<u>נספחים</u>

<u>הקוד של המחשב:</u>

<u>המסך הראשון:</u>

```
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))
```

<u>המסך השני:</u>

```
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("+")
```

```python
        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:

```python
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)
```

<u>מסך ההודעות:</u>

```
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)
```

<u>המסך הראשי:</u>

```
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)
```

<u>קובץ שבוא המחלקות האחראיות על התקשורת:</u>

```
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_lis
t))
```

```
            # 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).enco
de()))

            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
```

<u>אובייקט הפרויקט האחראי על הכל:</u>

```
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_fil
es)

self.main_receiving_socket.ready_for_files.connect(self.phone_ready_for_fil
es)

self.main_window.window2.finished_choosing_files.connect(self.finished_wind
ow2)

self.main_window.window3.all_files_have_location.connect(self.finished_wind
ow3)
        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()
```

```python
    project.main_window.show()

    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```

הקוד של הטלפון:

המסך הראשון:

```python
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)
```

<u>המסך השלישי:</u>

```
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)
```

<u>מסך ההודעות:</u>

```
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)
```

<u>המסך הראשי:</u>

```
#
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)
```

<u>קובץ שבוא המחלקות האחראיות על התקשורת:</u>

```
#
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_lis
t))

            # 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).enco
de()))

            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
```

<u>אובייקט הפרויקט האחראי על הכל:</u>

```
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_fil
es)

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_wind
ow3)


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()
```