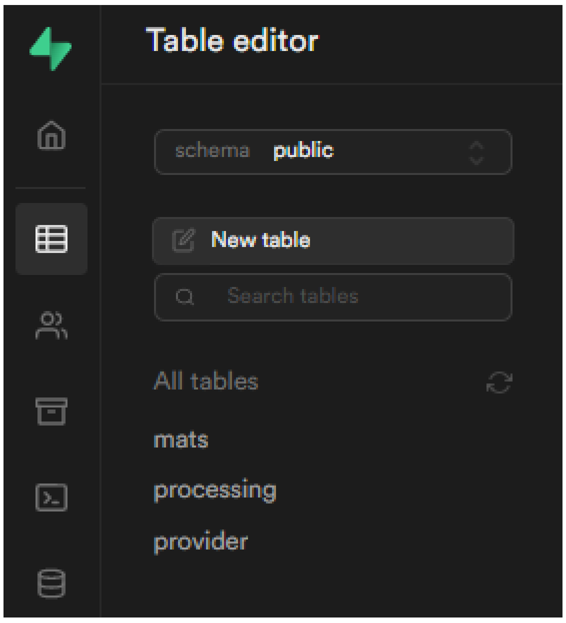
Appendix A : Programming code

In order to help the future developers of the application this appendix will explain all the steps needed to modify the database and the code of the application.

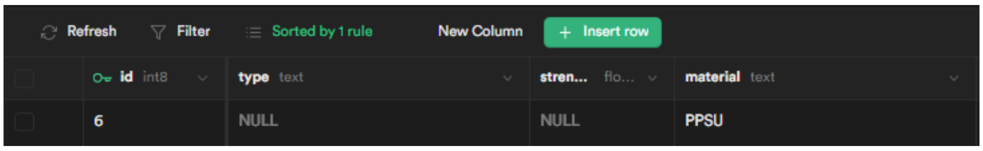
A.1 Database

To modify the database, during the phase of developing of the app the application that was used is Supabase. To access the database and modify it, it is necessary to write a mail to manelmartirosich@gmail.com and I will give the access. It is important to say that Supabase stores database for free, but when the application will be finished and more data need to be stored, a new database will be requested.

Once you have access to the database, you have to click on the “table editor” and then select the table that you want to modify.



Then in case you want to add a row just click on “insert row” and fulfill all the information. If you need to add information about a material that is already on the database just click on “NULL” and modify the value. It is important to consider before modifying the value if the column is a string or an integer. Also you can add a new parameter of the table using “new column”.



id int8	type text	stren... flo...	material text
6	NULL	NULL	PPSU

To modify the users database you don’t need to do anything, it is automatic with the programming code of the application.

A.2 Python files

A.2.1 admin.py

In this file you need to register every existing function that you import and register the models that you will use.

```
from pyexpat import features
from django.contrib import admin
from .models import Feature
# Register your models here.
admin.site.register(Feature)
```

A.2.2 models.py

In this file you can create and modify the models used in the application.

```
from django.db import models

# Create your models here.
class Feature(models.Model):
    name = models.CharField(max_length=100)
    details = models.CharField(max_length=500)
```

A.2.3 urls.py

In this file you have to add the entire html that will be used in the application. This is important because you have to specify values in order to have access and modify its tasks in the view.py file.

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('register', views.register, name='register'),
    path('login', views.login, name='login'),
    path('logout', views.logout, name='logout'),
    path('design', views.design, name='design'),
    path('results', views.results, name='results'),
    path('tech', views.tech, name='tech')
]
```

A.2.4 view.py

In this file is listed all the programming code. For every path that you have created in the urls.py you have to define the function here.

A.2.4.1 Index

Index is the main page of the application, so in this page the only request is that every time you click on a button it redirects you correctly. So it is necessary to use the features.

```
def index(request):
    features = Feature.objects.all()
    return render(request, 'index.html',{'features': features})
```

A.2.4.2 Login

This function is used to log in in the application in order to use it. So the first step is to establish connection with the users database. Also as is done before, you have to define the function. The DATABASE_PASSWORD, DATABASE_URL_U and DATABASE_PORT are defined in another file in order to protect the access to the database.

```
def login(request):
    conexion = psycopg2.connect(user='postgres',
password=os.getenv('DATABASE_PASSWORD'),
host=os.getenv('DATABASE_URL_U'),
port=int(os.getenv('DATABASE_PORT')), database='postgres')
    cursor = conexion.cursor()
```

In order to submit data to be processed for the server it is important to use the POST method instead of GET method. For processing which username and password is using the user, the method POST is needed.

```
 if request.method == 'POST':
        username = request.POST['username']
```

```
 else:
        conexion.close()
        return render (request, 'login.html')
```

Once the username is defined it is important to see if it is registered in the users database. To check it, the username is used to look in the database, if the username is in there, the password and the mail become a variable in the code. If it is not the length of the “l” variable becomes 0.

```
cursor.execute(
            "SELECT mail, password"
            " FROM usuaris"
            " WHERE usuari = %s"
            , (username,))
        l=cursor.fetchall()
```

Obviously the password is encrypted to protect the data of the users. So to check if it is the same password in the database that in the POST method is needed to encode the password of the database and hash and encode the password of the POST method. If those to passwords are different the function ends in here. Also if the length of “l” is 0 the function ends here.

```
if len(l) == 1:
            str_hash = l[0][1]
            hash = str_hash.encode('utf-8')
            if hash !=
bcrypt.hashpw(request.POST['password'].encode('utf-8'),hash):
                messages.info(request, 'Invalid Credentials')
                conexion.close()
                return redirect ('login')
```

```
else:
            messages.info(request, 'Invalid Credentials')
            conexion.close()
            return redirect ('login')
```

The application has an internal database of users in order to have control of who uses it. But every time you run a new version of the server it is deleted. So this is why the external database users is used. Once you have checked if the user exists in the external database it is important to see if it is in the internal database and if it has not been registered at another time and then make the internal log in.

```
if
User.objects.filter(username=request.POST['username']).exists()
:

                usero=auth.authenticate(username=username,
password=str(hash))
                auth.login(request, usero)
                conexion.close()
                return redirect('/')
            else:
                user =
User.objects.create_user(username=request.POST['username'],
password=str(hash))
                user.save();
                user=auth.authenticate(username=username,
password=str(hash))
                auth.login(request, user)
                conexion.close()
                return redirect('/')
```

A.2.4.3 Register

This time the first step is to check if the password matches with the parameter “repeat the password”. As is done before all the parameters introduced by the user, is treated with the POST method.

```
def register(request):
    if request.method == 'POST':
        if request.POST['password'] ==
request.POST['password2']:
```

As is done in the login function, the second step is getting connection with the database. If the username or the mail are already used the function ends here.

If there is no match between the data given by the user and the existing in the database, the user is stored in the database. The password is crypt before been introduced to the database.

Once the user is registered, the application will redirect it to the login page.

```
conexion = psycopg2.connect(user='postgres',
password=os.getenv('DATABASE_PASSWORD'),
host=os.getenv('DATABASE_URL_U'),
port=int(os.getenv('DATABASE_PORT')), database='postgres')
            cursor = conexion.cursor()
            cursor.execute(
                "SELECT mail, usuari"
                " FROM usuaris"
                " WHERE usuari = %s OR mail = %s"
                ,
(request.POST['username'],request.POST['email'],))
            l=cursor.fetchall()
            conexion.close()
            cursor.close()
            if len(l)!=0:
                messages.info(request, 'User Already Used')
                return redirect('register')
else:
                conexion = psycopg2.connect(user='postgres',
password=os.getenv('DATABASE_PASSWORD'),
host=os.getenv('DATABASE_URL_U'),
port=int(os.getenv('DATABASE_PORT')), database='postgres')
                cursor = conexion.cursor()
                salt = bcrypt.gensalt(10)
                password =
bcrypt.hashpw(request.POST['password'].encode('utf-8'),salt)
                cursor.execute("""
                    INSERT INTO usuaris ( usuari, mail,
password)
                    VALUES (%s,%s,%s);
                    """
                    , (request.POST['username'],
request.POST['email'], password.decode('utf-8'),))
                conexion.commit()
                conexion.close()
                cursor.close()
                return redirect('login')
```

A.2.4.4 Logout

This function has the objective of logging out the user and returning it to the main page.

```
def logout(request):
    auth.logout(request)
    return redirect('/')
```

A.2.4.5 Design

This function recollects all the requests that the user needs for the piece and compares those with the data stored in the materials database. As it has seen before, the first steps are connect with the database and process the data introduced by the user, but this time it is important to check if these data is introduced as an integer or string. Also the parameters which the user can choose options will be converted to true or false.

```
def design(request):
    conexion = psycopg2.connect(user='postgres',
password=os.getenv('DATABASE_PASSWORD'),
host=os.getenv('DATABASE_URL'),
port=int(os.getenv('DATABASE_PORT')), database='postgres')
    cursor = conexion.cursor()
    if request.method == 'POST':
        strengh = request.POST['strengh']
        if strengh.isnumeric() == False:
            messages.info(request, 'Invalid value, must be an
integer')
            return redirect('design')
        volum = request.POST['volum']
        if volum.isnumeric() == False:
            messages.info(request, 'Invalid value, must be an
integer')
            return redirect('design')
        strengh = float(strengh)
        volum = float(volum)
        temp_work = request.POST['temp_work']
        if temp_work.isnumeric() == False:
            messages.info(request, 'Invalid value, must be an
integer')
            return redirect('design')
        temp_work = float(temp_work)
        quality_grade = request.POST['quality_grade']
        if quality_grade.isnumeric() == False:
            messages.info(request, 'Invalid value, must be an 
integer')
            return redirect('design')
        quality_grade = float(quality_grade)
        isotropy0 = request.POST['isotropy']
        if isotropy0 == 'si':
            isotropy = True
        else:
            isotropy = False
        thick = request.POST['thick']
        if thick.isnumeric() == False:
            messages.info(request, 'Invalid value, must be an 
integer')
            return redirect('design')
        thick = float(thick)
        support0 = request.POST['support']
        if support0 == 'si':
            support=True
        else:
            support=False
```

Once all those checks are done, the next step is to check if there are any materials and processing types that match with those parameters and if not, give information to the user about if there is not any material on the database or if there is not any processing type on the database.

```
cursor.execute(
            "SELECT id, material"
            " FROM mats"
            " WHERE strenght > %s AND t_melting > %s "
            , (strengh,temp_work,))
        m=cursor.fetchall()
        if isotropy == False:
            cursor.execute(
                "SELECT id, name"
                " FROM processing"
                " WHERE quality_grade < %s AND min_thick < %s"
                , (quality_grade,thick))
            p=cursor.fetchall()
        else:
            cursor.execute(
                "SELECT id, name"
                " FROM processing"
                " WHERE quality_grade < %s AND isotropy = %s
AND min_thick < %s"
                , (quality_grade,isotropy,thick))
            p=cursor.fetchall()
        if len(m)==0:
            messages.info(request, 'No material in the database
by the moment. If you know any material that could be use with
those parameters please contact us')
            cursor.close()
            conexion.close()
            return render(request, 'design.html')
        elif len(p)==0:
            messages.info(request, 'No process in the database
by the moment. If you know any process that could be use with
those parameters please contact us')
            cursor.close()
            conexion.close()
            return redirect('design')
```

If there are possible options then the function will find if the materials can fit with the processing types. This is done with the id's of each of the materials and the processing type's. In the provider table every row is one connection between a material and a processing type using the id of each one. If there is a possible option it will be add to a list, and if it is not possible a list of the possible materials and the possible processing types will be shown.

```
llista=[]
            for z in m:
                for y in p:
                    cursor.execute(
                        "SELECT id"
                        " FROM provider"
                        " WHERE id_mats = %s AND id_processing
= %s"
```

```
                        , (z[0],y[0],))
                    t=cursor.fetchall()
                    if len(t) ==1:
                        llista.append(t[0][0])
if len(llista)==0:
                messages.info(request, 'Allowed materials')
                for x in m:
                    messages.info(request, x[1])
                messages.info(request, 'Allowed processes')
                for a in p:
                    messages.info(request, a[1])
                messages.info(request, 'Those materials can not
be used with those processes according to our database. If you
know that is possible please contact us')
                cursor.close()
                conexion.close()
                return redirect('design')
```

If there are possible options it will appear a display of all the parameters of the materials and the processing types in a table. The data of this table is written in this function but the shape of the table is set in the html file. This table is shown in the results page.

```
i=0
            for v in llista:
                i=i+1
                cursor.execute(
                    "SELECT id_mats, id_processing"
                    " FROM provider"
                    " WHERE id = %s"
                    , (v,))
                p=cursor.fetchall()
                for carm in p:
                    cursor.execute(
                        "SELECT *"
                        " FROM mats"
                        " WHERE id = %s"
                        , (carm[0],))
                    mate=cursor.fetchall()
                    messages.info(request, 'Characteristics')
```

```
                        messages.info(request, str(mate[0][3]))
                        messages.info(request, str(mate[0][1]))
                        messages.info(request, str(mate[0][4]))
                        messages.info(request, str(mate[0][2]))
                        messages.info(request, str(mate[0][5]))
                        coste = 
float(mate[0][6])*float(mate[0][8])*volum/1000
                        coste = round(coste,2)
                        messages.info(request, str(coste))
                        messages.info(request, str(mate[0][7]))
                        messages.info(request, str(mate[0][8]))

                        cursor.execute(
                            "SELECT *"
                            " FROM processing"
                            " WHERE id = %s"
                            , (carm[1],))
                        proce=cursor.fetchall()
                        messages.info(request, str(proce[0][1]))
                        costs=float(proce[0][2])/float(proce[0][5])
*volum
                        costs = round(costs,2)
                        messages.info(request, str(costs))
                        messages.info(request, str(proce[0][3]))
                        messages.info(request, str(proce[0][5]))
#speed
                        if support==True:
                            if proce[0][6]==True:
                                messages.info(request, 
str(support)) #support
                            else:
                                messages.info(request, str('Depend 
of the piece'))
                        else:
                            messages.info(request, str(support))
                        messages.info(request, str(proce[0][7]))
                        messages.info(request, str(proce[0][8]))
                        messages.info(request, str(proce[0][9]))
                        messages.info(request, str(proce[0][10]))
```

```
            return redirect('results')
```

A.2.4.6 Results and Tech

All the functions of those pages are programmed in the html files. Those functions are created for if the user refreshes the page it redirects you to the same page.

```
def results(request):
        return render(request, 'results.html')

def tech(request):
    return render(request, 'tech.html')
```

A.3 Html files

In the html files a template has been used, so the only things that will be commented on this chapter are inputs and the functions programmed manually. The things that have been modified in the template are considered that are not important in this annex because that code affects only the good look of the application.

A.3.1 Login

In order to display the message for an error, like “invalid credentials”. First of all a type of style is needed, and then makes functions for the messages.

```
<style>
h3{
    color: red;
}
</style>

{% for message in messages %}
<h3>{{message}}</h3>
{%endfor%}
```

Then create the form to allow the user to introduce all the inputs with the method POST.

```
<form action="login" method="POST">
    {% csrf_token %}
    <p>Username:</p>
    <input type="text" name="username" />
    <p>Password:</p>
    <input type="password" name="password" /><br>
    <input type="submit" />
</form>
```

A.3.2 Register

The html file for the register is almost the same as the login html, the only difference is the form that the user has to submit the inputs.

```
<form method= "POST" action="register">
    {% csrf_token %}
    <p>Username:</p>
    <input type="text" name="username" />
    <p>Email:</p>
    <input type="email" name="email" />
    <p>Password:</p>
    <input type="password" name="password" />
    <p> Repeat Password:</p>
    <input type="password" name="password2" /><br>
    <input type="submit" />
</form>
```

A.3.3 Design

In this html file, not considering the template, the form is different between the login and the register one. This time apart from the written inputs the user will have two inputs with a display of options.

```
<div class="container d-flex align-items-center justify-
content-between">
<form action="design" method="POST">
    {% csrf_token %}
    <br><br><br><br>
    <p>Strengh:</p>
    <input type="number" name="strengh" />
    <p>Working temperature:</p>
    <input type="number" name="temp_work" />
    <p>Quality grade</p>
    <input type="number" name="quality_grade" />
    <p>Minimun thickness</p>
    <input type="number" name="thick" />
    <p>Aproximate volum</p>
    <input type="number" name="volum" />
    <br><br>

    <p>Isotropy?</p>
    <select type="text" name="isotropy" id="isotropy">
        <option value="si">Yes</option>
        <option value="no">No</option>

    </select>
    <p>Need a support?</p>
    <select type="text" name="support" id="support">
        <option value="si">Yes</option>
        <option value="no">No</option>

        </select>
    <br>
    <br>

    <input type="submit" />
</form>
</div>
```

A.3.4 Results

In this html file, not considering the template lines, the first thing found is the creation of the table where all the parameters are shown. After that, all the options will be added on the table.

```
<html>
   <head>
      <style>
         table, th, td {
            border: 1px solid black;
         }
      </style>
   </head>

   <body>
      <h1>Options characteristics</h1>
      <table>
         <tr>
            <th></th>
            <th style="text-align:center">Material: </th>
            <th style="text-align:center">Material type:
</th>
```

```
            <th style="text-align:center">Young module: </th>
            <th style="text-align:center">Strengh that can
afford: </th>
            <th style="text-align:center">Specific deformation:
</th>
            <th style="text-align:center">Material cost: </th>
            <th style="text-align:center">Melting temperature:
</th>
            <th style="text-align:center">Density: </th>
            <th style="text-align:center">Processing
technology: </th>
            <th style="text-align:center">Processing cost:
</th>
            <th style="text-align:center">Quality grade: </th>
            <!--<th style="text-align:center">Surface
roughness: </th>-->
            <th style="text-align:center">Speed: </th>
            <th style="text-align:center">Needs post-processing
treatment? </th>
            <th style="text-align:center">Clearance between
moving parts: </th>
            <th style="text-align:center">Minimum thickness:
</th>
            <th style="text-align:center">Minimum hole size:
</th>
            <th style="text-align:center">Does the technology
manufacture with isotropy?: </th>
    {% for message in messages%}
    {% if message|truncatewords:15 == 'Characteristics'%}
         </tr><tr>
            <td style="text-align:center">{{message}}</td>
    {%else%}
    {% if message == Null%}
            <td style="text-align:center"> </td>
            {%else%}
            <td style="text-align:center">{{message}}</td>
            {%endif%}
    {%endif%}
    {%endfor%}
```

```
         </tr>
      </table>
   </body>
</html>
<br>
```

A.3.5 Tech

This html file has to different parts, the first one with the general advices, which are written directly to the html file. If you want to add more advices you only need to add another line to the code with the text needed.

```
<h1>General advices:</h1><br>
<li>Use AM if the shape is complex and difficult to manufacture
with conventional methods</li><br>
<li>AM is the perfect option for insert logos or figures on the
surface</li><br>
```

Then there is a display where the user can choose another technology to see specific advices for the corresponding processing technologies. If you want to add more technologies on the display you only need to add a line above </select>

```
<h1>Specific advices:</h1><br>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Chosing technology</title>
  </head>
  <body>
    <label for="techno">Select the processing technology:
</label>
    <select id="techno">
        <option value="">--Make a choice--</option>
        <option value="material_extrusion">Material
Extrusion</option>
        <option value="powder_bed_fusion">Powder Bed
Fusion</option>
```

```
        <option value="vat_photo">Vat
Photopolymerization</option>
        <option value="material_jetting">Material
Jetting</option>
        <option value="binder_jetting">Binder Jetting</option>
    </select>
```

After that, the parameters are defined. This number of parameters matches with the number of advices of the processing technology that has more advices. In this case there are the eleven advices of the material extrusion.

```
    <p id="para1"></p>
    <p id="para2"></p>
    <p id="para3"></p>
    <p id="para4"></p>
    <p id="para5"></p>
    <p id="para6"></p>
    <p id="para7"></p>
    <p id="para8"></p>
    <p id="para9"></p>
    <p id="para10"></p>
    <p id="para11"></p>

    <script>
      const select = document.getElementById('techno');
      const para1 = document.getElementById('para1');
      const para2 = document.getElementById('para2');
      const para3 = document.getElementById('para3');
      const para4 = document.getElementById('para4');
      const para5 = document.getElementById('para5');
      const para6 = document.getElementById('para6');
      const para7 = document.getElementById('para7');
      const para8 = document.getElementById('para8');
      const para9 = document.getElementById('para9');
      const para10 = document.getElementById('para10');
      const para11 = document.getElementById('para11');


      select.onchange = setWeather;
```

Once all the parameters are defined, the function setWeather is defined. Depending on which processing technology is selected the advices that are shown are different. It is important to put all the parameters defined previously in the function for every technology; if it is empty it is compulsory to define that parameter anyway.

```
function setWeather() {
        const choice = select.value;

        if(choice === 'material_extrusion') {
            para1.textContent = '-  Most used for
thermoplastics polymers';
            para2.textContent = '-  Recommended to use
different materials for the support and the part';
            para3.textContent = '-  Use thinner layer thickness
if the shape of the part is complex in order to avoid stair-
step effect and thicker if the surface is flat. The recommended
layer thickness is 0,25mm';
            para4.textContent = '-  Increase the infill
percentage if the part has to resist high tensile efforts';
            para5.textContent = '-  Vertical wall thickness:
1mm';
            para6.textContent = '-  Horizontal walls with at
least 4 layers of material';
            para7.textContent = '-  Minimum 45º respect the
horizontal plane for the overhanging parts in order to avoid
the use of supports';
            para8.textContent = '-  It is recommended to use a
soluble support, with that the clearance between moving parts
is 0,5mm in horizontal and 0,25mm in vertical';
            para9.textContent = '-  Minimum 5mm of diameter for
vertical holes and 0,2mm more in the CAD version';
            para10.textContent = '- Minimum 2mm of diameter for
circular pins';
            para11.textContent = '- Minimum 5mm of diameter for
screw threads and leave 1mm of separation between the start and
the base';
        } else if(choice === 'powder_bed_fusion') {
            para1.textContent = '-  It can avoid support if is
well designed';
```

```
            para2.textContent = '-  Recommended layer 
thickness: 0,1mm';
            para3.textContent = '-  Avoid large masses of 
material';
            para4.textContent = '-  Possible to recycle the 
powder';
            para5.textContent = '-  Recommended wall thickness: 
1mm';
            para6.textContent = '-  Minimum clearance between 
moving parts: 1mm in both horizontals and verticals directions 
(it has to be possible to remove the support material)';
            para7.textContent = '-  Minimum circular hole 
diameter: 0,5mm for vertical holes and 1,3mm for horizontal 
holes';
            para8.textContent = '-  Minimum square hole side 
size: 0,5mm for vertical holes and 0,8mm for horizontal holes';
            para9.textContent = '-  Minimum 0,8mm of diameter 
for circular pins';
            para10.textContent = '- 0,8mm of distance between 
the wall edge and a 2,5mm hole';
            para11.textContent = '';
        } else if(choice === 'vat_photo') {
          para1.textContent = '-  Minimum 0,1mm of height for 
the details on the surface';
          para2.textContent = '-  Recommended 0,6mm of wall 
thickness';
          para3.textContent = '-  Minimum 0,5mm of diameter for 
circular holes';
          para4.textContent = '';
          para5.textContent = '';
          para6.textContent = '';
          para7.textContent = '';
          para8.textContent = '';
          para9.textContent = '';
          para10.textContent = '';
          para11.textContent = '';
        } else if(choice === 'material_jetting') {
          para1.textContent = '-  Minimum 1mm of thickness for 
walls';
```

```
          para2.textContent = '-  Minimum 0,5mm of diameter for
pins and hole sizes';
          para3.textContent = '-  Minimum 0,5mm of height for
surface details';
          para4.textContent = '-  Minimum 0,2mm of clearance
between moving parts';
          para5.textContent = '-  Best option for a nice
surface quality but poor mechanical properties.';
          para6.textContent = '';
          para7.textContent = '';
          para8.textContent = '';
          para9.textContent = '';
          para10.textContent = '';
          para11.textContent = '';
        } else if(choice === 'binder_jetting') {
          para1.textContent = '-  It is important to be careful
during the green state phase.';
          para2.textContent = '-  It is recommended to use ribs
and reinforcements';
          para3.textContent = '-  Wall thickness (build size
=  minimum thickness): 3-75mm = 1mm; 75-152mm = 1,5mm; 152-
203mm = 2mm; 203-305mm = 3,2mm';
          para4.textContent = '-  2mm of thickness for a 25mm
of width overhanging part.';
          para5.textContent = '-  Minimum 2mm of diameter for
holes.';
          para6.textContent = '';
          para7.textContent = '';
          para8.textContent = '';
          para9.textContent = '';
          para10.textContent = '';
          para11.textContent = '';
        } else {
          para1.textContent = '';
        }
      }
```

A.4 Other files

Apart from the files commented before it is important to define the other files in order to protect the database. In order to not upload the password, the port and the url of the database to GitHub those variable are stored in a .env file.

Then in another file, which is .gitignore, you need to write .env and *sqlite3.