomicilio	INTEGER		PRIMARY	NO		AUTO INCREMENT
idCliente	INTEGER		FOREIGN	NO		
latitudine	FLOAT			NO		
longitudine	FLOAT			NO		
idLineaElettrica	INTEGER		FOREIGN	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 UNIQUE
username	VARCHAR	300		NO		
password	CHAR	128		NO		
nome	VARCHAR	64		NO		
cognome	VARCHAR	64		NO		
ruolo	INTEGER		FOREIGN	NO		

4.3 DDL

Traditional Method

The traditional method is not used in this project.

```

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 );
```

The SQLAlchemy Way

restricted/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
74     # Location of the power plant
75     latitude = db.Column(db.Float, nullable=False)
76     longitude = db.Column(db.Float, nullable=False)
77
78     def __repr__(self):
79         return '<PowerPlant %s %f:%f>' % (self.name, self.latitude, self.longitude)
80
81     def toApiData(self):
82         return {
83             'id': self.id,
84             'name': self.name,
85             'type': 'Power Plant',
86             'description': self.description,
87             'category': PowerPlantCategory.query.get(self.category).name,
88             'location': [self.latitude, self.longitude],
89             'lat': self.latitude,
90             'lng': self.longitude,
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)
198     destination = HighVoltagePowerCabin.query.get(self.destination)
199
200     # This is the start (the power plant)
201     routes.append([origin.latitude, origin.longitude])
202
203     first_item = None
204     while True:
205         foobar = Pylon1_Pylon2.query.filter_by(
206             line = self.id,
207             pylon1 = first_item
208         )
209
210         if foobar.count() == 0:
211             break
212
213         pylon_data = Pylon.query.get(foobar.first().pylon2)
214         if pylon_data is None:
215             break
216
217         routes.append([pylon_data.latitude, pylon_data.longitude])
218         first_item = pylon_data.id
219
220     # This is the end (the power cabin)
221     routes.append([destination.latitude, destination.longitude])
222
223     # Return the calculated routes.
224     return routes
225
226 def __repr__(self):
227     return '<PowerLine %f:%f>' % (self.source, self.destination)
228
229 def toApiData(self):
230     return {
231         'coords': self.calculateRoute(),
232         'properties': {
233             'id': self.id,
234             'name': self.line_name,
235             'color': 'DarkGray',
236             'type': 'Power Line',
237         }
238     }
239
240 class Pylon1_Pylon2(Base):
241     __tablename__ = 'pylon1_pylon2'
242
243     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     # 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
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 Inserting data

Traditional method

```

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 );

```

The SQLAlchemy Way

```

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 Some queries

4.5.1 Members of the financing team

Traditional Method

```

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");

```

The SQLAlchemy Way

```

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

```

4.5.2 Green Power Plants

Traditional Method

```
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     );
```

The SQLAlchemy Way

```
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 )
```