**נספחים**

**הקוד של המחשב:**

המסך הראשון:

from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
  
class Window1(QWidget):  
 # this window will present the QR code  
 def \_\_init\_\_(self, path):  
 super(Window1, self).\_\_init\_\_()  
 self.path = path  
 self.window1\_ui()  
  
 def window1\_ui(self):  
 # creating the label that will hold the QR code  
 self.photo = QLabel(self)  
 self.photo.setPixmap(QPixmap(self.path))  
 self.photo.setScaledContents(True)  
 self.photo.setGeometry(QRect(300, 0, 700, 700))

המסך השני:

from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
  
  
class Window2(QWidget):  
 # this is the window where you choose files  
 finished\_choosing\_files = pyqtSignal(list)  
  
 def \_\_init\_\_(self):  
 super(Window2, self).\_\_init\_\_()  
 self.setup\_ui()  
  
 def setup\_ui(self):  
 self.files = []  
 self.setObjectName("Form")  
  
 # creating the vertical layout and the actual widget  
 self.vertical\_layout\_widget = QWidget(self)  
 self.vertical\_layout\_widget.setGeometry(QRect(200, 100, 900, 500))  
 self.vertical\_layout\_widget.setObjectName("verticalLayoutWidget")  
  
 self.vertical\_layout = QVBoxLayout(self.vertical\_layout\_widget)  
 self.vertical\_layout.setContentsMargins(0, 0, 0, 0)  
 self.vertical\_layout.setObjectName("verticalLayout")  
  
 # creating the button that you press to add a file and adding it to the vertical layout  
 self.push\_button = QPushButton(self.vertical\_layout\_widget)  
 self.push\_button.setObjectName("pushButton")  
 self.push\_button.setText("+")  
 self.push\_button.setFont(QFont('Arial', 15))  
 self.push\_button.clicked.connect(self.get\_file\_path)  
 self.vertical\_layout.addWidget(self.push\_button)  
  
 # creating a label and adding it to the vertical layout  
 self.title\_label = QLabel(self.vertical\_layout\_widget)  
 self.title\_label.setText("files:")  
 self.title\_label.setFont(QFont('Arial', 15))  
 self.vertical\_layout.addWidget(self.title\_label)  
  
 # creating the area that you add the files to in the GUI  
 self.scroll\_area = QScrollArea(self.vertical\_layout\_widget)  
 self.scroll\_area.setWidgetResizable(True)  
 self.scroll\_area.setObjectName("scrollArea")  
  
 self.scroll\_area\_widget\_contents = QWidget()  
 self.scroll\_area\_widget\_contents.setGeometry(QRect(0, 0, 287, 153))  
 self.scroll\_area\_widget\_contents.setObjectName("scrollAreaWidgetContents")  
  
 layout = QVBoxLayout(self.scroll\_area\_widget\_contents)  
 layout.addStretch()  
  
 self.scroll\_area\_widget\_contents.setLayout(layout)  
 self.scroll\_area.setWidget(self.scroll\_area\_widget\_contents)  
 self.vertical\_layout.addWidget(self.scroll\_area)  
  
 # creating the button that you press when you're finished choosing files and adding it to the vertical layout  
 self.ok\_button = QPushButton(self.vertical\_layout\_widget)  
 self.ok\_button.setObjectName("okButton")  
 self.ok\_button.setText("OK")  
 self.ok\_button.setFont(QFont('Arial', 15))  
 self.ok\_button.clicked.connect(self.ok\_button\_clicked)  
 self.vertical\_layout.addWidget(self.ok\_button)  
  
 def add\_file(self, file\_path):  
 # adding the file to the list of files updating the GUI with the file  
 self.files.append(file\_path)  
 self.label = QLabel(file\_path, self.scroll\_area\_widget\_contents)  
 self.scroll\_area\_widget\_contents.layout().addWidget(self.label)  
  
 def get\_file\_path(self):  
 # opening the file system and getting the location of the file i choose  
 file\_app = QFileDialog(self)  
 file\_app.fileSelected.connect(self.add\_file)  
 file\_app.setFixedSize(1300, 700)  
 file\_app.show()  
  
 def ok\_button\_clicked(self):  
 self.finished\_choosing\_files.emit(self.files)

המסך השלישי:

from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
  
  
class Window3(QWidget):  
 # this is the window where you pick locations for the files the other device chose  
 all\_files\_have\_location = pyqtSignal(dict)  
 def \_\_init\_\_(self):  
 super(Window3, self).\_\_init\_\_()  
 self.file\_location\_dict = {}  
 self.setup\_ui()  
  
 def setup\_ui(self):  
 self.setObjectName("Form")  
  
 # creating the vertical layout and the actual widget  
 self.vertical\_layout\_widget = QWidget(self)  
 self.vertical\_layout\_widget.setGeometry(QRect(200, 100, 900, 500))  
 self.vertical\_layout\_widget.setObjectName("vertical\_layout\_widget")  
  
 self.vertical\_layout = QVBoxLayout(self.vertical\_layout\_widget)  
 self.vertical\_layout.setContentsMargins(0, 0, 0, 0)  
 self.vertical\_layout.setObjectName("vertical\_layout")  
  
 # creating a label and adding it to the vertical layout  
 self.title\_label = QLabel(self.vertical\_layout\_widget)  
 self.title\_label.setObjectName("label")  
 self.title\_label.setText("Files:")  
 self.title\_label.setFont(QFont('Arial', 15))  
 self.vertical\_layout.addWidget(self.title\_label)  
  
 # creating the area where the files that the other device chose will be  
 self.scroll\_area = QScrollArea(self.vertical\_layout\_widget)  
 self.scroll\_area.setWidgetResizable(True)  
 self.scroll\_area.setObjectName("scroll\_area")  
  
 self.scroll\_area\_widget\_contents = QWidget()  
 self.scroll\_area\_widget\_contents.setGeometry(QRect(0, 0, 417, 255))  
 self.scroll\_area\_widget\_contents.setObjectName("scroll\_area\_widget\_contents")  
  
 layout = QFormLayout(self.scroll\_area\_widget\_contents)  
  
 self.scroll\_area\_widget\_contents.setLayout(layout)  
 self.scroll\_area.setWidget(self.scroll\_area\_widget\_contents)  
 self.vertical\_layout.addWidget(self.scroll\_area)  
  
 # creating the button that you press when you're finished choosing files and adding it to the vertical layout  
 self.ok\_button = QPushButton(self.vertical\_layout\_widget)  
 self.ok\_button.setObjectName("ok\_button")  
 self.ok\_button.setText("OK")  
 self.ok\_button.setFont(QFont('Arial', 15))  
 self.ok\_button.clicked.connect(self.check\_all\_files\_have\_location)  
 self.vertical\_layout.addWidget(self.ok\_button)  
  
 def select\_directory(self, file\_name):  
 # creating the object of the file system  
 options = QFileDialog.Options()  
 options |= QFileDialog.ReadOnly  
 file\_dialog = QFileDialog(self, options=options)  
  
 # setting the object to a mode in which you choose a directory instead of a file  
 file\_dialog.setFileMode(QFileDialog.Directory)  
 file\_dialog.setOption(QFileDialog.ShowDirsOnly, True)  
 file\_dialog.setFixedSize(1300, 700)  
 file\_dialog.show()  
 try:  
 # it tries to see if you chose a directory or just closed the window  
 # if you chose a directory it would work and if not the try except will catch it  
 if file\_dialog.exec\_():  
 directory = file\_dialog.selectedFiles()[0]  
 self.file\_location\_dict[file\_name] = directory  
 except:  
 pass  
  
 def check\_all\_files\_have\_location(self):  
 files\_have\_location = True  
 # it goes over each file and checking if it has a location or is it a None type  
 for location in self.file\_location\_dict.values():  
 if location == None:  
 files\_have\_location = False  
 if files\_have\_location:  
 # if all files have a location then it emits a dictionary with the files and their location  
 self.all\_files\_have\_location.emit(self.file\_location\_dict)  
  
 def add\_files(self, files):  
 # here you add files to this window  
 for file in files:  
 # for each file it extracts the file name insert it to the dict and adds a label for it in the GUI  
 file = file.split("/")[-1]  
 self.file\_location\_dict[file] = None  
 label = QLabel(file)  
 button = QPushButton()  
 button.clicked.connect(self.create\_select\_directory\_function(label.text()))  
 button.setText("Choose Location")  
 self.scroll\_area\_widget\_contents.layout().addRow(label, button)  
  
 def create\_select\_directory\_function(self, label\_text):  
 return lambda: self.select\_directory(label\_text)

מסך ההודעות:

from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
import sys  
  
class MessageWindow(QWidget):  
 # this is the message window  
 def \_\_init\_\_(self, message):  
 super(MessageWindow, self).\_\_init\_\_()  
 self.message = message  
 self.setupUI()  
  
 def setupUI(self):  
 # we create the label that holds the message  
 self.message\_label = QLabel(self)  
 self.message\_label.setText(self.message)  
 self.message\_label.setFont(QFont('Arial', 10))  
 self.message\_label.setGeometry(QRect(300, 300, 700, 100))  
  
 def change\_message(self, message):  
 # with this function we can change that message  
 self.message\_label.setText(message)

המסך הראשי:

from window1 import \*  
from window2 import \*  
from window3 import \*  
from message\_win import \*  
  
  
class MainWindow(QMainWindow):  
 # this is the window that supervises the GUI  
 def \_\_init\_\_(self, path):  
 super(MainWindow, self).\_\_init\_\_()  
 self.path = path  
 self.setGeometry(100, 100, 1300, 700)  
 self.setWindowTitle("project")  
 self.setup\_ui()  
  
 def setup\_ui(self):  
 self.current\_win = 0  
  
 # here i create the windows  
 self.window1 = Window1(self.path)  
 self.window2 = Window2()  
 self.window3 = Window3()  
 self.message\_win = MessageWindow("")  
  
 # and add them to the stack  
 self.stack = QStackedWidget(self)  
 self.stack.setGeometry(0, 0, 1300, 700)  
 self.stack.addWidget(self.window1)  
 self.stack.addWidget(self.window2)  
 self.stack.addWidget(self.window3)  
 self.stack.addWidget(self.message\_win)  
  
 # with the stack i can replace which window is being presented to the user  
  
 self.hbox = QHBoxLayout(self)  
 self.hbox.addWidget(self.stack)  
  
 self.stack.setCurrentIndex(self.current\_win)  
  
 def change\_win(self):  
 # this function changes the window to the next one  
 self.current\_win += 1  
 self.stack.setCurrentIndex(self.current\_win)  
  
 def change\_to\_message\_win(self, message):  
 # this function changes the window to the message window  
 self.message\_win.change\_message(message)  
 self.stack.setCurrentIndex(3)

קובץ שבוא המחלקות האחראיות על התקשורת:

import time  
  
from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
import socket  
import pickle  
import os  
from cryptography.fernet import Fernet  
  
  
  
class MainSendingSocket(QThread):  
 got\_file\_list = pyqtSignal(list)  
 ready\_to\_send = pyqtSignal()  
 send\_massage = pyqtSignal(str)  
 done\_signal = pyqtSignal()  
 exception\_rose = pyqtSignal(str)  
  
 def \_\_init\_\_(self, ip, port, key):  
 super(MainSendingSocket, self).\_\_init\_\_()  
 # here i set up all the important information for the connection  
 self.ip = ip  
 self.port = port  
 self.sending\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.encrypting\_object = Fernet(key)  
  
 self.mutex = QMutex()  
 self.condition = QWaitCondition()  
  
 # here i connect the signals of this class to a slot  
 self.got\_file\_list.connect(self.got\_files)  
 self.ready\_to\_send.connect(self.ready\_to\_send\_files)  
 self.send\_massage.connect(self.send)  
 self.done\_signal.connect(self.done)  
 self.done\_condition = False  
  
 self.files = []  
  
 def run(self):  
 self.mutex.lock()  
 # here it connects to the phone and then waits until it has the list of files to send  
 self.connect\_to\_phone()  
 self.condition.wait(self.mutex)  
  
 # it converts the file list to bytes and sends it  
 serialized\_file\_list = pickle.dumps(self.files)  
  
 try:  
 self.sending\_socket.send(self.encrypting\_object.encrypt(serialized\_file\_list))  
  
 # here it wait again until the computer is ready to start sending the files  
 self.condition.wait(self.mutex)  
 self.sending\_socket.send(self.encrypting\_object.encrypt("ready for files".encode()))  
  
 while True:  
 # here it waits until the project tells it to send a message  
 # the message is that either a socket opened or that a socket connected  
 self.condition.wait(self.mutex)  
 self.sending\_socket.send(self.encrypting\_object.encrypt(self.message.encode()))  
 if self.done\_condition:  
 break  
  
 except Exception as exception:  
 self.exception\_rose.emit(str(exception))  
 self.mutex.unlock()  
  
 def connect\_to\_phone(self):  
 # this function connects to the socket on the phone  
 try:  
 self.sending\_socket.connect((self.ip, self.port))  
  
 except Exception as exception:  
 self.exception\_rose.emit(str(exception))  
  
  
 def got\_files(self, files):  
 # when the program will have the files a signal will be emitted to this slot and this will run  
 # this function sets the file list and tells the program to stop waiting and continue  
 self.files = files  
  
 self.mutex.lock()  
 self.condition.wakeAll()  
 self.mutex.unlock()  
  
 def ready\_to\_send\_files(self):  
 # when the program will be ready to send the files a signal will be emitted to this slot and this will run  
 # this function tells the program to stop waiting and continue  
 self.mutex.lock()  
 self.condition.wakeAll()  
 self.mutex.unlock()  
  
 def send(self, message):  
 # when the program needs to send a message (during the file sending) a signal will be emitted to this slot and this will run  
 # this function sets the message list and tells the program to stop waiting and continue  
 self.mutex.lock()  
 self.message = message  
 self.condition.wakeAll()  
 self.mutex.unlock()  
  
 def done(self):  
 # when the program is done a signal will be emitted to this slot and this will run  
 # this function sets the done condition to true list and tells the program to stop waiting and continue  
 self.done\_condition = True  
 self.mutex.lock()  
 self.condition.wakeAll()  
 self.mutex.unlock()  
  
  
class FileSendingSocket(MainSendingSocket):  
 def \_\_init\_\_(self, ip, port, file\_path, key):  
 super(FileSendingSocket, self).\_\_init\_\_(ip, port, key)  
 self.file\_path = file\_path  
 self.BUFFER\_SIZE = 1024  
  
 def run(self):  
 self.connect\_to\_phone()  
  
 # we send the socket the file name  
 file\_name = self.file\_path.split("/")[-1]  
 try:  
 self.sending\_socket.send(self.encrypting\_object.encrypt(str(file\_name).encode()))  
  
 with open(self.file\_path, "rb") as f:  
 while True:  
 # we read 1024 bytes from the file  
 bytes\_read = f.read(self.BUFFER\_SIZE)  
 if not bytes\_read:  
 # file transmitting is done  
 break  
  
 # we encrypt the data  
 encrypted\_bytes = self.encrypting\_object.encrypt(bytes\_read)  
 size = len(encrypted\_bytes)  
  
 # we send the size and then the data  
 self.sending\_socket.send(str(size).encode())  
 message = self.sending\_socket.recv(1024)  
 self.sending\_socket.send(encrypted\_bytes)  
  
  
 except Exception as exception:  
 self.exception\_rose.emit(str(exception))  
  
 self.sending\_socket.close()  
  
  
class MainReceivingSocket(QThread):  
 connection\_made = pyqtSignal(tuple)  
 got\_file\_list\_from\_phone = pyqtSignal(list)  
 ready\_for\_files = pyqtSignal()  
 receive = pyqtSignal(str)  
 done\_signal = pyqtSignal()  
 exception\_rose = pyqtSignal(str)  
  
 def \_\_init\_\_(self, ip, port, key):  
 super(MainReceivingSocket, self).\_\_init\_\_()  
 # here i set up all the important information for the connection  
 self.ip = ip  
 self.port = port  
 self.done\_signal.connect(self.done)  
 self.done\_condition = False  
 self.encrypting\_object = Fernet(key)  
  
  
  
 def run(self):  
 self.handle\_connection()  
 try:  
 # we receive the file list and emit to the project object  
 serialized\_file\_list = self.encrypting\_object.decrypt(self.receiving\_socket.recv(1024))  
 list\_of\_files = pickle.loads(serialized\_file\_list)  
  
 self.got\_file\_list\_from\_phone.emit(list\_of\_files)  
  
 # we wait to receive a message that the phone is ready for sending  
 message = self.encrypting\_object.decrypt(self.receiving\_socket.recv(1024)).decode()  
 if message == "ready for files":  
 self.ready\_for\_files.emit()  
  
 while True:  
 # here it waits to receive a message  
 # the message is that either a socket opened or that a socket connected  
 message = self.encrypting\_object.decrypt(self.receiving\_socket.recv(1024)).decode()  
 self.receive.emit(message)  
 if self.done\_condition:  
 break  
  
 except Exception as exception:  
 self.exception\_rose.emit(str(exception))  
 self.receiving\_socket.close()  
  
 def handle\_connection(self):  
 # here we open a socket and listen for connections on the address specified  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 sock.bind((self.ip, self.port))  
 sock.listen()  
 try:  
 self.receiving\_socket, self.address = sock.accept()  
 except Exception as exception:  
 self.exception\_rose.emit(str(exception))  
 sock.close()  
 self.handle\_address()  
  
 def handle\_address(self):  
 self.connection\_made.emit(self.address)  
  
  
 def done(self):  
 self.done\_condition = True  
  
class FileReceivingSocket(MainReceivingSocket):  
 def \_\_init\_\_(self, ip, port, files\_and\_paths, key):  
 super(FileReceivingSocket, self).\_\_init\_\_(ip, port, key)  
 self.file = None  
 self.BUFFER\_SIZE = 1024  
 self.files\_and\_paths = files\_and\_paths  
 self.finished = False  
  
 def run(self):  
 self.handle\_connection()  
 try:  
 # here we receive the file name form the socket  
 file\_name = self.encrypting\_object.decrypt(self.receiving\_socket.recv(self.BUFFER\_SIZE)).decode()  
 location = self.files\_and\_paths[file\_name]  
  
 time.sleep(1)  
  
 with open(f"{location}/{file\_name}", "wb") as file:  
 while True:  
 # we read the size of the part from the socket  
 size = self.receiving\_socket.recv(1024)  
 self.receiving\_socket.send(size)  
 if size == "":  
 # their is no next piece of data so it doesn't have a size  
 break  
 size = int(size.decode())  
  
 # we receive the data and decrypt it  
 encrypted\_bytes = self.receiving\_socket.recv(size)  
 bytes\_read = self.encrypting\_object.decrypt(encrypted\_bytes)  
  
 if not bytes\_read:  
 # nothing is received  
 # file transmitting is done  
 break  
 # write to the file the bytes we just received  
 file.write(bytes\_read)  
 except Exception as exception:  
 self.exception\_rose.emit(str(exception))  
 self.finished = True  
 self.receiving\_socket.close()  
  
 def handle\_address(self):  
 pass

אובייקט הפרויקט האחראי על הכל:

import time  
from cryptography.fernet import Fernet  
from gui import \*  
from connection import \*  
import qrcode  
from threading import Thread  
import sys  
  
class Project:  
 def \_\_init\_\_(self):  
 # we set all the important stuff here  
 self.key = Fernet.generate\_key()  
 self.ip = str(socket.gethostbyname(socket.gethostname()))  
 self.port = 6744  
 self.phone\_port = 6745  
 self.sending\_port = 6746  
 self.path = "qr.png"  
 self.qr\_image = qrcode.make(f"{self.ip} {str(self.port)} {self.key.decode()}")  
 self.qr\_image.save(self.path)  
 self.main\_window = MainWindow(self.path)  
 self.main\_receiving\_socket = MainReceivingSocket(self.ip, self.port, self.key)  
 self.got\_files = False  
 self.finished\_window2 = False  
 self.ready\_for\_the\_files = False  
 self.is\_phone\_ready\_for\_the\_files = False  
 self.mutex = QMutex()  
 self.condition = QWaitCondition()  
 self.files\_received = []  
  
 # here we connect all the signals from the sockets and windows  
 self.main\_receiving\_socket.connection\_made.connect(self.handle\_connection)  
 self.main\_receiving\_socket.got\_file\_list\_from\_phone.connect(self.handle\_files)  
 self.main\_receiving\_socket.ready\_for\_files.connect(self.phone\_ready\_for\_files)  
 self.main\_window.window2.finished\_choosing\_files.connect(self.finished\_window2)  
 self.main\_window.window3.all\_files\_have\_location.connect(self.finished\_window3)  
 self.main\_receiving\_socket.receive.connect(self.received\_message)  
 self.main\_receiving\_socket.exception\_rose.connect(self.exception\_rose)  
  
 self.main\_receiving\_socket.start()  
  
 def exception\_rose(self, error\_message):  
 # this function gets called in case of an exception  
 self.main\_window.change\_to\_message\_win(error\_message)  
  
 def handle\_connection(self, address):  
 # this gets called when the phone connected and now we need to connect to the phone  
 self.phone\_ip = address[0]  
 self.main\_sending\_socket = MainSendingSocket(self.phone\_ip, self.phone\_port, self.key)  
 self.main\_sending\_socket.exception\_rose.connect(self.exception\_rose)  
 self.main\_sending\_socket.start()  
 self.main\_window.change\_win()  
  
 def handle\_files(self, list\_of\_files):  
 # this handles the file list from the phone and adds it to the GUI  
 self.files\_from\_phone = list\_of\_files  
 self.main\_window.window3.add\_files(self.files\_from\_phone)  
 self.got\_files = True  
 if (self.finished\_window2):  
 self.main\_window.change\_win()  
  
 def phone\_ready\_for\_files(self):  
 # this gets called when the phone sends a message that it is read to send the files  
 self.is\_phone\_ready\_for\_the\_files = True  
 if self.ready\_for\_the\_files:  
 # if we are also ready to send the files the file sending starts  
 self.main\_window.change\_to\_message\_win("transferring files")  
 thread = Thread(target=self.send\_files)  
 thread.start()  
 else:  
 pass  
  
 def send\_files(self):  
 self.file\_sending\_sockets = []  
 self.file\_recving\_sockets = []  
  
 # sending the files  
 self.mutex.lock()  
 for file in self.files:  
 # waiting for the other device to tell us he opened a socket and is listening on it  
 self.condition.wait(self.mutex)  
 # connecting to that socket  
 sock = FileSendingSocket(self.phone\_ip, self.sending\_port, file, self.key)  
 self.sending\_port +=1  
 sock.start()  
 self.file\_sending\_sockets.append(sock)  
 # telling the other device we connected to the socket and he can open another one  
 self.main\_sending\_socket.send\_massage.emit("connected")  
 time.sleep(1)  
 i=0  
 self.sockets = ["" for f in self.files\_and\_paths.keys()]  
  
 # receiving the files  
 for file in self.files\_and\_paths:  
 # opening a listening socket  
 self.sockets[i] = FileReceivingSocket(self.ip, self.sending\_port, self.files\_and\_paths, self.key)  
 self.sending\_port +=1  
 self.sockets[i].start()  
 self.file\_recving\_sockets.append(self.sockets[i])  
 # telling the other device we opened a socket and he can connect to it  
 self.main\_sending\_socket.send\_massage.emit("socket opened")  
 time.sleep(1)  
 # waiting for the other device to tell us he opened connected to our socket  
 self.condition.wait(self.mutex)  
 i+=1  
   
 # emitting to the main sockets that we finished transferring so they can close   
 self.main\_sending\_socket.done\_signal.emit()  
 self.main\_receiving\_socket.done\_signal.emit()  
  
 # making sure all receiving socket have finished  
 all\_socket\_finished = True  
 while True:  
 for socket in self.sockets:  
 # going through each socket and making sure it's finished  
 if not socket.finished:  
 all\_socket\_finished = False  
 if all\_socket\_finished:  
 break  
 all\_socket\_finished = True  
  
 # updating the GUI that file transfer went well  
 self.main\_window.change\_to\_message\_win("all files received successfully")  
  
 self.mutex.unlock()  
  
  
  
 def finished\_window2(self, files):  
 # this happens when the second window finishes. it updates the files  
 self.files = files  
 self.finished\_window2 = True  
 # if the phone already sent the file than we can move to the next window and if not we wait for him  
 if (self.got\_files):  
 self.main\_window.change\_win()  
 else:  
 self.main\_window.change\_to\_message\_win("waiting for phone")  
 # sending to the phone the files we chose  
 self.main\_sending\_socket.got\_files(self.files)  
  
 def finished\_window3(self, files\_and\_paths):  
 # this happens when the third window is finished  
 # we update the locations of the files the phone sent  
 self.files\_and\_paths = files\_and\_paths  
 # sending that we are ready to send the files  
 self.main\_sending\_socket.ready\_to\_send\_files()  
 self.ready\_for\_the\_files = True  
 # if the phone already sent that he is ready to send files we start sending files  
 # if not we wait for it  
 if self.is\_phone\_ready\_for\_the\_files:  
 self.main\_window.change\_to\_message\_win("transferring files")  
 thread = Thread(target=self.send\_files)  
 thread.start()  
 else:  
 self.main\_window.change\_to\_message\_win("waiting for phone")  
  
 def received\_message(self, message):  
 # this get called when we receive a message during the file sending  
 self.mutex.lock()  
 self.condition.wakeAll()  
 self.mutex.unlock()  
  
def main():  
 app = QApplication(sys.argv)  
 project = Project()  
 project.main\_window.show()  
  
 sys.exit(app.exec\_())  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**הקוד של הטלפון:**

המסך הראשון:

import socket  
from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
import cv2  
  
class Window1(QWidget):  
 # this is window where we display the camera  
 def \_\_init\_\_(self):  
 super(Window1, self).\_\_init\_\_()  
 self.initUI()  
  
 def initUI(self):  
 self.VBL = QVBoxLayout()  
  
 # this is the label that will display the camera  
 self.FeedLabel = QLabel()  
 self.VBL.addWidget(self.FeedLabel)  
  
 # starting the camera  
 self.CameraThread = CameraThread()  
  
 self.CameraThread.start()  
 self.CameraThread.ImageUpdate.connect(self.ImageUpdateSlot)  
 self.setLayout(self.VBL)  
  
  
 def ImageUpdateSlot(self, Image):  
 # this will change the image displayed  
 self.FeedLabel.setPixmap(QPixmap.fromImage(Image))  
  
class CameraThread(QThread):  
 ImageUpdate = pyqtSignal(QImage)  
 got\_data = pyqtSignal(str)  
 def run(self):  
 # this is the object to capture an image  
 Capture = cv2.VideoCapture(0)  
 # this is the object to decode the qr code  
 detector = cv2.QRCodeDetector()  
 while True:  
 # it reads from the camera  
 ret, frame = Capture.read()  
 if ret:  
 # it's converting it to an image  
 Image = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  
 ConvertToQtFormat = QImage(Image.data, Image.shape[1], Image.shape[0], QImage.Format\_RGB888)  
 Pic = ConvertToQtFormat.scaled(1440, 2960, Qt.KeepAspectRatio)  
 # updating the GUI with the new image  
 self.ImageUpdate.emit(Pic)  
 data, bbox, \_ = detector.detectAndDecode(Image)  
 if data:  
 # if it decoded the qr code it emits the data to the project object  
 self.got\_data.emit(str(data))  
 break

המסך השני

from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
import sys  
  
  
class Window2(QWidget):  
 # this is the window where you choose files  
 finished = pyqtSignal(list)  
 def \_\_init\_\_(self):  
 super(Window2, self).\_\_init\_\_()  
 self.i = 0  
 self.setupUi()  
 def setupUi(self):  
 self.files = []  
 self.setObjectName("Form")  
 self.setGeometry(100, 100, 1300, 700)  
  
 # creating the vertical layout and the actual widget  
 self.verticalLayoutWidget = QWidget(self)  
 self.verticalLayoutWidget.setGeometry(QRect(100, 200, 870, 1500))  
 self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")  
  
 self.verticalLayout = QVBoxLayout(self.verticalLayoutWidget)  
 self.verticalLayout.setContentsMargins(0, 0, 0, 0)  
 self.verticalLayout.setObjectName("verticalLayout")  
  
 # creating the button that you press to add a file and adding it to the vertical layout  
 self.pushButton = QPushButton(self.verticalLayoutWidget)  
 self.pushButton.setObjectName("pushButton")  
 self.pushButton.setText("+")  
 self.pushButton.setFont(QFont('Arial', 15))  
 self.pushButton.clicked.connect(self.get\_file\_path)  
 self.verticalLayout.addWidget(self.pushButton)  
  
 # creating a label and adding it to the vertical layout  
 self.title\_label = QLabel(self.verticalLayoutWidget)  
 self.title\_label.setText("files:")  
 self.title\_label.setFont(QFont('Arial', 15))  
 self.verticalLayout.addWidget(self.title\_label)  
  
 # creating the area that you add the files to in the GUI  
 self.scrollArea = QScrollArea(self.verticalLayoutWidget)  
 self.scrollArea.setWidgetResizable(True)  
 self.scrollArea.setObjectName("scrollArea")  
  
 self.scrollAreaWidgetContents = QWidget()  
 self.scrollAreaWidgetContents.setGeometry(QRect(0, 0, 287, 153))  
 self.scrollAreaWidgetContents.setObjectName("scrollAreaWidgetContents")  
  
 layout = QVBoxLayout(self)  
 layout.addStretch()  
  
 self.scrollAreaWidgetContents.setLayout(layout)  
 self.scrollArea.setWidget(self.scrollAreaWidgetContents)  
  
 self.verticalLayout.addWidget(self.scrollArea)  
  
 # creating the button that you press when you're finished choosing files and adding it to the vertical layout  
 self.okButton = QPushButton(self.verticalLayoutWidget)  
 self.okButton.setObjectName("okButton")  
 self.okButton.setText("OK")  
 self.okButton.setFont(QFont('Arial', 15))  
 self.okButton.clicked.connect(self.ok\_button\_clicked)  
 self.verticalLayout.addWidget(self.okButton)  
  
 def add\_file(self, file\_path):  
 # adding the file to the list of files updating the GUI with the file  
 self.files.append(file\_path)  
 self.label = QLabel(file\_path)  
 self.scrollAreaWidgetContents.layout().addWidget(self.label)  
  
 def get\_file\_path(self):  
 # opening the file system and getting the location of the file i choose  
 file\_app = QFileDialog(self)  
 file\_app.fileSelected.connect(self.add\_file)  
 file\_app.setFixedSize(1000, 2000)  
 file\_app.show()  
   
 def ok\_button\_clicked(self):  
 self.finished.emit(self.files)

המסך השלישי:

from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
import sys  
  
class Window3(QWidget):  
 # this is the window where you pick locations for the files the other device chose  
 all\_files\_have\_location = pyqtSignal(dict)  
  
 def \_\_init\_\_(self):  
 super(Window3, self).\_\_init\_\_()  
 self.file\_location\_dict = {}  
 self.setupUi()  
  
 def setupUi(self):  
 self.setObjectName("Form")  
  
 # creating the vertical layout and the actual widget  
 self.verticalLayoutWidget = QWidget(self)  
 self.verticalLayoutWidget.setGeometry(QRect(100, 100, 900, 1500))  
 self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")  
  
 self.verticalLayout = QVBoxLayout(self.verticalLayoutWidget)  
 self.verticalLayout.setContentsMargins(0, 0, 0, 0)  
 self.verticalLayout.setObjectName("verticalLayout")  
  
 # creating a label and adding it to the vertical layout  
 self.label = QLabel(self.verticalLayoutWidget)  
 self.label.setObjectName("label")  
 self.label.setText("Files:")  
 self.label.setFont(QFont('Arial', 15))  
 self.verticalLayout.addWidget(self.label)  
  
 # creating the area where the files that the other device chose will be  
 self.scrollArea = QScrollArea(self.verticalLayoutWidget)  
 self.scrollArea.setWidgetResizable(True)  
 self.scrollArea.setObjectName("scrollArea")  
  
 self.scrollAreaWidgetContents = QWidget()  
 self.scrollAreaWidgetContents.setGeometry(QRect(0, 0, 417, 255))  
 self.scrollAreaWidgetContents.setObjectName("scrollAreaWidgetContents")  
  
 layout = QFormLayout(self.scrollAreaWidgetContents)  
  
 self.scrollAreaWidgetContents.setLayout(layout)  
 self.scrollArea.setWidget(self.scrollAreaWidgetContents)  
 self.verticalLayout.addWidget(self.scrollArea)  
  
 # creating the button that you press when you're finished choosing files and adding it to the vertical layout  
 self.ok\_button = QPushButton(self.verticalLayoutWidget)  
 self.ok\_button.setObjectName("ok\_button")  
 self.ok\_button.setText("OK")  
 self.ok\_button.setFont(QFont('Arial', 15))  
 self.ok\_button.clicked.connect(self.check\_all\_files\_have\_location)  
 self.verticalLayout.addWidget(self.ok\_button)  
  
 def select\_directory(self, file\_name):  
 # creating the object of the file system  
 options = QFileDialog.Options()  
 options |= QFileDialog.ReadOnly  
 file\_dialog = QFileDialog(self, options=options)  
 # setting the object to a mode in which you choose a directory instead of a file  
 file\_dialog.setFileMode(QFileDialog.Directory)  
 file\_dialog.setOption(QFileDialog.ShowDirsOnly, True)  
 file\_dialog.setFixedSize(1000, 2000)  
 file\_dialog.show()  
 try:  
 # it tries to see if you chose a directory or just closed the window  
 # if you chose a directory it would work and if not the try except will catch it  
 if file\_dialog.exec\_():  
 directory = file\_dialog.selectedFiles()[0]  
 self.file\_location\_dict[file\_name] = directory  
 except:  
 pass  
  
 def check\_all\_files\_have\_location(self):  
 files\_have\_location = True  
 # it goes over each file and checking if it has a location or is it a None type  
 for location in self.file\_location\_dict.values():  
 if location == None:  
 files\_have\_location = False  
 if files\_have\_location:  
 # if all files have a location then it emits a dictionary with the files and their location  
 self.all\_files\_have\_location.emit(self.file\_location\_dict)  
  
 def add\_files(self, files):  
 # here you add files to this window  
 for file in files:  
 # for each file it extracts the file name insert it to the dict and adds a label for it in the GUI  
 file = file.split("/")[-1]  
 self.file\_location\_dict[file] = None  
 label = QLabel(file)  
 button = QPushButton()  
 button.clicked.connect(self.create\_select\_directory\_function(label.text()))  
 button.setText("Choose Location")  
 self.scrollAreaWidgetContents.layout().addRow(label, button)  
  
 def create\_select\_directory\_function(self, label\_text):  
 return lambda: self.select\_directory(label\_text)

מסך ההודעות:

from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
import sys  
  
class MessageWindow(QWidget):  
 # this is the message window  
 def \_\_init\_\_(self, message):  
 super(MessageWindow, self).\_\_init\_\_()  
 self.message = message  
 self.setupUI()  
  
 def setupUI(self):  
 # we create the label that holds the message  
 self.message\_label = QLabel(self)  
 self.message\_label.setText(self.message)  
 self.message\_label.setFont(QFont('Arial', 10))  
 self.message\_label.setGeometry(QRect(300, 300, 700, 100))  
  
 def change\_message(self, message):  
 # with this function we can change that message  
 self.message\_label.setText(message)

המסך הראשי:

#  
from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
from message\_win import \*  
from window1\_phone import \*  
from window2\_phone import \*  
from window3\_phone import \*  
import time  
  
  
class MainWindow(QMainWindow):  
 # this is the window that supervises the GUI  
 def \_\_init\_\_(self):  
 super(MainWindow, self).\_\_init\_\_()  
 self.path = "qr.png"  
 self.setGeometry(0, 0, 1050, 2000)  
 self.setWindowTitle("project")  
 self.setupUI()  
  
 def setupUI(self):  
 self.current\_index = 0  
  
 # here i create the windows  
 self.window1 = Window1()  
 self.window2 = Window2()  
 self.window3 = Window3()  
 self.message\_win = MessageWindow("")  
  
 # and add them to the stack  
 self.Stack = QStackedWidget(self)  
 self.Stack.setGeometry(0, 0, 1050, 2000)  
 self.Stack.addWidget(self.window1)  
 self.Stack.addWidget(self.window2)  
 self.Stack.addWidget(self.window3)  
 self.Stack.addWidget(self.message\_win)  
  
 # with the stack i can replace which window is being presented to the user  
  
 self.hbox = QHBoxLayout(self)  
 self.hbox.addWidget(self.Stack)  
  
 self.Stack.setCurrentIndex(self.current\_index)  
  
 def change\_win(self):  
 # this function changes the window to the next one  
 self.current\_index += 1  
 self.Stack.setCurrentIndex(self.current\_index)  
  
 def change\_to\_message\_win(self, message):  
 # this function changes the window to the message window  
 self.message\_win.change\_message(message)  
 self.Stack.setCurrentIndex(3)

קובץ שבוא המחלקות האחראיות על התקשורת:

#  
import time  
from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
import socket  
import pickle  
import os  
import time  
from cryptography.fernet import Fernet  
  
  
  
class MainSendingSocket(QThread):  
 got\_file\_list = pyqtSignal(list)  
 ready\_to\_send = pyqtSignal()  
 send\_massage = pyqtSignal(str)  
 done\_signal = pyqtSignal()  
 exception\_rose = pyqtSignal(str)  
  
  
 def \_\_init\_\_(self, ip, port, key):  
 super(MainSendingSocket, self).\_\_init\_\_()  
 # here i set up all the important information for the connection  
 self.ip = ip  
 self.port = port  
 self.sending\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.encrypting\_object = Fernet(key)  
  
 self.mutex = QMutex()  
 self.condition = QWaitCondition()  
  
 # here i connect the signals of this class to a slot  
 self.got\_file\_list.connect(self.got\_files)  
 self.ready\_to\_send.connect(self.ready\_to\_send\_files)  
 self.send\_massage.connect(self.send)  
 self.done\_signal.connect(self.done)  
 self.done\_condition = False  
 self.files = []  
  
  
 def run(self):  
 self.mutex.lock()  
 # here it connects to the phone and then waits until it has the list of files to send  
 self.connect\_to\_phone()  
 self.condition.wait(self.mutex)  
  
 # it converts the file list to bytes and sends it  
 serialized\_file\_list = pickle.dumps(self.files)  
 try:  
 self.sending\_socket.send(self.encrypting\_object.encrypt(serialized\_file\_list))  
  
 # here it wait again until the computer is ready to start sending the files  
 self.condition.wait(self.mutex)  
 self.sending\_socket.send(self.encrypting\_object.encrypt("ready for files".encode()))  
  
 while True:  
 # here it waits until the project tells it to send a message  
 # the message is that either a socket opened or that a socket connected  
 self.condition.wait(self.mutex)  
 self.sending\_socket.send(self.encrypting\_object.encrypt(self.message.encode()))  
 if self.done\_condition:  
 break  
  
 except Exception as exception:  
 self.exception\_rose.emit(str(exception))  
  
 self.mutex.unlock()  
   
 self.sending\_socket.close()  
  
 def connect\_to\_phone(self):  
 # this function connects to the socket on the phone  
 try:  
 self.sending\_socket.connect((self.ip, self.port))  
 except Exception as exception:  
 self.exception\_rose.emit(str(exception))  
  
 def got\_files(self, files):  
 # when the program will have the files a signal will be emitted to this slot and this will run  
 # this function sets the file list and tells the program to stop waiting and continue  
 self.files = files  
  
 self.mutex.lock()  
 self.condition.wakeAll()  
 self.mutex.unlock()  
  
 def ready\_to\_send\_files(self):  
 self.mutex.lock()  
 self.condition.wakeAll()  
 self.mutex.unlock()  
  
 def send(self, message):  
 self.mutex.lock()  
 self.message = message  
 self.condition.wakeAll()  
 self.mutex.unlock()  
  
 def done(self):  
 self.done\_condition = True  
  
class FileSendingSocket(MainSendingSocket):  
 def \_\_init\_\_(self, ip, port, file\_path, key):  
 super(FileSendingSocket, self).\_\_init\_\_(ip, port, key)  
 self.file\_path = file\_path  
 self.BUFFER\_SIZE = 1024  
  
 def run(self):  
 self.connect\_to\_phone()  
  
 file\_name = self.file\_path.split("/")[-1]  
 try:  
 self.sending\_socket.send(self.encrypting\_object.encrypt(str(file\_name).encode()))  
  
 with open(self.file\_path, "rb") as f:  
 while True:  
 bytes\_read = f.read(1024)  
 if not bytes\_read:  
 # file transmitting is done  
 break  
 # we use sendall to assure transmission in  
 # busy networks  
 encrypted\_bytes = self.encrypting\_object.encrypt(bytes\_read)  
 size = len(encrypted\_bytes)  
 self.sending\_socket.send(str(size).encode())  
 message = self.sending\_socket.recv(1024)  
 self.sending\_socket.send(encrypted\_bytes)  
  
 except Exception as exception:  
 self.exception\_rose.emit(str(exception))  
  
 self.sending\_socket.close()  
  
  
class MainReceivingSocket(QThread):  
 connection\_made = pyqtSignal()  
 got\_file\_list\_from\_phone = pyqtSignal(list)  
 ready\_for\_files = pyqtSignal()  
 receive = pyqtSignal(str) #wrong name  
 done\_signal = pyqtSignal()  
 exception\_rose = pyqtSignal(str)  
  
  
  
 def \_\_init\_\_(self, ip, port, key=""):  
 super(MainReceivingSocket, self).\_\_init\_\_()  
 self.ip = ip  
 self.port = port  
 self.done\_signal.connect(self.done)  
 self.done\_condition = False  
 self.receiving\_socket=None  
 if key != "":  
 self.encrypting\_object = Fernet(key)  
  
 def run(self):  
 self.handle\_connection()  
 try:  
 serialized\_file\_list = self.encrypting\_object.decrypt(self.receiving\_socket.recv(1024))  
 list\_of\_files = pickle.loads(serialized\_file\_list)  
  
 self.got\_file\_list\_from\_phone.emit(list\_of\_files)  
  
 message = self.encrypting\_object.decrypt(self.receiving\_socket.recv(1024)).decode()  
 if message == "ready for files":  
 self.ready\_for\_files.emit()  
  
 while True:  
 message = self.encrypting\_object.decrypt(self.receiving\_socket.recv(1024)).decode()  
 self.receive.emit(message) # add signal connected  
 if self.done\_condition:  
 break  
  
 except Exception as exception:  
 self.exception\_rose.emit(str(exception))  
   
 self.receiving\_socket.close()  
  
 def handle\_connection(self):  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 sock.bind((self.ip, self.port))  
 sock.listen()  
 try:  
 self.receiving\_socket, self.address = sock.accept()  
 except Exception as exception:  
 self.exception\_rose.emit(str(exception))  
  
 sock.close()  
 self.handle\_address()  
  
 def handle\_address(self):  
 self.connection\_made.emit()  
  
 def done(self):  
 self.done\_condition = True  
  
 def add\_encrypting\_object(self, key):  
 self.encrypting\_object = Fernet(key)  
  
class FileReceivingSocket(MainReceivingSocket):  
 def \_\_init\_\_(self, ip, port, files\_and\_paths, key):  
 super(FileReceivingSocket, self).\_\_init\_\_(ip, port, key)  
 self.file = None  
 self.BUFFER\_SIZE = 1464  
 self.files\_and\_paths = files\_and\_paths  
 self.finished = False  
  
 def run(self):  
 self.handle\_connection()  
 error\_line = 0  
 try:  
 file\_name = self.encrypting\_object.decrypt(self.receiving\_socket.recv(self.BUFFER\_SIZE)).decode()  
 location = self.files\_and\_paths[file\_name]  
 time.sleep(1)  
  
 with open(f"{location}/{file\_name}", "wb") as file:  
 while True:  
 error\_line = 0  
 # read 1024 bytes from the socket (receive)  
 size = self.receiving\_socket.recv(1024)  
 self.receiving\_socket.send(size)  
 size = int(size.decode())  
 bytes\_encrypted = self.receiving\_socket.recv(size)  
   
 error\_line += 1  
 bytes\_read = self.encrypting\_object.decrypt(bytes\_encrypted)  
  
 error\_line += 1  
 if not bytes\_read:  
 # nothing is received  
 # file transmitting is done  
 break  
 # write to the file the bytes we just received  
 error\_line += 1  
 file.write(bytes\_read)  
 except Exception as exception:  
 self.exception\_rose.emit(str(exception))  
 self.finished = True  
 self.receiving\_socket.close()  
  
 def handle\_address(self):  
 pass

אובייקט הפרויקט האחראי על הכל:

from PyQt5.QtGui import \*  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
from GUI\_phone import \*  
from Connection\_phone import \*  
import netifaces  
from threading import Thread  
import time  
from cryptography.fernet import Fernet  
  
  
class Project:  
 def \_\_init\_\_(self):  
# Get a list of network interfaces on the device  
 interfaces = netifaces.interfaces()  
 # Loop through the interfaces and find the one that is active and has an IP address  
 for iface in interfaces:  
 addrs = netifaces.ifaddresses(iface)  
 if netifaces.AF\_INET in addrs:  
 self.ip = addrs[netifaces.AF\_INET][0]['addr']  
 if self.ip != '127.0.0.1':  
 break  
 # we set all the important stuff here  
 self.port = 6745  
 self.g\_port = 6746  
 self.main\_window = MainWindow()  
 self.main\_receiving\_socket = MainReceivingSocket(self.ip, self.port)  
 self.got\_files = False  
 self.finished\_window2 = False  
 self.ready\_for\_the\_files = False  
 self.is\_phone\_ready\_for\_the\_files = False  
 self.mutex = QMutex()  
 self.condition = QWaitCondition()  
 self.files\_received = []  
  
 # here we connect all the signals from the sockets and windows  
 self.main\_receiving\_socket.receive.connect(self.received\_message)  
 self.main\_receiving\_socket.connection\_made.connect(self.handle\_connection)  
 self.main\_receiving\_socket.got\_file\_list\_from\_phone.connect(self.handle\_files)  
 self.main\_receiving\_socket.ready\_for\_files.connect(self.computer\_ready\_for\_files)  
  
 self.main\_window.window1.CameraThread.got\_data.connect(self.handle\_data)  
 self.main\_window.window2.finished.connect(self.finished\_window2)  
 self.main\_window.window3.all\_files\_have\_location.connect(self.finished\_window3)  
  
 self.main\_receiving\_socket.exception\_rose.connect(self.exception\_rose)  
  
 self.main\_receiving\_socket.start()  
  
 def exception\_rose(self, error\_message):  
 # this function gets called in case of an exception  
 self.main\_window.change\_to\_message\_win(error\_message)  
  
 def handle\_data(self, data):  
 # scan the qr code and we need to connect the the computer  
 # the data we get is from the qr code  
 data = data.split()  
 self.computer\_ip = data[0]  
 self.computer\_port = int(data[1])  
 self.key = data[2].encode()  
 self.main\_receiving\_socket.add\_encrypting\_object(self.key)  
 # we start the sending socket so it will connect to the computer  
 self.main\_sending\_socket = MainSendingSocket(self.computer\_ip, self.computer\_port, self.key)  
 self.main\_sending\_socket.exception\_rose.connect(self.exception\_rose)  
 self.main\_sending\_socket.start()  
  
 def handle\_connection(self):  
 self.main\_window.change\_win()  
  
 def handle\_files(self, list\_of\_files):  
 # this handles the file list from the computer and adds it to the GUI  
 self.files\_from\_computer = list\_of\_files  
 self.main\_window.window3.add\_files(self.files\_from\_computer)  
 self.got\_files = True  
 if (self.finished\_window2):  
 self.main\_window.change\_win()  
  
 def computer\_ready\_for\_files(self):  
 # this gets called when the computer sends a message that it is read to send the files  
 self.is\_phone\_ready\_for\_the\_files = True  
 if self.ready\_for\_the\_files:  
 # if we are also ready to send the files the file sending starts  
 self.main\_window.change\_to\_message\_win("transferring files")  
 thread = Thread(target=self.send\_files)  
 thread.start()  
 else:  
 pass  
  
 def send\_files(self):  
 self.file\_sending\_sockets = []  
 self.file\_recving\_sockets = []  
 self.mutex.lock()  
 i=0  
 # receiving the files  
 self.sockets = ["" for f in self.files\_and\_paths.keys()]  
 for file in self.files\_and\_paths:  
 time.sleep(1)  
 # opening a listening socket  
 self.sockets[i] = FileReceivingSocket(self.ip, self.g\_port, self.files\_and\_paths, self.key)  
 self.g\_port+=1  
 self.sockets[i].start()  
 self.file\_recving\_sockets.append(self.sockets[i])  
 # telling the other device we opened a socket and he can connect to it  
 self.main\_sending\_socket.send\_massage.emit("socket opened")  
 # waiting for the other device to tell us he opened connected to our socket  
 self.condition.wait(self.mutex)  
 i += 1  
   
 # sending the files  
 for file in self.files:  
 # waiting for the other device to tell us he opened a socket and is listening on it  
 self.condition.wait(self.mutex)  
 # connecting to that socket  
 sock = FileSendingSocket(self.computer\_ip, self.g\_port, file, self.key)  
 self.g\_port+=1  
 sock.start()  
 self.file\_sending\_sockets.append(sock)  
 # telling the other device we connected to the socket and he can open another one  
 self.main\_sending\_socket.send\_massage.emit("connected")  
 time.sleep(1)  
   
 # emitting to the main sockets that we finished transferring so they can close   
 self.main\_sending\_socket.done\_signal.emit()  
 self.main\_receiving\_socket.done\_signal.emit()  
   
  
 # making sure all receiving socket have finished  
 all\_socket\_finished = True  
 while True:  
 for socket in self.sockets:  
 # going through each socket and making sure it's finished  
 if not socket.finished:  
 all\_socket\_finished = False  
 if all\_socket\_finished:  
 break  
 all\_socket\_finished = True  
  
 # updating the GUI that file transfer went well  
 self.main\_window.change\_to\_message\_win("all files received successfully")  
  
 self.mutex.unlock()  
  
 def finished\_window2(self, files):  
 # this happens when the second window finishes. it updates the files  
 self.files = files  
 self.finished\_window2 = True  
 # if the computer already sent the file than we can move to the next window and if not we wait for him  
 if (self.got\_files):  
 self.main\_window.change\_win()  
 else:  
 self.main\_window.change\_to\_message\_win("waiting for computer")  
 # sending to the computer the files we chose  
 self.main\_sending\_socket.got\_files(self.files)  
  
 def finished\_window3(self, files\_and\_paths):  
 # this happens when the third window is finished  
 # we update the locations of the files the computer sent  
 self.files\_and\_paths = files\_and\_paths  
 # sending that we are ready to send the files  
 self.main\_sending\_socket.ready\_to\_send\_files()  
 self.ready\_for\_the\_files = True  
 # if the computer already sent that he is ready to send files we start sending files  
 # if not we wait for it  
 if self.is\_phone\_ready\_for\_the\_files:  
 self.main\_window.change\_to\_message\_win("transferring files")  
 thread = Thread(target=self.send\_files)  
 thread.start()  
 else:  
 self.main\_window.change\_to\_message\_win("waiting for computer")  
  
 def received\_message(self, message):  
 # this get called when we receive a message during the file sending  
 self.mutex.lock()  
 self.condition.wakeAll()  
 self.mutex.unlock()  
  
def main():  
 app = QApplication(sys.argv)  
 project = Project()  
 project.main\_window.show()  
  
 sys.exit(app.exec\_())  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()