# DIGITAL PLATE LOAD TEST DEVICE

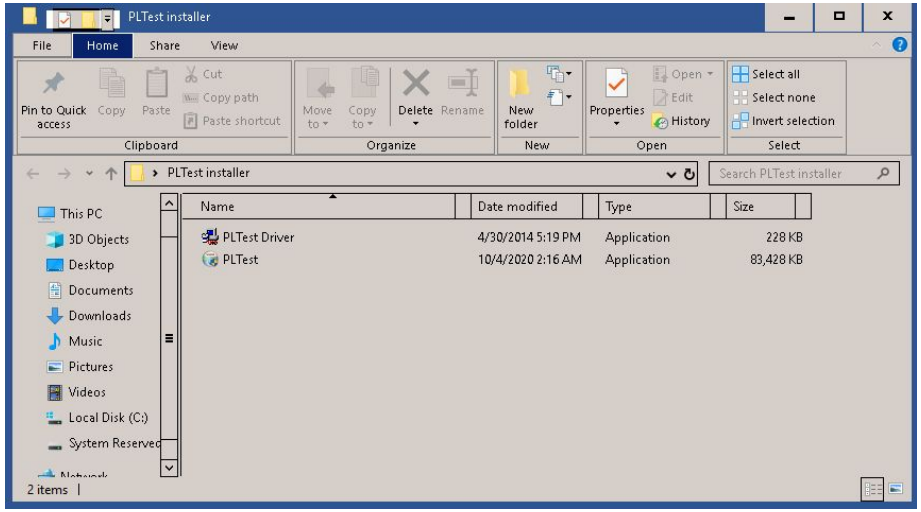
USER MANUAL

Please read this manual before operating the device and keep it for future reference

## TABLE OF CONTENTS

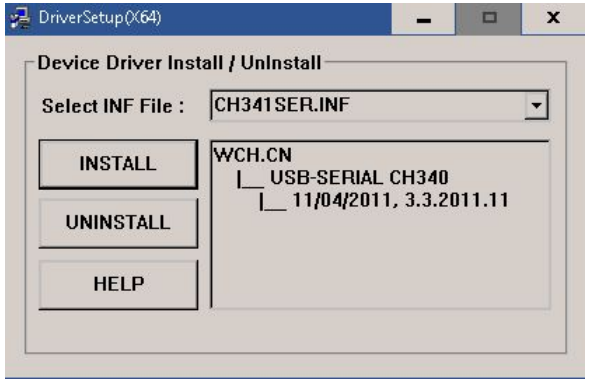
# INSTALLATION OF PLATE LOAD TEST APP

- **Open the PLTest Folder**




*Your PLTest folder should look like this and it must contain the installer app for the application.*

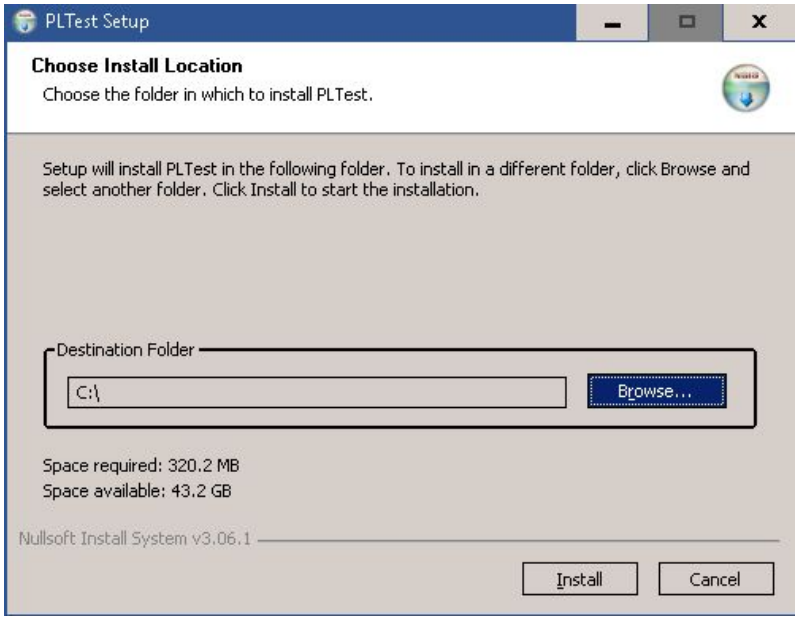
- **If you are using Windows 7 operating system, Open the PLTest Driver Installer.**




*Click install and wait for the installation to complete.*

- **After the driver installation, Open the PLTest Installer**

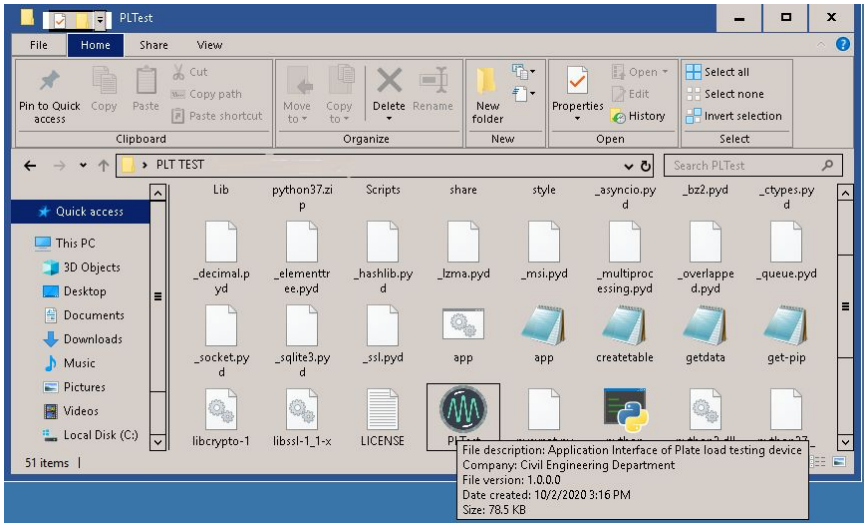
*Note: If you are using Windows 10 operating system, You don't need to install the PLTest Driver.*




- ➢ *Install System will pop up like the figure above, Click Browse and find "Local Disk C" on "This PC".*
- ➢ *Click install and wait for the installation process to complete.*
- ➢ *After the installation, you can now close the install system.*

# SHORTCUTS CONFIGURATION

- **Go to “This PC”, Find “Local Disk C” and Open it. It should contain a “PLTest” folder.**



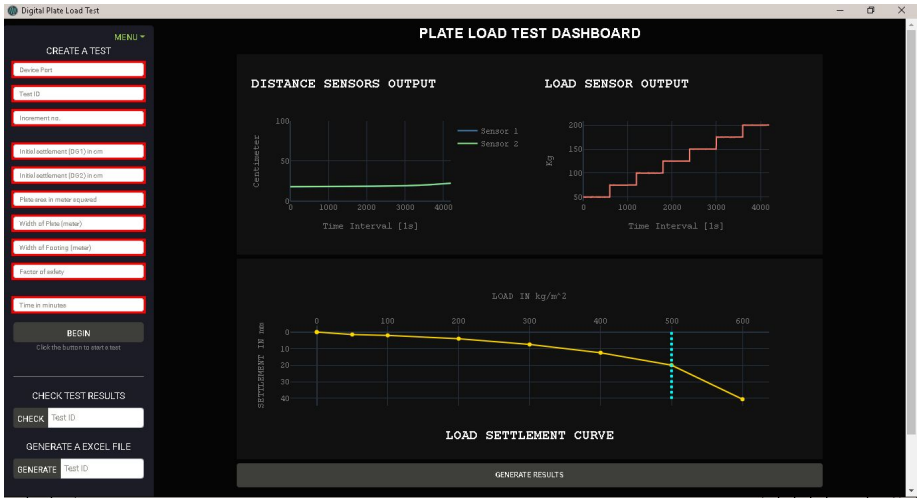

➢ *Open the “PLTest” folder and find the “PLTest.exe”, Then right-click, Look for “Send to” and select “Desktop”.*

➢ *Find the “excelfiles” folder (if not found, create one and name it as “excelfiles”) then create a shortcut of that folder on your Desktop.*

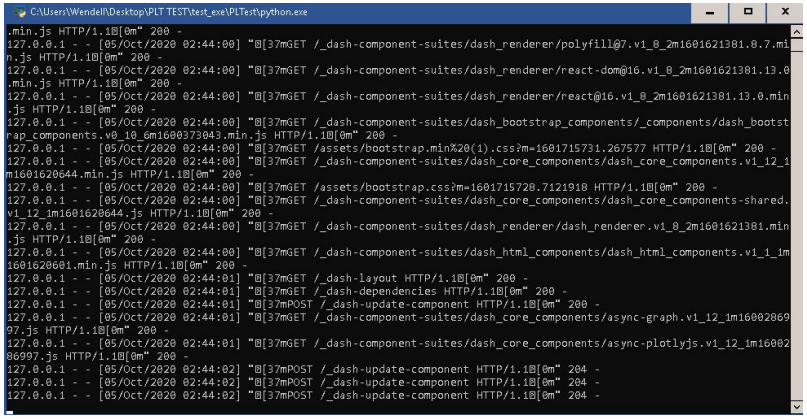
➢ *Open the “PLTest” folder again, Find “gui” folder and open it, Locate the ‘SQLiteStudio.exe’ and Send it to “Desktop”*


*You should now be able to find these shortcuts on your Desktop.*

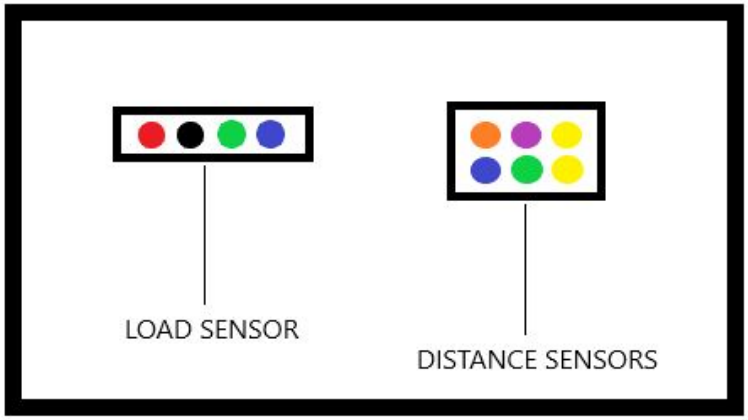
- **Open the “PLTest” shortcut on your Desktop to start the application.**




*Warning: Do not close the command line that looks like the figure below or else the application will not work.*



## SENSORS CONNECTION

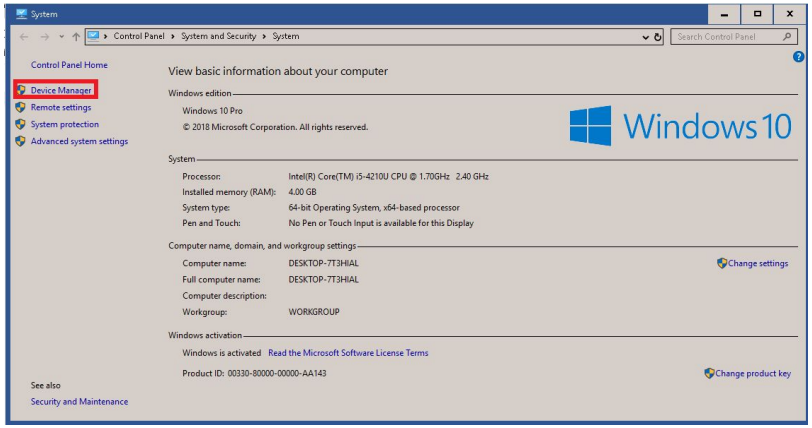



- **The figure above is the back representation of the device, Dots with colours corresponds to the colour of the wire that is connected to the sensor.**

Warning: Please be careful when connecting the sensors on the device. If you apply too much force the pins may bent.

## SERIAL PORT LOCATION

- **Open "This PC", then right-click select "Properties".**




➢ *A window will pop-up that looks like the figure above, Click on the "Device Manager".*


➢ *Find the "Ports (COM & LPT)" and click it. Locate the "USB SERIAL CH340(COM123)", and memorize the COM port eg.(COM123).*

*Note: COM port is different for every USB socket on your computer.*

# CALIBRATION OF SENSORS

- **Open the PLTest app on your Desktop, Click the "Menu" located at Top-left of the application and Select Calibration.**

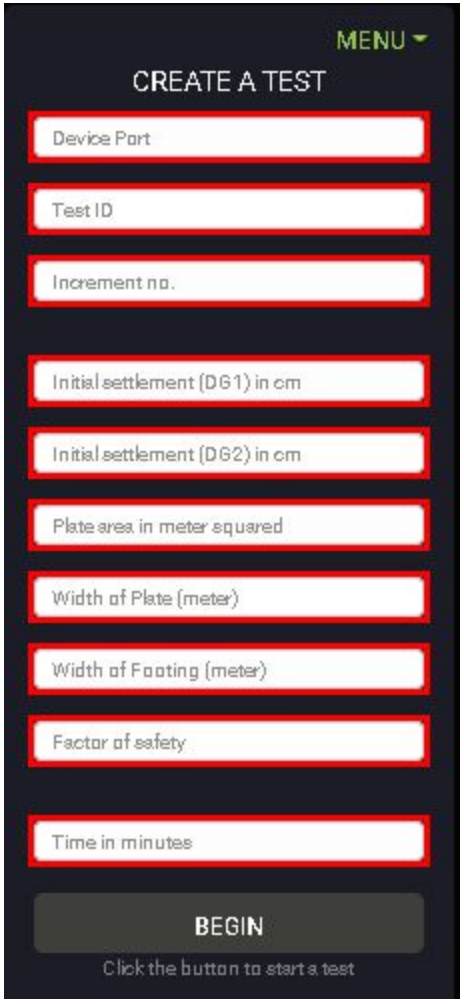

*Your Calibration page should look like the figure above.*

- *On the check device connection, Type in the COM port that you memorized eg. (COM123). By clicking the "Check" button, a text will pop-up below the form that indicates if your device is connected.*
- *Click the "Generate" button to generate data from the sensors.*
- *If you try to hover your mouse pointer on the plot, a box will pop-up that indicates the X and Y-axis of a given point.*
- *Memorize the Initial settlement of the distance sensors, You will use it later to create a test.*

Note: If you click the "Generate" button, it will only generate data for 10 seconds, you have to click it again if you change the Load and Initial settlement of the sensors.

# CREATE A TEST

- **On your PLTest Dashboard, you must be able to find the "Create a Test" form like the figure below.**




If you create a test, All inputs must be filled with their specified information and units.

You must always double-check the information that you provided for the reason that if you click the "Begin" button, you cannot stop the test unless you close the application and its terminal.

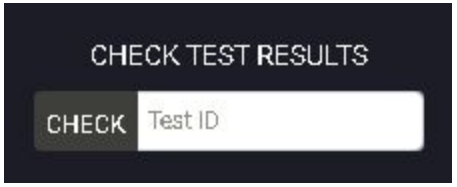
You can use the calibration on the menu to measure the initial settlement of the distance sensors.

Note: Memorize the "Test ID" of your test in view of the fact that you will use it later to generate a result or maybe save an excel file of the collected data summary.


Your PLTest App should look like this after clicking the “Begin” button if the provided information is valid.

Warning: DO NOT move or touch the distance sensors when the test begins, it may result in a data error owing to the fact that the distance sensors are very sensitive and the initial settlement will change when moved.

## CHECK THE DATA OF A GIVEN TEST

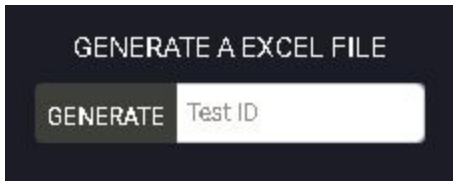



On the PLTest Dashboard find the “Check Test Results”.

➢ Type the “Test ID” of the test that you want to check and click the “Check” button.

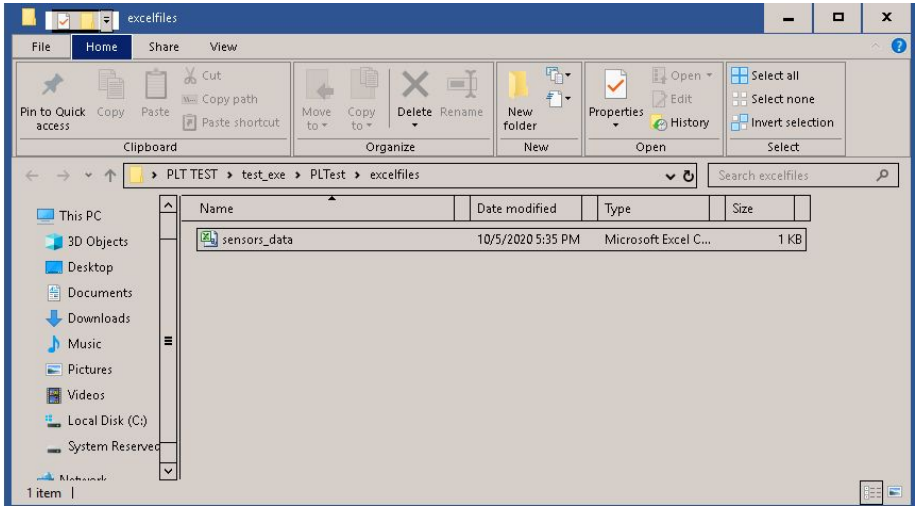

➢ The data of a given test will be loaded on the “Distance Sensors Output” and “Load Sensor Output” if provided “Test ID” is correct or exists in the Database.

## GENERATE AN EXCEL FILE




On the PLTest Dashboard find “Generate a Excel File”.

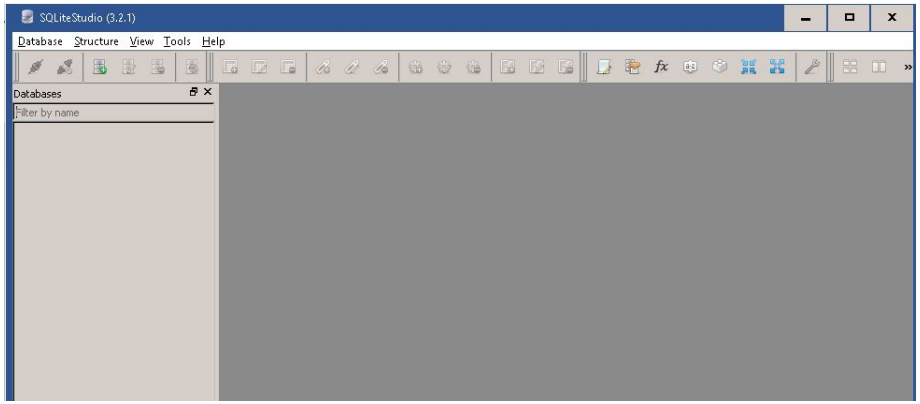
- Type the “Test ID” and click the “Generate” button to generate an excel file.
- You can find the excel file that you generated on the “excelfiles” folder shortcut you created on the Desktop.




Note: The filename of the excel file you generated is the “Test ID” you provided at the input form.

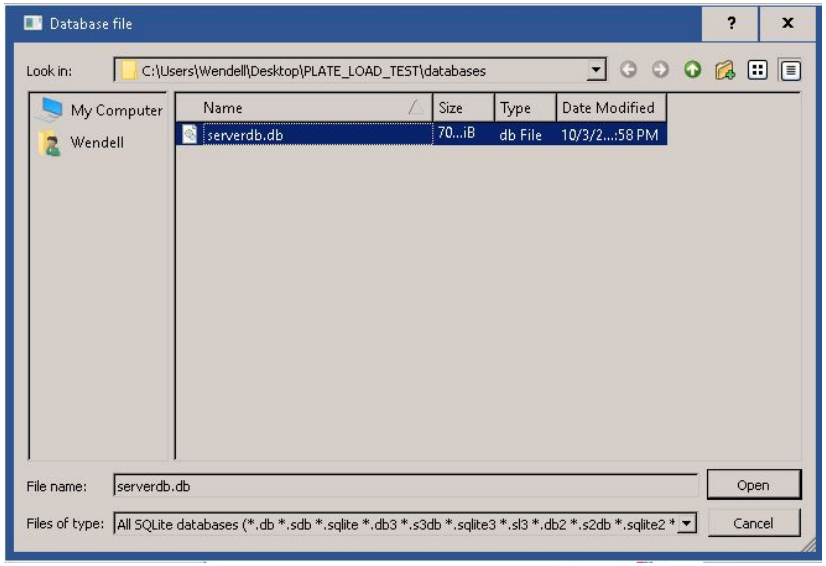
## DATABASE SYSTEM

On your desktop, open the “SQLiteStudio” shortcut you created earlier.

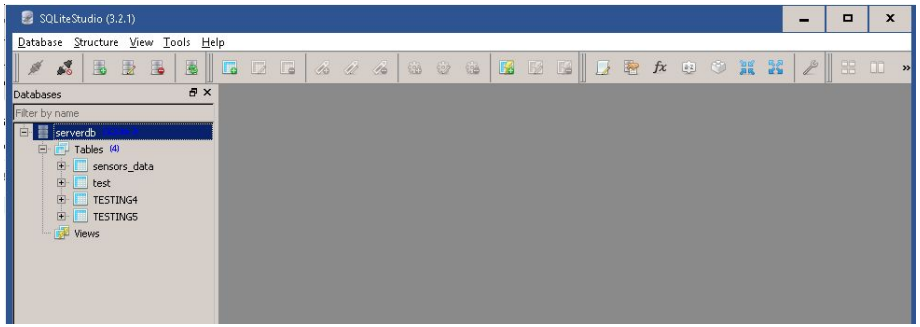



- To add a database file, Click “Database” and select “Add a database”.


- Click the folder icon and go to the PLTest folder, then open the “databases” folder.



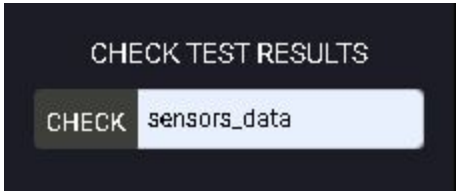

➢ “databases” folder should contain a “serverdb.db” file, select it and click “Open”.




➢ All the tests that were created on the application should be found on the “Table”.

Note: SQLiteStudio is a third-party application that was packaged on the app, you can also find some basic tutorial on google or youtube on how to use it.

# TEST RESULTS




On the PLTest app, find the “Check Test Results” and type the Test ID that you want the results to be generated.


➢ Click the “Generate Results” button that looks like the figure above.


➢ Results should pop-up above the load settlement curve plot that gives you the following:
  - Ultimate Bearing Capacity in Kg/m^2
  - Safe Bearing Capacity in Kg/m^2
  - Settlement of Footing in mm.
➢ Also, the application will plot a load settlement curve based on the data of a given test.
➢ The broken line in the load settlement curve indicates the ultimate bearing capacity of the soil.

# PROGRAMMING SECTION

**Logic and Control System**

**Data Processing**

**Plot Generator**

**Application Layout**

## LOGIC AND CONTROL SYSTEM

### CONTROLLER PROGRAM

```
#include "HX711.h"
#define sensor1 A0
#define sensor2 A1
#define calibration_factor 7050.0 //This value is obtained using
the SparkFun_HX711_Calibration sketch

#define DOUT  3
#define CLK  2

int SIZE = 100;

HX711 scale;

void setup() {
  Serial.begin(9600);
  scale.begin(DOUT, CLK);
  scale.set_scale(calibration_factor); //This value is obtained by
using the SparkFun_HX711_Calibration sketch
 //Assuming there is no weight on the scale at start up, reset the
scale to 0
}

void loop() {

  float disp1_val[SIZE];
  float disp2_val[SIZE];
  float loadc = scale.get_units() / 2.2;

  for (int i =0; i<SIZE; i++){
    float volts1 = analogRead(sensor1)*0.0048828125;
    float volts2 = analogRead(sensor2)*0.0048828125;

    float disp1 = 60.374 * pow(volts1, -1.16);
    float disp2 = 60.374 * pow(volts2, -1.16);
```

```
    disp1_val[i] = disp1;
    disp2_val[i] = disp2;
  }

  float d1= 0;
  float d2= 0;

  for (int i =0; i<SIZE; i++){
    d1 = d1 + disp1_val[i];
    d2 = d2 + disp2_val[i];
  }

  d1 = (d1/SIZE);
  d2 = (d2/SIZE);
  loadc = loadc - 0.40;
  if (loadc <0){
    loadc = 0.0;
  }


  Serial.print(d1);
  Serial.print('\t');
  Serial.print(d2);
  Serial.print('\t');
  Serial.println(loadc);
  delay(1000);
}
```

# LOGIC OF THE SYSTEM

```
import numpy as np
import serial
import pandas as pd
import time
import sqlite3

import flask
from flask import request
import dash
import dash_core_components as dcc
import dash_bootstrap_components as dbc
from dash.dependencies import Input, Output
import dash_html_components as html
from dash.exceptions import PreventUpdate
from dash.dash import no_update

import chart_studio.plotly as py
import plotly.express as px
import plotly.graph_objects as go

#Initialize getdata
from getdata import GetData
getdata = GetData()
#plot generator
from components.PlotGenerator.PlotGenerator import scatter_data,
get_lsc


#Get Bootstrap
# from style.bootstrap import Bootstrap
# external_scripts = Bootstrap().getScripts()
# external_stylesheets = Bootstrap().getStylesheet()

#Initilize Layouts
from components.Home.Home import home_layout
from components.CalibrationLayout.CalibrationLayout import
calibration_layout

from flaskwebgui import FlaskUI #get the FlaskUI class
server = flask.Flask(__name__)
```

```
ui = FlaskUI(server, port=2020)

app = dash.Dash(
    __name__,
    server = server,
    title='Digital Plate Load Test',
    suppress_callback_exceptions = True,
    # external_scripts=external_scripts,
    # external_stylesheets=external_stylesheets
    #external_stylesheets = [dbc.themes.BOOTSTRAP]
)


def shutdown_server():
    func = request.environ.get('werkzeug.server.shutdown')
    if func is None:
        raise RuntimeError('Not running with the Werkzeug Server')
    func()

#[APP MAIN LAYOUT]
app.layout = html.Div([
    dcc.Location(id='url', refresh=False),
    dcc.Location(id='url2', refresh=False),
    html.Div(id='page-content')
])



#START TEST CALLBACK
@app.callback(
    [Output('disp-sensor', 'figure'),
     Output('load-sensor', 'figure'),
     Output('test-indicator', 'children'),
     Output('start-an-btn', 'style'),
     Output('loader_csv', 'children')],
    [Input('start-btn', 'n_clicks'),
     Input('port', 'value'),
     Input('test-id', 'value'),
     Input('inc-no', 'value'),
     Input('time-input', 'value'),
     Input('view1', 'n_clicks'),
     Input('inp-sc', 'value'),
     Input('view2', 'n_clicks'),
     Input('inp-csv', 'value'),
     Input('plate-area', 'value'),
     Input('ini-set-1', 'value'),
     Input('ini-set-2', 'value'),
     Input('factor-safety', 'value'),
     Input('plate-width', 'value'),
     Input('width-footing', 'value')]
 )
def startTestHandler(btn1, port, id, inc, time_s, btn2, t_id,
btn3, t_csv, p_area, ini1, ini2, fs, pw, wf):
    changed_id = [p['prop_id'] for p in
dash.callback_context.triggered][0]
    try:
        try:
          if ('start-btn' in changed_id
              and port is not None
              and id is not None
              and inc is not None
              and time_s is not None
              and p_area is not None
              and ini1 is not None
              and ini2 is not None
              and fs is not None
              and pw is not None
              and wf is not None):
              df = getdata.upload_generate(port= port, baud=9600,
n=time_s, table=id, inc=inc, area=p_area, set1=ini1, set2=ini2,
                                           w_footing=wf, w_plate =
pw, fs = fs)
              ind = 'Process running.'
              fig1, fig2 = scatter_data(df, plotType = 'normal')
              return  fig1, fig2, ind, {'display': 'block'},
no_update
        except:
          return no_update, no_update, 'Error Detected',
no_update, no_update

        if 'view1' in changed_id:
            try:
                df = getdata.get_dataframe(table=t_id)
                ind = 'Click the button to start a test.'
                fig1, fig2 = scatter_data(df, plotType = 'normal',
table = t_id)
```

```
                return  fig1, fig2, ind, {'display': 'none'},
no_update
            except:
                raise PreventUpdate

        elif 'view2' in changed_id:
            df = getdata.get_dataframe(table=t_csv)
            df,_, __ = getdata.get_PS(df)
            df = df[['P', 'INC_NO','TIME','S1', 'S2', 'S',
'TS']].round(2)
            df.columns = ['Pressure (Kg/m^2)', 'Increment No.',
                          'Time (minute/s)','Dial Gauge-1
Reading',
                          'Dial Gauge-2 Reading',
                          'Avereage Settlement (mm)',
                          'Total Settlement (mm)']
            time.sleep(1)
            df.to_csv('./excelfiles/'+t_csv+'.csv')
            raise PreventUpdate
        else:
            raise PreventUpdate
    except:
        raise PreventUpdate


#[TIMER HANDLER]
@app.callback([Output('start-test', 'style'),
               Output('timer-div', 'children'),
               Output('p-test-id', 'children'),
               Output('p-inc-no', 'children'),
               Output('dg-1', 'children'),
               Output('dg-2', 'children'),
               Output('p-fs', 'children'),
               Output('p-time', 'children'),
               Output('summary', 'style'),
               Output('s-width-footing', 'children')],
        [Input('start-btn', 'n_clicks'),
        Input('port', 'value'),
        Input('test-id', 'value'),
        Input('inc-no', 'value'),
        Input('time-input', 'value'),
        Input('view1', 'n_clicks'),
        Input('inp-sc', 'value'),
        Input('view2', 'n_clicks'),
        Input('inp-csv', 'value'),
        Input('ini-set-1', 'value'),
        Input('ini-set-2', 'value'),
        Input('plate-area', 'value'),
        Input('factor-safety', 'value'),
        Input('plate-width', 'value'),
        Input('width-footing', 'value')])
def showtimer(btn1, port,
              id, inc, time_s,
              btn2, t_id, btn3,
              t_csv, set1, set2,
              p_area, fs, pw, wf):
    changed_id = [p['prop_id'] for p in
dash.callback_context.triggered][0]
    if ('start-btn' in changed_id
    and port is not None
    and id is not None
    and inc is not None
    and time_s is not None
    and p_area is not None
    and set1 is not None
    and set2 is not None
    and fs is not None
    and pw is not None
    and wf is not None):
        try:

          style_test = {'display': 'none'}
          id = 'Test ID: ' + id
          inc = 'Increment no: ' + str(inc)
          set1 = 'Initial DG-1: ' + str(set1) + 'cm'
          set2 = 'Initial DG-2: ' + str(set2) + 'cm'
          wf = 'Width of Footing: ' + str(wf) + 'm'
          fs = 'Factor of Safety: ' + str(fs)
          time_x = 'Time: ' + str(time_s) + ' minute/s'
          plus = 1100
          return style_test,[dbc.Progress(value=0,
id='progressb',color='success',
                                          style={'height': '30px',
                                                 'fontSize':
'10px'}),
                             dcc.Interval(id="progress-interval",
```

```
n_intervals=0, interval=plus)], \
                            id, inc,set1,set2,fs, time_x,
{'display': 'inline-block'}, wf
        except:
          raise PreventUpdate
    else:
        raise PreventUpdate

#PROGRESS BAR CALLBACK
@app.callback([Output('progressb', 'value'),
               Output('progressb', 'children')],
              [Input('time-input', 'value'),
               Input('progress-interval', 'n_intervals')])
def update_progress(time_s,n):
    time_s = time_s * 60
    coef = 100 / int(time_s)
    value = coef * n
    if value <=100:
        return value,str(round(value)) +'%'
    else:
        raise PreventUpdate

#RESULTS CALLBACK
@app.callback([Output('lsc', 'figure'),
               Output('measurements', 'style'),
               Output('m1', 'children'),
               Output('m2', 'children'),
               Output('m3', 'children')],
              [Input('gen-res', 'n_clicks'),
               Input('inp-sc', 'value')])
def generateUBC(n, inp):
    changed_id = [p['prop_id'] for p in
dash.callback_context.triggered][0]
    if 'gen-res' in changed_id:
        try:
          data = [[11.5, 23, 35,46,57.5, 80.5, 103.5, 120],
              [0.07,0.34,0.845,1.55,2.09,3.2,6.06,7.55]]
          dummy = pd.DataFrame(data).T
          dummy.columns = ['P', 'S']

          dataf  = getdata.get_dataframe(table=inp)
          fos = dataf.FOS.iloc[-1]
          ps, bp, b = getdata.get_PS(dataf)
          ubc,sett, idx = getdata.get_ubc(df=ps)
          sett = sett * 0.7
          sf = round(((sett/1000) * ((b*(bp+0.3)) /
(bp*(b+0.3)))**2)*1000,2)
          m1 = 'Ultimate Bearing Capacity:' + str(round(ubc, 2)) +
' Kg/m^2'
          m2 = 'Safe Bearing Capacity: ' + str(round(ubc/fos, 2))
+ ' Kg/m^2'
          m3 = 'Settlement of Footing: ' + str(sf) + ' mm'
          fig = get_lsc(ps, ubc=ubc, ubc_s=sett)
          return fig, {'display': 'block', 'marginLeft': '10%'},
m1, m2, m3
        except:
          raise PreventUpdate

    else:
        raise PreventUpdate

#CALIBRATION CALLBACK
@app.callback([Output('ca-ind', 'children'),
               Output('ca-ind', 'style')],
              [Input('check-port', 'n_clicks'),
               Input('ca-port-id', 'value')])
def calibrationHandler(btn1, inp):
    changed_id = [p['prop_id'] for p in
dash.callback_context.triggered][0]
    if 'check-port' in changed_id:
        try:
            if inp:
              serial.Serial(inp, 9600)
              ind = 'Device found'
              style={'color': 'lightgreen', 'fontSize':'15px'}
              time.sleep(1)
              return ind, style
            else:
              raise PreventUpdate
        except:
            ind = 'Device not found'
            style={'color': 'salmon', 'fontSize':'15px'}
            time.sleep(1)
            return ind, style
```

```
    else:
        raise PreventUpdate

#[RUN CALIBRATION]
@app.callback([Output('ca-disp-sensor', 'children'),
               Output('ca-test-indicator', 'children')],
              [Input('ca-start-btn', 'n_clicks'),
               Input('ca-port-id', 'value'),
               Input('ca-clear-btn', 'n_clicks')])
def run_calibration(btn,port, clear):
    changed_id = [p['prop_id'] for p in
dash.callback_context.triggered][0]
    if 'ca-start-btn' in changed_id:

        try:
            time.sleep(0)
            getdata.upload_data(port= port, baud=9600, n=10,
table='test', inc='test')
            raise PreventUpdate
        except:
            raise PreventUpdate
    elif 'ca-clear-btn' in changed_id:
        try:
            conn = sqlite3.connect('./databases/serverdb.db')
            c = conn.cursor()

            # delete all rows from table
            c.execute('DELETE FROM test;',);
            conn.commit()
            conn.close()
            raise PreventUpdate
        except:
            raise PreventUpdate
    else:
        raise PreventUpdate

@app.callback([Output('ca-disp-sensor', 'figure'),
               Output('ca-load-sensor', 'figure')],
               [Input('interval-component1', 'n_intervals')])
def update_plot(n):
    try:
        df = getdata.get_dataframe(table='test')
        fig1, fig2 = scatter_data(df, height=400)
        return fig1, fig2
    except:
        raise PreventUpdate

#ROUTING CALLBACK
@app.callback(Output('page-content', 'children'),
              [Input('url', 'pathname')])
def display_page(pathname):

    if pathname == '/calibration':
        return calibration_layout
    elif pathname == '/shutdown':
        shutdown_server()
        return 'Server shutting down...'
    else:
        return home_layout

if __name__ == '__main__':
  #app.run_server(debug=True, port=2020)
  ui.run()
```

# DATA PROCESSING

```
import serial
import numpy as np
import sqlite3
import pandas as pd
from tqdm import tqdm
import plotly.express as px
import plotly.graph_objects as go

from createtable import create_table


class GetData:
    def __init__(self, path='./databases/serverdb.db'):
        try:
            self.conn = sqlite3.connect(path)
            self.c = self.conn.cursor()

        except:
            print('Device not found.')


    def get_dataframe(self, table='sensors_data',
path='./databases/serverdb.db'):
        conn = sqlite3.connect(path)
        df = pd.read_sql('SELECT * FROM {}'.format(table),
con=conn)
        conn.close()
        return df

    def upload_data(self, port='', baud=9600, n=10,
                        table = 'table',
                        inc = 'none',
                        path='./databases/serverdb.db'):

        arduino = serial.Serial(port, baud)
        conn = sqlite3.connect(path)
        c = conn.cursor()
        readings = []
        for i in tqdm(range(n)):
            data = arduino.readline()[:-2].decode('utf-8')
            data = [float(i) for i in data.split('\t')]
            data.append(inc)
            readings.append(data)

            c.execute('INSERT INTO '+ table +'
VALUES(?,?,?,?);',tuple(data));
            conn.commit()
        conn.close()

    def generate_df(self, port='', baud=9600, n=10):
        arduino = serial.Serial(port, baud)
        readings = []
        for i in tqdm(range(n)):
            data = arduino.readline()[:-2].decode('utf-8')
            data = [float(i) for i in data.split('\t')]
            readings.append(data)
        df = pd.DataFrame(readings, columns=['S1', 'S2', 'S3'])
        return df

    def upload_generate(self, port='', baud=9600, n=10,
                        table = 'table',
                        inc = 0,
                        area = 0,
                        set1 =0,
                        set2=0,
                        w_plate = 0,
                        w_footing = 0,
                        fs = 1,
                        path='./databases/serverdb.db'):
        arduino = serial.Serial(port, baud)
        n = int(n * 60)
        try:
            conn = sqlite3.connect(path)
            c = conn.cursor()
            readings = []

            for i in tqdm(range(n)):
                data = arduino.readline()[:-2].decode('utf-8')
                data = [float(i) for i in data.split('\t')]
                data.append(inc)
                data.append(round(data[2]/ area,1))
                data.append(set1)
```

```
                data.append(set2)
                data.append(area)
                data.append(w_plate)
                data.append(w_footing)
                data.append(fs)
                data.append(round(n/60))
                readings.append(data)

                c.execute('INSERT INTO '+ table +'
VALUES(?,?,?,?,?,?,?,?,?,?,?,?);',tuple(data));
                conn.commit()
            df = pd.DataFrame(readings, columns=['S1', 'S2', 'S3',
'INCREMENT', 'PRESSURE', 'SET1', 'SET2',
                                                 'PLATE_AREA',
'WIDTH_PLATE', 'WIDTH_FOOTING', 'FOS', 'TIME'])
            return df
            conn.close()
        except:
            conn.close()
            query = '''CREATE TABLE IF NOT EXISTS {0} (
                                    S1 REAL,
                                    S2 REAL,
                                    S3 REAL,
                                    INCREMENT REAL,
                                    PRESSURE REAL,
                                    INITIAL_SET1 REAL,
                                    INITIAL_SET2 REAL,
                                    PLATE_AREA REAL,
                                    WIDTH_PLATE REAL,
                                    WIDTH_FOOTING REAL,
                                    FOS REAL,
                                    TIME_OF_TEST REAL

                                );'''.format(table)
            conn = sqlite3.connect(path)
            c = conn.cursor()
            c.execute(query)
            readings = []
            for i in tqdm(range(n)):
                data = arduino.readline()[:-2].decode('utf-8')
                data = [float(i) for i in data.split('\t')]
                data.append(inc)
                data.append(round(data[2]/ area,1))
                data.append(set1)
                data.append(set2)
                data.append(area)
                data.append(w_plate)
                data.append(w_footing)
                data.append(fs)
                data.append(round(n/60))
                readings.append(data)

                c.execute('INSERT INTO '+ table +'
VALUES(?,?,?,?,?,?,?,?,?,?,?,?);',tuple(data));
                conn.commit()
            df = pd.DataFrame(readings, columns=['S1', 'S2', 'S3',
'INCREMENT', 'PRESSURE', 'SET1', 'SET2',
                                                 'PLATE_AREA',
'WIDTH_PLATE', 'WIDTH_FOOTING', 'FOS', 'TIME'])
            return df
            conn.close()


    def get_ubc(self,df):
        df['lag'] = df.S.diff()
        idx = df[df.lag ==df.lag.max()].index[0] -1
        ubc = df.iloc[idx]['P']
        ubc_set = df.iloc[idx]['S']
        return ubc, ubc_set, idx


    def get_PS(self,df):
        summmary_df = pd.DataFrame(columns = ['P', 'S1', 'S2',
'S1_S2'])
        df['diff1'] = (df.S1.values - df.INITIAL_SET1.values) * 10
        df['diff2'] = (df.S2.values - df.INITIAL_SET2.values) * 10

        unik = df.INCREMENT.unique()
        set1_per_inc = []
        set2_per_inc = []
        time_per_inc = []
        for increment in unik:
            percent_to_ave1  =
int(len(df[df['INCREMENT']==increment]['diff1']) * 0.05)
            percent_to_ave2  =
int(len(df[df['INCREMENT']==increment]['diff2']) * 0.05)
```

```
set1_per_inc.append(df[df['INCREMENT']==increment]['diff1'].iloc[-
percent_to_ave1:].mean())

set2_per_inc.append(df[df['INCREMENT']==increment]['diff2'].iloc[-
percent_to_ave2:].mean())

time_per_inc.append(int(df[df['INCREMENT']==increment]['TIME_OF_TE
ST'].iloc[-1]))

        summmary_df['S1'] = np.sort(np.array(set1_per_inc))
        summmary_df['S2'] = np.sort(np.array(set2_per_inc))
        summmary_df['S1_S2'] = np.array(summmary_df.S1 +
summmary_df.S2)
        summmary_df = summmary_df.sort_values('S1_S2')
        summmary_df['P'] =
np.sort(df.groupby('INCREMENT').mean()['PRESSURE'].values)

        summmary_df['S'] = np.sort((summmary_df.S1.values +
summmary_df.S2.values) / 2)
        summmary_df['TS'] = summmary_df['S'].cumsum()
        summmary_df['TIME'] = time_per_inc
        summmary_df['INC_NO'] = ['Increment '+ str(int(inc)) for
inc in unik]

        return summmary_df.reset_index(),
df['WIDTH_PLATE'].iloc[-1], df['WIDTH_FOOTING'].iloc[-1]
```

# PLOT GENERATOR

```
import chart_studio.plotly as py
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd
import numpy as np
import sys

sys.path.insert(1, '../../')
from getdata import GetData

getdata = GetData()

def scatter_data(df, height=300, plotType = '', table = ''):
      y1 = df.S1
      y2 = df.S2

      if plotType == 'normal' and table != 'test':
         y1 = np.sort(df.S1.values)
         y2 = np.sort(df.S2.values)

      fig1 = go.Figure()
      fig1.add_trace(go.Scatter(x=np.arange(len(df)),
                                y= y1,
                                name='Sensor 1',
                                marker_color='steelblue'))
      fig1.add_trace(go.Scatter(x=np.arange(len(df)),
                                y=y2,
                                name='Sensor 2',
                                marker_color='lightgreen'))
      fig1.update_layout(title=dict(
                        text='<b>DISTANCE SENSORS OUTPUT</b>',
                        font=dict(size=20, color='white')),
                        template='plotly_dark',
                        height=height,
                        yaxis=dict(range=[0,100]),
                        font=dict(family="Courier",
                                  size=12, color='gray'))
      fig1.update_xaxes(title='Time Interval [1s]')
      fig1.update_yaxes(title='Centimeter')
```

```
    #[Scatterplot 2]
    fig2 = go.Figure()
    fig2.add_trace(go.Scatter(x=np.arange(len(df)),
                              y=df.S3,
                              name='Sensor 3',
                              marker_color='salmon'))

    fig2.update_layout(title=dict(
                       text='<b>LOAD SENSOR OUTPUT</b>',
                       font=dict(size=20, color='white')),
                       template='plotly_dark',
                       height=height,
                       font=dict(family="Courier",
                                 size=12, color='gray'))
    fig2.update_xaxes(title='Time Interval [1s]')
    fig2.update_yaxes(title='Kg')

    return fig1, fig2
```

```
def get_lsc(df, x=[0,50,100,200,300,400,500,600],
            y=[0,1.5,2,4,7.5,12.5, 20, 40.6],
            ubc=500, ubc_s=20):

    x = np.append(0,df.P)
    y = np.append(0,df.S)
    max_settlement = df.S.max()
    fig = go.Figure()
    fig.add_trace(go.Scatter(x=x,
                             y=y,
                             name='Result',
                             marker_color='gold'))
    fig.add_shape(
        # Line Vertical
        dict(
            type="line",
            x0=ubc,
            y0=0,
            x1=ubc,
            y1=max_settlement,
            line=dict(
                color="cyan",
                width=4,
                dash='dot'
            )))
    fig.update_shapes(dict(xref='x', yref='y'))
    fig.update_layout(title=dict(
                            text='<b>LOAD SETTLEMENT CURVE</b>',
                            x=0.5,
                            y=0.1,
                            font=dict(size=20, color='white')),
                          template='plotly_dark',
                          height=300,
                          xaxis={ 'side': 'top'},
                          yaxis={'autorange':'reversed', 'side':
'left'},
                          font=dict(family="Courier",
                                    size=12, color='gray'))
    fig.update_xaxes(title='LOAD IN kg/m^2')
    fig.update_yaxes(title='SETTLEMENT IN mm')
    return fig
```

# APPLICATION LAYOUT

## HOME LAYOUT

```
import dash_html_components as html
import dash_core_components as dcc
import dash_bootstrap_components as dbc

import pandas as pd
import sys
# insert at 1, 0 is the script path (or '' in REPL)
sys.path.insert(1, '../../')
from getdata import GetData
sys.path.insert(1, './components/PlotGenerator')
from PlotGenerator import scatter_data, get_lsc
getdata = GetData(path='./databases/serverdb.db')

df = getdata.get_dataframe(path='./databases/serverdb.db')
#[Scatterplot 1]
fig1, fig2 = scatter_data(df)

dummy_lsc = [[0,50,100,200,300,400,500,600],
             [0,1.5,2,4,7.5,12.5, 20, 40.6]]
dummy_lsc = pd.DataFrame(dummy_lsc).T
dummy_lsc.columns = ['P', 'S']
lsc = get_lsc(dummy_lsc)

items = [
    dbc.DropdownMenuItem(
            html.A("Dashboard", id = 'dash-app', href='/',
                     style = {'color': 'black',
                              ':hover': {'color': 'black'}}),
            style={'height':'20px', 'width': '200px' }
        ),
    dbc.DropdownMenuItem(divider=True),
    dbc.DropdownMenuItem(
        dcc.Link("Calibration", id = 'calib-link',
href='/calibration',
                 style = {'color': 'black',
                          ':hover': {'color': 'black'}}),
        style={'height':'20px', 'width': '200px' }
    ),
    dbc.DropdownMenuItem(divider=True),
    dbc.DropdownMenuItem(
        dcc.Link("Close application", id = 'exit-app',
href='/shutdown',
                 style = {'color': 'black',
                          ':hover': {'color': 'black'}}),
        style={'height':'20px', 'width': '200px' }
    )
]

#HOME LAYOUT
home_layout = html.Div(
    id = 'home-id',
    style={
        'textAlign': 'center',
        'margin': '0 auto',
        'backgroundColor': '#050505',
        'position': 'relative',
        'padding': '0',
        'paddingBottom': '50px',
        'height': '100%'
    },
    className='row',

    children=[

        html.Div([
            html.Div([
                dbc.DropdownMenu(label="Menu",
                                 bs_size="sm",
                                 children=items,
                                 className='m-1',
                                 color='link')
            ], style={'position': 'absolute',
                      'top': '0px',
                      'right': '0px'}),
            html.Div([

                #[Device info]
                html.H6('CREATE A TEST',
                        style={'display': 'block',
                               'color': 'white',
```

```
                        'margin': '10px auto'}),
                html.Div([
                    dcc.Input(id='port',
                        debounce=True,
                        placeholder='Device Port',
                        type='text',
                        className='form-control',
                        required=True,
                        style={'height': '20px',
'fontSize':'10px'}),
                        ],
                    className='input-group input-group-sm mb-3
sm',
                    style={'width': '100%', 'margin': '0 auto'}
                ),

                html.Div([
                    dcc.Input(id='test-id',
                        debounce=True,
                        placeholder='Test ID',
                        type='text',
                        className='form-control',
                        required=True,
                        style={'height': '20px',
'fontSize':'10px'}),
                        ],
                    className='input-group input-group-sm mb-3',
                    style={'width': '100%', 'margin': '0 auto'}
                ),
                html.Div([
                    dcc.Input(id='inc-no',
                        debounce=True,
                        placeholder='Increment no.',
                        type='number',
                        className='form-control',
                        required=True,
                        style={'height': '20px',
'fontSize':'10px'}),
                        ],
                    className='input-group input-group-sm mb-3',
                    style={'width': '100%', 'margin': '0 auto'}
                ),
```

```
                #[CONSTANTS]
                html.Div([
                    dcc.Input(id='ini-set-1',
                        debounce=True,
                        placeholder='Initial settlement (DG1) in
cm',
                        type='number',
                        className='form-control',
                        required=True,
                        style={'height': '20px',
'fontSize':'10px'}),
                        ],
                    className='input-group input-group-sm mb-3',
                    style={'width': '100%', 'margin': '30px auto 0
auto'}
                ),
                html.Div([
                    dcc.Input(id='ini-set-2',
                        debounce=True,
                        placeholder='Initial settlement (DG2) in
cm',
                        type='number',
                        className='form-control',
                        required=True,
                        style={'height': '20px',
'fontSize':'10px'}),
                        ],
                    className='input-group input-group-sm mb-3',
                    style={'width': '100%', 'margin': '0 auto'}
                ),
                html.Div([
                    dcc.Input(id='plate-area',
                        debounce=True,
                        placeholder='Plate area in meter squared',
                        type='number',
                        className='form-control',
                        required=True,
                        style={'height': '20px',
'fontSize':'10px'}),
                        ],
                    className='input-group input-group-sm mb-3',
                    style={'width': '100%', 'margin': '0 auto'}
                ),
```

```
                html.Div([
                    dcc.Input(id='plate-width',
                        debounce=True,
                        placeholder='Width of Plate (meter)',
                        type='number',
                        className='form-control',
                        required=True,
                        style={'height': '20px',
'fontSize':'10px'}),
                        ],
                    className='input-group input-group-sm mb-3',
                    style={'width': '100%', 'margin': '0 auto'}
                ),
                html.Div([
                    dcc.Input(id='width-footing',
                        debounce=True,
                        placeholder='Width of Footing (meter)',
                        type='number',
                        className='form-control',
                        required=True,
                        style={'height': '20px',
'fontSize':'10px'}),
                        ],
                    className='input-group input-group-sm mb-3',
                    style={'width': '100%', 'margin': '0 auto'}
                ),
                html.Div([
                    dcc.Input(id='factor-safety',
                        debounce=True,
                        placeholder='Factor of safety',
                        type='number',
                        className='form-control',
                        required=True,
                        style={'height': '20px',
'fontSize':'10px'}),
                        ],
                    className='input-group input-group-sm mb-3',
                    style={'width': '100%', 'margin': '0 auto'}
                ),
                #[TIMER]
                html.Div([
                    dcc.Input(id='time-input',
                        debounce=True,
                        placeholder='Time in minutes',
                        type='number',
                        className='form-control',
                        required=True,
                        style={'height': '20px',
'fontSize':'10px'}),
                        ],
                    className='input-group input-group-sm mb-3',
                    style={'width': '100%', 'margin': '30px auto
10px auto'}
                ),



                    html.Button(
                        'Begin',
                        className='btn btn-dark btn-lg btn-block',
                        id = 'start-btn',
                        n_clicks=0,
                        style = {
                            'display': 'block',
                            'position': 'relative',
                            'height': '30px',
                            'fontSize': '12px',
                            'padding': '5px'
                        }),
                    html.Div(children=
                         html.P(['Click the button to start a test'],
                                id='test-indicator',
                                style = {'color': 'gray', 'fontSize':
'10px'}
                                ),
                         style={'borderBottom': '1px solid grey',
                                'paddingBottom': '20px'}),
                    ],
                id='start-test'
                ),
                html.Div([
                    html.H5('SUMMARY OF INPUTS',
                             style={'margin': '0 0 20px 0',
                                    'color': 'white',
                                    'borderBottom': '1px solid gray',
                                    'marginTop': '20px'}),
```

```
                html.P(id='p-test-id', style={'float': 'left',
                                'display': 'block',
                                'color': 'gray',
                                'margin': '0',
                                'fontSize': '15px'}),
                html.Br(),
                html.P(id='p-inc-no', style={'float': 'left',
                                'display': 'block',
                                'color': 'gray',
                                'margin': '0',
                                'fontSize': '15px'}),
                html.Br(),
                html.P(id='dg-1', style={'float': 'left',
                                'display': 'block',
                                'color': 'gray',
                                'margin': '0',
                                'fontSize': '15px'}),
                html.Br(),
                html.P(id='dg-2', style={'float': 'left',
                                'display': 'block',
                                'color': 'gray',
                                'margin': '0',
                                'fontSize': '15px'}),
                html.Br(),
                html.P(id='s-width-footing', style={'float':
'left',
                                'display': 'block',
                                'color': 'gray',
                                'margin': '0',
                                'fontSize': '15px'}),
                html.Br(),
                html.P(id='p-fs', style={'float': 'left',
                                'display': 'block',
                                'color': 'gray',
                                'margin': '0',
                                'fontSize': '15px'}),
                html.Br(),
                html.P(id='p-time', style={'float': 'left',
                                'display': 'block',
                                'color': 'gray',
                                'margin': '0',
                                'fontSize': '15px',
                                'paddingBottom': '50px'}),
```

```
            ], style={'display': 'none'}, id='summary'),

            html.Div(children=[
            ],id='timer-div'),
            html.Button(
                    html.A('Create another test?', href ='/'),
                    className='btn btn-dark btn-lg btn-block',
                    id = 'start-an-btn',
                    n_clicks=0,
                    style = {
                        'display': 'none',
                        'position': 'relative',
                        'height': '30px',
                        'fontSize': '12px',
                        'padding': '5px'
                    }),

            html.H6('CHECK TEST RESULTS',
                        style={'display': 'block',
                                'color': 'white',
                                'marginTop': '20px'}),

            html.Div(
                children=[
                    html.Div(
                        children=[
                            html.Button(
                                children='Check',
                                id = 'view1',
                                className='btn btn-dark',
                                style={'height': '30px',
                                        'fontSize': '12px',
                                        'padding': '5px'}
                            )
                        ],
                        className='input-group-prepend'
                    ),
                    dcc.Input(
                        id= 'inp-sc',
                        placeholder='Test ID',
                        type='text',
                        className='form-control',
```

```
                style={'height': '30px',
                        'fontSize': '12px',
                        'padding': '5px'}
            )
        ],
        className='input-group mb-3',
        style={'width': '100%', 'margin': '10px auto 30px
auto'}
    ),

    html.H6('GENERATE A EXCEL FILE',
                style={'display': 'block',
                        'color': 'white',
                        'marginTop': '20px'}),
    html.Div(
            children=[
                html.Div(
                    children=[
                        html.Button(
                            children='Generate',
                            id = 'view2',
                            className='btn btn-dark',
                            style={'height': '30px',
                                    'fontSize': '12px',
                                    'padding': '5px'}
                        )
                    ],
                    className='input-group-prepend'
                ),
                dcc.Input(
                    id= 'inp-csv',
                    placeholder='Test ID',
                    type='text',
                    className='form-control',
                    style={'height': '30px',
                            'fontSize': '12px',
                            'padding': '5px'}
                ),
            ],
            className='input-group mb-3',
            style={'width': '100%',
                   'margin': '10px auto 30px auto'}),
    html.Br(),
    dcc.Loading(
        html.Div(id= 'loader_csv'),
        type='dot',
        color = 'lightgreen'
    ),
    html.P('Back to dashboard',
            id='back-dash',
            style={'display': 'none'})
],
    style={'width': '50%',
           'margin': '0 auto',
           'marginTop':'10px',
           'paddingTop': '20px',
           'paddingBottom': '30px',
           'backgroundColor': '#1b1c25',
           'borderRadius': '5px'},
    className='col-6 col-md-2'
),



html.Div([
    html.H4(children='PLATE LOAD TEST DASHBOARD',
            style={'color':'white',
                   'fontFamily': 'sans-serif',
                   'fontWeight': '700',
                   'padding': '10px'}),

    html.Div(
        children=[
            dcc.Graph(
                id = 'disp-sensor',
                figure= fig1
            )
        ],
        id = 'graph-1',
        style={'width': '40%', 'display':'inline-block'}

    ),
    html.Div(
        children=[
            dcc.Graph(
                id = 'load-sensor',
```

```
                        figure= fig2
                    )
                ],
                id='graph-2',
                style={'width': '40%', 'display':'inline-block'}),
            html.Div([
                html.Ul(
                    [html.Li('Ultimate Bearing Capacity',
id='m1',style={'margin': '0px', 'width': '80%'}),
                    html.Li('Safe Bearing Capacity',
id='m2',style={'margin': '0px', 'width': '80%'}),
                    html.Li('Settlement of Footing',
id='m3',style={'margin': '0px'})],
                    style={'textAlign': 'left', 'color': '#ccc'}
                    )
                ], id='measurements',style={'marginLeft': '10%',
'display': 'none'}),

            html.Div(
                children=[
                    dcc.Graph(
                        id = 'lsc',
                        figure= lsc
                    )
                ],
                id = 'graph-2',
                style={'width': '80%', 'display':'inline-block'}

            ),
            html.Div([
                dbc.Button("Generate Results", color="dark",
block=True, id='gen-res')
            ], style={'margin': '0 auto', 'width': '80%'})

        ],
            className='col-md-10'
        ),


])
```

# CALIBRATION LAYOUT

```
import dash_html_components as html
import dash_core_components as dcc
import dash_bootstrap_components as dbc

import sys
# insert at 1, 0 is the script path (or '' in REPL)
sys.path.insert(1, '../../')
from getdata import GetData
sys.path.insert(1, './components/PlotGenerator')
from PlotGenerator import scatter_data, get_lsc
getdata = GetData(path='./databases/serverdb.db')

df =
getdata.get_dataframe(table='test',path='./databases/serverdb.db')
fig1,fig2 = scatter_data(df, height=400)

calibration_layout = html.Div(
    id = 'calib-id',
    style={
        'textAlign': 'center',
        'margin': '0 auto',
        'backgroundColor': '#050505',
        'position': 'relative',
        'height': '100vh'
    },
    className='row',

    children=[

        html.Div([
            html.Button(id='start-an-btn', style={'display':
'none'}),
            html.Button(id='calib-link', style={'display':
'none'}),

            html.H6('CHECK DEVICE CONNECTION',
                        style={'display': 'block',
                                'color': 'white',
                                'marginTop': '20px',
                                'fontSize': '14px'}),

            html.Div(
                children=[
                    html.Div(
                        children=[
                            html.Button(
                                children='Check',
                                id = 'check-port',
                                className='btn btn-dark'
                            )
                        ],
                        className='input-group-prepend'
                    ),
                    dcc.Input(
                        id= 'ca-port-id',
                        placeholder='Port',
                        type='text',
                        className='form-control'
                    )
                ],
                className='input-group mb-3',
                style={'width': '100%', 'margin': '10px auto 30px
auto'}
            ),
            html.Div(style={'paddingTop': '5px'}),
            html.Div([
                dcc.Loading(
                    html.Div(children='',
                             id='ca-ind',
                             style={'color': 'lightgreen',
'fontSize':'15px'}),
                    type= 'dot',
                    color='steelblue'
                )]
            ),
            html.Div(style={'paddingTop': '5px'}),
            html.H6('CHECK SENSORS OUTPUT',
                        style={'display': 'block',
                                'color': 'white',
                                'margin': '10px auto',
                                'fontSize': '14px'}),

            html.Div(
                html.Button('Generate',
```

```
                classNameName='btn btn-dark btn-lg btn-block',
                id = 'ca-start-btn',
                n_clicks=0,
                style = {
                    'display': 'block',
                    'position': 'relative'
                })),

            dcc.Loading(
                    children= html.Div(children='Click the button
to start a test',
                                       id='ca-test-indicator',
                                       style={'color': 'white',
                                              'borderBottom': '1px
solid grey',

                                              'padding': '10px 0
20px 0',

                                              'fontSize': '10px',
'color': 'grey'}),
                    type='default',
                    color='lightgreen'
                ),


            dcc.Link('Back to Dashboard',
                    id='back-dash',
                    style={'textDecoration': 'underline',
                           'color':'white',
                           'fontSize': '15px',
                           'position':'absolute',
                           'bottom': '20px',
                           'cursor': 'pointer',
                           'left': '10%'},
                    href = '/'
            ),

        ],
            style={'width': '50%',
                   'margin': '0 auto',
                   'marginTop':'10px',
                   'paddingTop': '120px',
                   'backgroundColor': '#1b1c25',
                   'paddingTop': '100px'},
            classNameName='col-6 col-md-2'
    ),

    html.Div([
        html.H2(children='LIVE PLOTTER',
                style={'color':'white',
                       'fontFamily': 'sans-serif',
                       'fontWeight': '700',
                       'padding': '10px'}),

        html.Div(
            children=[
                dcc.Graph(
                    id = 'ca-disp-sensor',
                    figure=fig1
                ),
                dcc.Interval(
                    id='interval-component1',
                    interval=1000, # in milliseconds
                    n_intervals=0
                )
            ],
            id = 'ca-1',
            style={'width': '40%', 'display':'inline-block'}

        ),
        html.Div(
            children=[
                dcc.Graph(
                    id = 'ca-load-sensor',
                    figure=fig2
                ),
                dcc.Interval(
                    id='interval-component2',
                    interval=1000, # in milliseconds
                    n_intervals=0
                )
            ],
            id='ca-2',
            style={'width': '40%', 'display':'inline-block'}),
        html.Button(
            'Clear output',
            classNameName='btn btn-dark btn-lg btn-block',
```

```
            id = 'ca-clear-btn',
            n_clicks=0,
            style = {
                'display': 'block',
                'position': 'relative',
                'width': '80%',
                'margin': '0 auto'
            }),

        ],
            className='col-md-10',
            style={'padding': '50px 0 0 0'}
        ),


])
```