```python
from kivy.uix.boxlayout import BoxLayout
from popups import ModbusPopup, ScanPopup, DataGraphPopup, HistGraphPopup
from pyModbusTCP.client import ModbusClient
from kivy.core.window import Window
from threading import Thread
from time import sleep
from datetime import datetime
import random
from timeseriesgraph import TimeSeriesGraph
from bdhandler import BDHandler
from kivy_garden.graph import LinePlot

class MainWidget(BoxLayout):
    """
    Widget principal da aplicação
    """
    _updateThread = None
    _updateWidgets = True
    _tags = {}
    _max_points = 20

    def __init__(self, **kwargs):
        """
        Construtor do widget principal
        """
        super().__init__()
        self._scan_time = kwargs.get('scan_time')
        self._serverIP = kwargs.get('server_ip')
        self._serverPort = kwargs.get('server_port')
        self._modbusPopup = ModbusPopup(self._serverIP,self._serverPort)
        self._scanPopup = ScanPopup(self._scan_time)
        self._modbusClient = ModbusClient(host=self._serverIP,port=self._serverPort)
        self._meas = {}
        self._meas['timestamp'] = None
        self._meas['values'] = {}
        for key,value in kwargs.get('modbus_addrs').items():
            if key == 'fornalha':
                plot_color = (1,0,0,1)
            else:
                plot_color = (random.random(),random.random(),random.random(),1)
            self._tags[key] = {'addr': value, 'color':plot_color}
        self._graph = DataGraphPopup(self._max_points,self._tags['fornalha']
['color'])
        self._hgraph = HistGraphPopup(tags=self._tags)
        self._db = BDHandler(kwargs.get('db_path'),self._tags)

    def startDataRead(self, ip, port):
        """
        Método utilizado para a configuração do IP e porta do servidor MODBUS e
        inicializar uma thread para a leitura dos dados e atualização da interface
        gráfica
        """
        self._serverIP = ip
        self._serverPort = port
        self._modbusClient.host = self._serverIP
        self._modbusClient.port = self._serverPort
```

```
        try:
            Window.set_system_cursor("wait")
            self._modbusClient.open()
            Window.set_system_cursor("arrow")
            if self._modbusClient.is_open():
                self._updateThread = Thread(target=self.updater)
                self._updateThread.start()
                self.ids.img_con.source = 'imgs/conectado.png'
                self._modbusPopup.dismiss()
            else:
                self._modbusPopup.setInfo("Falha na conexão com o servidor")
        except Exception as e:
            print("Erro: ",e.args)

    def updater(self):
        """
        Método que invoca as rotinas de leitura dos dados, atualização da interface e
        inserção dos dados no Banco de dados
        """
        try:
            while self._updateWidgets:
                self.readData()
                self.updateGUI()
                self._db.insertData(self._meas)
                sleep(self._scan_time/1000)
        except Exception as e:
            self._modbusClient.close()
            print("Erro: ",e.args)

    def readData(self):
        """
        Método para a leitura dos dados por meio do protocolo MODBUS
        """
        self._meas['timestamp'] = datetime.now()
        for key,value in self._tags.items():
            self._meas['values'][key] =
self._modbusClient.read_holding_registers(value['addr'],1)[0]

    def updateGUI(self):
        """
        Método para atualização da interface gráfica a partir dos dados lidos
        """
        #Atualização dos labels das temperaturas
        for key,value in self._tags.items():
            self.ids[key].text = str(self._meas['values'][key]) + ' ºC'

        #Atualização do nível do termômetro
        self.ids.lb_temp.size = (self.ids.lb_temp.size[0],self._meas['values']
['fornalha']/450*self.ids.termometro.size[1])

        #Atualização do gráfico

 self._graph.ids.graph.updateGraph((self._meas['timestamp'],self._meas['values']
['fornalha']),0)

    def stopRefresh(self):
        self._updateWidgets = False
```

```
109
110     def getDataDB(self):
111         """
112         Método que coleta as informações da interface fornecidas pelo usuário
113         e requisita a busca no BD
114         """
115         try:
116             init_t = self.parseDTString(self._hgraph.ids.txt_init_time.text)
117             final_t = self.parseDTString(self._hgraph.ids.txt_final_time.text)
118             cols = []
119             for sensor in self._hgraph.ids.sensores.children:
120                 if sensor.ids.checkbox.active:
121                     cols.append(sensor.id)
122 
123             if init_t is None or final_t is None or len(cols)==0:
124                 return
125 
126             cols.append('timestamp')
127 
128             dados = self._db.selectData(cols, init_t, final_t)
129 
130             if dados is None or len(dados['timestamp'])==0:
131                 return
132 
133             self._hgraph.ids.graph.clearPlots()
134 
135             for key,value in dados.items():
136                 if key == 'timestamp':
137                     continue
138                 p = LinePlot(line_width=1.5, color=self._tags[key]['color'])
139                 p.points = [(x, value[x]) for x in range(0,len(value))]
140                 self._hgraph.ids.graph.add_plot(p)
141             self._hgraph.ids.graph.xmax = len(dados[cols[0]])
142             self._hgraph.ids.graph.update_x_labels([datetime.strptime(x,"%Y-%m-%d
    %H:%M:%S.%f") for x in dados['timestamp']])
143         except Exception as e:
144             print("Erro: ", e.args)
145 
146     def parseDTString(self, datetime_str):
147         "Método que converte a string inserida pelo usuário para o formato utilizado
    na busca dos dados no BD"
148         try:
149             d = datetime.strptime(datetime_str, '%d/%m/%Y %H:%M:%S')
150             return d.strftime("%Y-%m-%d %H:%M:%S")
151         except Exception as e:
152             print("Erro: ", e.args)
153 
154 
```