Criterion C: Development

Technologies

All the code is written in Python 3.11 with additional libraries such as *QT, sys*, *Pandas*, *copy*. I use: the *copy* library to copy predefined arrays that I want to keep vital, the *Pandas* library to collect data from Excel, PySide6 (*QT*), to create User Interface, and the *sys* library to open and close the application's Iser Interface. Pandas, QT, and sys are the results of planning described in Criterion B, while during coding, I did not want to rewrite once again the same Pandas query. Thus, I used a copy library and no other development tools were used.

Program files structure

Modular Programming presented on the diagram:

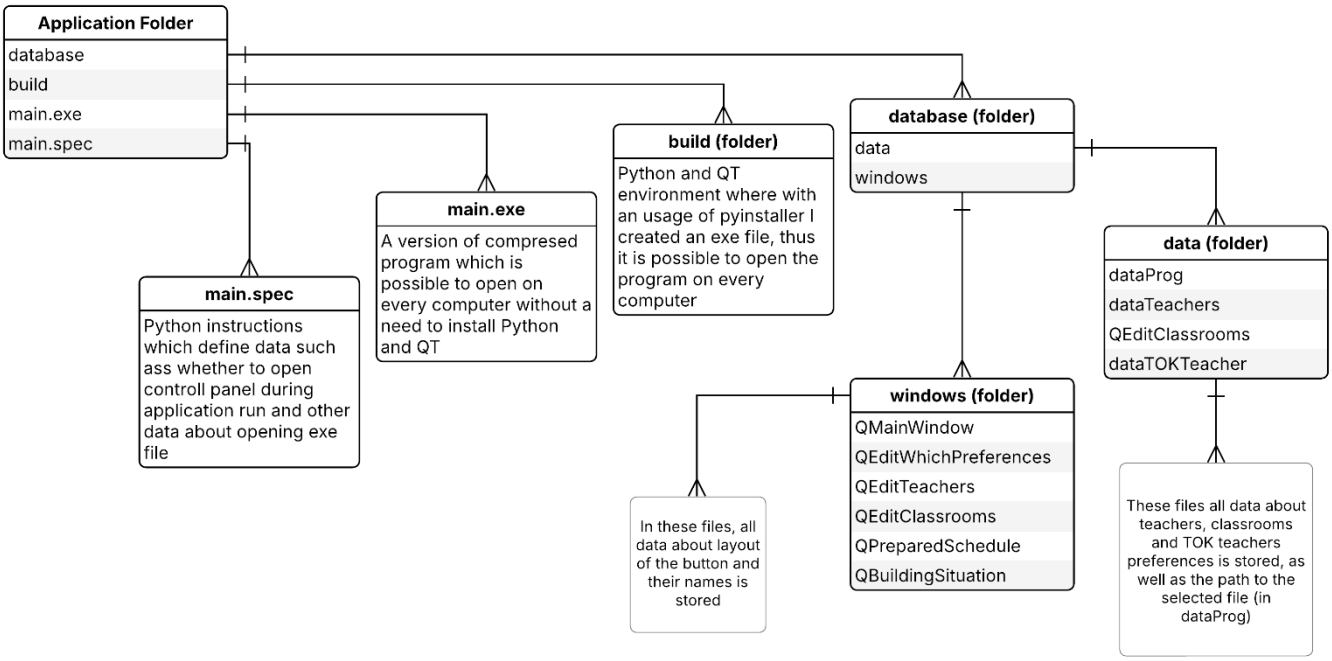


Fig. 1 File structure

To maintain order in created environment, I use Modular Programming which allows for improved readability.

UML Diagrams

UML diagrams of classes with methods:

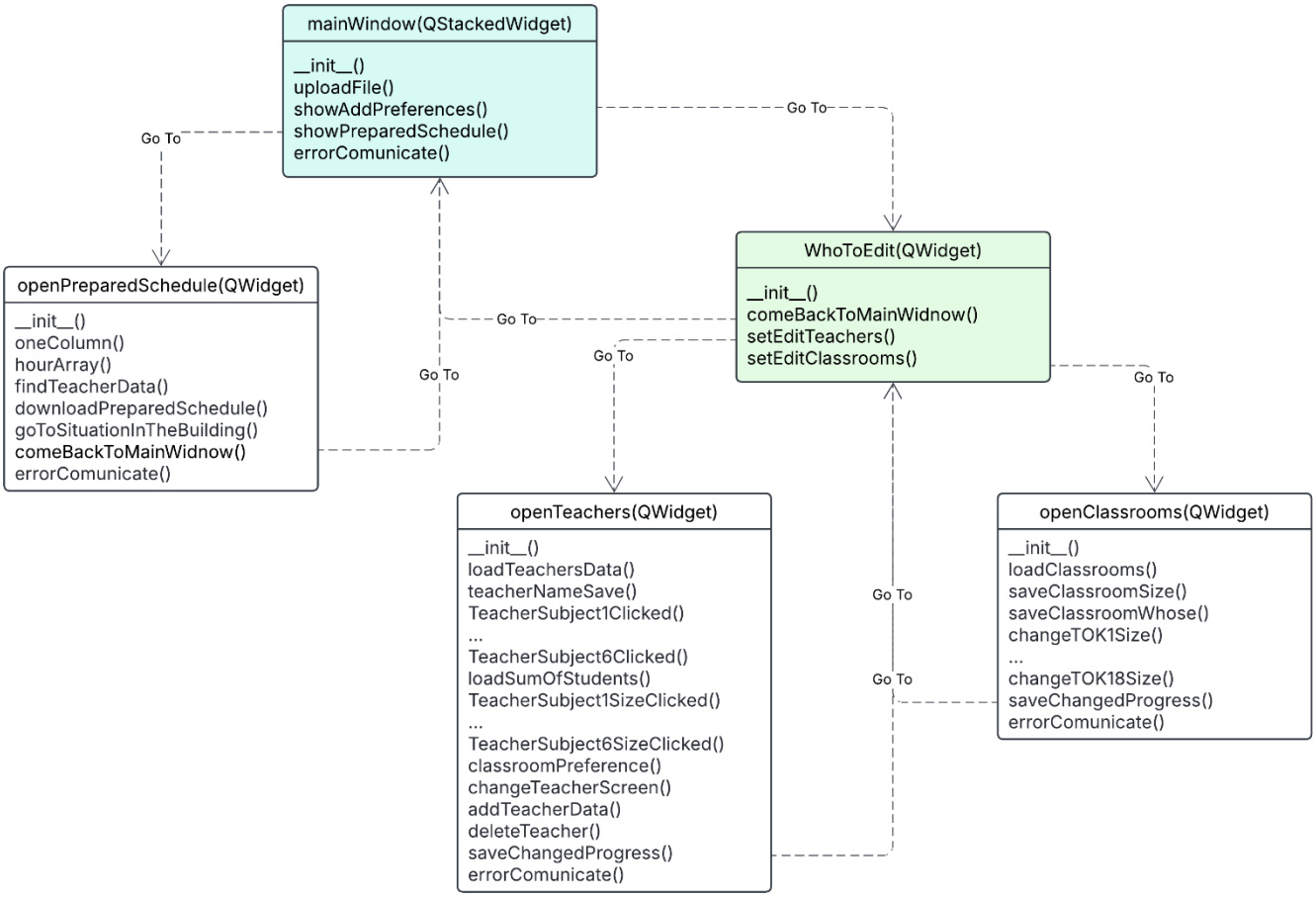
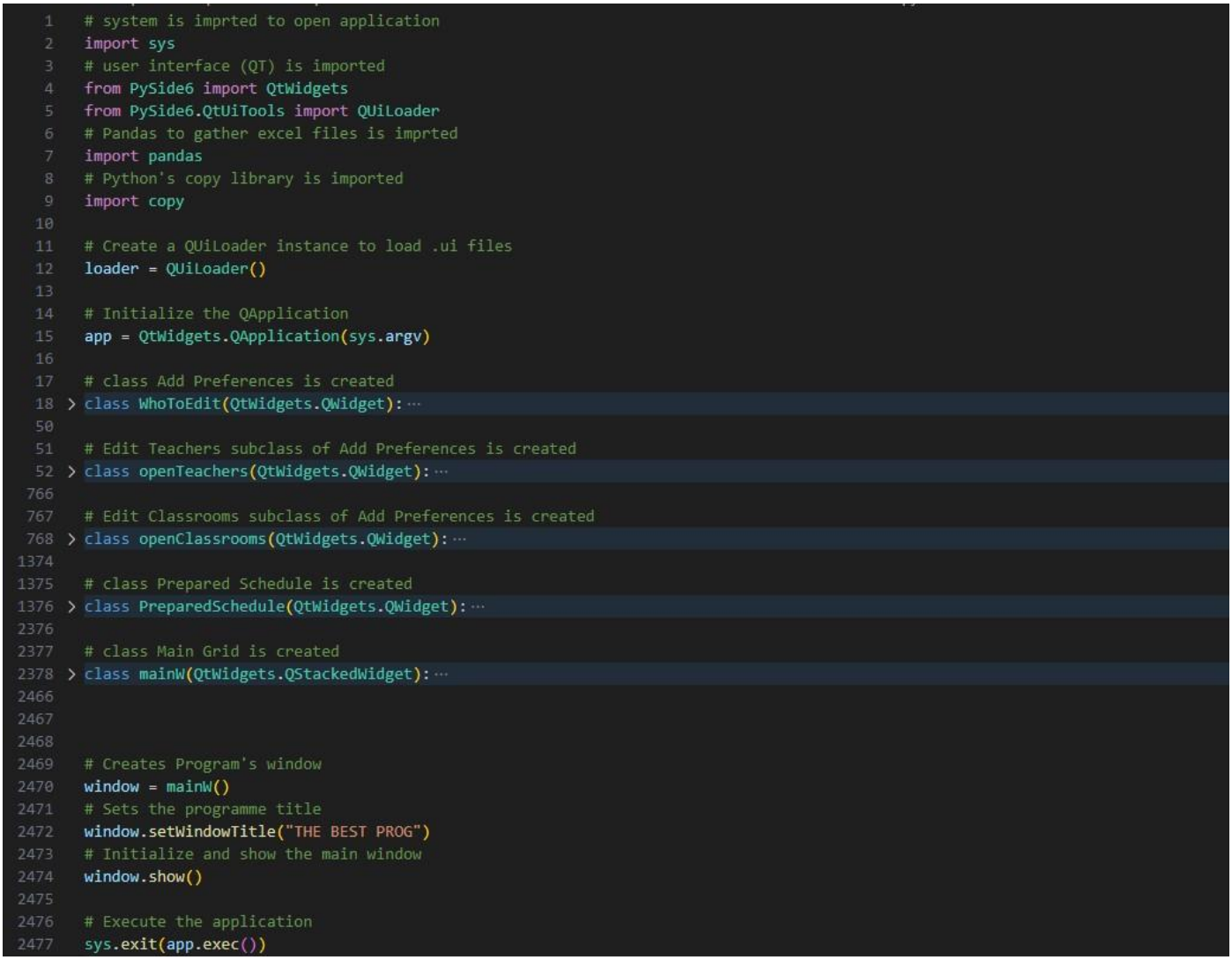


Fig. 2 Program's UML diagrams

The general structure of defined classes is demonstrated. I resigned from adding names of variables as diagrams would become unreadable. Such structure was introduced to maintain order in the code and allow for simplification in window transitions.

Class Structure (code)

Please familiarize with the comments on figures with code each time.



```
# system is imprted to open application
import sys
# user interface (QT) is imported
from PySide6 import QtWidgets
from PySide6.QtUiTools import QUiLoader
# Pandas to gather excel files is imprted
import pandas
# Python's copy library is imported
import copy

# Create a QUiLoader instance to load .ui files
loader = QUiLoader()

# Initialize the QApplication
app = QtWidgets.QApplication(sys.argv)

# class Add Preferences is created
class WhoToEdit(QtWidgets.QWidget):···

# Edit Teachers subclass of Add Preferences is created
class openTeachers(QtWidgets.QWidget):···

# Edit Classrooms subclass of Add Preferences is created
class openClassrooms(QtWidgets.QWidget):···

# class Prepared Schedule is created
class PreparedSchedule(QtWidgets.QWidget):···

# class Main Grid is created
class mainW(QtWidgets.QStackedWidget):···



# Creates Program's window
window = mainW()
# Sets the programme title
window.setWindowTitle("THE BEST PROG")
# Initialize and show the main window
window.show()

# Execute the application
sys.exit(app.exec())
```

Fig. 3 Python class definition

Logic from UML diagram is transferred to Python code.

User Interface

Please familiarize yourself with User Interface (UI) nomenclature.

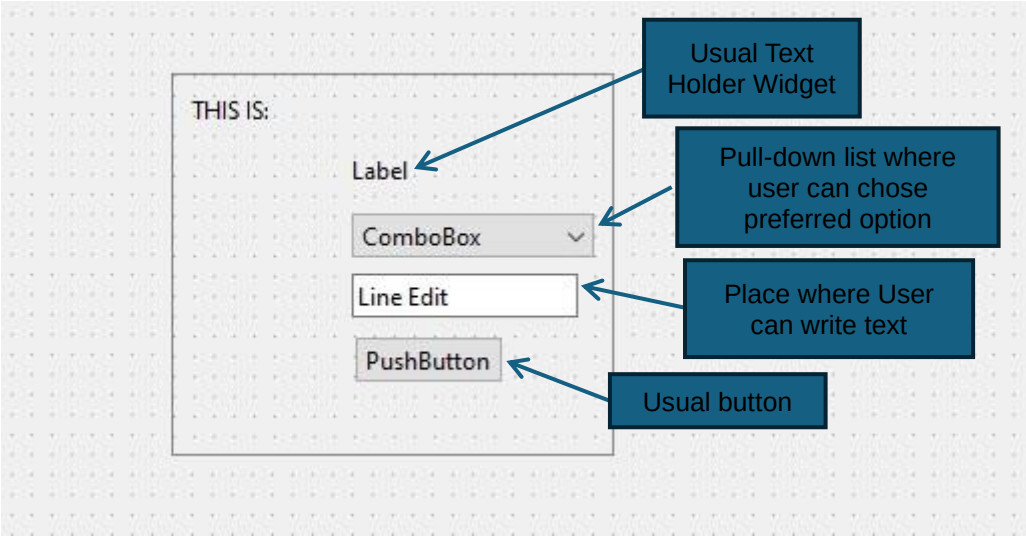


Fig. 4 User Interface nomenclature

Presented nomenclature is used throughout all Criterion.

Creating User Interface

Instead of coding user interface manually, I used QT's design tool. Below overview of creating main grid is presented:

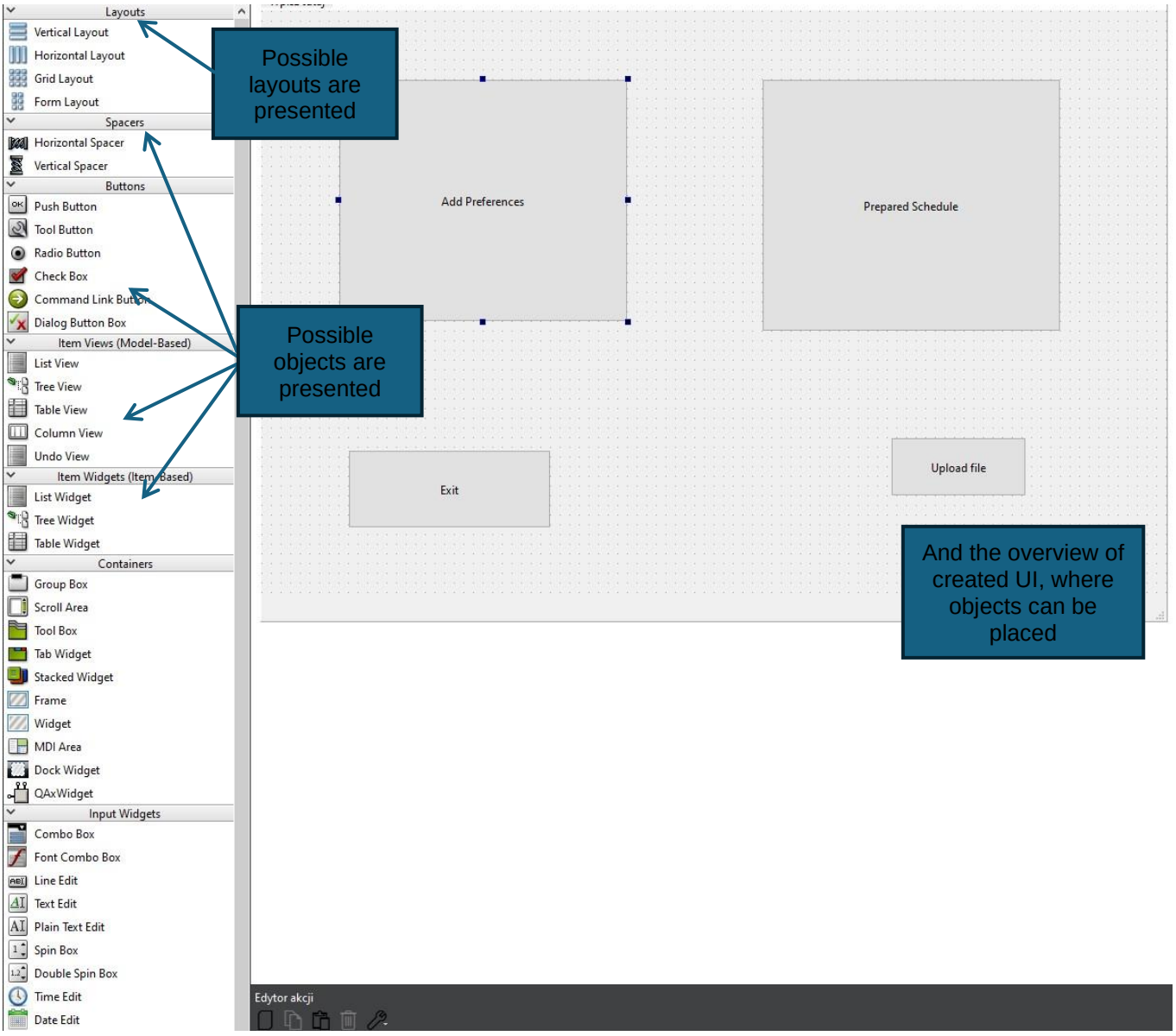
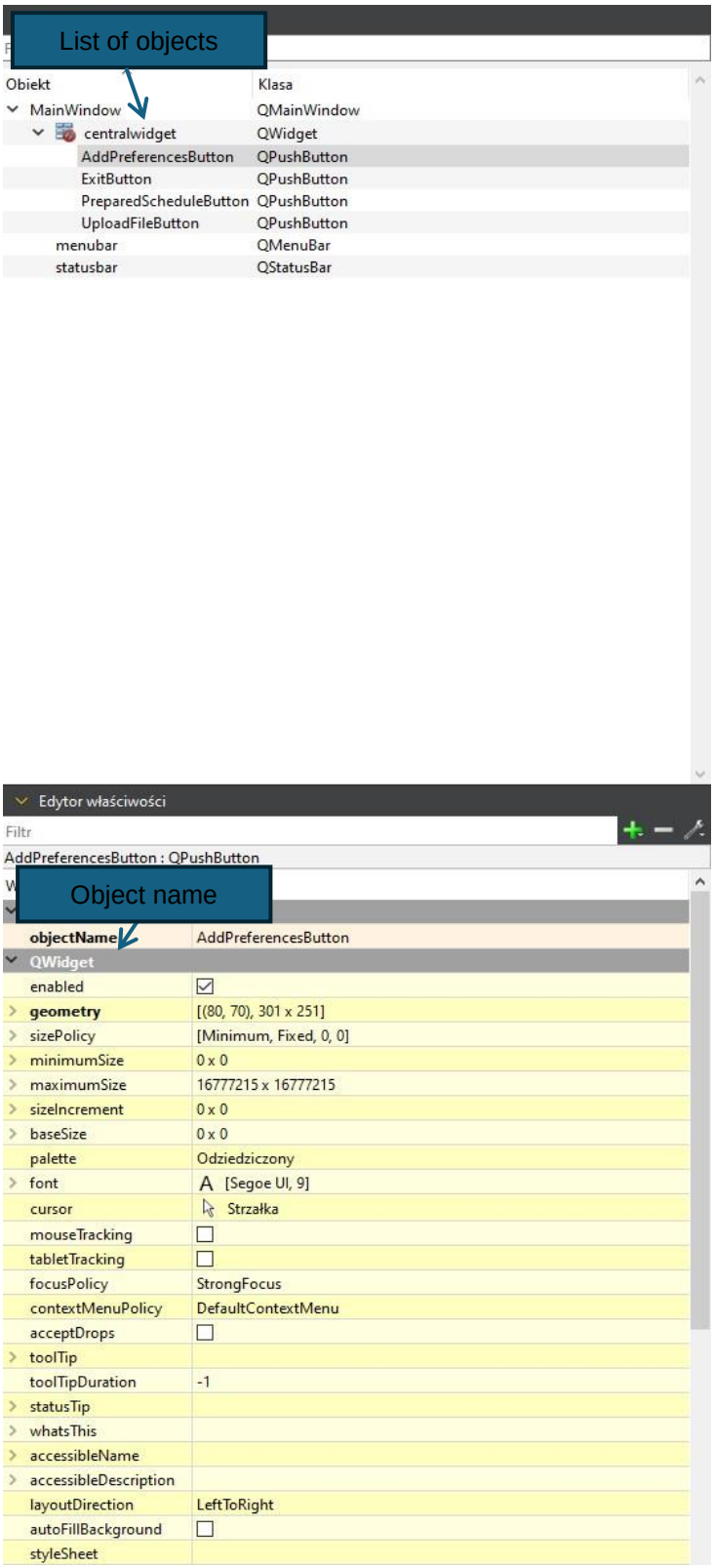


Fig. 5 QT's design tool



This section allow for connecting created object in QT UI with python code, by assigning *objectName*.

Fig. 6 Connection between QT and Python

This section shows how UI was created with an using QT's design tool.

Python Code Section

ErrorCommunicate method is created, thus the User know what went wrong and the program does not crack.

```
2720    # type of prevention from chosing improper file
2721    def errorCommunicate(self, whatError):
2722      messageText = ""
2723      requestText = ""
2724      if (whatError == "not plan file"):
2725        messageText = "chosen file is not named 'plan'"
2726        requestText = "if you are aware of that, please proceed"
2727      elif (whatError == "file not found"):
2728        messageText = "file not found"
2729        requestText = "please make sure the file exists"
2730      elif (whatError == "not plan file"):
2731        messageText = "chosen file is not named 'plan'"
2732        requestText = "if you are aware of that, please proceed"
2733      elif (whatError == "file already exist"):
2734        messageText = "file already has been created"
2735        requestText = ""
2736
2737      # creates message window Widget
2738      message = QtWidgets.QMessageBox()
2739      # sets the window title
2740      message.setWindowTitle("Error")
2741      # informs about the issue encountered
2742      message.setText(messageText)
2743      # request to rectify the problem
2744      message.setInformativeText(requestText)
2745      # sets red exclamation mark in the message
2746      message.setIcon(QtWidgets.QMessageBox.Critical)
2747      # sets buttons which allow for closing message
2748      message.setStandardButtons(QtWidgets.QMessageBox.Ok)
2749      message.exec()
```

Fig. 7 ErrorCommunicate method

I define this method at the beginning of the code section, as it is used in every class.

Main Window class (crit. B - Section C)

Below code of class Main Window is presented:

```
2377  # class Main Grid is created
2378  class mainW(QtWidgets.QStackedWidget):
2379      # Class is initialised as a parent class
2380      def __init__(self, parent=None):
2381          super().__init__(parent)
2382
2383          # Loads the UI of main grid
2384          self.currentlyVisibleWidget = loader.load("database/windows/QMainWindow.ui", None)
2385          # Adds the main window widget to widgets database
2386          self.addWidget(self.currentlyVisibleWidget)
2387
2388          # Adds the Add preferences widget to widgets database
2389          self.addPreferencesWidget = WhoToEdit(self)
2390          self.addWidget(self.addPreferencesWidget)
2391          # Adds the Edit teachers widget to widgets database
2392          self.openTeachersWidget = openTeachers(self)
2393          self.addWidget(self.openTeachersWidget)
2394          # Adds the Edit Classrooms widget to widgets database
2395          self.openClassroomsWidget = openClassrooms(self)
2396          self.addWidget(self.openClassroomsWidget)
2397          # Adds the Prepared Schedule widget to widgets database
2398          self.openPreparedScheduleWidget = PreparedSchedule(self)
2399          self.addWidget(self.openPreparedScheduleWidget)
2400
2401          # Set the initial visible widget as Main Grid
2402          self.setCurrentWidget(self.currentlyVisibleWidget)
2403          # Set the size of the Program window
2404          self.resize(950, 650)
2405
2406          # Find the button from Main Grid UI
2407          self.OpenWhoToEditButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "AddPreferencesButton")
2408          # Perscribes the method to the found button
2409          self.OpenWhoToEditButton.clicked.connect(self.showAddPreferences)
2410
2411          # Similarly for the rest of buttons
2412          self.OpenPreparedScheduleButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "PreparedScheduleButton")
2413          self.OpenPreparedScheduleButton.clicked.connect(self.showPreparedSchedule)
2414
2415          self.uploadFileButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "UploadFileButton")
2416          self.uploadFileButton.clicked.connect(self.uploadFile)
2417
2418          self.ExitButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "ExitButton")
2419          # app.exit function closes the program
2420          self.ExitButton.clicked.connect(app.exit)
```

Fig. 8 Main Window class assignment

The __init__() method is Python's convention of class definition and the structure of this particular method scheme will repeat throughout the code explanation. I create *mainW* as a kind of parenting class that stores transitions between Sections of the program to maintain structuralism.

Then, uploadFile Method (Section C) is created

```
2680      # Method which gathers path of an Excel file
2681      def uploadFile(self):
2682          # Open a file dialog to select an Excel file
2683          file_path, _ = QtWidgets.QFileDialog.getOpenFileName(
2684              self.currentlyVisibleWidget,
2685              "Select an Excel File",
2686              "",
2687              "Excel Files (*.xlsx *.xls);;All Files (*)"
2688          )
2689
2690          # if the new path was gathered
2691          if (file_path):
2692              # prevention of chosing inproper file
2693              if not("plan" in file_path.lower()):
2694                  # informs user that chosen file isn't named plan
2695                  self.errorComunicate("not plan file")
2696              # program opens stored data using pandas
2697              dfPrevious = pandas.read_excel("database\data\dataProg.xlsx", "Sheet1")
2698              # creates header for updated file
2699              dataGathered = [["path", "notifications"]]
2700              # converts Pandas' array into python one
2701              for i in range(len(dfPrevious)):
2702                  dataGathered.append(dfPrevious.iloc[i].tolist())
2703              # updates the new path
2704              dataGathered[1][0] = file_path
2705              # stores the updated path into program's memory
2706              dfChanged = pandas.DataFrame(dataGathered[1:], columns=dataGathered[0])
2707              # edits program memory and saves new file path
2708              dfChanged.to_excel("database\data\dataProg.xlsx", index=False)
2709
2710      # method which transfers the user to Prepared Schedule section
2711      def showPreparedSchedule(self):
2712          self.setCurrentWidget(self.openPreparedScheduleWidget)
2713
2714      # method which transfers the user to Add Preferences section
2715      def showAddPreferences(self):
2716          # Switch to the AddPreferences widget
2717          self.setCurrentWidget(self.addPreferencesWidget)
```

Fig. 9 Upload file method

Program gathers the path of the inputted Excel file to improve efficiency. Program already has moderate speed, mainly due to .exe formatting, but also Pandas library themselves is not rapid. That is why this kind of optimization was implemented.

Who To Edit class (crit. B - Section A)

Below code of class Who To Edit is presented:

```
17  # class Add Preferences is created
18  class WhoToEdit(QtWidgets.QWidget):
19      # Class is initialized in the conected classes family
20      def __init__(self, parent=None):
21          super().__init__(parent)
22
23          # Loads the UI of Add Preferences section
24          self.currentlyVisibleWidget = loader.load("database/windows/QEditWhichPreferences.ui", None)
25
26          # creates a variable of QT Layout type
27          layout = QtWidgets.QVBoxLayout()
28          # Adds the previously uploaded widget
29          layout.addWidget(self.currentlyVisibleWidget)
30          # Sets the layout of previously added widget
31          self.setLayout(layout)
32
33          # Finds button in UI which is assigned to Python variable
34          goToEditTeachersButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "editTeachers")
35          # Connects the button to later defined method
36          goToEditTeachersButton.clicked.connect(self.setEditTeachers)
37
38          # This methodology is used throughout the programme
39          goToEditClassroomsButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "editClassrooms")
40          goToEditClassroomsButton.clicked.connect(self.setEditClassrooms)
41
42          comeBackButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "ComeBackButton")
43          comeBackButton.clicked.connect(self.comeBackToMainWindow)
44
45      # Method which transfer user to Main Grid
46      def comeBackToMainWindow(self):
47          # Parenting class, sets widget method
48          self.parent().setCurrentWidget(self.parent().currentlyVisibleWidget)  # Sets the Main Grid Widget
49
50      # Method which transfer user to Edit Teachers subclass
51      def setEditTeachers(self):
52          self.parent().setCurrentWidget(self.parent().openTeachersWidget)
53
54      # Method whichh transfer user to Edit Classrooms subclass
55      def setEditClassrooms(self):
56          self.parent().setCurrentWidget(self.parent().openClassroomsWidget)
```

Fig. 10 Who To Edit class assignment

A kind of gateway was implemented, as all code concerning both EditTeachers and EditClassrooms code was too complex for one class.

Open Teachers class (crit. B - Section A)

Below code of class Edit Teachers is presented:

```
60  class openTeachers(QtWidgets.QWidget):
61      # Class is initialized in the conected classes family
62      def __init__(self, parent=None):
63          super().__init__(parent)
64
65          # Load the UI of Edit Teachers
66          self.currentlyVisibleWidget = loader.load("database/windows/QEditTeachers.ui", None)
67
68          # Finds the ComeBack button
69          comeBackButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "ComeBackButton")
70          # and assigns it to the ComeBack method to WhoToEdit
71          comeBackButton.clicked.connect(self.comeBackToWhoToEdit)
72
73          # Defines ComboBoxes of Load and Remove teachers
74          self.ComboBoxTeachers = self.currentlyVisibleWidget.findChild(QtWidgets.QComboBox, "comboBoxTeachers")
75          self.ComboBoxDeleteTeachers = self.currentlyVisibleWidget.findChild(QtWidgets.QComboBox, "comboBoxDeleteTeachers")
76          # Adds primary item
77          self.ComboBoxTeachers.addItem("ADD TEACHER")
78          self.ComboBoxDeleteTeachers.addItem("WHICH TEACHER")
```

Fig. 11 Open Teachers class assignment

Class definition is conducted usually, but then in __init__ method the calculations proceed. “in the connected classes family” refers to the retention of all these classes in the MainWindow QStackedWidget class.

Class operation is shown in the following figures

```
        # Loads the saved data about teachers using Pandas
        dfTeachers = pandas.read_excel("database\data\dataTeachers.xlsx", "Sheet1")
        # In Teachers array currently edited data is stored
        self.teachers = []
        # Loop to gather all teachers from Excel file
        for i in range(len(dfTeachers)):
          # Gathers data to tempoary variables
          nameT = dfTeachers.iloc[i,0]
          screenT = dfTeachers.iloc[i,1]
          subject1T = dfTeachers.iloc[i,2]
          subject2T = dfTeachers.iloc[i,3]
          subject3T = dfTeachers.iloc[i,4]
          subject4T = dfTeachers.iloc[i,5]
          subject5T = dfTeachers.iloc[i,6]
          subject6T = dfTeachers.iloc[i,7]
          size1T = dfTeachers.iloc[i,8]
          size2T = dfTeachers.iloc[i,9]
          size3T = dfTeachers.iloc[i,10]
          size4T = dfTeachers.iloc[i,11]
          size5T = dfTeachers.iloc[i,12]
          size6T = dfTeachers.iloc[i,13]
          classroomPreferenceT = dfTeachers.iloc[i,14]

          # Appends the array with gathered teacher data
          self.teachers.append([
            str(nameT),
            bool(screenT),
            str(subject1T),
            str(subject2T),
            str(subject3T),
            str(subject4T),
            str(subject5T),
            str(subject6T),
            int(size1T),
            int(size2T),
            int(size3T),
            int(size4T),
            int(size5T),
            int(size6T),
            str(classroomPreferenceT)
          ])

        # Bubble sorting that sorts alphabetises teachers
        for i in range(len(self.teachers)):
          for j in range(len(self.teachers)-1-i):
            if (self.teachers[j][0] > self.teachers[j+1][0]):
              self.teachers[j], self.teachers[j+1] = self.teachers[j+1], self.teachers[j]

        # Teachers are added to earlier defined ComboBoxes
        for i in range(len(self.teachers)):
          self.ComboBoxTeachers.addItem(self.teachers[i][0])
          self.ComboBoxDeleteTeachers.addItem(self.teachers[i][0])
```

Fig. 12 Edit Teachers variables assignment v1

Where previously added teachers are gathered

And the rest of buttons are defined:

```
# Labels of:
# Teacher's name
self.textHolderNameT = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "valueName")
# Total number of students
self.textHolderNumberOfStudents = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "labelSumOfStudents")
# Screen availability preference
self.textHolderScreenT = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "valuseTScreen")
# are defined, as well as
# LineEdit to enter teacher's new name
self.TeacherNameSaveLineEdit = self.currentlyVisibleWidget.findChild(QtWidgets.QLineEdit, "lineEditName")
# and load teacher button
loadTeachersButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "loadTeachers")
# which is connected to loadTeacher'sData is defined
loadTeachersButton.clicked.connect(self.loadTeachersData)

# and the rest of objects are defined
self.TeacherSubject1ComboBox = self.currentlyVisibleWidget.findChild(QtWidgets.QComboBox, "comboBoxSubject1")
self.textHolderSubject1T = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "valueSubject1")
self.TeacherSubject1SIZELineEdit = self.currentlyVisibleWidget.findChild(QtWidgets.QLineEdit, "lineEditSubject1Size")
self.textHolderSubject1SIZET = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "valueSubject1Size")

TeacherSubject1SaveButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "saveSubject1")
TeacherSubject1SaveButton.clicked.connect(self.TeacherSubject1Clicked)

TeacherSubject1SIZESaveButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "saveSubject1Size")
TeacherSubject1SIZESaveButton.clicked.connect(self.TeacherSubject1SizeClicked)
```

Fig. 13 Edit Teachers variables assignment v2

Both TeacherSubjectSaveButton and TeacherSubjectSIZESaveButton x6

```
TeacherScreenButtonFalse = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "TScreenFalse")
TeacherScreenButtonFalse.clicked.connect(lambda: self.changeTeacherScreen(False))

TeacherScreenButtonTrue = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "TScreenTrue")
TeacherScreenButtonTrue.clicked.connect(lambda: self.changeTeacherScreen(True))

TeacherNameSaveButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "saveName")
TeacherNameSaveButton.clicked.connect(self.TeacherNameSave)

deleteTeacherButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "deleteTeacher")
deleteTeacherButton.clicked.connect(self.deleteTeacher)

self.classroomPreferenceComboBox = self.currentlyVisibleWidget.findChild(QtWidgets.QComboBox, "classroomPreferenceComboBox")
self.classroomPreferenceComboBox.addItem("Favourite classroom")
```

Fig. 14 Edit Teachers variables assignment v3

After the definition of all buttons, the methods and the proper definition of class operations can be defined.

Then, program gathers chosen Excel file

```
230         # Path of the chosen Excel file is gathered
231         dfPrevious = pandas.read_excel("database\data\dataProg.xlsx", "Sheet1")
232         # Python array is defined
233         dataGathered = [["path", "other"]]
234         # to which data is gathered
235         for i in range(len(dfPrevious)):
236           dataGathered.append(dfPrevious.iloc[i].tolist())
237         # Path of Excel gather is defined from Python array
238         file_path = dataGathered[1][0]
239         # Catch error is implemented
240 >       try: # So if the path is invalid, program will not crack···
380
381         except FileNotFoundError: # If path is invalid
382           # Error comunicate is displayed
383           self.errorCommunicate("file not found")
384           # then program transfers user to Add Preferences choice
385           self.comeBackToWhoToEdit()
386
387         # Similarly as in Main Grid Widget
388         layout = QtWidgets.QVBoxLayout()
389         # Layout from gathered UI is set
390         layout.addWidget(self.currentlyVisibleWidget)
391         self.setLayout(layout)
```

Fig. 15 Edit Teachers data gathering v1

and case of file invalidity, proper message is displayed. I aim to gather all classes conducted, therefore they can be placed in ComboBoxes from which the Client can choose classes held by a particular teacher.

Code of earlier described classes gathering is presented:

```
try: # So if the path is invalid, program will not crack
  # Both DP1 and DP2 schedules
  RawDP1Schedule = pandas.read_excel(file_path, sheet_name="dp1") # are gathered from input Excel file
  RawDP2Schedule = pandas.read_excel(file_path, sheet_name="dp2") # which are put in ComboBoxes in Edit Teachers subsection
  # One dimentional array for DP1 classes is defined
  ALLlessons_DP1_array_Gathered = []

  # Gathering nest loop is implemented
  for day in range (len(RawDP1Schedule.columns)-2):
    for hour in range (10):
      # Explanation 1
      cell = RawDP1Schedule.iloc[hour,day+2]
      ALLlessons_DP1_array_Gathered.append(cell)

  # Loop which will delete TOK classes
  i = len(ALLlessons_DP1_array_Gathered) - 1
  while i >= 0:
    # Class variable
    subjectCeck = str(ALLlessons_DP1_array_Gathered[i])
    # if tok is in Class variable (lower case)
    if ("tok" in subjectCeck.lower()):
      del ALLlessons_DP1_array_Gathered[i]
    i -= 1

  # Array with no repeated classes is defined
  JustLessons_DP1_array_sorted = []
  JustLessons_DP1_array_sorted.append("chose subject")
  # Nested loop which will searches for non-repeated classes
  for i in range(len(ALLlessons_DP1_array_Gathered)):
    presentOnList = False
    for j in range(len(JustLessons_DP1_array_sorted)):
      if (ALLlessons_DP1_array_Gathered[i] == JustLessons_DP1_array_sorted[j]):
        presentOnList = True
                    # Excel assigns the variable NaN to the empty cells that must be excluded
    if not(presentOnList) and pandas.notna(ALLlessons_DP1_array_Gathered[i]):
      JustLessons_DP1_array_sorted.append(ALLlessons_DP1_array_Gathered[i])

  # Bubble sorting that sorts alphabetises classes
  for i in range(1, len(JustLessons_DP1_array_sorted)): # from 1 becasue we do not sort "chose subject"
    for j in range(1, len(JustLessons_DP1_array_sorted)-i):
      if (JustLessons_DP1_array_sorted[j] > JustLessons_DP1_array_sorted[j+1]):
        JustLessons_DP1_array_sorted[j], JustLessons_DP1_array_sorted[j+1] = JustLessons_DP1_array_sorted[j+1], JustLessons_DP1_array_sorted[j]

  # Adds DP1 at the end of each class
  for i in range(1, len(JustLessons_DP1_array_sorted)):
    JustLessons_DP1_array_sorted[i] = JustLessons_DP1_array_sorted[i] + " DP 1"

  # Exactly the same as for DP1
  ALLlessons_DP2_array_Gathered = []
```

Fig. 16 Edit Teachers data gathering v2

After gathering sorted classes assigned whether it is a class of DP1 or DP2 without TOK teacher classes as they belong to the EditClassrooms section, data is placed in the ComboBoxes. I use bubble sort, as, during data gathering, loaded data is already sorted, and adding a new teacher will cause only a one cycle of swapping. While the selection sort would need to change the order of every teacher each time, which is less efficient.

Similarly, the same procedure occurs for classroom preference option:

```
        # Gathered classes are added to ComboBoxes
        for j in range(len(JustLessons_DP1_array_sorted)): # for DP1
          self.TeacherSubject1ComboBox.addItem(JustLessons_DP1_array_sorted[j])
          self.TeacherSubject2ComboBox.addItem(JustLessons_DP1_array_sorted[j])
          self.TeacherSubject3ComboBox.addItem(JustLessons_DP1_array_sorted[j])
          self.TeacherSubject4ComboBox.addItem(JustLessons_DP1_array_sorted[j])
          self.TeacherSubject5ComboBox.addItem(JustLessons_DP1_array_sorted[j])
          self.TeacherSubject6ComboBox.addItem(JustLessons_DP1_array_sorted[j])

        for j in range(1, len(JustLessons_DP2_array_sorted)): # and DP2
          self.TeacherSubject1ComboBox.addItem(JustLessons_DP2_array_sorted[j])
          self.TeacherSubject2ComboBox.addItem(JustLessons_DP2_array_sorted[j])
          self.TeacherSubject3ComboBox.addItem(JustLessons_DP2_array_sorted[j])
          self.TeacherSubject4ComboBox.addItem(JustLessons_DP2_array_sorted[j])
          self.TeacherSubject5ComboBox.addItem(JustLessons_DP2_array_sorted[j])
          self.TeacherSubject6ComboBox.addItem(JustLessons_DP2_array_sorted[j])

        # Classrooms are gathered from program memory
        dfClasrooms = pandas.read_excel("database\data\dataClassrooms.xlsx", "Sheet1")
        # Classroom array is defined
        self.classrooms = [
          ["class", 0, False, "non"],
          ["class", 0, False, "non"],
          ["class", 0, False, "non"],
          ["class", 0, False, "non"],
          ["class", 0, False, "non"],
          ["class", 0, False, "non"],
          ["class", 0, False, "non"],
          ["class", 0, False, "non"],
          ["class", 0, False, "non"],
          ["class", 0, False, "non"],
          ["class", 0, False, "non"]
        ]
        # Gathered data is placed in Python array
        for i in range(len(dfClasrooms)): # note that we will never exceed the bounaried
          # as there is no option of adding classroom
          nameC = dfClasrooms.iloc[i,0]
          size = dfClasrooms.iloc[i,1]
          screenC = dfClasrooms.iloc[i,2]
          homeroom = dfClasrooms.iloc[i,3]
          self.classrooms[i] = [str(nameC), int(size), bool(screenC), str(homeroom)]

        # Classrooms' names are added to ClassroomPreference ComboBox
        for i in range(len(self.classrooms)):
          self.classroomPreferenceComboBox.addItem(self.classrooms[i][0])
```

Fig. 17 Edit Teachers data gathering v3

Up until now, all these procedures are conducted and displayed after the Client enters the class. Now, methods which allow for data manipulation are defined, each of them is assigned to particular button object

Starting with a method which loads teacher's data:

```
429     #====ALL==METHODS==BELOW==ARE==CONNECTED==TO==PROPER==BUTTONS======
430     #====================LOAD====DATA================================
431     # Loading method is define
432     def loadTeachersData(self): # which is connected to LoadButton
433       # Global TeacherIndex is defined, which stores the index of the teacher in the Teachers array
434       global TeachersIndex
435       TeachersIndex = -1
436       # Loop searching for the selected teacher who is loaded
437       for i in range(len(self.teachers)):
438         # If chosen teacher is Add Teacher Option
439         if (self.ComboBoxTeachers.currentText() == "ADD TEACHER"):
440           # Then all labels come back to initial settings
441           self.textHolderNameT.setText("Teacher Name")
442           self.textHolderScreenT.setText("True/False")
443           self.textHolderSubject1T.setText("subject name")
444           self.textHolderSubject2T.setText("subject name")
445           self.textHolderSubject3T.setText("subject name")
446           self.textHolderSubject4T.setText("subject name")
447           self.textHolderSubject5T.setText("subject name")
448           self.textHolderSubject6T.setText("subject name")
449           self.textHolderSubject1SIZET.setText("0")
450           self.textHolderSubject2SIZET.setText("0")
451           self.textHolderSubject3SIZET.setText("0")
452           self.textHolderSubject4SIZET.setText("0")
453           self.textHolderSubject5SIZET.setText("0")
454           self.textHolderSubject6SIZET.setText("0")
455           self.textHolderPreferedClassroom.setText("no preference")
456           self.textHolderNumberOfStudents.setText("number of students: 00")
457           # ComboBoxes are set to initial position
458           self.TeacherSubject1ComboBox.setCurrentIndex(0)
459           self.TeacherSubject2ComboBox.setCurrentIndex(0)
460           self.TeacherSubject3ComboBox.setCurrentIndex(0)
461           self.TeacherSubject4ComboBox.setCurrentIndex(0)
462           self.TeacherSubject5ComboBox.setCurrentIndex(0)
463           self.TeacherSubject6ComboBox.setCurrentIndex(0)
464           self.classroomPreferenceComboBox.setCurrentIndex(0)
465           # And writing spaces are reset
466           self.TeacherSubject1SIZELineEdit.clear()
467           self.TeacherSubject2SIZELineEdit.clear()
468           self.TeacherSubject3SIZELineEdit.clear()
469           self.TeacherSubject4SIZELineEdit.clear()
470           self.TeacherSubject5SIZELineEdit.clear()
471           self.TeacherSubject6SIZELineEdit.clear()
472         # If teacher is found in Teacher array
473         elif (self.ComboBoxTeachers.currentText() == self.teachers[i][0]):
474           # Global index is updated
475           TeachersIndex = i
476           # and all data is loaded to proper Labels
477           self.textHolderNameT.setText(self.teachers[i][0])
478           self.textHolderScreenT.setText(str(self.teachers[i][1]))
479           self.textHolderSubject1T.setText(self.teachers[i][2])
480           self.textHolderSubject2T.setText(self.teachers[i][3])
481           self.textHolderSubject3T.setText(self.teachers[i][4])
482           self.textHolderSubject4T.setText(self.teachers[i][5])
483           self.textHolderSubject5T.setText(self.teachers[i][6])
```

Fig. 18 Edit Teachers load teachers method v2

Similarly, rest of data is updated in above presented objects.

```
501             self.TeacherSubject5SIZELineEdit.clear()
502             self.TeacherSubject6SIZELineEdit.clear()
503             self.classroomPreferenceComboBox.setCurrentIndex(0)
504             self.textHolderPreferedClassroom.setText(str(self.teachers[i][14]))
505             # Sum of students under chosen teacher is defined
506             sum = 0
507             # and calculated using loop
508             for j in range(8, 14):
509               sum = sum + self.teachers[i][j]
510             self.textHolderNumberOfStudents.setText("number of students: " + str(sum))
511             # Loop is beaked to boost program efficiency
512             break
513         #====================LOAD====DATA============================
```

Fig. 19 Edit Teachers load teachers method v2

After loading teacher's data, Client can:

- Change teacher's name

```
515     # Method which saves edited teacher name
516     def TeacherNameSave(self):
517       # Text is gathered from a proper LineEdit
518       text = self.TeacherNameSaveLineEdit.text()
519       # Gathered text is set in
520       self.textHolderNameT.setText(text)
521       # If ComboBox is not placed at initial position
522       if (self.ComboBoxTeachers.currentText() != "ADD TEACHER"):
523         # Note that editing a teacher's name is only possible
524         self.teachers[TeachersIndex][0] = text # After loading the proper teacher
525         # Thus, the TeacherIndex is valid
526
527       # ComboBoxes are reset
528       self.ComboBoxTeachers.clear()
529       self.ComboBoxDeleteTeachers.clear()
530       # and loaded once again
531       self.ComboBoxTeachers.addItem("ADD TEACHER")
532       self.ComboBoxDeleteTeachers.addItem("WHICH TEACHER")
533       self.ComboBoxTeachers.setCurrentIndex(0)
534       self.ComboBoxDeleteTeachers.setCurrentIndex(0)
535
536       for i in range(len(self.teachers)):
537         self.ComboBoxTeachers.addItem(self.teachers[i][0])
538         self.ComboBoxDeleteTeachers.addItem(self.teachers[i][0])
539       # maintaining good order among teachers
```

Fig. 20 Edit Teachers edit data v1

As described in the figure, the edition of any kind of data can only be done after loading a particular teacher, thus we can freely use TeacherIndex.

- Change teacher’s class held

```
541     # Method which saves edited teacher subject
542     def TeacherSubject1Clicked(self):
543       # Chosen text from ComboBox is stored
544       text = self.TeacherSubject1ComboBox.currentText()
545       # If it is equal to primary option
546       if (text == "chose subject"):
547         # Text is set to the diffult one
548         text = "subject name"
549       # and the ComboBox is updated
550       self.textHolderSubject1T.setText(text)
551       # When teacher was loaded
552       if (self.ComboBoxTeachers.currentText() != "ADD TEACHER"):
553         # His/Her subject is updated
554         self.teachers[TeachersIndex][2] = text
555         # for the same reason as earlier TeacherIndex is valid
```

Fig. 21 Edit Teachers edit data v2

- Change teacher’s total number of students

```
595     # Update total number of students under teacehr
596     def loadSumOfStudents(self, whichTeacher):
597       # Similarl analogy as in loading teacher
598       sum = 0
599       for j in range(8, 14):
600         sum = sum + self.teachers[whichTeacher][j]
601       self.textHolderNumberOfStudents.setText("number of students: " + str(sum))
602
603     # Method which saves edited teacher subject size
604     def TeacherSubject1SizeClicked(self):
605       # Catch inproper data type
606       try:
607         # Entered value is gathered and changed into integer
608         value = int(self.TeacherSubject1SIZELineEdit.text())
609         # Labels is updated
610         self.textHolderSubject1SIZET.setText(str(value)) # it requires string variable
611         # If ComboBox is not on initial possition
612         if (self.ComboBoxTeachers.currentText() != "ADD TEACHER"):
613           # Teacher's data is updated
614           self.teachers[TeachersIndex][8] = int(value) # TeacherIndex is valid, for the same reason as above
615           self.loadSumOfStudents(TeachersIndex) # I put the method after loadSumOfStudents method
616       except ValueError: # If inproper data type, the message is displayed
617         self.errorCommunicate("inapropriate type")
```

Fig. 22 Edit Teachers edit data v3

- Change teacher's class’s size

- Change teacher's classroom preference

```
677     # Method which saves edited teacher classroom preference
678     def classroomPreference(self):
679       # Gathers chosen text from ComboBox
680       text = self.classroomPreferenceComboBox.currentText()
681       # If ComboBox is placed at initial position
682       if (text == "Favourite classroom"):
683         # Variable is changed to initial name
684         text = "no preference"
685       # Variable is set at proper Label
686       self.textHolderPreferedClassroom.setText(text)
687       # If ComboBox is not placed at initial position
688       if (self.ComboBoxTeachers.currentText() != "Favourite classroom"):
689         # Teacher data is updated
690         self.teachers[TeachersIndex][14] = text # TeacherIndex is valid, for the same reason
```

Fig. 23 Edit Teachers edit data v4

- Change teacher's screen availability preference

```
698     # Method which saves edited teacher screen preference
699     def changeTeacherScreen(self, boolean):
700       # Changed True/False value is updated in proper Label
701       self.textHolderScreenT.setText(str(boolean))
702       # If ComboBox is not placed at initial position
703       if (self.ComboBoxTeachers.currentText() != "ADD TEACHER"):
704         # Teacher data is updated
705         self.teachers[TeachersIndex][1] = boolean # TeacherIndex is valid, for the same reason
```

Fig. 24 Edit Teachers edit data v5

All these methods are based on the same analogy, program gathers text entered in LineEdit, checks if it is valid and then updates data in the Teachers array. I decided to store all data in the Teachers array to improve efficiency, as consistent saving and reading data from Excel files from Program memory would be inefficient. All data is saved at the end.

AddTeacher method is presented:

```
714     # Method which adds entered teacher
715     def addTeachersData(self):
716       # All data is gathered from Labels
717       nameCURRENT = self.textHolderNameT.text()
718       screenCURRENT = self.textHolderScreenT.text()
719       if (screenCURRENT == "True/False"):
720         screenCURRENT = ""
721       subjectsCURRENT = ["","","","","","",""]
722       subjectsCURRENT[0] = self.textHolderSubject1T.text()
723       subjectsCURRENT[1] = self.textHolderSubject2T.text()
724       subjectsCURRENT[2] = self.textHolderSubject3T.text()
725       subjectsCURRENT[3] = self.textHolderSubject4T.text()
726       subjectsCURRENT[4] = self.textHolderSubject5T.text()
727       subjectsCURRENT[5] = self.textHolderSubject6T.text()
728
729       size1T = int(self.textHolderSubject1SIZET.text())
730       size2T = int(self.textHolderSubject2SIZET.text())
731       size3T = int(self.textHolderSubject3SIZET.text())
732       size4T = int(self.textHolderSubject4SIZET.text())
733       size5T = int(self.textHolderSubject5SIZET.text())
734       size6T = int(self.textHolderSubject6SIZET.text())
735
736       favouriteClassroom = self.textHolderPreferedClassroom.text()
737
738       # New teacher is added to Teachers array
739       self.teachers.append([str(nameCURRENT),
740         bool(screenCURRENT),
741         str(subjectsCURRENT[0]),
742         str(subjectsCURRENT[1]),
743         str(subjectsCURRENT[2]),
744         str(subjectsCURRENT[3]),
745         str(subjectsCURRENT[4]),
746         str(subjectsCURRENT[5]),
747         int(size1T),
748         int(size2T),
749         int(size3T),
750         int(size4T),
751         int(size5T),
752         int(size6T),
753         str(favouriteClassroom)
754       ])
755       # Last index of Teachers array is gathered
756       addedTeacherIndex = len(self.teachers)-1
757       # Teacher is added to ComboBoxes
758       self.ComboBoxTeachers.addItem(self.teachers[addedTeacherIndex][0])
759       self.ComboBoxDeleteTeachers.addItem(self.teachers[addedTeacherIndex][0])
760
761       # All Labels and ComboBoxes are set to initial possitions
762       self.TeacherNameSaveLineEdit.clear()
763       self.textHolderScreenT.setText("True/False")
764       self.TeacherSubject1ComboBox.setCurrentIndex(0)
765       self.TeacherSubject2ComboBox.setCurrentIndex(0)
766       self.TeacherSubject3ComboBox.setCurrentIndex(0)
767       self.TeacherSubject4ComboBox.setCurrentIndex(0)
```

Fig. 25 Edit Teachers add teacher method

AddTeacher method gathers all data set in proper Labels and adds teacher to Teachers array, because of earlier described efficiency reasons.

Then, bubble sort is used, due to the earlier described reason.

```
793         # Teachers array is sorted using Bubble sort
794         for i in range(len(self.teachers)):
795           for j in range(len(self.teachers)-1-i):
796             if (self.teachers[j][0] > self.teachers[j+1][0]):
797               self.teachers[j], self.teachers[j+1] = self.teachers[j+1], self.teachers[j]
```

Fig. 26 Edit Teachers bubble sort

Delete teacher method is defined:

```
804         # Method which removes teacher from database
805         def deleteTeacher(self):
806           # Loop which searches for chosen teacher
807           for i in range(len(self.teachers)):
808             # Teacher's name is gathered from ComboBox
809             name = self.ComboBoxDeleteTeachers.currentText()
810             # If found the teacher is deleted from Teachers array
811             if (name == self.teachers[i][0]):
812               del self.teachers[i]
813               break
814           # ComboBoxes are cleared
815           self.ComboBoxTeachers.clear()
816           self.ComboBoxDeleteTeachers.clear()
817
818           self.ComboBoxTeachers.setCurrentIndex(0)
819           self.ComboBoxDeleteTeachers.setCurrentIndex(0)
820
821           # And updated without removed teacher
822           self.ComboBoxTeachers.addItem("ADD TEACHER")
823           self.ComboBoxDeleteTeachers.addItem("WHICH TEACHER")
824
825           for i in range(len(self.teachers)):
826             self.ComboBoxTeachers.addItem(self.teachers[i][0])
827             self.ComboBoxDeleteTeachers.addItem(self.teachers[i][0])
828
829           # Rest of Labels are set to initial positions
830           self.textHolderNameT.setText("Teacher Name")
831           self.textHolderScreenT.setText("True/False")
832           self.textHolderSubject1T.setText("subject name")
833           self.textHolderSubject2T.setText("subject name")
834           self.textHolderSubject3T.setText("subject name")
835           self.textHolderSubject4T.setText("subject name")
836           self.textHolderSubject5T.setText("subject name")
837           self.textHolderSubject6T.setText("subject name")
838           self.textHolderSubject1SIZET.setText("0")
839           self.textHolderSubject2SIZET.setText("0")
840           self.textHolderSubject3SIZET.setText("0")
841           self.textHolderSubject4SIZET.setText("0")
842           self.textHolderSubject5SIZET.setText("0")
843           self.textHolderSubject6SIZET.setText("0")
```

Fig. 27 Edit Teachers delete teacher method

At the end, saveChangedProgress method is defined:

```
846     #====================SAVE=DATA=GATHERED==========================
847     # Method which saves all edited data to Program's memory
848     def saveChangedProgress(self):
849       # Changed data array is defined
850       changedTeachers = []
851       # and structured properly for Pandas requirements
852       changedTeachers.append(["name", "screen", "subject1", "subject2", "subject3", "subject4", "subjec
853       # Changed data array is filled
854       for i in range(len(self.teachers)):
855         changedTeachers.append(self.teachers[i])
856       # And the proper shcemat for Pandas requirements is implemented
857       dfTeach = pandas.DataFrame(changedTeachers[1:], columns=changedTeachers[0])
858       dfTeach.to_excel("database\data\dataTeachers.xlsx", index=False)
859       # Program comes back to Add Preferences section
860       self.comeBackToWhoToEdit()
861
862     #====================SAVE=DATA=GATHERED==========================
```

Fig. 28 Edit Teachers save data

The structure of the Excel file edition is learned from Pandas' library and the changes are updated to the Program memory only at the stage when the Client leaves EditTeachers class to improve efficiency.

Open Classrooms class (crit. B - Section A)

Below code of class Edit Teachers is presented:

```
878   # Edit Classrooms subclass of Add Preferences is created
879   class openClassrooms(QtWidgets.QWidget):
880       # Class is initialized in the conected classes family
881       def __init__(self, parent=None):
882           super().__init__(parent)
883
884           # Load the UI file of Edit Classrooms
885           self.currentlyVisibleWidget = loader.load("database/windows/QEditClassrooms.ui", None)
886
887           # Comeback button is defined
888           comeBackButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "ComeBackButton")
889           comeBackButton.clicked.connect(self.saveChangedProgress)
890
891           # Chosing classroom ComboBox is defined
892           self.ComboBoxClassrooms = self.currentlyVisibleWidget.findChild(QtWidgets.QComboBox, "comboBoxClassrooms")
893
894           # Program gathers data about classrooms from Program memory
895           dfClasrooms = pandas.read_excel("database\data\dataClassrooms.xlsx", "Sheet1")
896           # Global ClassroomIndex is defined
897           global Clasroomindex # similar to the global TeacherIndex
898           Clasroomindex = -1
899           # Classrooms array is defined
900           self.classrooms = [
901             ["class", 0, False, "non"],
902             ["class", 0, False, "non"],
903             ["class", 0, False, "non"],
904             ["class", 0, False, "non"],
905             ["class", 0, False, "non"],
906             ["class", 0, False, "non"],
907             ["class", 0, False, "non"],
908             ["class", 0, False, "non"],
909             ["class", 0, False, "non"],
910             ["class", 0, False, "non"],
911             ["class", 0, False, "non"]
912           ]
913           # and filled with a data from Program memory
914           for i in range(len(dfClasrooms)):
915             nameC = dfClasrooms.iloc[i,0]
916             size = dfClasrooms.iloc[i,1]
917             screenC = dfClasrooms.iloc[i,2]
918             homeroom = dfClasrooms.iloc[i,3]
919             self.classrooms[i] = [str(nameC), int(size), bool(screenC), str(homeroom)]
```

Fig. 29 Edit Classrooms class assignment

Class definition is almost identical to the definition of EditTeachers, however, now it is proceeding in the context of classrooms and TOK teacher preferences.

Similarly, all objects are defined

```
921         # Initial possition of ComboBox is assigned
922         self.ComboBoxClassrooms.addItem("CHOSE CLASSROOM")
923         # and the rest of classrooms' names are added
924         for i in range(len(self.classrooms)):
925             self.ComboBoxClassrooms.addItem(self.classrooms[i][0])
926 
927         # The Labels are defined
928         self.textHolderSizeC = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "valueSize")
929         self.textHolderScreenC = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "valuseCScreen")
930         self.textHolderWhoseC = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "valueWhose")
931         # as well as the buttons
932         loadClassroomButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "loadClassrooms")
933         loadClassroomButton.clicked.connect(self.loadClassroomData) # and proper methods are assigned
934 
935         saveClassroomSizeButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "saveSize")
936         saveClassroomSizeButton.clicked.connect(self.saveClassroomSize)
937 
938         saveClassroomWhoseButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "saveWhose")
939         saveClassroomWhoseButton.clicked.connect(self.saveClassroomWhose)
940 
941         lassroomScreenButtonFalse = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "CScreenFalse")
942         lassroomScreenButtonFalse.clicked.connect(lambda: self.saveClassroomScreen(False))
943         lassroomScreenButtonTrue = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "CScreenTrue")
944         lassroomScreenButtonTrue.clicked.connect(lambda: self.saveClassroomScreen(True))
```

Fig. 29 Edit Classrooms variables assignment v1

Catch error is included in case of file invalidity.

```
943         # Similarly as earlier
944         # Path of the chosen Excel file is gathered
945         dfPrevious = pandas.read_excel("database\data\dataProg.xlsx", "Sheet1")
946         dataGathered = [["path", "notifications"]]
947         for i in range(len(dfPrevious)):
948             dataGathered.append(dfPrevious.iloc[i].tolist())
949         # ans saved in file_path variable
950         file_path = dataGathered[1][0]
951         # Catch error if path is no longer valid
952 >       try:···
1238 
1239         except FileNotFoundError: # If the path is not valid
1240             self.errorCommunicate("file not found") # Proper message is displayed
1241             self.comeBackToWhoToEdit() # and Program comes back to Add Preferences section
```

Fig. 30 Edit Classrooms variables assignment v2

Similarly, data about classes is gathered:

```
        # Similar, but exactly opposite situation is happening as earlier
        RawDP1Schedule = pandas.read_excel(file_path, sheet_name="dp1")
        RawDP2Schedule = pandas.read_excel(file_path, sheet_name="dp2")
        # All classes are gathered from a chosen file
        ALLlessons_DP1_array_Gathered = []
        # They are placed in one-dimentional array
        for day in range (len(RawDP1Schedule.columns)-2):
          for hour in range (10):
              cell = RawDP1Schedule.iloc[hour,day+2]
              ALLlessons_DP1_array_Gathered.append(cell)
        # But this time we take all classess
        i = len(ALLlessons_DP1_array_Gathered) - 1
        while i >= 0:
          # In which TOK is present
          subjectCeck = str(ALLlessons_DP1_array_Gathered[i])
          if not("tok" in subjectCeck.lower()):
            del ALLlessons_DP1_array_Gathered[i]
          i -= 1
        # Then the nonrepeated TOKs are placed in array
        JustTOK_DP1_array_sorted = []
        for i in range(len(ALLlessons_DP1_array_Gathered)):
          presentOnList = False
          for j in range(len(JustTOK_DP1_array_sorted)):
            if (ALLlessons_DP1_array_Gathered[i] == JustTOK_DP1_array_sorted[j]):
              presentOnList = True
          if not(presentOnList):
            JustTOK_DP1_array_sorted.append(ALLlessons_DP1_array_Gathered[i])
        # and sorted using Bubble sort
        for i in range(len(JustTOK_DP1_array_sorted)):
          for j in range(len(JustTOK_DP1_array_sorted)-i-1):
            if (JustTOK_DP1_array_sorted[j] > JustTOK_DP1_array_sorted[j+1]):
              JustTOK_DP1_array_sorted[j], JustTOK_DP1_array_sorted[j+1] = JustTOK_DP1_array_sorted[j+1], JustTOK_DP1_array_sorted[j]

        for i in range(len(JustTOK_DP1_array_sorted)):
          JustTOK_DP1_array_sorted[i] = JustTOK_DP1_array_sorted[i] + " DP 1"

        # Exactly the same for DP2
        ALLlessons_DP2_array_Gathered = []

        for day in range (len(RawDP2Schedule.columns)-2):
```

Fig. 31 Edit Classrooms data gathering v1

However, this time only for TOK teacher.

```
        # Array with all TOKs is created
        ALLTOKs = [""] * 18
        # All TOKs are placed to singular array
        lastDP1TOK = 0
        for i in range(len(JustTOK_DP1_array_sorted)):
          ALLTOKs[i] = JustTOK_DP1_array_sorted[i]
          lastDP1TOK = i

        for i in range(len(JustTOK_DP2_array_sorted)):
          ALLTOKs[lastDP1TOK+i+1] = JustTOK_DP2_array_sorted[i]
```

Fig. 32 Edit Classrooms data gathering v2

As agreed with the client, there is no possibility for TOK teacher to have more than 18 classes a week, thus the size of the array.

Then, previous data about TOK teacher is gathered

```
1038            # TOK teacher saved data is gathered
1039            dfTOKteacher = pandas.read_excel("database\data\dataTOKTeacher.xlsx", "Sheet1")
1040            # TOK teacher data Python array is defined
1041            self.TOKTeacherData = []
1042            for i in range(18):
1043              subjectName = dfTOKteacher.iloc[i,0]
1044              size = dfTOKteacher.iloc[i,1]
1045              self.TOKTeacherData.append([str(subjectName), int(size)])
1046
1047            # Search for new TOK classes
1048            for i in range(18):
1049              # Boolean to distingush new TOK
1050              isTOKClassPresent = False
1051              # With its index
1052              index = -1
1053              # Searches for not present TOK classes
1054              for j in range(18):
1055                if (ALLTOKs[j] == self.TOKTeacherData[i][0]) and (ALLTOKs != ""):
1056                  isTOKClassPresent = True
1057                  index = j
1058              # If TOK class is not present
1059              if (isTOKClassPresent):
1060                # then it is added to first empty place
1061                for k in range(18):
1062                  if (self.TOKTeacherData[k][0] == ""):
1063                    self.TOKTeacherData[k][0] = ALLTOKs[index]
1064                    break
```

Fig. 33 Edit Classrooms data gathering v2

Data is updated according to the information from the inputted Excel file with schedule.

```
1066            # Then TOK teacher Classroom is defined
1067            self.TOKTeacherClassroomPreference = ""
1068            # Boolean variable is define to once gather proper datatype
1069            loadTOKteacherClassroomPreference = True
1070            if (loadTOKteacherClassroomPreference):
1071              # Classroom preference value is gathered
1072              valueOfTOKpreference = dfTOKteacher.iloc[0,2]
1073              # If the gathered variable is string
1074              if isinstance(valueOfTOKpreference, str):
1075                # Then program normaly gathers the value
1076                self.TOKTeacherClassroomPreference = valueOfTOKpreference
1077              else: # However, because of unknown reason
1078                # TOK teacher's classroom preference is stored as boolean
1079                self.TOKTeacherClassroomPreference = str(int(valueOfTOKpreference)) # that is why it is converted to integer and later string
1080              loadTOKteacherClassroomPreference = False
1081            # Label with TOK teacher classroom preference is updated
1082            self.TOKTeacherClassroomPreferenceLabel = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "classroomPreferenceLabel")
1083            self.TOKTeacherClassroomPreferenceLabel.setText(self.TOKTeacherClassroomPreference)
1084
1085
1086            # Then the data is loaded to proper Labels
1087            whichTOKLabel1 = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "whichTOK1")
1088            whichTOKLabel1.setText(self.TOKTeacherData[0][0])
1089            whichTOKLabel2 = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "whichTOK2")
1090            whichTOKLabel2.setText(self.TOKTeacherData[1][0])
1091            whichTOKLabel3 = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "whichTOK3")
1092            whichTOKLabel3.setText(self.TOKTeacherData[2][0])
1093            whichTOKLabel4 = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "whichTOK4")
```

Fig. 34 Edit Classrooms data gathering v3

All necessary objects are defined

```
1224        # Rest of objects are defined and set
1225        saveTOK1sizeButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "saveSize1")
1226        saveTOK1sizeButton.clicked.connect(self.changeTOK1Size)
1227
1228        self.sizeTOK1LineEdit = self.currentlyVisibleWidget.findChild(QtWidgets.QLineEdit, "numberOfStudents1")
1229
1230        self.textHolderTOK1Size = self.currentlyVisibleWidget.findChild(QtWidgets.QLabel, "valueOfsize1")
1231        self.textHolderTOK1Size.setText(str(self.TOKTeacherData[0][1]))
1232
1233        self.classroomPreferenceTOKComboBox = self.currentlyVisibleWidget.findChild(QtWidgets.QComboBox, "classroomPreferenceComboBox")
1234        classroomTOKPreferenceSaveButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "saveClassroomPreference")
1235        classroomTOKPreferenceSaveButton.clicked.connect(self.classroomPreference)
1236
1237        self.classroomPreferenceTOKComboBox.addItem("Prefered classroom")
1238
1239        # ComboBox is filled
1240        for i in range(len(self.classrooms)):
1241            self.classroomPreferenceTOKComboBox.addItem(self.classrooms[i][0])
```

Fig. 35 Edit Classrooms variables assignment v3

Similarly, as in the EditTeachers class, all these procedures take place after the Client opens the EditClassrooms section. Now we move to the edition aspect of the class which is almost identical to the one from EditTeachers that is why I do not get into the details, as the argumentation is exactly the same as in the EditTeachers class.

Client can:

- load classroom’s data

```
1255    # Method which loads chosen classroom
1256    def loadClassroomData(self):
1257        # ClassroomIndex works on the same basis as Teacher Index
1258        self.Clasroomindex = -1
1259        # Loop searches for chosen classroom
1260        for i in range(len(self.classrooms)):
1261            if (self.ComboBoxClassrooms.currentText() == self.classrooms[i][0]):
1262                # ClassroomIndex is updated
1263                self.Clasroomindex = i
1264                # and data is loaded to Labels
1265                self.textHolderSizeC.setText(str(self.classrooms[i][1]))
1266                self.textHolderScreenC.setText(str(self.classrooms[i][2]))
1267                self.textHolderWhoseC.setText(self.classrooms[i][3])
1268                break # Break for improved efficiency of the Program
```

Fig. 36 Edit Classrooms edit data v1

- change classroom’s size

```
1273    # Method which saved edited classroom size
1274    def saveClassroomSize(self):
1275      # Catch improper data types
1276      try:
1277        # Updates ClassroomIndex
1278        self.loadClassroomData()
1279        self.Clasroomindex
1280        # Gathers text from LineEdit
1281        firstSize = self.currentlyVisibleWidget.findChild(QtWidgets.QLineEdit, "lineEditSize")
1282        # Converts to integer
1283        intSize = int(firstSize.text())
1284        # and stores in Classrooms array
1285        self.classrooms[self.Clasroomindex][1] = intSize
1286        # Then updates Label
1287        self.textHolderSizeC.setText(str(intSize))
1288      except ValueError: # in case of improper data type
1289        self.errorComunicate("inapropriate type") # Same shcemat as earlier
```

Fig. 37 Edit Classrooms edit data v2

- change whose classroom is it; in context of homeroom hours

```
1294    # Method which saved edited homeroom hour value
1295    def saveClassroomWhose(self):
1296      # Updates ClassroomIndex
1297      self.loadClassroomData()
1298      self.Clasroomindex
1299      # Defines LineEdit
1300      firstWhose = self.currentlyVisibleWidget.findChild(QtWidgets.QLineEdit, "lineEditWhose")
1301      # and gathers text from it
1302      strWhose = firstWhose.text()
1303      # Updates value in Clasroom array
1304      self.classrooms[self.Clasroomindex][3] = strWhose
1305      # and updates Label
1306      self.textHolderWhoseC.setText(strWhose)
```

Fig. 38 Edit Classrooms edit data v3

- change TOK class's size x 18)

```
1328    # Method which saved edited TOK class' size
1329    def changeTOK1Size(self):
1330      try: # Catch improper data types
1331        # Gathers changed value as integer
1332        value = int(self.sizeTOK1LineEdit.text())
1333        # Updates Label
1334        self.textHolderTOK1Size.setText(str(value))
1335        # and data value in TOK teacher array
1336        self.TOKTeacherData[0][1] = int(value)
1337      except ValueError: # in case of improper data type
1338        self.errorComunicate("inapropriate type") # Same shcemat as earlier
```

Fig. 39 Edit Classrooms edit data v4

Program saves changed progress:

```
1501      # Method which saves edited data
1502      def saveChangedProgress(self):
1503          # Changed data array is defined
1504          changedClassrooms = []
1505          # and structured properly for Pandas requirements
1506          changedClassrooms.append(["classroom", "size", "screen", "HR"])
1507          # Data is loaded to changed data array
1508          for i in range(len(self.classrooms)):
1509              changedClassrooms.append(self.classrooms[i])
1510          # The proper shcemat for Pandas requirements is implemented
1511          dfClass = pandas.DataFrame(changedClassrooms[1:], columns=changedClassrooms[0])
1512          dfClass.to_excel("database\data\dataClassrooms.xlsx", index=False)
1513
1514          # Similarly for TOK teacher
1515          changedTOKteacher = []
1516          changedTOKteacher.append(["tok name", "size", "preference"])
1517
1518          for i in range(18):
1519              tokName = ""
1520              size = ""
1521              preference = ""
1522              if (i == 0):
1523                  tokName = self.TOKTeacherData[i][0]
1524                  size = str(self.TOKTeacherData[i][1])
1525                  self.TOKTeacherClassroomPreference
1526                  preference = self.TOKTeacherClassroomPreference
1527              else:
1528                  tokName = self.TOKTeacherData[i][0]
1529                  size = str(self.TOKTeacherData[i][1])
1530                  preference = "null"
1531              changedTOKteacher.append([tokName, size, preference])
1532          dfTOKt = pandas.DataFrame(changedTOKteacher[1:], columns=changedTOKteacher[0])
1533          dfTOKt.to_excel("database\data\dataTOKTeacher.xlsx", index=False)
1534          # Similarly for TOK teacher
1535
1536          # Comes back to Add Preferences section
1537          self.comeBackToWhoToEdit()
```

Fig. 40 Edit Classrooms save progress

This class operates on the same basis as the EditTeachers class, as it corresponds to exactly the same task – gathering preference data. However, now we move on to the main task of the Program which is classroom assignment.

Prepared Schedule class (crit. B - Section B)

Class is defined as the EditTeachers and EditClassrooms classes

```
1564  # class Prepared Schedule is created
1565  class PreparedSchedule(QtWidgets.QWidget):
1566      # Class is initialized in the conected classes family
1567      def __init__(self, parent=None):
1568          super().__init__(parent)
1569
1570          # Load the UI file of Prepared Schedule
1571          self.currentlyVisibleWidget = loader.load("database/windows/QPreparedSchedule.ui", None)
1572
1573          # All data about teachers and TOK teacher is loaded
1574
1575          #================LOAD===ADD=PREFERENCES===DATA============================
1576          dfClasrooms = pandas.read_excel("database\data\dataClassrooms.xlsx", "Sheet1")
1577          dfTeachers = pandas.read_excel("database\data\dataTeachers.xlsx", "Sheet1")
1578
1579          self.classrooms = [
1580            ["class", 0, False, "non"],
1581            ["class", 0, False, "non"],
```

Fig. 41 Prepared schedule class assignment v1

All data regarding teachers, TOK teacher and classrooms is gathered and placed to arrays of the structure identical as earlier. Then, the path of the inputted Excel file is gathered

```
1661          # Similarly, the path of inputed Excel file
1662          dfPrevious = pandas.read_excel("database\data\dataProg.xlsx", "Sheet1")
1663          dataGathered = [["path", "notifications"]]
1664
1665          for i in range(len(dfPrevious)):
1666            dataGathered.append(dfPrevious.iloc[i].tolist())
1667
1668          file_path = dataGathered[1][0]
1669
1670 >        try: # Catch error if the path is not valid···
2273
2274
2275          except FileNotFoundError: # If path no longer valid
2276            self.errorComunicate("file not found")  # then the proper message is displayed
2277            self.comeBackToMainWindow # and Program transfers user to Main Grid
```

Fig. 42 Prepared schedule class assignment v2

Catch error is used in case of path invalidity.

Now, all data regarding the schedule is gathered and transferred to Python array.

At the beginning, the headers are gathered:

```
1671        # Data from chosen Excel file is gathered
1672        RawDP1Schedule = pandas.read_excel(file_path, sheet_name="dp1")
1673        RawDP2Schedule = pandas.read_excel(file_path, sheet_name="dp2")
1674        # Separetly Headers are stored
1675        headers_DP1_df = pandas.read_excel(file_path, sheet_name="dp1", nrows=0)
1676        headers_DP2_df = pandas.read_excel(file_path, sheet_name="dp2", nrows=0)
1677        # Headers are the days of the week
1678        #=================HEADERS========DP1================
1679        # Proper header DP1 array is defined
1680        headers_DP1_array_Gathered = []
1681        for i in range(len(headers_DP1_df.columns)):
1682          headers_DP1_array_Gathered.append(headers_DP1_df.columns[i])
1683        # First cell is deleted, due to the layout of the chosen Excel file
1684        del headers_DP1_array_Gathered[0] # Look the form of Excel file in Crit B
1685        # Unnamed variables are deleted
1686        for i in range(1, len(headers_DP1_array_Gathered)): # because Pandas read merged cells as "Unnamed"
1687          for j in range(len(headers_DP1_array_Gathered)+5):
1688            if (headers_DP1_array_Gathered[i] == f"Unnamed: {j}"):
1689              # The missing names of the week is assigned
1690              headers_DP1_array_Gathered[i] = headers_DP1_array_Gathered[i-1]
1691              break
1692        # Additional few places are created for Friday
1693        for i in range(7): # because Pandas read merged cells as "Unnamed i"
1694          headers_DP1_array_Gathered.append(headers_DP1_array_Gathered[len(headers_DP1_array_Gathered)-1])
1695
1696        #=================HEADERS========DP2================
1697        # Exactly the same for DP 2
1698        headers_DP2_array_Gathered = []
1699        for i in range(len(headers_DP2_df.columns)):
1700          headers_DP2_array_Gathered.append(headers_DP2_df.columns[i])
```

Fig. 43 Prepared schedule data gathering v1

Then, classes are gathered to Python array:

```
1715        #=======GATHERS==THE==ARRAY==OF==DP1==CLASSES=======
1716        # 11 due to maximum of 10 hours in our schedule
1717        lessons_DP1_array_Gathered = [[""] * len(headers_DP1_array_Gathered) for _ in range(11)] # and the maximal length of gathered headers
1718
1719        # Loop to gather classess
1720        for day in range (len(headers_DP1_array_Gathered)-3):
1721          for hour in range (10):
1722              # +2 due to the layout of Excel form
1723              cell = RawDP1Schedule.iloc[hour,day+2]
1724              # Leaving first raw and column
1725              lessons_DP1_array_Gathered[hour+1][day+1] = cell
1726        # for numbers of hours
1727        for i in range(10):
1728          lessons_DP1_array_Gathered[i+1][0] = i+1
1729        # and the days of the week
1730        for i in range(len(lessons_DP1_array_Gathered[0])-1):
1731          lessons_DP1_array_Gathered[0][i+1] = headers_DP1_array_Gathered[i]
1732        #=======GATHERS==THE==ARRAY==OF==DP1==CLASSES=======
1733        # Similarlt for DP2
1734        #=======GATHERS==THE==ARRAY==OF==DP2==CLASSES=======
1735        lessons_DP2_array_Gathered = [[""] * len(headers_DP2_array_Gathered) for _ in range(11)]
```

Fig. 44 Prepared schedule data gathering v2

After finishing the data gathering to Python arrays of both DP1 and DP2, array which combines both annuals is created.

```
# Creates an array of all classes
whole_week_array_combined = [[""] * (len(lessons_DP1_array_Gathered[0]) + len(lessons_DP2_array_Gathered[0])) for _ in range(11)]
# Adds numbers of hours
for i in range(11):
  whole_week_array_combined[i][0] = i
# and defines the days of the week
days_of_the_week_array = ['MONDAY', 'TUESDAY', 'WEDNESDAY', 'THURSDAY', 'FRIDAY']
# Array of days' lengths
length_of_day = [0,0,0,0,0,0]
# First day of the week
day = "MONDAY"
# Index of first
from_day = 1
# and last day of given day of the week
to_day = 0
# Algorithm which puts all headers into one array
for j in range (4):
```

Fig. 45 Prepared schedule data interpretation v1

For proper assignment, headers are counted and transferred to combined array.

```
for j in range (4):
  # Variable which counts the number of days
  day_count = 0 # of given day of the week
  # Counting algorithm
  for k in range(len(headers_DP1_array_Gathered)-2):
    if (days_of_the_week_array[j] == headers_DP1_array_Gathered[k]):
      day_count += 1
  for k in range(len(headers_DP2_array_Gathered)-2):
    if (days_of_the_week_array[j] == headers_DP2_array_Gathered[k]):
      day_count += 1
  # Length of calculated day is updated
  length_of_day[j+1] = day_count
  # +1 to prevent filling first columnt
  if j == 0:
    to_day = to_day + day_count + 1
  else:
    to_day = to_day + day_count
  # First raw is filled with proper day of the week
  for i in range(from_day, to_day):
    whole_week_array_combined[0][i] = days_of_the_week_array[j]
  from_day = to_day

# To this point, all middle days of the week are filled
for i in range(len(whole_week_array_combined[0])): # thus it fills the empty Fridays
  if (whole_week_array_combined[0][i]==""):
    whole_week_array_combined[0][i] = days_of_the_week_array[4]
# Number of Friday's columns is calculated
length_of_day[5] = len(whole_week_array_combined[0]) - length_of_day[0] - length_of_day[1] - length_of_day[2] - length_of_day[3]- length_of_day[4]
```

Fig. 46 Prepared schedule data interpretation v2

Method which gathers one column is defined:

```
# Method which gathers one column
def oneColumn (index, array):
  # There are 10 hours in our schedule
  one_hour_array = [""] * 10
  # Gathers all classes from one column
  for j in range(len(array)-1):  # without day of the week
    one_hour_array[j] = str(array[j+1][index])
  return one_hour_array # and returns it
```

Fig. 47 Prepared schedule data interpretation v3

The method is created for more transparent combined array creation.

Then, the columns are placed in proper places:

```
1811            # DP1 classes are putted to the array
1812            additional = 0
1813            for day in range (len(days_of_the_week_array)):
1814              # because we do not want to set first found proper day of the week
1815              additional = additional + length_of_day[day] # an additional and deviation variables are added
1816              # So when the column is placed at first proper place
1817              deviation = 0
1818              # it disables the possibility to overlap columns
1819              for i in range(len(headers_DP1_array_Gathered)-1):
1820                # If day of the week equals the one in DP1
1821                if(days_of_the_week_array[day]==headers_DP1_array_Gathered[i]):
1822                  # Specific column array is gathered
1823                  temp_array = oneColumn(i+1, lessons_DP1_array_Gathered)
1824                  for k in range(len(temp_array)):
1825                    # Program adds DP 1 to distinguish them
1826                    whole_week_array_combined[k + 1][i +1+ additional-deviation] = str(temp_array[k]) + " DP 1"
1827                else:
1828                  deviation += 1
1829
1830            # DP2 classes are putted to the array
1831            for i in range(len(lessons_DP2_array_Gathered[0])-1):
1832              # Specific column array is gathered
1833              temp_array = oneColumn(i+1, lessons_DP2_array_Gathered)
1834              for k in range(len(whole_week_array_combined[0])-1):
1835                # and is placed in first empty place
1836                if ((whole_week_array_combined[1][k+1] == "")):
1837                  for j in range(len(temp_array)):
1838                    # Program adds DP 1 to distinguish them
1839                    whole_week_array_combined[j + 1][k + 1] = str(temp_array[j]) + " DP 2"
1840                  break
```

Fig. 48 Prepared schedule data interpretation v4

After creating the array of both DP1 and DP2 classes we are able to begin the classroom assignment.

Method which gathers specific hour at particular day is defined:

```
# CREATE A SEARCH ALGORITHM FOR EACH HOUR
def hourArray(hour_id, day):
  # Gathers all classes which are held
  this_hour_array = [] # for specific hour and day
  for i in range(len(whole_week_array_combined[0])):
    # If day of the week match
    if ((whole_week_array_combined[0][i] == days_of_the_week_array[day])
        #hour_id is the same for all defined arrays
        and not(whole_week_array_combined[hour_id+1][i] in [" DP 1", " DP 2", "nan DP 1", "nan DP 2"])): # excludes empty cells
      # appends the array with found classes
      this_hour_array.append(whole_week_array_combined[hour_id+1][i])
  return this_hour_array # and returns full array
# CREATE A SEARCH ALGORITHM FOR EACH HOUR
```

Fig. 48 Prepared schedule data interpretation v4

Finally, after the preparation of all data gathered by the program, the main algorithm is created. All process is conducted for 5 days of the week and 10 rows of available hours (our school's characteristics). For this part of the Program, please extra carefully familiarize yourself with comments in the code.

```
# Classroom to class assigning algorithm
for day in range(5):
  for hour in range(10):
    # Array with classes to assign
    hour_day_classroom_assigned = []
    # Gathered classes at specific hour
    hour_day_array_gather = hourArray(hour,day)
    # Available classrooms array       copied version of Classrooms array
    self.classrooms_avaiable_array = copy.deepcopy(self.classrooms)

    # Creates an array with all objects which seek assignment
    for i in range(len(hour_day_array_gather)):
      subject = hour_day_array_gather[i] # defines subject
      arrayOfData = findTeacherData(subject) # searches for teacher which teaches this subject
      teacherName = arrayOfData[0] # Teacher's name
      if (teacherName != "not found"): # For all teacehrs beside TOK teacehr
        # Gathers data associated with this class
        size = arrayOfData[1]
        screen_av = arrayOfData[2]
        classroomPreference = arrayOfData[3]
        #            appends the array                                                    classroom
        hour_day_classroom_assigned.append([subject, teacherName, screen_av, classroomPreference, size, ""])
      elif (teacherName == "not found"): # If it is a TOK teacehr
        for i in range(len(TOKTeacherData)):
          if (subject == TOKTeacherData[i][0]): # Searches for the proper TOK class
            teacherName = "TOK teacher"
            size = TOKTeacherData[i][1]
            classroomPreference = ""
            if isinstance(TOKTeacherClassroomPreferencearedSchedule, str): # If the classroom preference is string
              classroomPreference = TOKTeacherClassroomPreferencearedSchedule
            elif TOKTeacherClassroomPreferencearedSchedule: # If it is not a string
              classroomPreference = str(TOKTeacherClassroomPreferencearedSchedule) # changes to string
            #        appends the array                                                  classroom
            hour_day_classroom_assigned.append([subject, teacherName, True, classroomPreference, size, ""])
```

Fig. 49 Prepared schedule main algorithm v1

At the beginning of the algorithm, all data is arranged in familiar structure, therefore, operating on this amount of variables is easier and clearer.

Now, the proper assignment takes place:

```
1913            # Sorts the array with classes by size
1914            for i in range(len(hour_day_classroom_assigned)):
1915              for j in range(len(hour_day_classroom_assigned)-i-1):
1916                if (hour_day_classroom_assigned[j][4] < hour_day_classroom_assigned[j+1][4]):
1917                  hour_day_classroom_assigned[j], hour_day_classroom_assigned[j+1] = hour_day_classroom_assigned[j+1], hour_day_classroom_assigned[j]
1918
1919            # assign homehours
1920            for b in range(len(hour_day_classroom_assigned)):
1921              # Checks for classes which has homeroom hour
1922              for a in range(len(self.classrooms)):
1923                if (hour_day_classroom_assigned[b][0] == ("HR " + self.classrooms[a][3]) # in [3] data about homeroom hour is stored
1924                        and (hour_day_classroom_assigned[b][4] < self.classrooms[a][1])):
1925                  # If found homeroom hour is assigned
1926                  hour_day_classroom_assigned[b][5] = self.classrooms[a][0]
1927                  # Deletes assigned classroom, thus it is not able to
1928                  self.classrooms_avaiable_array[a][0] = "" # set class over there
1929                  break
```

Fig. 50 Prepared schedule main algorithm v2

Classrooms are sorted using bubble sort, and homeroom hours are searched and assigned as it is the most significant requirement of the Client.

```
1931            # Assign other classes by size
1932            for c in range (len(hour_day_classroom_assigned)):
1933              # If classroom is not yet assigned
1934              if (hour_day_classroom_assigned[c][5] == ""):
1935                # then searches for available classroom
1936                for m in range(len(self.classrooms_avaiable_array)):
1937                  # and if hour is available
1938                  if not(self.classrooms_avaiable_array[m][0] == ""):
1939                    # Assigns classroom and deletes it from available ones
1940                    hour_day_classroom_assigned[c][5] = self.classrooms_avaiable_array[m][0]
1941                    self.classrooms_avaiable_array[m][0] = ""
1942                    break
```

Fig. 51 Prepared schedule main algorithm v3

Then the classrooms are assigned by size, as it might occur that some classes are too large for the capabilities of the building and the Program should no longer trouble with them, as it seeks to assign classes.

Then, the process of trying to swap some classrooms by the preferences of the teachers:

```
# Algorithm which searches for a change in favourite classroom
for i in range(len(hour_day_classroom_assigned)):
  # If there is a need for looking
  if (hour_day_classroom_assigned[i][3] != "no preference"):
    # Stores which classroom does teacher seek to have class
    preferedClassroom = hour_day_classroom_assigned[i][3]
    # Changed variable
    successfulSearch = False
    # Classrooms which are empty
    for j in range(len(self.classrooms_avaiable_array)):
      # If favourite classroom is empty                                    and whether it is big enough
      if (self.classrooms_avaiable_array[j][0] == preferedClassroom) and (hour_day_classroom_assigned[i][4] < self.classrooms_avaiable_array[j][1]):
        # Classroom which will become empty
        previousClassroom = hour_day_classroom_assigned[i][5]
        # Change in classrooms
        hour_day_classroom_assigned[i][5] = self.classrooms_avaiable_array[j][0]
        # Classroom is no longer empty
        self.classrooms_avaiable_array[j][0] = ""
        # Searches for an empty place to put previous classroom
        for n in range(len(self.classrooms_avaiable_array)):
          if (self.classrooms_avaiable_array[n][0] == ""):
            self.classrooms_avaiable_array[n][0] = previousClassroom
            # Updates boolean variable
            successfulSearch = True
            break
        break
    if not (successfulSearch): # If not found, for efficiency
      for j in range(len(hour_day_classroom_assigned)):
        # Searches favourite classroom
        if (hour_day_classroom_assigned[j][5] == preferedClassroom):
          # Teacher's classroom size
          preferedClassroomSize = 0
          # Replacing with classroom size
          replacingClassroomSize = 0
          for m in range(len(self.classrooms)):
            # Searches for classrooms' sizes
            if (self.classrooms[m][0] == preferedClassroom):
              preferedClassroomSize = self.classrooms[m][1]
            elif(self.classrooms[m][0] == hour_day_classroom_assigned[j][5]):
              replacingClassroomSize = self.classrooms[m][1]
          # If both classes are small enough for change
          if (replacingClassroomSize > hour_day_classroom_assigned[i][4]) and (preferedClassroomSize > hour_day_classroom_assigned[j][4]) :
            # and it is not homeroom hour
            if not("HR" in hour_day_classroom_assigned[j][0]):
              # then they are swapped
              hour_day_classroom_assigned[i][5],  hour_day_classroom_assigned[j][5] = hour_day_classroom_assigned[j][5],  hour_day_classroom_assigned[i][5]
              break

# Exactly the same for Screen preference
for i in range(len(hour_day_classroom_assigned)):
```

Fig. 52 Prepared schedule main algorithm v4

I am aware that the last teacher in the array is in the best position to satisfy its preferences. Nevertheless, the program prevents from swapping already satisfied teachers. I do not show the swapping algorithm of screen availability preference as it is identical to the classroom preferences one.

Then, not used classrooms are saved:

```
2030            # Empty classrooms array
2031            self.empty_classroonms_array = []
2032            # Not used classrooms are put to the array
2033            for f in range(len(self.classrooms_avaiable_array)):
2034              if (self.classrooms_avaiable_array[f][0] != ""):
2035                self.empty_classroonms_array.append(self.classrooms_avaiable_array[f][0])
2036
2037            # Found data is stored in the arrays
2038            self.full_this_hour_arr[hour][day] = hour_day_classroom_assigned
2039            self.empty_classroonms[hour][day] = self.empty_classroonms_array
```

Fig. 53 Prepared schedule main algorithm v5

And the Program begins the final stage of schedule preparation which is gathering incompatibilities of the teachers.

```
2055        # Incompatibilities array is created
2056        self.full_notification_array = [[None for _ in range(5)] for _ in range(10)]
2057        # For all classes
2058        for day in range(5):
2059          for hour in range(10):
2060            # Notification array for specific cell is created
2061            notifications = []
2062            # Assigned classrooms array is gathered
2063            check_hour_problems = self.full_this_hour_arr[hour][day]
2064            if (check_hour_problems): # If array exists
2065              # For each earlier defined object
2066              for i in range(len(check_hour_problems)):
2067                # Teacher screen availability preference is gathered
2068                teacher_screen_preference = check_hour_problems[i][2]
2069                # and the classroom at which class is held
2070                teacher_assigned_classroom = check_hour_problems[i][5]
2071                # Searches for this classroom
2072                for j in range(len(self.classrooms)):
2073                  # If classroom is found                              and   teacher preference NOR classroom specifications
2074                  if (self.classrooms[j][0] == teacher_assigned_classroom) and not(teacher_screen_preference ^ self.classrooms[j][2]):
2075                    # Notification is added
2076                    text_of_notification = ("teacher: " + check_hour_problems[i][1] + " could not have a screen available")
2077                    notifications.append(text_of_notification)
2078                    break
2079              # For each earlier defined object
2080              for i in range(len(check_hour_problems)):
2081                # Teacher classroom preference is gathered
2082                teacher_clasroom_preference = check_hour_problems[i][3]
2083                # Searches for this classroom
2084                teacher_assigned_classroom = check_hour_problems[i][5]
2085                # If the class is not in favourite classroom
2086                if (teacher_clasroom_preference != teacher_assigned_classroom):
2087                  # Notification is added
2088                  text_of_notification = ("teacher: " + check_hour_problems[i][1] + " could not have classes at preferred classroom")
2089                  notifications.append(text_of_notification)
2090            # All notifications for specific hour are stored
2091            self.full_notification_array[hour][day] = notifications
```

Fig. 54 Prepared schedule data representation v1

Finally, method which creates prepared schedule is defined:

```
2273        # Method which creates Schedule with assigned classrooms
2274        def downloadPreparedSchedule():
2275          # Schedules are copied
2276          preparedDP1Schedule = copy.deepcopy(lessons_DP1_array_Gathered)
2277          preparedDP2Schedule = copy.deepcopy(lessons_DP2_array_Gathered)
2278          # Days of the week array is defined
2279          days_of_the_week_array = ['MONDAY', 'TUESDAY', 'WEDNESDAY', 'THURSDAY', 'FRIDAY']
2280          # For each cell
2281          for day in range(5): # 5 days of the week
2282            for hour in range(10): # 10 hours
2283              # Array with assigned object is gathered
2284              assigned_classrooms_to_classes = self.full_this_hour_arr[hour][day]
2285              # Day of the week variable
2286              day_of_the_week = days_of_the_week_array[day]
2287              # Searches for DP1 class
2288              for i in range(len(preparedDP1Schedule[0])):
2289                # If proper day of the week
2290                if (day_of_the_week == preparedDP1Schedule[0][i]):
2291                  # Class which seeks for classroom assignment
2292                  class_which_seeks_classroom = str(preparedDP1Schedule[hour+1][i])
2293                  # Searches for this class data
2294                  for j in range(len(assigned_classrooms_to_classes)):
2295                    # If it is searched class
2296                    if (class_which_seeks_classroom + " DP 1" == assigned_classrooms_to_classes[j][0]):
2297                      # classroom is assigned to class
2298                      subjects_classroom = assigned_classrooms_to_classes[j][5]
2299                      preparedDP1Schedule[hour+1][i] = class_which_seeks_classroom + " " + subjects_classroom
2300                      break
2301
2302              # Exactly the same for DP2
2303              for i in range(len(preparedDP2Schedule[0])):
2304                if (day_of_the_week == preparedDP2Schedule[0][i]):
2305                  class_which_seeks_classroom = str(preparedDP2Schedule[hour+1][i])
2306                  for j in range(len(assigned_classrooms_to_classes)):
2307                    if (class_which_seeks_classroom + " DP 2" == assigned_classrooms_to_classes[j][0]):
2308                      subjects_classroom = assigned_classrooms_to_classes[j][5]
2309                      preparedDP2Schedule[hour+1][i] = class_which_seeks_classroom + " " + subjects_classroom
2310                      break
```

Fig. 55 Prepared schedule data representation v2

And converted to Pandas requirements:

```
2312            # Array is converted into Pandas requirements
2313            preparedDP1Schedule = pandas.DataFrame(preparedDP1Schedule)
2314            preparedDP2Schedule = pandas.DataFrame(preparedDP2Schedule)
2315            # DP1 headers are stored
2316            headersDP1Prep = preparedDP1Schedule.iloc[0].tolist()
2317            # Remove first row from array as it is now header
2318            preparedDP1Schedule = preparedDP1Schedule[1:].reset_index(drop=True)
2319            # DP2 headers are stored
2320            headersDP2Prep = preparedDP2Schedule.iloc[0].tolist()
2321            # Remove first row from array as it is now header
2322            preparedDP2Schedule = preparedDP2Schedule[1:].reset_index(drop=True)
2323            # Headers are added
2324            preparedDP1Schedule.columns = headersDP1Prep
2325            preparedDP2Schedule.columns = headersDP2Prep
2326            # File name is defined
2327            fileName = "preparedSchedule.xlsx"
2328            try: # In case of error, so the program does not crack
2329              # Excel file with assigned classrooms is created
2330              with pandas.ExcelWriter(fileName, engine="xlsxwriter") as writer:
2331                preparedDP1Schedule.to_excel(writer, sheet_name="dp1", index=False)
2332                preparedDP2Schedule.to_excel(writer, sheet_name="dp2", index=False)
2333            except FileExistsError or Exception:
2334              self.errorComunicate("file already exist")
```

Fig. 56 Prepared schedule data representation v3

That is how classrooms are assigned to classes.

Building Situation subclass (crit. B - Section B)

Necessary transfer Buttons for Building Situation section are defined:

```
2243        # Transfer buttons to Building Situation subclass are defined
2244        GoToDay5hour10Button = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "hourF_10")
2245        GoToDay5hour10Button.clicked.connect(lambda: self.goToSituationInTheBuilding(4,9))
2246        # and the Download Prepared Schedule button
2247        downloadPreparedScheduleButton = self.currentlyVisibleWidget.findChild(QtWidgets.QPushButton, "DownloadSchedule")
2248        downloadPreparedScheduleButton.clicked.connect(lambda: downloadPreparedSchedule())
```

Fig. 57 Prepared schedule data representation v4

I decided to create Building Situation subclass as a method inside Prepared Schedule class to reduce the complexity of the Program. The method of subclass is defined:

```
2354    # Method which transfer to Building Situation subclass
2355    def goToSituationInTheBuilding(self, day, hour): # day and hour correspond to data in each button
2356        # Load the UI for the building situation
2357        building_ui = QtWidgets.QWidget()
2358        self.buildingSituationWidget = loader.load("database/windows/QBuildingSituation.ui", None)
2359
2360        # Set layout
2361        layout = QtWidgets.QVBoxLayout()
2362        layout.addWidget(self.buildingSituationWidget)
2363        building_ui.setLayout(layout)
2364
2365        # Adds the widget to the stacked widget (parent)
2366        self.parent().addWidget(building_ui)
2367
2368        # Switch to the new widget
2369        self.parent().setCurrentWidget(building_ui)
```

Fig. 58 Prepared schedule data representation v5

Then data is loaded to proper Labels

```
2493        # Similarly for empty classrooms array is gathered
2494        array_of_empty_classess = self.empty_classroonms[hour][day]
2495        if (array_of_empty_classess): # If array is not empty
2496          for i in range(len(array_of_empty_classess)):
2497            # Empty classrooms array is gathered
2498            emptyClassroomName = array_of_empty_classess[i]
2499            if (i == 0): # and proper Labels are updated
2500              emptyClassroom1NameLabel = self.buildingSituationWidget.findChild(QtWidgets.QLabel, "emptyClassroom1")
2501              emptyClassroom1NameLabel.setText(emptyClassroomName)
2502            elif (i == 1):
2503              emptyClassroom2NameLabel = self.buildingSituationWidget.findChild(QtWidgets.QLabel, "emptyClassroom2")
2504              emptyClassroom2NameLabel.setText(emptyClassroomName)
2505            elif (i == 2):
2506              emptyClassroom3NameLabel = self.buildingSituationWidget.findChild(QtWidgets.QLabel, "emptyClassroom3")
```

Fig. 59 Prepared schedule data representation v6

```
2372        array_of_classess = self.full_this_hour_arr[hour][day]
2373        if (array_of_classess): # If array is not empty
2374          # For each object
2375          for i in range(len(array_of_classess)):
2376            # Teacher name is gathered
2377            teacherName = array_of_classess[i][1]
2378            # Class name is gathered
2379            subjectName = array_of_classess[i][0]
2380            # Classroom name is gathered
2381            classroomName = array_of_classess[i][5]
2382            # and its size
2383            classroomSize = 0
2384            # Search for classroom data
2385            for j in range(len(self.classrooms)):
2386              if (self.classrooms[j][0] == classroomName):
2387                classroomSize = self.classrooms[j][1]
2388                break
2389            # Number of empty places is defined
2390            emptyPlaces = classroomSize - int(array_of_classess[i][4])
2391            if (i == 0):
2392              # Proper Labels are updated
2393              subject1NameLabel = self.buildingSituationWidget.findChild(QtWidgets.QLabel, "className1")
2394              subject1NameLabel.setText(subjectName)
2395              teacher1NameLabel = self.buildingSituationWidget.findChild(QtWidgets.QLabel, "teacherName1")
2396              teacher1NameLabel.setText(teacherName)
2397              classroom1NameLabel = self.buildingSituationWidget.findChild(QtWidgets.QLabel, "classroomName1")
2398              classroom1NameLabel.setText(classroomName)
2399              emptyPlaces1NameLabel = self.buildingSituationWidget.findChild(QtWidgets.QLabel, "classroomPlaces1")
2400              emptyPlaces1NameLabel.setText(str(emptyPlaces))
2401            elif (i == 1):
2402              subject2NameLabel = self.buildingSituationWidget.findChild(QtWidgets.QLabel, "className2")
2403              subject2NameLabel.setText(subjectName)
2404              teacher2NameLabel = self.buildingSituationWidget.findChild(QtWidgets.QLabel, "teacherName2")
```

Fig. 60 Prepared schedule data representation v7

```
2533        # The same as for empty classrooms
2534        array_of_notifications = self.full_notification_array[hour][day]
2535        if (array_of_notifications):
2536          for i in range(len(array_of_notifications)):
2537            notificationText = array_of_notifications[i]
2538            if (i == 0):
2539              notification1Label = self.buildingSituationWidget.findChild(QtWidgets.QLabel, "notification1")
2540              notification1Label.setText(notificationText)
2541            elif (i == 1):
2542              notification2Label = self.buildingSituationWidget.findChild(QtWidgets.QLabel, "notification2")
2543              notification2Label.setText(notificationText)
2544            elif (i == 2):
2545              notification3Label = self.buildingSituationWidget.findChild(QtWidgets.QLabel, "notification3")
```

Fig. 61 Prepared schedule data representation v8

Loading data to proper Labels is done in such inefficient way, because we never know how many empty/occupied classes or notifications we have.

The important aspect is that the Main class is defined at the end of the program, because all classes must be defined earlier, thus it is possible to later refer to them and transfer the user to the appropriate place.

In retrospect, I can see the ineffectiveness of some of the methods, but due to lack of time made me to leave them as they are.

Compression to .exe extension

I compress my Python Qt code to exe, hence it is possible to open the program on every computer. I move main it into the Scripts folder in the python environment and using:

the ./pyinstaller –onefile –windowed main.py

command I compressed the program into exe.

Word count: 1143

Bibliography

“1. Command Line and Environment.” *Python Documentation*, docs.python.org/3/using/cmdline.html.

“Convert Python Script to .Exe File.” *GeeksforGeeks*, 3 Feb. 2020, www.geeksforgeeks.org/convert-python-script-to-exe-file/.

“Introduction to Qt | Qt 6.8.” *Doc.qt.io*, 2025, doc.qt.io/qt-6/qt-intro.html.

“Learn Python GUI Development for Desktop – PySide6 and Qt Tutorial.” *Www.youtube.com*, www.youtube.com/watch?v=Z1N9JzNax2k.

“Pandas.read_excel — Pandas 1.1.4 Documentation.” *Pandas.pydata.org*, pandas.pydata.org/docs/reference/api/pandas.read_excel.html.

PYTHON. “Python.” *Python.org*, Python.org, 29 May 2019, www.python.org/.

“Qt Widgets 6.4.1.” *Doc.qt.io*, doc.qt.io/qt-6/qtwidgets-index.html.

“Widgets Classes | Qt Widgets 6.8.2.” *Doc.qt.io*, 2025, doc.qt.io/qt-6/widget-classes.html.

“Working with Excel Files Using Pandas.” *GeeksforGeeks*, 13 Feb. 2020, www.geeksforgeeks.org/working-with-excel-files-using-pandas/.