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

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

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 affilated 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 acces 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.

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

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.

```
version: '3.4'

```

```
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
```

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

        networks:
            database_net:
              ipv4_address: 172.22.4.5

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

        networks:
            mail_net:
                ipv4_address: 172.22.5.5



volumes:
    psql_database:
    mail:



networks:
    www_net:
        external:
            name: www

    api_net:
        external:
            name: api

    admin_net:
        external:
            name: admin

    database_net:
        external:
            name: database

    mail_net:
        external:
            name: mail
```

... and the /etc/hosts file (C: \Windows \system32 \etc \hosts)

```
127.0.0.1 localhost
127.0.1.1 PwnStation

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



# Progetto
172.22.2.5  api.electrocorp.com
```

```
172.22.1.5  electrocorp.com www.electrocorp.com
172.22.3.5  admin.electrocorp.com
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.

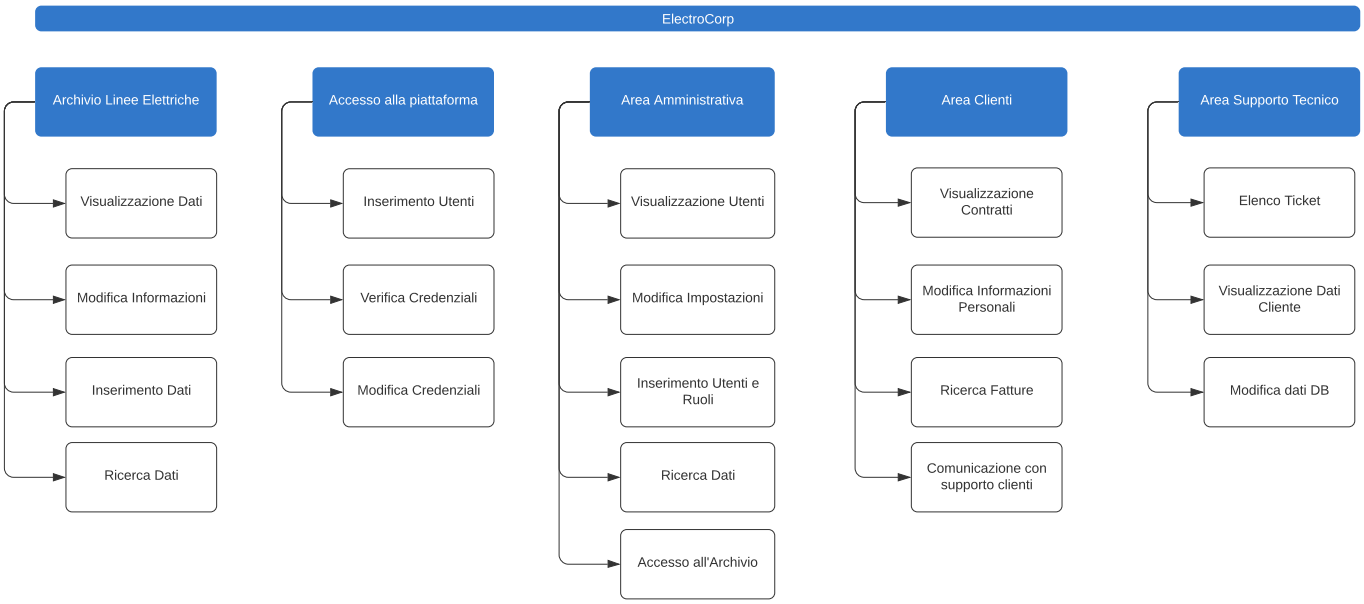
Access Control (admin.electrocorp.com)					
	Archive	Control Panel	Login	Finances	Logs
Administrator	ALL	ALL	AUTHENTICATE	ALL	READ
Finances			AUTHENTICATE	ALL	
not auth			AUTHENTICATE		

This is only accessible by the employees.

This is the block diagrams.

Archivio

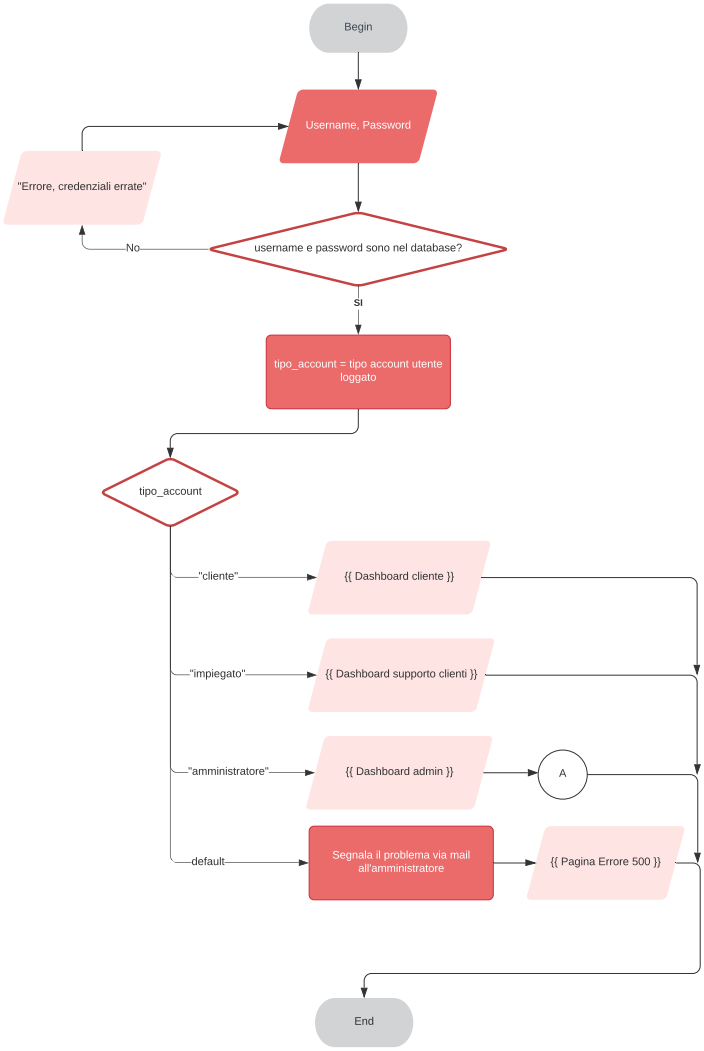
Zmlyc3Q bGFzdA | May 15, 2021

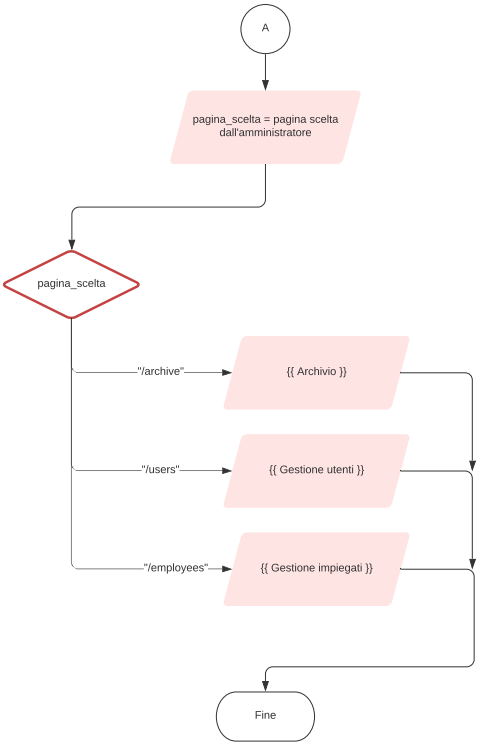


And this is a Flow Chart.

Diagramma di flusso dei moduli

Zmlyc3Q 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.

```
import json

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


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

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

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

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

        else:
            error = 'Authentification Failed'

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

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


# Power Plant endpoints.
```

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

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

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



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

    return jsonify({
        'data': data
    })


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

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

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

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


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

```

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

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


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

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

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


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

    try:
        foo = models.PowerPlant.query.get(id)
        db.session.delete(foo)
        db.session.commit()
        return jsonify({
            'error': None,
        }), 200

    except Exception as e:
        return jsonify({
            'error': str(e)
        }), 404



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

    data = []
    for x in models.HighVoltagePowerCabin.query.all():
        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),
```

```
            'data': None
        }), 404


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

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

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


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

    try:
        foo = models.Pylon.query.get(id)
        db.session.delete(foo)
        db.session.commit()
        return jsonify({
            'error': None,
        }), 200

    except Exception as e:
        return jsonify({
            'error': str(e)
        }), 404


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

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

        for x in models.LowVoltagePowerCabin.query.all():
            source = models.HighVoltagePowerCabin.query.get(x.power_cabin)
            if source is None: continue
            data.append({
                'coords': [
                    [
```

```
                        source.latitude, source.longitude
                    ],

                    [
                        x.latitude, x.longitude
                    ]
                ],

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

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

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


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

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


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

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

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


# Useful functions
def check_authorization(user_roles):
    if g.data == None or g.data.get('role', None) not in user_roles:
        return jsonify({
            'error': 'Unauthorized',
            'data': str(g.data),
        }), 401

```

```
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';
```

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

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


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


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

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

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

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

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

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

                <Route path='*'>
                    <Error404 />
                </Route>
            </Switch>
        </Router>
    );
}

export default App;
```

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.

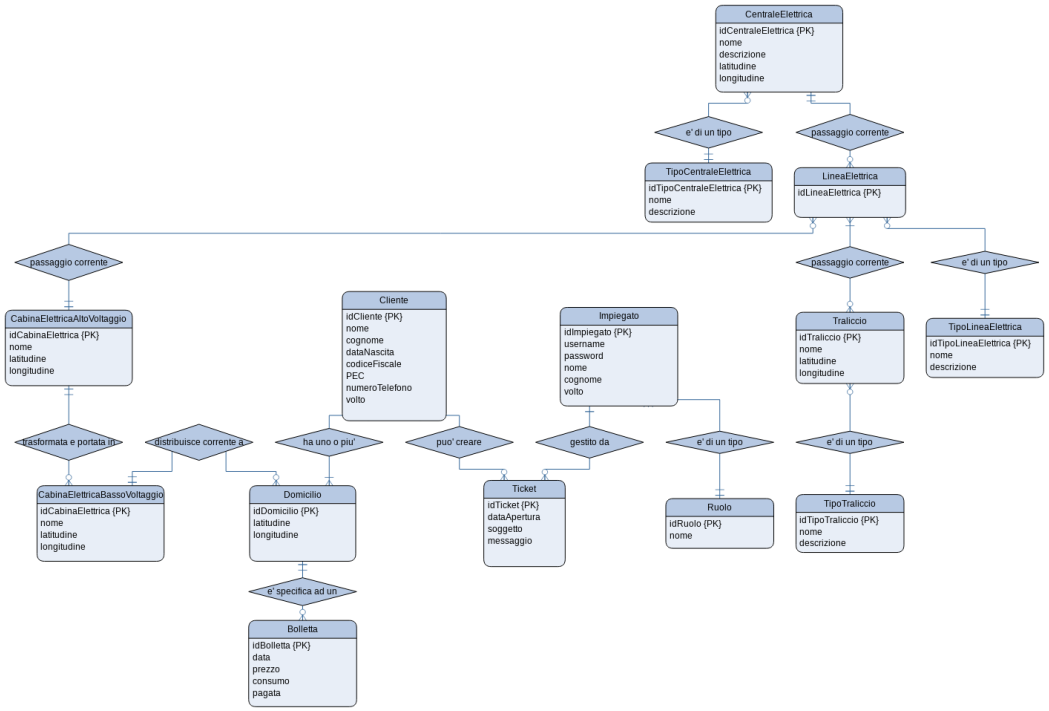
iptables requires elevated privileges to operate and must be executed by user root, otherwise it fails to function. On most Linux systems, iptables is installed as /usr/sbin/iptables and documented in its man pages, which can be opened using

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 compount 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		PRIMARY	NO		AUTO INCREMENT
nome	VARCHAR	128		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		PRIMARY	NO		AUTO INCREMENT
nome	VARCHAR	128		YES		
latitudine	FLOAT			NO		
longitudine	FLOAT			NO		

PassaggioLinea						
Name	Type	Size	Key	Null	Default	Extra
idPassaggio	INTEGER		PRIMARY	NO		AUTO INCREMENT
idLinea	INTEGER		FOREIGN	NO	1	
iterazione	INTEGER			NO		AUTO INCREMENT
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		PRIMARY	NO		AUTO INCREMENT
nome	VARCHAR	64		NO		
cognome	VARCHAR	64		NO		
dataNascita	DATE			NO		
codiceFiscale	VARCHAR	16		NO		UNIQUE
numeroTelefono	VARCHAR	16		NO		UNIQUE
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
username	VARCHAR	300		NO		UNIQUE
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.

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

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

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

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

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


CREATE TABLE roles (
        id INTEGER NOT NULL,
```

```
        date_created DATETIME,
        date_modified DATETIME,
        name VARCHAR(128) NOT NULL,
        description VARCHAR(409) NOT NULL,
        PRIMARY KEY (id),
        UNIQUE (name)
);


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


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


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


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


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

```

```
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,
```

```
        longitude FLOAT NOT NULL,
        power_cabin INTEGER NOT NULL,
        PRIMARY KEY (id),
        FOREIGN KEY(power_cabin) REFERENCES high_voltage_power_cabins (id)
);


CREATE TABLE power_lines (
        id INTEGER NOT NULL,
        date_created DATETIME,
        date_modified DATETIME,
        line_name VARCHAR(128),
        line_type INTEGER NOT NULL,
        source INTEGER NOT NULL,
        destination INTEGER NOT NULL,
        PRIMARY KEY (id),
        FOREIGN KEY(line_type) REFERENCES power_line_categories (id),
        FOREIGN KEY(source) REFERENCES power_plants (id),
        FOREIGN KEY(destination) REFERENCES high_voltage_power_cabins (id)
);


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


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


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

The SQLAlchemy Way

`restricted/code/models.py`

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

from app import db


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

    id            = db.Column(db.Integer, primary_key=True)
    date_created  = db.Column(db.DateTime, default=db.func.current_timestamp())
    date_modified = db.Column(db.DateTime, default=db.func.current_timestamp(),
    onupdate=db.func.current_timestamp())



# Define an Employee model
class Employee(Base):
    __tablename__ = 'employees'

    # User Name
    name    = db.Column(db.String(128), nullable=False)
    surname = db.Column(db.String(128), nullable=False)

    # Identification Data: email & password
    email    = db.Column(db.String(128), nullable=False, unique=True)
    password = db.Column(db.String(192), nullable=False)

    # Authorisation Data: role & status
    role     = db.Column(db.Integer, db.ForeignKey('roles.id'), nullable=False)

    def __repr__(self):
        return '<User %r>' % (self.name)

    def toApiData(self):
        return {
            'name': '%s %s' % (self.surname, self.name),
            'email': '%s' % self.email,
            'role': '%s' % Role.query.get(self.role).name
        }


# Defining the Role model.
class Role(Base):
    __tablename__ = 'roles'

    # Role information
    name            = db.Column(db.String(128), nullable=False, unique=True)
    description     = db.Column(db.String(409), nullable=False)

    # Relationship data
    users           = db.relationship('Employee', backref='roles', lazy=False)

    def __repr__(self):
        return '<Role %r>' % (self.name)

    def toApiData(self):
```

```
        return {
            'name': self.name,
            'description': self.description
        }


# Defining the PowerPlant model.
class PowerPlant(Base):
    __tablename__ = 'power_plants'

    # Information about the power plant
    name        = db.Column(db.String(128), nullable=False)
    description = db.Column(db.String(4096), nullable=True)
    category    = db.Column(db.Integer, db.ForeignKey('power_plant_categories.id'),
    nullable=False)

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

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

    def toApiData(self):
        return {
            'id': self.id,
            'name': self.name,
            'type': 'Power Plant',
            'description': self.description,
            'category': PowerPlantCategory.query.get(self.category).name,
            'location': [self.latitude, self.longitude],
            'lat': self.latitude,
            'lng': self.longitude,
        }


class Pylon(Base):
    __tablename__ = 'pylons'

    # Information about the pylon
    name        = db.Column(db.String(128), nullable=True)
    category    = db.Column(db.Integer, db.ForeignKey('pylon_categories.id'))

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

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

    def toApiData(self):
        return {
            'id': self.id,
            'name': self.name,
            'type': 'Pylon',
            'category': PylonCategory.query.get(self.category).name,
            'color': '#00ff00',
            'lat': self.latitude,
            'lng': self.longitude,
            'size': 0.0051215125,
            'radius': 0.03,
        }

```

```
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')
    , nullable=False)
160
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'),
        nullable=False)
186
187     # Path of the electric line
188     source      = db.Column(db.Integer, db.ForeignKey('power_plants.id'), nullable=
        False)
189     destination = db.Column(db.Integer, db.ForeignKey('high_voltage_power_cabins.id')
        , nullable=False)
190
191     # Method to calculate the path.
192     def calculateRoute(self):
193         routes = []
194         origin = PowerPlant.query.get(self.source)
195         destination = HighVoltagePowerCabin.query.get(self.destination)
196
197         # This is the start (the power plant)
198         routes.append([origin.latitude, origin.longitude])
199
200         first_item = None
201         while True:
202             foobar = Pylon1_Pylon2.query.filter_by(
203                 line = self.id,
204                 pylon1 = first_item
205             )
206
207             if foobar.count() == 0:
208                 break
209
210             pylon_data = Pylon.query.get(foobar.first().pylon2)
211             if pylon_data is None:
212                 break
213
214             routes.append([pylon_data.latitude, pylon_data.longitude])
215             first_item = pylon_data.id
216
217         # This is the end (the power cabin)
218         routes.append([destination.latitude, destination.longitude])
219
220         # Return the calculated routes.
221         return routes
222
223     def __repr__(self):
224         return '<PowerLine %f:%f>' % (self.source, self.destination)
225
226     def toApiData(self):
227         return {
228             'coords': self.calculateRoute(),
229             'properties': {
230                 'id': self.id,
231                 'name': self.line_name,
232                 'color': 'DarkGray',
233                 'type': 'Power Line',
234             }
235         }
236
237
238 class Pylon1_Pylon2(Base):
239     __tablename__ = 'pylon1_pylon2'
240
241     line   = db.Column(db.Integer, db.ForeignKey('power_lines.id'), nullable=False)
```

```
242     pylon1 = db.Column(db.Integer, db.ForeignKey('pylons.id'), nullable=True)
243     pylon2 = db.Column(db.Integer, db.ForeignKey('pylons.id'), nullable=True)
244 
245     def __repr__(self) -> str:
246         return '<PylonConnection from %d to %d>' % (self.pylon1 or 0, self.pylon2 or
        0)
247 
248 
249 # User informations
250 class Customer(Base):
251     __tablename__ = 'customers'
252 
253     # Informations about the customer.
254     name = db.Column(db.String(64), nullable=False)
255     surname = db.Column(db.String(128), nullable=False)
256     birthdate = db.Column(db.DateTime, nullable=False)
257     fiscal_code = db.Column(db.String(16), nullable=False)
258     pec = db.Column(db.String(300), nullable=False)
259     phone_number = db.Column(db.String(20), nullable=False)
260 
261     def __repr__(self) -> str:
262         return '<Customer@%d %s %s>' % (self.id, self.name, self.surname)
263 
264     def toApiData(self):
265         return {
266             'id': self.id,
267             'name': self.name,
268             'surname': self.surname,
269             'birthdate': self.birthdate,
270             'fiscal_code': self.fiscal_code,
271             'pec': self.pec,
272             'phone_number': self.phone_number
273         }
274 
275 
276 class Residence(Base):
277     __tablename__ = 'residences'
278 
279     # Information
280     latitude = db.Column(db.Float, nullable=False)
281     longitude = db.Column(db.Float, nullable=False)
282 
283     # Relationships
284     owner = db.Column(db.Integer, db.ForeignKey('customers.id'), nullable=False)
285     power_cabin = db.Column(db.Integer, db.ForeignKey('low_voltage_power_cabins.id'),
         nullable=False)
286 
287     def __repr__(self) -> str:
288         return '<Residence %d of %s at %f:%f>' % (self.id, repr(Customer.query.get(
        self.owner)), self.latitude, self.longitude)
289 
290     def toApiData(self):
291         return {
292             'id': self.id,
293             'owner': Customer.query.get(self.owner).toApiData(),
294             'location': [self.latitude, self.longitude],
295             'cabin': LowVoltagePowerCabin.query.get(self.power_cabin).toApiData()
296         }
297 
298 
299 class Bill(Base):
300     __tablename__ = 'bills'
301 
```

```
302     # Information
303     consumption = db.Column(db.Float, nullable=False)
304     paid = db.Column(db.Boolean, default=False)
305 
306     # Relationship
307     residence = db.Column(db.Integer, db.ForeignKey('residences.id'))
308 
309     def __repr__(self) -> str:
310         return '<Bill %d to %s>' % (self.id, repr(Residence.query.get(self.residence)
        ))
311 
312 
313 class Ticket(Base):
314     __tablename__ = 'tickets'
315 
316     # Information
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 deleetus
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 
```

```
class PowerLineCategory(Base):
    __tablename__ = 'power_line_categories'

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

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

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

Per creare le tabelle del database:

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

4.4 Inserting data

Traditional method

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

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


-- Add the users.
INSERT INTO Users (
    NAME,
    SURNAME,
    EMAIL,
    PASSWORD,
    ROLE
) VALUES (
    "Anderson",
    "Smith",
    "admin@localhost",
    "admin",
```

```
    (SELECT ID FROM Roles WHERE NAME = "Administrator" LIMIT 1)
);

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

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

The SQLAlchemy Way

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

from app import app, db
from models import *


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

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

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



## Create the administrator user.
user_admin      = Employee(
    surname='Anderson',
    name='Smith',
    email='admin@localhost',
```

```
    password='admin',
    role=role_admin.id
)

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


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

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

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

4.5 Some queries

4.5.1 Members of the financing team

Traditional Method

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

FROM
    Users


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

The SQLAlchemy Way

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

4.5.2 Green Power Plants

Traditional Method

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

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

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

The SQLAlchemy Way

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