

Code Listing

```
# file - Meetings.py - 49 lines
# Meetings.py - Class definition for use with meetings stuff

from GlobalResources import *
from DatabaseInit import MeetingsInfo, UsersInfo

class Meeting:
    def __init__(self, title="[Blank Meeting]", place="[Nowhere]", attendees=["[Demo Person 1]", "[Demo Person 2]"], when="[Sometime]", meeting_id=1):
        #Attendees to be a list of people objects

        self.title = title
        self.place = place
        self.attendees = attendees
        self.when = when
        self.meeting_id = meeting_id

        self.info = {"Title":title, "Location":place, "Attendees":attendees, "ISOTime":when,
"MeetingID":meeting_id}

        self.load_meeting_from_database()

        self._update_info()

    def load_meeting_from_database(self, info=None):
        #provide functionality to get an individual meeting from a database.

        if info == None:
            info = self.info

        dbmeeting = MeetingsInfo(info)

        raw_info = dbmeeting.get_meeting_info()
        self.info = {"Title":raw_info[2], "Location":raw_info[4], "ISOTime":raw_info[3],
"MeetingID":raw_info[0], "OwnerID":raw_info[1]}

    def _update_info(self):
        self.title = self.info["Title"]
        self.place = self.info["Location"]
        self.attendees = [] # this'll be the entry point for the DB query
        self._get_attendees_from_database()
        self.when = self.info["ISOTime"]
        self.meeting_id = self.info["MeetingID"]

    def _get_attendees_from_database(self):
        attendees = MeetingsInfo().get_meeting_attendees(self.meeting_id)

        for a in attendees:
```

```

        attendee_id = a[0]
        # Lookup the username
        username = UsersInfo().get_username_by_uid(attendee_id)
        self.attendees.append(username)
# file - setup.py - 30 lines
# This is the setup script which will install the software package into the
# wherever it's meant to be.

import sys, uuid, random, os
import httpplib

Dependancies = True

# Check that the user has PyQt installed
try:
    import PyQt4.QtCore
except ImportError:
    print("You do not have the correct version of PyQt installed :)")
    Dependancies = False
    # TODO: Automatically download the PyQt installation file for the
    # detected OS

# Check that the user has access to SQLite

try:
    import sqlite3
except ImportError:
    print("You are unable to use databases because you're a spaz")
    Dependancies = False

# The dependencies checks are done, now for the

if Dependancies:
    pass
    # TODO: Implement the PyPacker code to unpack the squashed project code.
    # file - NewResourceDialog.py - 27 lines
from Databaselnit import ResourcesInfo
from PyQt4 import QtCore, QtGui
from GlobalResources import *

class NewResourceDialog(QDialog):
    def __init__(self, user):
        super().__init__()

        self.setWindowTitle("Add a Resource")
        self.layout = QVBoxLayout()

        self.title = QLabel("Add a resource")
        self.title.setFont(GTitleFont)
        self.layout.addWidget(self.title)

        self.name_label = QLabel("Resource Name:")
        self.layout.addWidget(self.name_label)

```

```
self.name_input = QLineEdit()
self.layout.addWidget(self.name_input)

self.cost_input = QLineEdit()
self.layout.addWidget(self.cost_input)

self.current_quantity = QLineEdit()
self.layout.addWidget(self.current_quantity)
self.setLayout(self.layout)
```

```
# file - NewMeetingDialog.py - 119 lines
# New Meeting Dialog
```

```
import re
```

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
```

```
from GlobalResources import *
from DatabaseInit import *
from UsernameLookupDialog import *
```

```
class NewMeetingDialog(QDialog):
    def __init__(self, user = None):
        super().__init__()
        self.user = user
        self.main_layout = QVBoxLayout()

        self.setWindowTitle("Add New Meeting")

        self.title = QLabel("Add New Meeting")
        self.title.setFont(GTitleFont)
        self.main_layout.addWidget(self.title)

        self.meeting_title_label = QLabel("Meeting Title:")
        self.main_layout.addWidget(self.meeting_title_label)
        self.meeting_title_entry = QLineEdit()
        self.main_layout.addWidget(self.meeting_title_entry)

        self.attendees_label = QLabel("Attendees:")
        self.main_layout.addWidget(self.attendees_label)

        self.attendees_container = QWidget()
        self.attendees_layout = QHBoxLayout()

        self.attendees_entry = QLineEdit()
        self.attendees_layout.addWidget(self.attendees_entry)
        # self.attendees_entry.textChanged.connect(self._add_attendees)

        self.username_lookup_button = QPushButton("...")
        self.username_lookup_button.setFixedWidth(30)
        self.username_lookup_button.clicked.connect(self.show_username_lookup)
        self.attendees_layout.addWidget(self.username_lookup_button)
```

```

self.attendees_container.setLayout(self.attendees_layout)
self.main_layout.addWidget(self.attendees_container)
self.attendees_info_label = QLabel("A list of usernames seperated by semicolons")
self.attendees_info_label.setFont(GSmallText)

```

```

self.where_label = QLabel("Where")
self.main_layout.addWidget(self.where_label)
self.where_entry = QLineEdit()
self.main_layout.addWidget(self.where_entry)

```

```

self.when_label = QLabel("When")
self.main_layout.addWidget(self.when_label)
self.when_entry = QDateTimeEdit()
self.when_entry.setMinimumDate(QDate.currentDate())
self.when_entry.setMinimumTime(QTime.currentTime())
self.main_layout.addWidget(self.when_entry)

```

```

self.button_container_widget = QWidget()
self.button_container_layout = QHBoxLayout()

```

```

self.save_button = QPushButton("Save")
self.save_button.clicked.connect(self.add_meeting)
self.button_container_layout.addWidget(self.save_button)
self.cancel_button = QPushButton("Cancel")
self.cancel_button.clicked.connect(self.close)
self.button_container_layout.addWidget(self.cancel_button)

```

```

self.button_container_widget.setLayout(self.button_container_layout)
self.main_layout.addWidget(self.button_container_widget)

```

```

self.setLayout(self.main_layout)

```

```

def add_meeting(self):
    info = {"OwnerID": self.user.id, # This is where to do the username lookup
           "Title":self.meeting_title_entry.text(),
           "ISOTime":self.when_entry.text(),
           "Location":self.where_entry.text()
          }
    meeting = MeetingsInfo(info)
    meeting.add_meeting()
    if self._add_attendees(meeting):
        pass
    self.close()

def _add_attendees(self, meeting):
    valid = True
    raw_attendee_list = self.attendees_entry.text()
    #Parse this into a list of attendees, seperated by eiter semicolons or commas.
    pattern = re.compile("[a-zA-Z]+;?")
    # Iterate through the list of attendees
    print(pattern.findall(raw_attendee_list))

```

```

for string in pattern.findall(raw_attendee_list):

    if string[-1] == ";":
        string = string[0:-1]
    try:
        attendeeID = UsersInfo().get_uid_by_username(string)
    except IndexError:
        print("[WARN] Username '{0}' not recognised".format(string))
        attendeeID = 0
        valid = False
        # Show a warning dialog and prevent the form from completing.
    if attendeeID:
        meeting.add_meeting_attendee(attendeeID)
return valid

```

```

def show_username_lookup(self):
    u = UsernameLookup(self)
    #u.show()
    #u.raise_()
    u.exec_()

```

file - MainScreenGui_DiaryView.py - 143 lines

```

from PyQt4.QtCore import *
from PyQt4.QtGui import *

```

```

from GlobalResources import *

```

```

from MeetingWidget import *
from NewMeetingDialog import *
from Meetings import Meeting
from RespondToPendingRequestsDialog import *
from IndicatorBadge import *

```

```

class DiaryView(QWidget):
    def __init__(self, user = None):
        super().__init__()
        self.user = user
        print("[INFO] Created MainScreenGuiDiaryView")

        self.main_layout = QVBoxLayout()

        self.title = QLabel("Upcoming Meetings & Appointments")
        self.title.setFont(GTitleFont)

        self.main_layout.addWidget(self.title)

        self.middle_widget = QWidget()
        self.middle_layout = QHBoxLayout()

        # Define the left-side stuff
        self.left_side_widget = QWidget()
        self.left_side_layout = QVBoxLayout()

        self.meetings_widget = QWidget()

```

```

self.meetings_layout = QVBoxLayout()

# Start to do a slightly more indepth meetings code
# Get meetings from the database
# Enumerate these meetings
# have an array of meetings objects

# #Get a list of meetings
# self.meetings_list = MeetingsInfo(None).get_meetings_by_owner(user.id)
# print("{0} Meeting(s) found.".format(len(self.meetings_list)))
# self.meetings_widgets = []
# for m in self.meetings_list:
#     self.meetings_widgets.append(MeetingOverview(Meeting(meeting_id=m[0])))
#     self.meetings_layout.addWidget(self.meetings_widgets[-1])

self.meetings_widget.setLayout(self.meetings_layout)
self.meetings_widget.setMinimumSize(300, 700)

self.meetings_container_widget = QScrollArea()
self.meetings_container_widget.setWidget(self.meetings_widget)
self.meetings_container_widget.setVerticalScrollBarPolicy(2)
self.left_side_layout.addWidget(self.meetings_container_widget)

self.left_side_widget.setLayout(self.left_side_layout)
self.middle_layout.addWidget(self.left_side_widget)

# End of left side
# Define the right side stuff

self.right_side_widget = QWidget()
self.right_side_layout = QVBoxLayout()#

# Right side widgets...
self.button_container = QWidget()
self.button_container_layout = QVBoxLayout()

self.add_new_appointment_button = QPushButton("New Appointment")
self.add_new_appointment_button.clicked.connect(self.display_new_meeting_dialog)
self.button_container_layout.addWidget(self.add_new_appointment_button)

# TODO: Add some kind of indicator to show the amount of unread pending
meetings

self.response_container = QWidget()
self.response_layout = QHBoxLayout()

self.respond_to_pending_appointments_button = QPushButton("Respond to Pending
Appointments")

self.respond_to_pending_appointments_button.clicked.connect(self.display_respond_to_meetin
gs_dialog)
self.respond_to_pending_appointments_button.setFixedHeight(30)
self.response_layout.addWidget(self.respond_to_pending_appointments_button)

```

```

        self.pending_number =
Indicator(len(MeetingsInfo().get_outstanding_meetings(self.user.id)))
        self.response_layout.addWidget(self.pending_number)

        self.response_container.setLayout(self.response_layout)
        self.button_container_layout.addWidget(self.response_container)

        self.button_container.setLayout(self.button_container_layout)
        self.right_side_layout.addWidget(self.button_container)

        self.spacer1 = QLabel(" ")
        self.spacer1.setFixedHeight(270)
        self.right_side_layout.addWidget(self.spacer1)

        self.right_side_widget.setLayout(self.right_side_layout)
        self.middle_layout.addWidget(self.right_side_widget)

        self.middle_widget.setLayout(self.middle_layout)
        self.main_layout.addWidget(self.middle_widget)
        self.setLayout(self.main_layout)

# Update the list of meetings
        self._update_meeting_list()

def _update_meeting_list(self):

    # This needs to be re-written so it works.

    # Remove all widgets
    for index in range(self.meetings_layout.count()):
        self.meetings_layout.removeItem(self.meetings_layout.itemAt(index))
    self.meetings_layout.update()
    # Start to do a slightly more indepth meetings code
    # Get meetings from the database
    # Enumerate these meetings
    # have an array of meetings objects

    #Get a list of meetings
    self.meetings_list = MeetingsInfo(None).get_meetings_by_owner(self.user.id)
    print("{0} Meeting(s) found.".format(len(self.meetings_list)))
    self.meetings_widgets = []
    for m in self.meetings_list:
        self.meetings_widgets.append(MeetingOverview(Meeting(meeting_id=m[0])))
        self.meetings_layout.addWidget(self.meetings_widgets[-1])
    self.meetings_layout.update()

self.pending_number.update(len(MeetingsInfo().get_outstanding_meetings(self.user.id)))

def display_new_meeting_dialog(self):

```

```

        new_meeting_dialog = NewMeetingDialog(self.user)
        new_meeting_dialog.show()
        new_meeting_dialog.exec_()
        self._update_meeting_list()

    def display_respond_to_meetings_dialog(self):
        respond_to_meetings_dialog = RespondToPendingMeetingDialog(self.user)
        respond_to_meetings_dialog.exec_()
        self._update_meeting_list()
# file - GlobalResources.py - 16 lines
# Global resources

from PyQt4.QtCore import *
from PyQt4.QtGui import *

from LoginAction import User

GTitleFont = QFont("Segoe UI Light", 24)
#GTitleFont.setHintingPreference(QFont.PreferFullHinting)
GTitleFont.setStyleStrategy(QFont.PreferAntialias)
GBodyFont = QFont("Segoe UI Light", 12)
GBodyFont.setStyleStrategy(QFont.PreferAntialias)
GSmallText = QFont("Segoe UI Light", 10)
GSmallText.setStyleStrategy(QFont.PreferAntialias)

# userinfo = User()
# file - LoginScreenGui.py - 178 lines
import sys

# import Qt dependencies
from PyQt4.QtCore import *
from PyQt4.QtGui import *

from MainScreenGui import *
from GlobalResources import *

from LoginAction import *
from DatabaseInit import *

class PasswordWarningDialog(QDialog):
    def __init__(self, errmsg="", buttonmsg="Dismiss"):
        super().__init__()
        self.setModal(True)
        print("[INFO] Created Password Warning Dialog")

        self.setWindowTitle("Access Denied")

        self.main_layout = QVBoxLayout()

        self.title = QLabel("Access Denied")
        self.title.setFont(GTitleFont)
        self.main_layout.addWidget(self.title)

        self.text = QLabel(errormsg)

```



```

self.text.setFont(GBodyFont)
self.main_layout.addWidget(self.text)

self.dismiss_button = QPushButton(buttonmsg)
self.dismiss_button.clicked.connect(lambda: self.close())
self.main_layout.addWidget(self.dismiss_button)

self.subtext = QLabel("If the problem persists, please contact the system
administrator.")
self.subtext.setFont(GSmallText)
self.main_layout.addWidget(self.subtext)

self.setLayout(self.main_layout)

class LoginWindow(QDialog):
    def __init__(self):
        super().__init__()
        print("[INFO] Created Login window")

        self.setWindowTitle("[CMS] Login")

        self.main_title = QLabel("Welcome.")

        self.main_title.setFont(GTitleFont)

        # Create a labeled username feild:

        self.username_item = QWidget()

        self.username_layout = QHBoxLayout()

        self.username_label = QLabel("Username:")
        self.username_label.setFont(GBodyFont)
        self.username_label.setFixedWidth(150)

        self.username_input = QLineEdit()

        self.username_input.setWhatsThis("Enter your username here:")

        self.username_layout.addWidget(self.username_label)
        self.username_layout.addWidget(self.username_input)

        self.username_item.setLayout(self.username_layout)

        # Create a password feild

        self.password_item = QWidget()

        self.password_layout = QHBoxLayout()

        self.password_label = QLabel("Password:")

```

```

self.password_label.setFont(GBodyFont)
self.password_label.setFixedWidth(150)

self.password_input = QLineEdit()
self.password_input.setEchoMode(QLineEdit.Password)
self.password_input.returnPressed.connect(self._enter_submit)

self.password_input.setWhatsThis("Enter your password here")

self.password_layout.addWidget(self.password_label)
self.password_layout.addWidget(self.password_input)

self.password_item.setLayout(self.password_layout)

self.button_layout = QHBoxLayout()

self.submit_button = QPushButton("Login")
self.submit_button.clicked.connect(self.login_action)
self.submit_button.setDefault(True)

self.help_button = QPushButton("?")
self.help_button.setFixedWidth(30)

self.quit_button = QPushButton("Quit")

self.button_layout.addWidget(self.submit_button)

self.button_layout.addWidget(self.help_button)
self.button_layout.addWidget(self.quit_button)

# Add actions to the buttons within buttons_widget

self.quit_button.clicked.connect(lambda: sys.exit(1))

self.buttons_widget = QWidget()
self.buttons_widget.setFixedWidth(300)
self.buttons_widget.setLayout(self.button_layout)

self.layout = QVBoxLayout()

self.layout.addWidget(self.main_title)
self.layout.addWidget(self.username_item)
self.layout.addWidget(self.password_item)
self.layout.addWidget(self.buttons_widget)

self.setLayout(self.layout)
self.setFont(GBodyFont)

self.setFixedWidth(600)

def _show_error_dialog(self, text="", button="Dismiss"):
    er = PasswordWarningDialog(text, button)
    er.show()
    er.raise_()

```

```

er.exec_()

def login_action(self, e = None):
    try:
        # Get the user's id
        userid = UsersInfo().get_uid_by_username(self.username_input.text())

        # Create a User object with data related to the current userid
        user = User(userid) # User - class defined in LoginAction.py

        # If the hash of the input password does not match the stored hash
        if not user.password_hash_cmp(self.password_input.text()):

            # Show the warning dialog
            self._show_error_dialog("Password not recognised")
            # End the execution of the function.
            return 0

    else:

        # Create and run an instance of the MainScreen GUI.
        self.main_screen = MainScreen(user, self)
        self.main_screen.show()
        self.main_screen.raise_()

        # Hide this window.
        self.hide()

        # If the search for the username does not return any results:
        except IndexError:
            self._show_error_dialog("Username not recognised")

def reset(self):

    # Set both the input fields to ""
    self.password_input.setText("")
    self.username_input.setText("")

    # Event handler for pressing enter key.
    def _enter_submit(self, e = None):

        self.login_action(self)
# file - MainScreenGui_ResourcesView.py - 83 lines
from PyQt4.QtCore import *
from PyQt4.QtGui import *

from GlobalResources import *
from DatabaseInit import ResourcesInfo
from NewResourceDialog import *

class ResourcesView(QWidget):

```

```

def __init__(self, user):

    # `user` - reference to instance created in
    self.user = user
    super().__init__()

    print("[INFO] Created ResourcesView")

    self.main_layout = QVBoxLayout()

    self.title = QLabel("Resources")
    self.title.setFont(GTitleFont)
    self.main_layout.addWidget(self.title)

    self.pane_container = QWidget()
    self.pane_container_layout = QHBoxLayout()

    self.left_pane_container = QWidget()
    self.left_pane_layout = QVBoxLayout()

    self.table = QTableWidgetItem(3, 4)
    self.table.setHorizontalHeaderLabels(["Name", "Cost", "Quantity", "Quantity
Needed"])
    for col in range(4):
        self.table.setColumnWidth(col, 580/4)
    self.table.setFixedSize(601, 400)
    self.left_pane_layout.addWidget(self.table)
    self.update_table()
    # This is where I'll define the tabular view.
    # there also needs to be some code for getting sections of information
    # from the database - otherwise it won't work.

    self.left_pane_container.setLayout(self.left_pane_layout)
    self.pane_container_layout.addWidget(self.left_pane_container)
    self.right_pane_container = QWidget()
    self.right_pane_layout = QVBoxLayout()

    self.add_items_button = QPushButton("Add Resource")
    self.right_pane_layout.addWidget(self.add_items_button)
    self.add_items_button.clicked.connect(self._open_new_resource_dialog)
    self.edit_resource_button = QPushButton("Edit Resource")
    self.right_pane_layout.addWidget(self.edit_resource_button)
    self.view_urgent_requirements_button = QPushButton("View urgent requirements")
    self.right_pane_layout.addWidget(self.view_urgent_requirements_button)

    self.right_pane_container.setLayout(self.right_pane_layout)
    self.pane_container_layout.addWidget(self.right_pane_container)

    self.pane_container.setLayout(self.pane_container_layout)
    self.main_layout.addWidget(self.pane_container)

    self.setLayout(self.main_layout)

```

```

def _open_new_resource_dialog(self):
    dg = NewResourceDialog(self.user)
    dg.show()
    dg.exec_()

def update_table(self):
    self.table.clear()

    # Get information from database

    raw_data = ResourcesInfo().get_all_resources()

    # Table dimentions
    x_total = self.table.columnCount()
    y_total = len(raw_data)
    self.table.setRowCount(y_total)
    self.table.setHorizontalHeaderLabels(["Name", "Cost", "Quantity", "Quantity
Needed"])
    for x in range(x_total):
        for y in range(y_total):
            self.table.setItem(y, x, QTableWidgetItem(str(raw_data[y][x+1])))

# file - compile.py - 87 lines
# PyPackager - a simple utility designed to compile and decompile a set of
# python files.

# Program stages
# Iterate through a python file, searching for import statements
# get the module name from those statements and check if it's in the
# module's directory
# open that file & find repeat the process

# once a list of filenames (&paths) has been collected,
# open the files
# remove whitespace and comments
# (maybe) rename the variables to one or two character names
# write the squashed code to a (binary?) file or database ready for
# extraction

import os, sys
import re as regex

def parse_file(file_data, filenames_set):
    pattern = regex.compile("from +[a-zA-Z\\_]* import +[a-zA-Z\\_]*\\n")
    for string in pattern.findall(file_data):
        filename = string[string.find(" ") + 1: string.find(" ", string.find(" ") + 1)] + ".py"
        filenames_set.add(filename)
        parse_file(open(filename).read(), filenames_set)
        del filename
    pattern = regex.compile("import +[a-zA-Z\\_]*\\n")
    for string in pattern.findall(file_data):
        # find all the instances where the user uses import not from
        # whatever import class
        # This will also include all the imports of builtin classes

```

```

        # so we're going to have to do some error handling! yay
        filename = string[string.find(" ")+1:-1]+".py"
        try:
            open(filename)
            filenames_set.add(filename)
            parse_file(open(filename).read(), filenames_set)
        except FileNotFoundError:
            pass

def squash_file(filename):
    # returns a string of the squashed file
    f = open(filename)
    lines = f.readlines()
    f.close()
    outlines = ['@@@{0}@@@\\n'.format(filename)]
    for l in lines:
        tmp1 = l.strip()
        if tmp1 == "":
            pass
        elif tmp1[0] == "#":
            pass
        else:
            outlines+=l
    return "".join(outlines)

def conjoin_files(filenames_set):
    # returns a string of the conjoined file
    outfiles = []
    for f in filenames_set:
        outfiles.append(squash_file(f))
    return "\\n".join(outfiles)

def unconjoin_files(conjoined_file_data):
    # creates a directory containing the various files that have been squashed
    inlines = conjoined_file_data.split("\\n")
    startline, endline, alternator = 0, 0, False

    outfiles = []
    startlines = []

    for index, line in enumerate(inlines):
        if line[:3] == "@@@" :
            filename = line[3:line.find("@@@", 4)]
            startlines.append(index)

    print("len slines:", len(startlines))
print(squash_file("main.py"))
#Testing stuff
filenames_set = {"main.py"}
f = open("main.py")
parse_file(f.read(), filenames_set)
print(filenames_set)
print(len(filenames_set))
for f in filenames_set:
    print(f)
c = conjoin_files(filenames_set)

```

```

print(c, file=open("allc.py", "w"))
unconjoin_files(c)
# file - main.py - 38 lines
#!/usr/bin/env python3
# import core dependencies
import sys, time, random

# import Qt dependencies
from PyQt4.QtCore import *
from PyQt4.QtGui import *

# import custom classes
from LoginScreenGui import LoginWindow
from Databaselnit import UsersInfo

# Main Program

def main():

    print("[INFO] System Startup")

    print("[INFO] Initiate UsersInfo database table")
    u = UsersInfo()

    application = QApplication(sys.argv)
    login_window = LoginWindow()

    login_window.show()
    login_window.raise_()

    application.exec_()

    print("[INFO] Execution complete")

if __name__ == "__main__":
    # Launch the application.
    main()
# file - MainScreenGui_UserAdminView.py - 85 lines
from PyQt4.QtCore import *
from PyQt4.QtGui import *

from LoginAction import User
from GlobalResources import *

from ChangePasswordDialog import *

class UserOverview(QGroupBox):
    def __init__(self, user):
        self.user = user
        super().__init__()

```

```

self.layout = QVBoxLayout()

self.username_label = QLabel(user.info["Name"])
self.username_label.setFont(GTitleFont)
self.layout.addWidget(self.username_label)
# Maybe add an image to represent the user's privileges

self.priv_label = QLabel("Standard User")
self.priv_label.setFont(GSmallText)
self.layout.addWidget(self.priv_label)

self.control_bar = QWidget()
self.controls_layout = QHBoxLayout()

#Controls:
# self.rename_button = QPushButton("Change Name")
# self.controls_layout.addWidget(self.rename_button)
# self.rename_button.setDisabled(not self.user.permissions["ChangeOwnData"])

self.change_password_button = QPushButton("Change Password")
self.controls_layout.addWidget(self.change_password_button)
self.change_password_button.clicked.connect(self._show_change_password_window)
self.change_password_button.setDisabled(not

self.user.permissions["ChangeOwnData"])
print(not self.user.permissions["ChangeOwnData"])
print(self.user.permissions)

self.control_bar.setLayout(self.controls_layout)
self.control_bar.setFixedWidth(300)
self.control_bar.setFont(GSmallText)
self.layout.addWidget(self.control_bar)

self.setLayout(self.layout)
self.setFixedHeight(160)
self.setTitle("Me")

def _show_change_password_window(self):
    pwdwindow = ChangePasswordDialog(self.user)

    pwdwindow.show()
    pwdwindow.exec_()

class UserAdminView(QWidget):
    def __init__(self, user):
        self.user = user
        super().__init__()

        print("[INFO] Created mainScreenGuiUserAdminView")

        self.master_layout = QVBoxLayout()

        self.this_user_header = UserOverview(self.user)

```



```

        self.master_layout.addWidget(self.this_user_header)

        # Add the admin controls - which will either be hidden or disabled for the users
without
        # the proper privellages

        self.admin_tools = QGroupBox()
        self.admin_tools.setTitle("Administrative Tools") #Set this to change depending on if
the thing is locked or not
        self.admin_tools_layout = QVBoxLayout()

        # Show a scrollable list of the UserOverview(s) for all the registered users, and add
management
        # tools, eg "Delete User", "Add new user"

        # It miggh be a good idea to wait until the card updating system is sorted.
        self.admin_tools.setLayout(self.admin_tools_layout)

        self.master_layout.addWidget(self.admin_tools)

        self.admin_tools.setFont(GBodyFont)

        self.setLayout(self.master_layout)
# file - MainScreenGui.py - 104 lines
# Copyright (c) 2015 Peter East All Rights Reserved.

```

import sys

```

from PyQt4.QtCore import *
from PyQt4.QtGui import *

```

Import custom classes for this project

try:

```

    from MainScreenGui_DiaryView import *
    from MainScreenGui_TaskView import *
    from MainScreenGui_ResourcesView import *
    from MainScreenGui_UserAdminView import *
    from GlobalResources import *

```

except ImportError:

```

    print("[ERROR] Error loading modules")
    sys.exit(-1)

```

Requires seperate widgets for each view in the tabbed layout

class MainScreen(QMainWindow):

```

    def __init__(self, user=None, parent=None):
        self.user = user
        self.parent = parent
        super().__init__()
        print("[INFO] Created MainScreenGui")

```

```

        self.setWindowTitle("[CMS] Main View")

```

```

self.main_layout = QVBoxLayout()

self.central_widget = QWidget()
# Define the topbar
self.topbar = QWidget()

self.topbar_layout = QHBoxLayout()

self.tb_help_button = QPushButton("?")
self.tb_help_button.setFixedWidth(30)

# Create a spacer to keep the thing spaced out
self.tb_spacer = QLabel(" ")
self.tb_spacer.setFixedWidth(600)

self.tb_logout_button = QPushButton("Logout")
self.tb_logout_button.setFixedWidth(100)
self.tb_logout_button.clicked.connect(self._logout)

self.topbar_layout.addWidget(self.tb_help_button)
self.topbar_layout.addWidget(self.tb_spacer)
self.topbar_layout.addWidget(self.tb_logout_button)

self.topbar.setLayout(self.topbar_layout)

# End of the topobar widget
self.main_layout.addWidget(self.topbar)

# Finish defining the topbar
self.view_switcher = QTabWidget()

# Define the views for the view_switcher
# Diary View
if self.user.permissions["Meetings"]:
    self.diary_view = DiaryView(self.user)
    self.view_switcher.addTab(self.diary_view, "Planner")
# Task View
if self.user.permissions["Tasks"]:
    self.task_view = TaskView(self.user)
    self.view_switcher.addTab(self.task_view, "Tasks")

# Resources view
if self.user.permissions["Resources"]:
    self.resources_view = ResourcesView(self.user)
    self.view_switcher.addTab(self.resources_view, "Resources")

# User Admin View
if self.user.permissions["ChangeOwnData"] or self.user.permissions["Admin"]:
    self.user_admin_view = UserAdminView(self.user)
    self.view_switcher.addTab(self.user_admin_view, "User Admin")

```

```

        # finish defining the view switcher

        self.view_switcher.setFont(GBodyFont)

        # End the definitions for the view switcher
        # Add the task view to the window
        self.main_layout.addWidget(self.view_switcher)

        # Fill in the window
        self.central_widget.setLayout(self.main_layout)
        self.setCentralWidget(self.central_widget)

    def _logout(self):
        self.user = None
        self.close()
        self.parent.show()
        self.parent.reset()

# file - Databaselnit.py - 166 lines
# Databaselnit
# main program

#This tidies all of the SQL queries into another namespace.
import SqlDictionary
import random
import hashlib
import sqlite3

def gen_pw_hash(password):
    phash = hashlib.md5()
    phash.update(bytes(password, "UTF-8"))
    return phash.hexdigest()

class Database:
    """This is the general database wrapper that I'll use throughout the system"""
    def __init__(self, child, database_name="cmsdb.db"):
        #print("[INFO] Created database object")

        self.db_name = database_name

    def _connect_and_execute(self, sql="", database_name=None):
        #print(sql)
        if database_name == None:
            database_name = self.db_name

        with sqlite3.connect(self.db_name) as dbcon:
            cursor = dbcon.cursor()
            cursor.execute(sql)
            results = cursor.fetchall()

        #print("[INFO] Executed SQL query \"{0}\".format(sql))
        return results

```

```

class ResourcesInfo(Database):
    def __init__(self):
        super().__init__(self)
        self.create_table()

    def create_table(self):
        self._connect_and_execute(SqlDictionary.CREATE_RESOURCES)

    def get_all_resources(self):
        return self._connect_and_execute(SqlDictionary.GET_ALL_RESOURCES)

    def add_resources(self):
        pass

class UsersInfo(Database):
    def __init__(self, uid = 0):
        super().__init__(self)
        self.create_table()

    # NB: all input MUST be sanitized at this point.
    def create_table(self):
        self._connect_and_execute(SqlDictionary.CREATE_USERS)
        self._create_initial_admin_user()

    def _create_initial_admin_user(self):
        # This is to create an initial administrative user in case something happens to the
        database
        pwd = "".join([chr(random.choice(range(ord('A'), ord('z')))) for c in range(10)])

        new_user_info = {"Name": "ADMIN - TMP", "Username": "default_admin", "Password":
        gen_pw_hash(pwd), "Permissions": 29}

        if len(self.get_all_users("WHERE(Username = 'default_admin')")) == 0:
            print("[INFO] Empty users table detected, adding default user...")
            self.add_user(new_user_info)
            print("[INFO] Default user added,\n\tUsername: 'default_admin'\n\tPassword:
'{0}'".format(pwd))

    def get_all_users(self, condition = ""): # Add a SQL condition? maybe? TODO: refactor
this bit
        return self._connect_and_execute(SqlDictionary.GET_ALL_USERS.format(condition))

    def get_uid_by_username(self, username=""):
        return self._connect_and_execute((SqlDictionary.GET_USER_ID.format(username)))
[0][0]

    def get_username_by_uid(self, uid=None):
        return
self._connect_and_execute(SqlDictionary.GET_USERNAME_BY_UID.format(uid))[0][0]

    def add_user(self, info):

```

```

        #info follows the format {"SQL value":Data value}
        values = "{0}', '{1}', '{2}', {3}".format(info["Name"], info["Username"],
info["Password"], info["Permissions"])

        return self._connect_and_execute(SqlDictionary.ADD_USER.format(values))

def update_user_password(self, password, uid):
    sql = SqlDictionary.UPDATE_PASSWORD.format("{0}".format(password), uid)
    return self._connect_and_execute(sql)

class TasksInfo(Database):

    def __init__(self):
        super().__init__(self)
        self.create_table()

    def create_table(self):
        self._connect_and_execute(SqlDictionary.CREATE_TASKS)
        self._connect_and_execute(SqlDictionary.CREATE_TASKATTENDEE)

    def get_info_by_id(self, task_id):
        sql = SqlDictionary.GET_TASK.format("WHERE (TaskID = {0})".format(task_id))
        raw = self._connect_and_execute(sql)[0]
        return {"TaskID":raw[0], "Title":raw[1], "Description":raw[2], "OwnerID":raw[3],
"Attendees":raw[4]}

    def get_ids_by_owner(self, owner_id):
        sql = SqlDictionary.GET_TASK_ID_LIST.format("WHERE Owner =
{0}".format(owner_id))
        output_ids = []
        for row in self._connect_and_execute(sql):
            output_ids.append(row[0])
        return output_ids

    def add_task(self, info):
        SQL_DATA = "''''{0}', '{1}', {2}, {3}''''".format(info["Title"], info["Description"],
info["OwnerID"], info["Attendees"])
        self._connect_and_execute(SqlDictionary.ADD_TASK.format(SQL_DATA))

class MeetingsInfo(Database):
    def __init__(self, meeting_info = None):
        super().__init__(self)
        self.create_table()
        self.meeting_info = meeting_info
        self.id = None

    def create_table(self):
        self._connect_and_execute(SqlDictionary.CREATE_MEETINGS)
        self._connect_and_execute(SqlDictionary.CREATE_MEETINGS_ATTENEDEES)

    def add_meeting(self):

```

```

SQL_DATA = "{0}, '{1}', '{2}', '{3}'".format(self.meeting_info["OwnerID"],
    self.meeting_info["Title"],
    self.meeting_info["ISOTime"],
    self.meeting_info["Location"])
self._connect_and_execute(SqlDictionary.ADD_MEETING.format(SQL_DATA)) #TODO
Sort out the formatting.
self.id = self._connect_and_execute("SELECT Max(MeetingID) FROM Meetings;")[0][0]

def get_meeting_info(self):
    sql_condition = "WHERE (MeetingID = {0})".format(self.meeting_info['MeetingID'])
    q = SqlDictionary.GET_MEETING.format(sql_condition)
    results = self._connect_and_execute(q)
    return results[0]

def add_meeting_attendee(self, user_id):
    sql_values = "{0}, {1}, 0".format(self.id, user_id)
    return
self._connect_and_execute(SqlDictionary.ADD_MEETING_ATTENDEE.format(sql_values))

def get_meetings_by_owner(self, OwnerID):
    sql_condition = "WHERE (OwnerID = {0})".format(OwnerID)
    q = SqlDictionary.GET_MEETING_ID_LIST.format(sql_condition)
    return self._connect_and_execute(q)

def get_outstanding_meetings(self, OwnerID):
    results =
self._connect_and_execute(SqlDictionary.GET_OUTSTANDING_MEETINGS_TO_BE_ATTENDED.for
mat(OwnerID))
    return results

def get_meeting_attendees(self, MeetingID):
    return
self._connect_and_execute(SqlDictionary.GET_MEETING_ATTENDEES.format(MeetingID))

def respond_to_attendance_request(self, attending, MeetingID, UserID):
    if attending:
        self._connect_and_execute(SqlDictionary.ACCEPT_MEETING.format("UserID =
{0} AND MeetingID = {1}".format(UserID, MeetingID)))
    else:
        self._connect_and_execute(SqlDictionary.REJECT_MEETING.format("UserID =
{0} AND MeetingID = {1}".format(UserID, MeetingID)))
# file - IndicatorBadge.py - 40 lines
# Indicator Badge - a PyQt widget to display a numerical indicator Badge

from PyQt4.QtCore import *
from PyQt4.QtGui import *

from GlobalResources import *

class Indicator(QFrame):
    def __init__(self, value = 0):
        super().__init__()
        self.value = value

```

```

        self.main_layout = QVBoxLayout()

        self.value_display = QLabel(str(self.value))

        self.main_layout.addWidget(self.value_display)

        self.setFixedHeight(30)
        #self.setFixedWidth(30)
        self.colour = QColor(255, 61, 0)

        self.setStyleSheet("QFrame {background-color: %s; border-radius: 15; background-
clip: margin;} " % self.colour.name())

        self.text_colour = QColor(0xFF, 0xFF, 0xFF)

        self.value_display.setStyleSheet("QLabel {color: %s }" % self.text_colour.name())

        self.setFrameStyle(QFrame.StyledPanel + QFrame.Plain)
        self.setLayout(self.main_layout)

    def update(self, new_value):

        self.value = new_value

        self.value_display.setText(str(self.value))

        # do some cool stuff here :-))
    def getValue(self):
        return int(self.value)
# file - MainScreenGui_TaskView.py - 92 lines
from PyQt4.QtCore import *
from PyQt4.QtGui import *

from NewTaskDialog import *

from GlobalResources import *
from DatabaseInit import *
from Tasks import *

class TaskView(QWidget):
    def __init__(self, user = None):
        super().__init__()
        self.user = user

        self.main_layout = QVBoxLayout()

        self.title = QLabel("Outstanding Tasks")
        self.title.setFont(GTitleFont)

        self.main_layout.addWidget(self.title)

        # Set up the two-pane view

```

```

self.pane_container = QWidget()
self.pane_layout = QHBoxLayout()

# Left side
self.left_widget = QWidget()
self.left_layout = QVBoxLayout()
##Task list

self.task_list_view = QListView()

self.update_task_list()

self.left_layout.addWidget(self.task_list_view)
self.left_widget.setLayout(self.left_layout)

self.pane_layout.addWidget(self.left_widget)
# End Left pane
# Start right pane

self.right_widget = QWidget()
self.right_layout = QVBoxLayout()

self.add_new_task_button = QPushButton("Add new Task")
self.add_new_task_button.setFixedWidth(150)
self.add_new_task_button.clicked.connect(self.display_new_task_dialog)
self.right_layout.addWidget(self.add_new_task_button)

self.spacer = QLabel(" ")
self.spacer.setFixedHeight(300)
self.right_layout.addWidget(self.spacer)

self.right_widget.setLayout(self.right_layout)
self.pane_layout.addWidget(self.right_widget)
# End right pane

self.pane_container.setLayout(self.pane_layout)
self.main_layout.addWidget(self.pane_container)

self.setLayout(self.main_layout)

def display_new_task_dialog(self):
    new_task_dialog = NewTaskDialog(self.user)

    new_task_dialog.exec_()
    self.update_task_list()

def update_task_list(self):

    # Make it so that when the item is clicked, and the check box is

    print("[INFO] Updating task list... ")

```



```

# Add some data from the database
data = QStandardItemModel()

#Database fetch example
ids = TasksInfo().get_ids_by_owner(self.user.id)
print("[INFO] {0} Tasks found for user id: {0}".format(len(ids), self.user.id))
self.tasks = []
for taskID in ids:
    self.tasks.append(Task(databaseid=taskID))

    tmp = QStandardItem(self.tasks[-1].text)
    tmp.setCheckable(True)
    data.appendRow(tmp)
self.task_list_view.setModel(data)
print("[INFO] Task view update successful")

```

file - Tasks.py - 25 lines

This one's for taking the data out of the database

```

from DatabaseInit import TasksInfo

```

```

class Task:

```

```

    def __init__(self, title="", subtitle="", priority=1, databaseid=None):
        self.text = ""
        if databaseid == None:
            self.title = title
            self.priority = priority
            self.subtitle = subtitle
            self.text = self.title + "\n " + self.subtitle
        else:
            self.info = TasksInfo().get_info_by_id(databaseid)
            self.text = self.info["Title"] + "\n " + self.info["Description"]
            # TODO: The tasks are gonna need a due date and priority in the database so
            # the listings can be ordered by priority.
            pass

```

that

```

    def load_task_from_database(self, taskid):
        pass

    def complete(self):
        self.title+= " [Done]"
        self.priority = -1

```

file - ChangePasswordDialog.py - 135 lines

The dialog for changing a user's password

```

from PyQt4 import QtCore
from PyQt4 import QtGui

```

```

from DatabaseInit import UsersInfo
from GlobalResources import *

```

```

class _PWErrorDialog(QDialog): #Blank error dialog
    def __init__(self):
        super().__init__()

```

```
self.setWindowTitle("Error")
self.layout = QVBoxLayout()
self.title = QLabel("Password Error")
self.title.setFont(GTitleFont)
self.layout.addWidget(self.title)
self.label = QLabel("")
self.layout.addWidget(self.label)

self.setLayout(self.layout)
```

```
class _PWMismatchErrorDialog(_PWErrorDialog):
    def __init__(self):
        super().__init__()
        self.label.setText("New passwords do not match, try again")
```

```
class _PWCurrentIncorrectError(_PWErrorDialog):
    def __init__(self):
        super().__init__()
        self.label.setText("You password is incorrect")
```

```
class ChangePasswordDialog(QDialog):
    def __init__(self, user):
        super().__init__()

        INPUT_WIDTH = 400
        LABEL_WIDTH = 200

        self.user = user

        self.setWindowTitle("Change your password")

        self.layout = QVBoxLayout()

        self.title = QLabel("Change your password")
        self.title.setFont(GTitleFont)
        self.layout.addWidget(self.title)

        self.pw_current_container = QWidget()
        self.pw_current_entry = QLineEdit()
        self.pw_current_entry.setFixedWidth(INPUT_WIDTH)
        self.pw_current_entry_label = QLabel("Enter your current password:")
        self.pw_current_entry_label.setFixedWidth(LABEL_WIDTH)
        self.pw_current_entry.setEchoMode(QLineEdit.Password)

        self.pw_container_layout = QHBoxLayout()
        self.pw_container_layout.addWidget(self.pw_current_entry_label)
        self.pw_container_layout.addWidget(self.pw_current_entry)

        self.pw_current_container.setLayout(self.pw_container_layout)
        self.layout.addWidget(self.pw_current_container)

        self.new_pw_container = QWidget()
        self.new_pw_label = QLabel("Enter your new password:")
```

```

self.new_pw_label.setFixedWidth(LABEL_WIDTH)
self.new_pw_entry = QLineEdit()
self.new_pw_entry.setFixedWidth(INPUT_WIDTH)
self.new_pw_entry.setEchoMode(QLineEdit.Password)

self.new_pw_layout = QHBoxLayout()
self.new_pw_layout.addWidget(self.new_pw_label)
self.new_pw_layout.addWidget(self.new_pw_entry)

self.new_pw_container.setLayout(self.new_pw_layout)
self.layout.addWidget(self.new_pw_container)

self.confirm_pw_container = QWidget()
self.confirm_pw_label = QLabel("Confirm your new password:")
self.confirm_pw_label.setFixedWidth(LABEL_WIDTH)
self.confirm_pw_entry = QLineEdit()
self.confirm_pw_entry.setFixedWidth(INPUT_WIDTH)
self.confirm_pw_entry.setEchoMode(QLineEdit.Password)
self.confirm_pw_entry.returnPressed.connect(self._pwchange_action)

self.confirm_pw_layout = QHBoxLayout()
self.confirm_pw_layout.addWidget(self.confirm_pw_label)
self.confirm_pw_layout.addWidget(self.confirm_pw_entry)

self.confirm_pw_container.setLayout(self.confirm_pw_layout)
self.layout.addWidget(self.confirm_pw_container)

self.button_container = QWidget()
self.button_container_layout = QHBoxLayout()
self.accept_button = QPushButton("Accept")
self.accept_button.clicked.connect(self._pwchange_action)
self.button_container_layout.addWidget(self.accept_button)

self.reject_button = QPushButton("Cancel")
self.reject_button.clicked.connect(lambda: self.close())
self.button_container_layout.addWidget(self.reject_button)

self.button_container.setLayout(self.button_container_layout)
self.layout.addWidget(self.button_container)

self.setFont(GBodyFont)

self.setLayout(self.layout)

```

```

def _pwchange_action(self):
    # Generate password hash
    # Get password hash from the database
    # Compare the two
    # if correct, update the database entry for the password
    # should be easy

    # Check that the pw entry and the pwconfirm are equal

    passwords_the_same = self.confirm_pw_entry.text() == self.new_pw_entry.text()

```

```

        current_pw_correct = self.user.password_hash_cmp(self.pw_current_entry.text())

        if passwords_the_same and current_pw_correct:
            UsersInfo().update_user_password(User.gen_pw_hash(None,
self.new_pw_entry.text()), self.user.id)
            self.close()
            # Proceed to change the password
            print("[DEBUG] Passwords changed successfully
(ChangePasswordDialog.py:126)")
            elif not passwords_the_same and current_pw_correct:
                e = _PWMismatchErrorDialog()
                e.show()
                e.exec_()
                # Display a messagebox explaining wtf is wrong.
            else:
                e = _PWCurrentIncorrectError()
                e.show()
                e.exec_()

# file - MeetingWidget.py - 103 lines
# Meeting overview widget

from PyQt4.QtCore import *
from PyQt4.QtGui import *

from GlobalResources import *
from Meetings import Meeting
from DatabaseInit import UsersInfo, MeetingsInfo
#from Meetings import *

class MeetingOverview(QFrame):
    def __init__(self, meeting):
        """
        Should take a Meeting object as a parameter rather than passing all of the individual
parameters in.
        """
        super().__init__()
        self.meeting = meeting
        self.layout = QVBoxLayout()

        self.setFrameStyle(QFrame.StyledPanel + QFrame.Sunken)

        # Define the widgets

        self.title = QLabel(meeting.title)
        self.title.setFont(GTitleFont)
        self.layout.addWidget(self.title)

        self.place_title = QLabel("At: " + meeting.place)
        self.layout.addWidget(self.place_title)

        self.when_title = QLabel(meeting.when)
        self.layout.addWidget(self.when_title)

        self.owner_label = QLabel()

```

```

self.layout.addWidget(self.owner_label)

self.attendees_title = QLabel("Attendees:")
self.layout.addWidget(self.attendees_title)

self.attendees_list = []
for index, person in enumerate(meeting.attendees):
    self.attendees_title.setText(self.attendees_title.text()+"\n"+person)

#self.setMinimumHeight(100)
self.buttons_widget = QWidget()
self.buttons_layout = QHBoxLayout()

self.delete_button = QPushButton("Delete")
self.buttons_layout.addWidget(self.delete_button)

self.edit_button = QPushButton("Edit")
self.edit_button.setFixedWidth(150)
self.buttons_layout.addWidget(self.edit_button)

self.buttons_widget.setLayout(self.buttons_layout)
self.layout.addWidget(self.buttons_widget)
self.setLayout(self.layout)
self.setMinimumSize(400, 200)

def _edit_button_action(self):
    #Create a NewMeetingDialog with the information from this meeting
    pass

```

```

class PendingMeetingOverview(MeetingOverview):
    def __init__(self, meeting, user):
        super().__init__(meeting)
        self.meeting = meeting
        self.user = user
        # Get the owner's name
        owner_name = UsersInfo().get_username_by_uid(meeting.info["OwnerID"])
        self.owner_label.setText("From: {0}".format(owner_name))

        self.edit_button.setText("Respond - Confirm")
        self.edit_button.clicked.connect(self._accept_meeting)

        self.deny_button = QPushButton("Respond - Deny")
        self.deny_button.clicked.connect(self._reject_meeting)
        self.deny_button.setFixedWidth(150)
        self.buttons_layout.addWidget(self.deny_button)

    def _accept_meeting(self):
        MeetingsInfo().respond_to_attendance_request(True, self.meeting.meeting_id,
self.user.id)
        print("[INFO] Meeting accepted")

    def _reject_meeting(self):
        MeetingsInfo().respond_to_attendance_request(False, self.meeting.meeting_id,

```

```

self.user.id)
    print("[INFO] Meeting Rejected")

class PleaseSelectMeetingPlaceholder(QFrame):
    def __init__(self, empty = False):
        super().__init__()
        self.layout = QHBoxLayout()

        if not empty:
            self.label = QLabel("Please select\ na meeting")
        else:
            self.label = QLabel("You have no\ nnew meetings")
            self.label.setFont(GTitleFont)
            self.layout.addWidget(self.label)
            self.setLayout(self.layout)
            self.setFixedWidth(300)

# file - RespondToPendingRequestsDialog.py - 80 lines
# Respond to meeting requests

from PyQt4.QtCore import *
from PyQt4.QtGui import *

from GlobalResources import *
from Meetings import Meeting
from MeetingWidget import *
from DatabaseInit import MeetingsInfo

class RespondToPendingMeetingDialog(QDialog):
    def __init__(self, user):
        self.user = user
        super().__init__()

        #Respond to meeting request,
        # two panes, one for a list of pending meetings,
        # another for a tools and controls to deal with those
        # pending meetings
        self.setWindowTitle("Respond to Pending Requests")
        self.main_layout = QVBoxLayout()

        self.title = QLabel("Respond to Pending Requests")
        self.title.setFont(GTitleFont)
        self.main_layout.addWidget(self.title)

        self.pane_container = QWidget()
        self.pane_container_layout = QHBoxLayout()
        self.left_pane = QWidget()
        self.left_pane_layout = QVBoxLayout()

        self.meetings_list_view = QListView()
        self.meetings_list_view.clicked.connect(self._switch_right_stack)

        # End of 'following code'

```

```

self.left_pane_layout.addWidget(self.meetings_list_view)

self.left_pane.setLayout(self.left_pane_layout)

self.right_pane = QWidget()
self.right_pane_layout = QStackedLayout()

self.meeting_view =
PleaseSelectMeetingPlaceholder(len(MeetingsInfo({}).get_meetings_by_owner(self.user.id))==
0)

self.right_pane_layout.addWidget(self.meeting_view)
self.update_pending_meeting_list()

self.right_pane.setLayout(self.right_pane_layout)
self.pane_container_layout.addWidget(self.left_pane)
self.pane_container_layout.addWidget(self.right_pane)
self.pane_container.setLayout(self.pane_container_layout)
self.main_layout.addWidget(self.pane_container)

self.setLayout(self.main_layout)

def update_pending_meeting_list(self):
    print("[INFO] Updating list of pending appointments")

    self.data = QStandardItemModel()

    ids = MeetingsInfo().get_outstanding_meetings(self.user.id)
    print("[INFO] {0} meetings found for user id: {1}".format(len(ids), self.user.id))

    self.meetings = []
    for meetingID in ids:
        self.meetings.append(Meeting(meeting_id=meetingID[0]))
        meeting = self.meetings[-1]
        self.right_pane_layout.addWidget(PendingMeetingOverview(meeting, self.user))
        tmp =
QStandardItem(meeting.title+"\n"+meeting.place+"\n"+meeting.when+"\n")
        tmp.setCheckable(False)
        self.data.appendRow(tmp)
    self.meetings_list_view.setModel(self.data)

    # Use a QStackedLayout to have the stack of meeting widgets

def _switch_right_stack(self):
    index = self.meetings_list_view.selectedIndexes()[0].row()
    self.right_pane_layout.setCurrentIndex(index+1)
# file - LoginAction.py - 81 lines
#Login action:
# The code and database actions performed when the user attempts to log into the system.

import hashlib

from Databaselnit import *
```

import SqlDictionary

class User:

```
def __init__(self, uid=0):
    self.info = {} # info to be retrieved from the database
    self.permissions = {}
    self.user_id = uid

    self.dbinterface = UsersInfo()

    self.update_user_info()

def gen_pw_hash(self, password):
    # Create a md5 hash of the password
    phash = hashlib.md5()
    # (Encode the password - the python md5 implementation only accepts binary
data.)
    phash.update(bytes(password, "UTF-8"))
    # return a hexadecimal representation of the md5 hash.
    return phash.hexdigest()

def password_hash_cmp(self, password_input):
    currenthash = self.info["Password"]
    return currenthash == self.gen_pw_hash(password_input)

def add_user(self, info=None):
    # If there's no info input, use the existing info for this instance of the
    # class.
    if info == None:
        info = self.info

    # Use the database module to add the user's info to the database.
    UsersInfo().add_user(info)

def update_user_info(self):
    # get the first item in the list of users which have the UserID of
    # `self.user_id` (the length of the list should be 1)
    raw_info = self.dbinterface.get_all_users("WHERE(UserID =
{0})".format(self.user_id))[0]

    # raw_info follows the format [id, Name, Username, Password, Permissions]
    # put the individual parts of the raw data into a python dictionary
    self.info["UserID"] = self.user_id
    self.info["Name"] = raw_info[1]
    self.info["Username"] = raw_info[2]
    self.info["Password"] = raw_info[3]
    self.info["Permissions"] = raw_info[4]

    # Generate the permissions array for this user.
    self.gen_permissions()
    self.id = self.info["UserID"]

def gen_permissions(self):
```



```
# Get the denary integer value for the user's permissions
perm = self.info["Permissions"]

# Create a list of the default values for the user's permissions
blist = [False, False, False, False, False]

# For each item in a list of the individual binary digits
# The python `bin` function outputs a string in the format '0b10101'
# which is why we need to get rid of the first two characters
for index, digit in enumerate(bin(int(perm))[2:]):
    # set each item in the b(inary)list as a python Bool so it can easily
    # be used in selection statements
    blist[index] = (bool(int(digit)))
permissions = {}
permissions["Meetings"] = blist[0]
permissions["Tasks"] = blist[1]
permissions["Resources"] = blist[2]
permissions["ChangeOwnData"] = blist[3]
permissions["Admin"] = blist[4]

# Overwrite the existing permissions number with a dictionary.
self.permissions = permissions
return permissions

# file - SqlDictionary.py - 110 lines
# sql dictionary
# This file will contain all of the SQL references used throughout the system, with string
# formatting already
# added

#initialisation scripts

CREATE_USERS = """CREATE TABLE IF NOT EXISTS Users
    (UserID INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT,
    Username TEXT,
    Password TEXT,
    Permissions INTEGER
    );

"""

#Permissions: Like unix file permissions but using denary instead of octal
# and there are 5 bits rather than several.
# eg 0b11010 - will give the user permission to use the meetings, tasks and user admin, and
# not resources management or privac.

# time to design databases - NOW!

CREATE_MEETINGS = """CREATE TABLE IF NOT EXISTS Meetings
    (MeetingID INTEGER PRIMARY KEY AUTOINCREMENT,
    OwnerID INTEGER,
    Title TEXT,
    ISOTime TEXT,
    Location TEXT
    );
```

```
""""
```

```
CREATE_MEETINGS_ATTENEDEES = """CREATE TABLE IF NOT EXISTS MeetingAttendee(  
    MeetingID INTEGER,  
    UserID INTEGER,  
    Confirmed BOOLEAN  
);"""
```

```
CREATE_TASKS = """CREATE TABLE IF NOT EXISTS Tasks  
    (TaskID INTEGER PRIMARY KEY AUTOINCREMENT,  
    Title TEXT,  
    Description TEXT,  
    Owner INTEGER,  
    Attendees INTEGER  
);"""
```

```
CREATE_TASKATTENDEE = """CREATE TABLE IF NOT EXISTS TaskAttendee  
    (  
    TaskId INTEGER,  
    UserId INTEGER  
    );
```

```
""""
```

```
CREATE_RESOURCES = """CREATE TABLE IF NOT EXISTS Resources  
    (ResourceID INTEGER PRIMARY KEY AUTOINCREMENT,  
    Name TEXT,  
    Cost INTEGER,  
    QuantityAvailable INTEGER,  
    QuantityRequired INTEGER  
);  
"""
```

Users

```
GET_ALL_USERS = """ SELECT * FROM Users {0}; """
```

```
GET_USER_ID = """ SELECT UserID FROM Users WHERE (Username = '{0}');"""
```

```
ADD_USER = """INSERT INTO Users(Name, Username, Password, Permissions) VALUES({0});"""
```

```
UPDATE_PASSWORD = """UPDATE Users SET Password = {0} WHERE UserID = {1};"""
```

Meetings

```
ADD_MEETING = """INSERT INTO Meetings(OwnerID, Title, ISOTime, Location) VALUES({0})"""
```

```
GET_MEETING = """SELECT * FROM Meetings {0};"""
```

```
GET_OUTSTANDING_MEETINGS_TO_BE_ATTENDED = """SELECT * FROM MeetingAttendee  
WHERE (UserID = {0} AND Confirmed = 0);"""
```

```
GET_MEETING_ID_LIST = """SELECT MeetingID FROM Meetings {0};"""
```

```
ADD_MEETING_ATTENDEE = """INSERT INTO MeetingAttendee(MeetingID, UserID, Confirmed)
```

```
VALUES({0})"""
```

```
GET_MEETING_ATTENDEES = """SELECT UserID FROM MeetingAttendee WHERE (MeetingID = {0})"""
```

```
ACCEPT_MEETING = """UPDATE MeetingAttendee SET Confirmed = 1 WHERE {0}"""
```

```
REJECT_MEETING = """UPDATE MeetingAttendee SET Confirmed = 0 WHERE {0}"""
```

```
DELETE_MEETING = """DELETE FROM MeetingAttendee WHERE {0}"""
```

```
# Tasks
```

```
GET_TASK = """SELECT * FROM Tasks {0};"""
```

```
GET_TASK_ID_LIST = "SELECT TaskID FROM Tasks {0};"""
```

```
ADD_TASK = """INSERT INTO Tasks(Title, Description, Owner, Attendees) VALUES({0});"""
```

```
GET_USERNAME_BY_UID = """SELECT Name FROM Users WHERE(UserID = {0})"""
```

```
# Resources
```

```
CREATE_RESOURCES = """CREATE TABLE IF NOT EXISTS Resources  
    (ResourceID INTEGER PRIMARY KEY AUTOINCREMENT,  
    ResourceName TEXT,  
    ResourceCost INTEGER,  
    ResourceQuantity INTEGER,  
    ResourceRequiredQuantity INTEGER);  
"""
```

```
GET_ALL_RESOURCES = """SELECT * FROM Resources;"""
```

```
# file - NewTaskDialog.py - 88 lines
```

```
# NewTaskDialogGui
```

```
from PyQt4.QtCore import *
```

```
from PyQt4.QtGui import *
```

```
from GlobalResources import *
```

```
from Tasks import *
```

```
from Databaselnit import TasksInfo
```

```
class NewTaskDialog(QDialog):
```

```
    def __init__(self, user):
```

```
        self.user = user
```

```
        super().__init__()
```

```
        self.main_layout = QVBoxLayout()
```

```
        self.setWindowTitle("Add new task")
```

```
        self.smalltitle = QLabel("Create new task")
```

```
        self.smalltitle.setFont(GBodyFont)
```

```
        self.title = QLabel("New Task")
```

```
        self.title.setFont(GTitleFont)
```

```

self.main_layout.addWidget(self.smalltitle)
self.main_layout.addWidget(self.title)

self.title_entry_label = QLabel("Title:")
self.main_layout.addWidget(self.title_entry_label)
self.title_entry = QLineEdit("")
self.title_entry.textChanged.connect(self.update_window_title)
self.main_layout.addWidget(self.title_entry)

#self.people_entry_label = QLabel("With whom:")
#self.main_layout.addWidget(self.people_entry_label)

#self.people_entry = QLineEdit()
#self.people_entry.setPlaceholderText("Type usernames here")
#self.people_entry.textChanged.connect(self.check_names)
#self.main_layout.addWidget(self.people_entry)

self.description_entry_label = QLabel("Description:")
self.main_layout.addWidget(self.description_entry_label)

self.description_entry = QLineEdit()
self.main_layout.addWidget(self.description_entry)

self.submit_button = QPushButton("Submit")
self.submit_button.clicked.connect(self.submit)
self.main_layout.addWidget(self.submit_button)

self.setLayout(self.main_layout)
self.setFont(GBodyFont)

def update_window_title(self):
    newtext = self.title_entry.text()[0:45]
    if len(newtext) == 45:
        newtext += "..."
    if newtext == "":
        self.title.setText("New Task")
    else:
        self.title.setText(newtext)
    self.title_entry.setText(newtext)
    self.setWindowTitle(newtext)

    self.title_entry.setFixedWidth(self.title.width())

def check_names(self):
    # Add some name checking functionality
    pass

def validate(self):
    valid = True
    valid *= not (self.title_entry.text().strip() == "")
    # valid *= not (self.people_entry.text().strip() == "")
    valid *= not (self.description_entry.text().strip() == "")

```

```

        return valid

    def submit(self):
        if self.validate():
            info = {"Title":self.title_entry.text(), "Description":self.description_entry.text(),
"OwnerID":self.user.id, "Attendees":""}
            TasksInfo().add_task(info)
            self.close()
        else:
            self.title_entry.setPlaceholderText("Required")
            #self.people_entry.setPlaceholderText("Required")
            self.description_entry.setPlaceholderText("Required")
# file - UsernameLookupDialog.py - 35 lines
from PyQt4.QtCore import *
from PyQt4.QtGui import *

from GlobalResources import *

from DatabaseInit import *

class UsernameLookup(QDialog):
    def __init__(self, parent):
        super().__init__()
        self.setWindowTitle("Select a user")

        self.raise_()
        self.parent = parent
        self.layout = QVBoxLayout()
        self.list = QListView()
        self.list.clicked.connect(self._select_user)
        self.users = UsersInfo().get_all_users()

        data = QStandardItemModel()

        for user in self.users:
            data.appendRow(QStandardItem(user[1]))

        self.list.setModel(data)

        self.layout.addWidget(self.list)
        self.setLayout(self.layout)

        # Get a list of names,
        # Onclick pass names down to the parent.
    def _select_user(self):
        user = self.users[self.list.selectedIndex()[0].row()]
        self.parent.attendees_entry.setText(self.parent.attendees_entry.text()+"",
{0}".format(user[2]))
        self.close()

```