

Übung 8

Table of Contents

1. Lösungsidee	2
2. Tests.....	3
3. Verwendete Bibliotheken	8
4. Quellcode	9

1. Lösungsidee

Es gibt ein Hauptfenster, wo eine Übersicht aller Files angezeigt wird. Für die Detailansicht wird ein weiterer Dialog geöffnet, wo die Tabelle des Files entsprechend angezeigt wird. Dies funktioniert mittels Doppelklick auf die entsprechende Zeile. Dort wird dann eine Tabelle mit den Daten angezeigt und mittels Buttons können Werte gelöscht, aktualisiert (hier wird eigener Dialog geöffnet) bzw. Zeilen, Spalten entfernt werden. Für die Verwendung der `TableView` mit Pandas wurde ein eigenes Model erstellt, welches von `QAbstractTableModel` ableitet.

Files werden im Folder `uploaded_files` abgespeichert. Exportierte Files werden einfach in den Folder `export` im Projektverzeichnis abgespeichert.

Folgende Annahmen wurden für die Lösung getroffen:

- Es werden nur CSV-Files mit Beistrichen als Trennzeichen verwendet. Das Trennzeichen kann nicht umgestellt werden.

Für das Speichern, Validieren und Laden von Files wurden die Methoden aus den anderen Übungen übernommen.

Die GUI wurde mit dem Qt-Designer erstellt und wurden unter dem Ordner `ui` gespeichert.

2. Tests

Zusätzlich zum Hinzufügen aller Testfiles aus Moodle wurde ein kompletter Programmdurchlauf getestet und mit Screenshots dokumentiert.

Neues File "belgium.db" wird hinzugefügt.

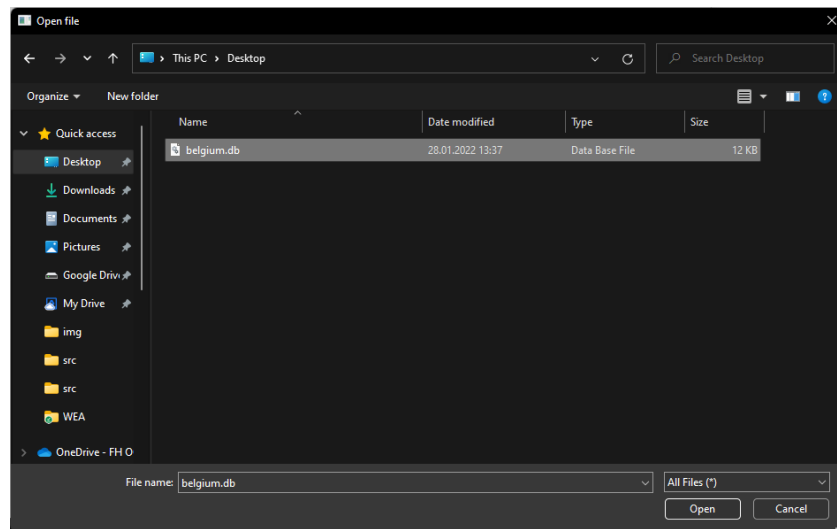


Figure 1. Hinzufügen eines neuen Files

Übersicht wurde aktualisiert um Eintrag "belgium.db".

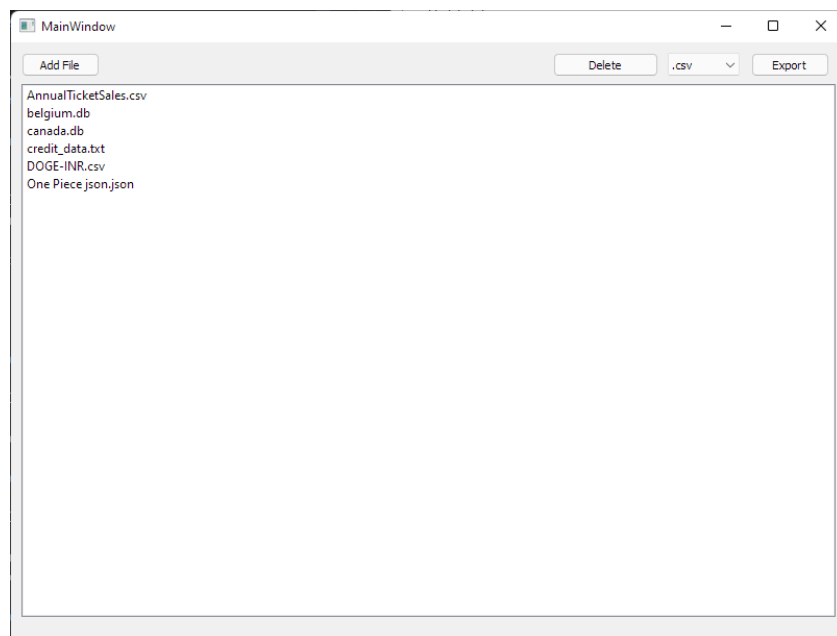


Figure 2. Übersicht nach Hinzufügen

File "AnnualTicketSales.csv" wird gelöscht.

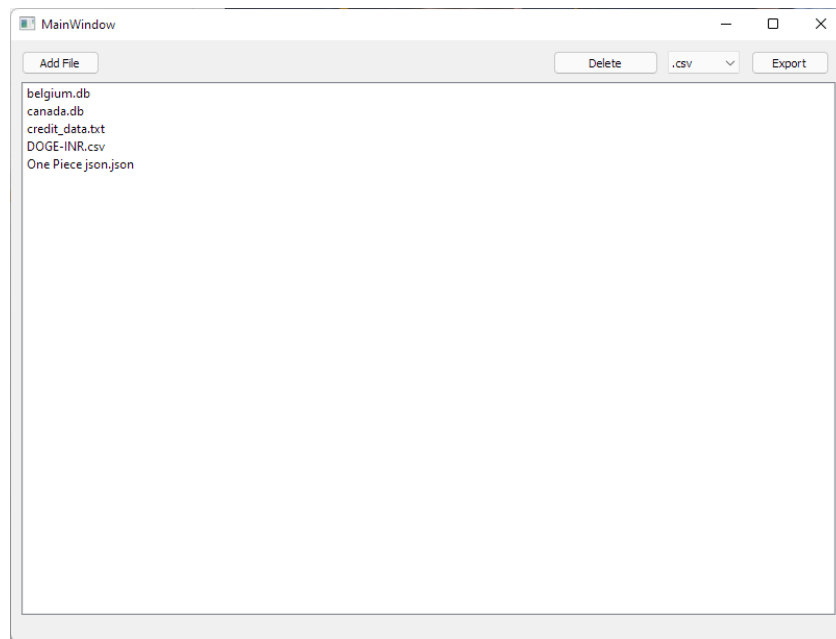


Figure 3. Übersicht nach Löschen

File "belgium.db" wird als csv exportiert.

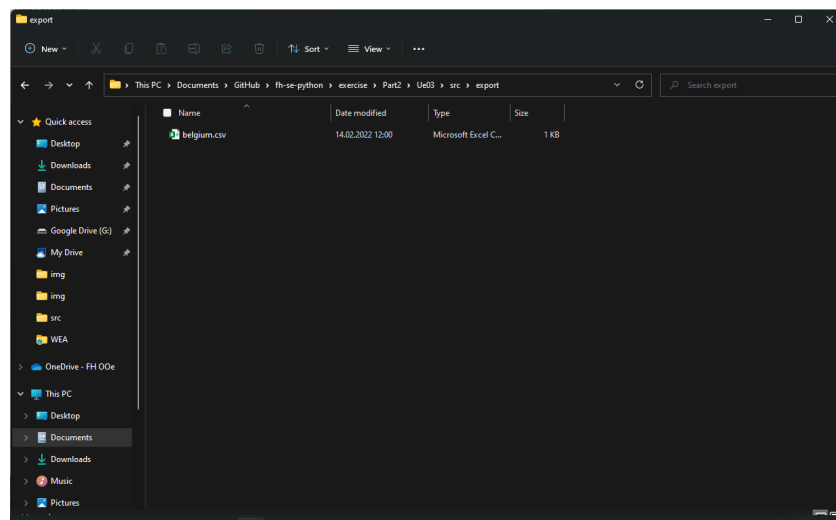
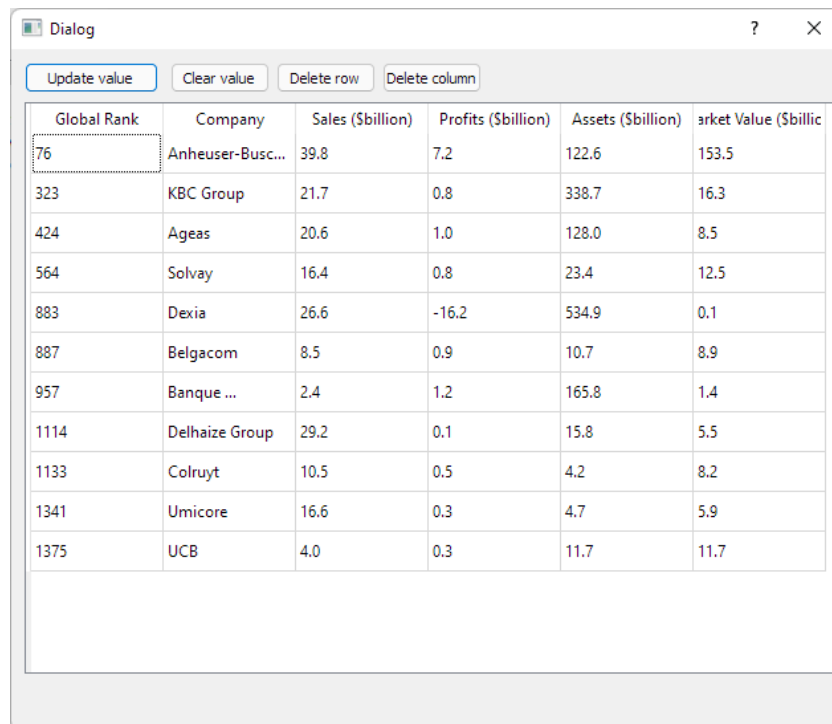


Figure 4. Export

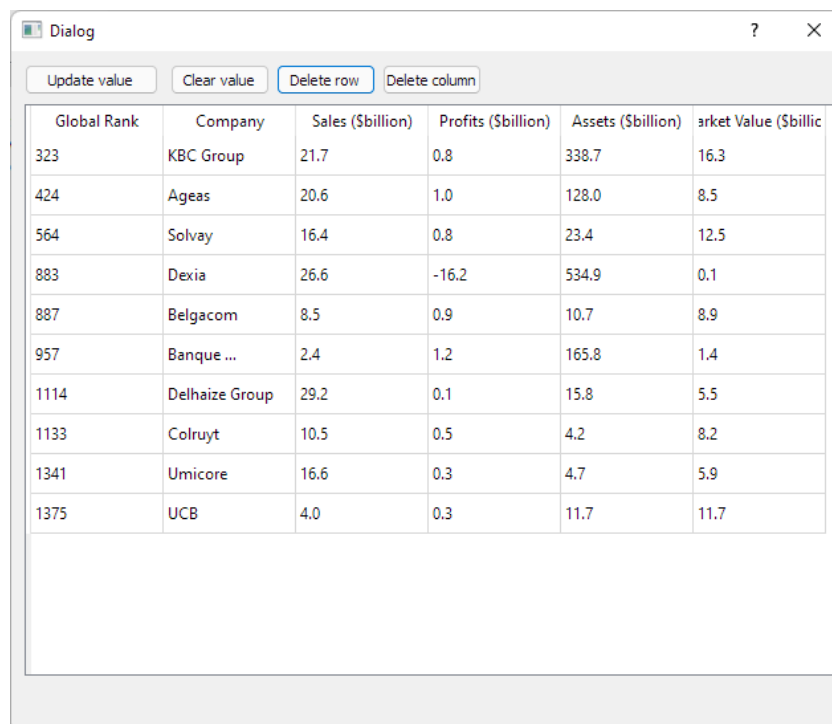
Detailansicht wird mit Doppelklick geöffnet.



Global Rank	Company	Sales (\$billion)	Profits (\$billion)	Assets (\$billion)	arket Value (\$billion)
76	Anheuser-Busc...	39.8	7.2	122.6	153.5
323	KBC Group	21.7	0.8	338.7	16.3
424	Ageas	20.6	1.0	128.0	8.5
564	Solvay	16.4	0.8	23.4	12.5
883	Dexia	26.6	-16.2	534.9	0.1
887	Belgacom	8.5	0.9	10.7	8.9
957	Banque ...	2.4	1.2	165.8	1.4
1114	Delhaize Group	29.2	0.1	15.8	5.5
1133	Colruyt	10.5	0.5	4.2	8.2
1341	Umicore	16.6	0.3	4.7	5.9
1375	UCB	4.0	0.3	11.7	11.7

Figure 5. Detailansicht

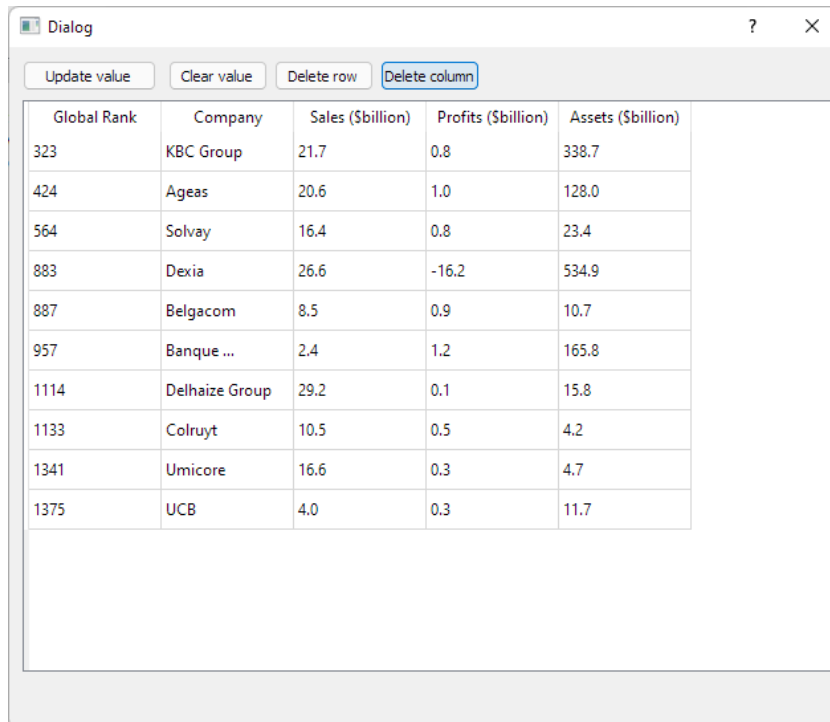
Erste Zeile wird gelöscht (Global Rank 76).



Global Rank	Company	Sales (\$billion)	Profits (\$billion)	Assets (\$billion)	arket Value (\$billion)
323	KBC Group	21.7	0.8	338.7	16.3
424	Ageas	20.6	1.0	128.0	8.5
564	Solvay	16.4	0.8	23.4	12.5
883	Dexia	26.6	-16.2	534.9	0.1
887	Belgacom	8.5	0.9	10.7	8.9
957	Banque ...	2.4	1.2	165.8	1.4
1114	Delhaize Group	29.2	0.1	15.8	5.5
1133	Colruyt	10.5	0.5	4.2	8.2
1341	Umicore	16.6	0.3	4.7	5.9
1375	UCB	4.0	0.3	11.7	11.7

Figure 6. Zeile wird gelöscht

Letzte Spalte wird entfernt (Market Value).



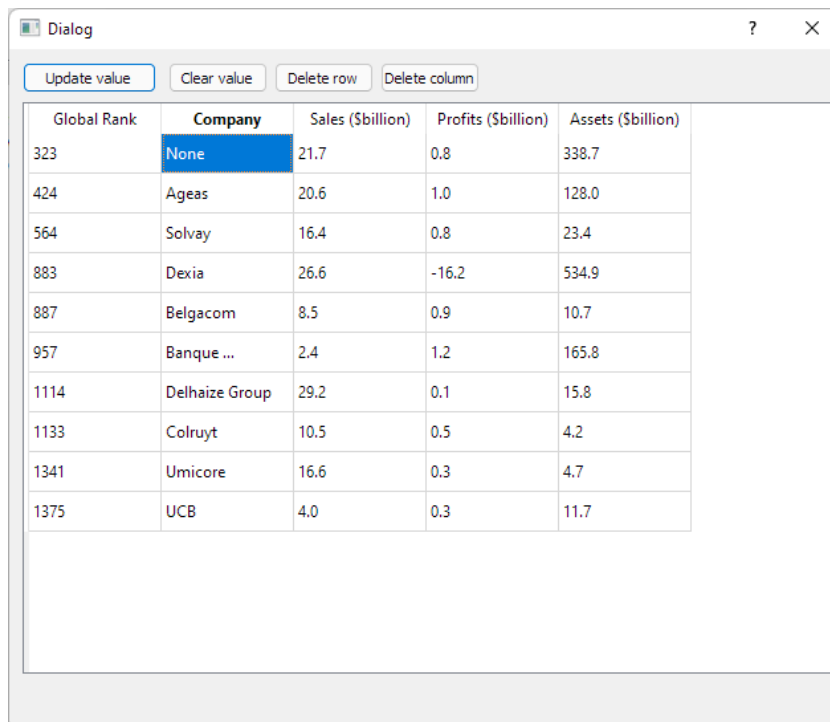
Dialog

Update value Clear value Delete row **Delete column**

Global Rank	Company	Sales (\$billion)	Profits (\$billion)	Assets (\$billion)
323	KBC Group	21.7	0.8	338.7
424	Ageas	20.6	1.0	128.0
564	Solvay	16.4	0.8	23.4
883	Dexia	26.6	-16.2	534.9
887	Belgacom	8.5	0.9	10.7
957	Banque ...	2.4	1.2	165.8
1114	Delhaize Group	29.2	0.1	15.8
1133	Colruyt	10.5	0.5	4.2
1341	Umicore	16.6	0.3	4.7
1375	UCB	4.0	0.3	11.7

Figure 7. Zeile wird gelöscht

Company-Name der ersten Zeile wird gelöscht.



Dialog

Update value Clear value Delete row Delete column

Global Rank	Company	Sales (\$billion)	Profits (\$billion)	Assets (\$billion)
323	None	21.7	0.8	338.7
424	Ageas	20.6	1.0	128.0
564	Solvay	16.4	0.8	23.4
883	Dexia	26.6	-16.2	534.9
887	Belgacom	8.5	0.9	10.7
957	Banque ...	2.4	1.2	165.8
1114	Delhaize Group	29.2	0.1	15.8
1133	Colruyt	10.5	0.5	4.2
1341	Umicore	16.6	0.3	4.7
1375	UCB	4.0	0.3	11.7

Figure 8. Zellenwert wird entfernt

Zellenwert von Global-Rank der ersten Zeile wird aktualisiert.

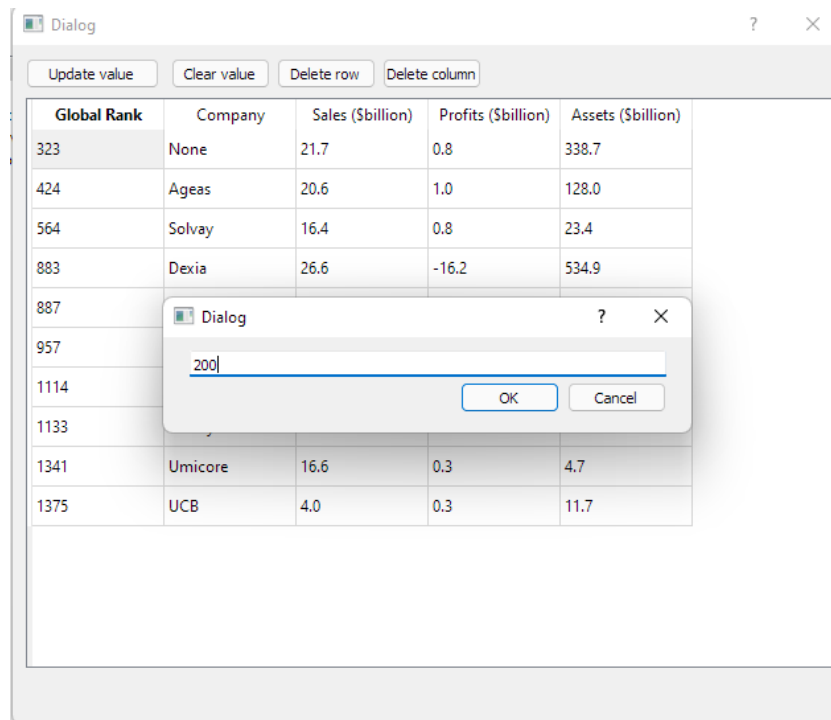


Figure 9. Edit-Dialog

Daten in der ersten Zeile wurden erfolgreich aktualisiert.

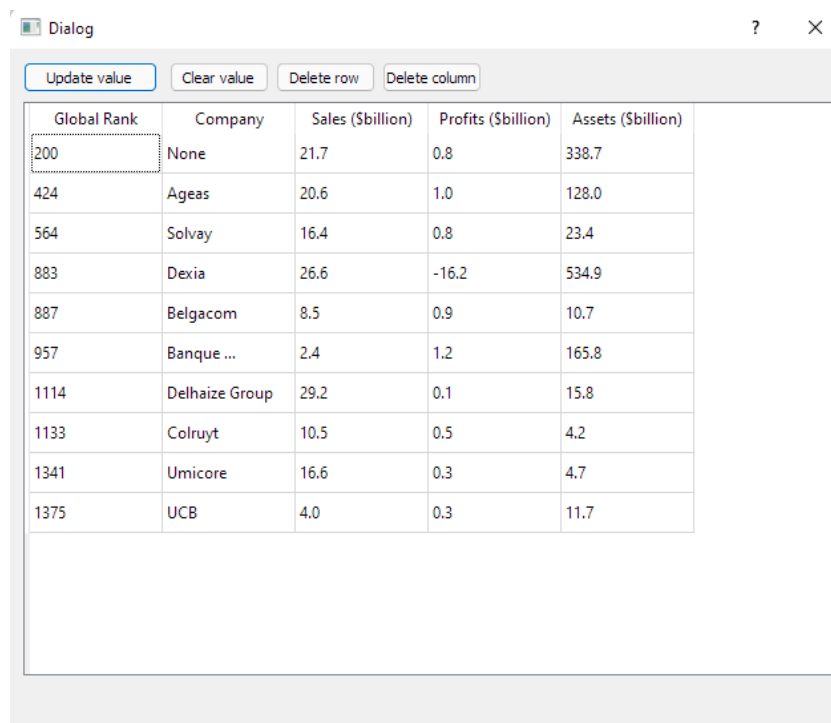


Figure 10. Detailansicht mit aktualisiertem Wert

3. Verwendete Bibliotheken

Listing 1. requirements.txt

```
pip~=21.1.2  
wheel~=0.36.2  
PyQt5~=5.15.6  
setuptools~=57.0.0  
pandas~=1.4.1
```


4. Quellcode

Listing 2. app.py

```
import os
import pathlib

from PyQt5 import QtCore, QtWidgets
from PyQt5.QtWidgets import QFileDialog, QFileDialog

import detail_dialog
import file_helper

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(800, 577)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.listView = QtWidgets.QListWidget(self.centralwidget)
        self.listView.setGeometry(QtCore.QRect(10, 40, 781, 511))
        self.listView.setObjectName("lvFile")
        self.listView.doubleClicked.connect(self.show_details)
        self.btnAddFile = QtWidgets.QPushButton(self.centralwidget, clicked=lambda: self.add())
        self.btnAddFile.setGeometry(QtCore.QRect(10, 10, 75, 23))
        self.btnAddFile.setObjectName("btnAddFile")
        self.btnDelete = QtWidgets.QPushButton(self.centralwidget, clicked=lambda: self.
delete())
        self.btnDelete.setGeometry(QtCore.QRect(520, 10, 101, 23))
        self.btnDelete.setObjectName("btnDelete")
        self.btnExport = QtWidgets.QPushButton(self.centralwidget, clicked=lambda: self.
export())
        self.btnExport.setGeometry(QtCore.QRect(710, 10, 75, 23))
        self.btnExport.setObjectName("btnExport")
        self.comboBox = QtWidgets.QComboBox(self.centralwidget)
        self.comboBox.setGeometry(QtCore.QRect(630, 10, 69, 22))
        self.comboBox.setObjectName("comboBox")
        MainWindow.setCentralWidget(self.centralwidget)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)
        self.init()

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.btnAddFile.setText(_translate("MainWindow", "Add File"))
        self.btnDelete.setText(_translate("MainWindow", "Delete"))
        self.btnExport.setText(_translate("MainWindow", "Export"))

    def init(self):
        self.update_list()
        self.comboBox.addItem(file_helper.filetypes)

    def update_list(self):
        self.listView.clear()
```

```
self.listView.addItem(file_helper.get_files())

def export(self):
    if self.listView.currentItem() is not None:
        file_type = self.comboBox.currentText()
        source_filename = self.listView.currentItem().text()

        filename = os.path.join(file_helper.output_directory, source_filename)
        filepath = pathlib.Path(filename)
        source_filetype = filepath.suffix
        df = file_helper.parse_file(filename, source_filetype)
        target_file_name = f"{filepath.stem}.{file_type}"
        file_helper.get_dataframe(source_filename)
        file_helper.save_file(df, target_file_name, file_type, target_directory='export')

def add(self):
    full_filename = QFileDialog.getOpenFileName(self.btnAddFile, 'Open file')[0]

    try:
        if full_filename == '':
            raise ValueError('No file specified')
        else:
            filepath = pathlib.Path(full_filename)
            target_filename = filepath.name
            file_type = filepath.suffix
            if file_type not in file_helper.filetypes:
                raise ValueError(f'File {target_filename} has invalid type')
            df = file_helper.parse_file(full_filename, file_type)
            file_helper.save_file(df, target_filename, file_type)
            self.update_list()
    except ValueError as e:
        print(f'Error: {e}')
    except BaseException as e:
        print(f'Exception details: {e}')

def delete(self):
    if self.listView.currentItem() is not None:
        file_helper.remove_file(self.listView.currentItem().text())
        self.update_list()

def show_details(self):
    dialog = QDialog()
    dialog.ui = detail_dialog.Ui_Dialog(self.listView.currentItem().text())
    dialog.ui.setupUi(dialog)
    dialog.exec_()

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Listing 3. detail_dialog.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file './detail_dialog.ui'
#
# Created by: PyQt5 UI code generator 5.15.6
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again. Do not edit this file unless you know what you are doing.
import os
import pathlib
import sys

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QAbstractItemView, QDialog

import edit_cell_dialog
import file_helper
from pandas_model import PandasModel

class Ui_Dialog(object):

    def __init__(self, filename):
        self.filename = filename

    def setupUi(self, Dialog):
        Dialog.setObjectName("File details")
        Dialog.resize(629, 514)
        self.tableView = QtWidgets.QTableView(Dialog)
        self.tableView.setGeometry(QtCore.QRect(10, 40, 611, 431))
        self.tableView.setObjectName("tableView")
        self.tableView.setSelectionMode(QAbstractItemView.SelectionMode.SingleSelection)
        self.tableView.doubleClicked.connect(self.update_value)
        self.btnUpdateValue = QtWidgets.QPushButton(Dialog, clicked=lambda: self.update_value())
        self.btnUpdateValue.setGeometry(QtCore.QRect(10, 10, 101, 23))
        self.btnUpdateValue.setObjectName("btnUpdateValue")
        self.btnDeleteValue = QtWidgets.QPushButton(Dialog, clicked=lambda: self.delete_value())
        self.btnDeleteValue.setGeometry(QtCore.QRect(120, 10, 75, 23))
        self.btnDeleteValue.setObjectName("btnDeleteValue")
        self.btnDeleteRow = QtWidgets.QPushButton(Dialog, clicked=lambda: self.delete_row())
        self.btnDeleteRow.setGeometry(QtCore.QRect(200, 10, 75, 23))
        self.btnDeleteRow.setObjectName("btnDeleteRow")
        self.btnDeleteColumn = QtWidgets.QPushButton(Dialog, clicked=lambda: self.
delete_column())
        self.btnDeleteColumn.setGeometry(QtCore.QRect(280, 10, 75, 23))
        self.btnDeleteColumn.setObjectName("btnDeleteColumn")

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)
        self.update_data()

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dialog"))
        self.btnUpdateValue.setText(_translate("Dialog", "Update value"))
        self.btnDeleteValue.setText(_translate("Dialog", "Clear value"))
        self.btnDeleteRow.setText(_translate("Dialog", "Delete row"))
        self.btnDeleteColumn.setText(_translate("Dialog", "Delete column"))

    def update_data(self):
```

```
full_filename = os.path.join(file_helper.output_directory, self.filename)
filepath = pathlib.Path(full_filename)
file_type = filepath.suffix
df = file_helper.parse_file(full_filename, file_type)

model = PandasModel(df)
self.tableView.setModel(model)

def delete_value(self):
    indexes = self.tableView.selectionModel().selectedIndexes()

    if len(indexes) > 0:
        index = indexes[0]
        file_helper.update_cell_value(self.filename, index.row(), index.column(), None)
        self.update_data()

def delete_row(self):
    indexes = self.tableView.selectionModel().selectedIndexes()

    if len(indexes) > 0:
        index = indexes[0]
        file_helper.delete_col_or_row(self.filename, index.row(), is_row=True)
        self.update_data()

def delete_column(self):
    indexes = self.tableView.selectionModel().selectedIndexes()

    if len(indexes) > 0:
        index = indexes[0]
        file_helper.delete_col_or_row(self.filename, index.column(), is_row=False)
        self.update_data()

def update_value(self):
    indexes = self.tableView.selectionModel().selectedIndexes()

    if len(indexes) > 0:
        index = indexes[0]
        value = self.tableView.model().index(index.row(), index.column()).data()
        dialog = QDialog()
        edit_cell_dialog_object = edit_cell_dialog.Ui_Dialog(value)
        dialog.ui = edit_cell_dialog_object
        dialog.ui.setupUi(dialog)

        if dialog.exec_():
            file_helper.update_cell_value(self.filename, index.row(), index.column(),
            edit_cell_dialog_object.updated_text())
            self.update_data()
```

Listing 4. `edit_cell_dialog.py`

```
from PyQt5 import QtCore, QtWidgets

class Ui_Dialog(object):

    def __init__(self, value):
        self.value = value

    def setupUi(self, Dialog):
        Dialog.setObjectName("Edit cell value")
        Dialog.resize(400, 72)
        self.dialogActions = QtWidgets.QDialogButtonBox(Dialog)
        self.dialogActions.setGeometry(QtCore.QRect(40, 30, 341, 32))
        self.dialogActions.setOrientation(QtCore.Qt.Horizontal)
        self.dialogActions.setStandardButtons(QtWidgets.QDialogButtonBox.Cancel|QtWidgets
.QDialogButtonBox.Ok)
        self.dialogActions.setObjectName("dialogActions")
        self.txtValue = QtWidgets.QLineEdit(Dialog)
        self.txtValue.setGeometry(QtCore.QRect(20, 10, 361, 20))
        self.txtValue.setObjectName("txtValue")
        self.txtValue.setText(self.value)

        self.retranslateUi(Dialog)
        self.dialogActions.accepted.connect(Dialog.accept) # type: ignore
        self.dialogActions.rejected.connect(Dialog.reject) # type: ignore
        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dialog"))

    def updated_text(self):
        return self.txtValue.text()
```

Listing 5. *pandas_model.py*

```
from PyQt5.QtCore import QAbstractTableModel, Qt

class PandasModel(QAbstractTableModel):

    def __init__(self, data):
        QAbstractTableModel.__init__(self)
        self._data = data

    def rowCount(self, parent=None):
        return self._data.shape[0]

    def columnCount(self, parent=None):
        return self._data.shape[1]

    def data(self, index, role=Qt.DisplayRole):
        if index.isValid():
            if role == Qt.DisplayRole:
                return str(self._data.iloc[index.row(), index.column()])
        return None

    def headerData(self, col, orientation, role):
        if orientation == Qt.Horizontal and role == Qt.DisplayRole:
            return self._data.columns[col]
        return None
```

Listing 6. *file_helper.py*

```
import os
import pathlib
import sqlite3

import pandas as pd
from pandas import DataFrame

output_directory = "uploaded_files"
filetypes = (".csv", ".txt", ".json", ".db")

def remove_file(filename):
    full_filename = os.path.join(output_directory, filename)
    if os.path.exists(full_filename):
        os.remove(full_filename)

def get_dataframe(filename):
    full_filename = os.path.join(output_directory, filename)
    file_path = pathlib.Path(full_filename)
    return parse_file(full_filename, file_path.suffix)

def save_file(df: DataFrame, filename: str, file_type: str, table_name: str = None,
              target_directory: str = None):

    if target_directory is None:
        target_directory = output_directory
```

```
if not os.path.exists(target_directory):
    os.mkdir(target_directory)

if file_type == '.csv' or file_type == '.txt':
    df.to_csv(os.path.join(target_directory, filename), index=False)
elif file_type == '.json':
    df.to_json(os.path.join(target_directory, filename), orient="records")
else:
    conn = sqlite3.connect(os.path.join(target_directory, filename))

    # Generate table name
    if table_name is None:
        table_name = filename.removesuffix(file_type)

    df.to_sql(table_name, con=conn, index=False, if_exists='replace')
    conn.close()

def get_sqlite_table_name(filename, conn) -> str:
    c = conn.cursor()
    c.execute(f"SELECT name FROM sqlite_master WHERE type='table';")
    results = c.fetchall()
    if len(results) == 0:
        conn.close()
        raise ValueError(f"{filename} has no table")
    elif len(results) == 2:
        conn.close()
        raise ValueError(f"{filename} has more than one table")
    # Get first column in first row
    return results[0][0]

def parse_sqlite(filename: str) -> DataFrame:
    conn = sqlite3.connect(filename)
    table_name = get_sqlite_table_name(filename, conn)

    df = pd.read_sql(f"select * from {table_name}", con=conn)
    conn.close()
    return df

def parse_csv(filename: str, separator: str) -> DataFrame:
    return pd.read_csv(filename, encoding="latin-1", delimiter=separator)

def parse_json(filename) -> DataFrame:
    return pd.read_json(filename, encoding="latin-1", orient='records')

def parse_file(name: str, file_type: str) -> DataFrame:
    if not os.path.isfile(name):
        raise ValueError('File does not exist')

    if file_type == '.csv' or file_type == '.txt':
        return parse_csv(name, separator=',')
    elif file_type == '.json':
        return parse_json(name)
    else:
        return parse_sqlite(name)
```

```
def get_files():  
  
    if not os.path.exists(output_directory):  
        return []  
  
    return [f for f in os.listdir(output_directory) if os.path.isfile(os.path.join  
(output_directory, f))]  
  
def get_cell_value(filename: str, row_index: int, column_index: int):  
    full_filename = os.path.join(output_directory, filename)  
    file_path = pathlib.Path(full_filename)  
    df = parse_file(full_filename, file_path.suffix)  
  
    return df.iloc[row_index, column_index]  
  
def update_cell_value(filename: str, row_index: int, column_index: int, value):  
    full_filename = os.path.join(output_directory, filename)  
    file_path = pathlib.Path(full_filename)  
    file_type = file_path.suffix  
    df = parse_file(full_filename, file_path.suffix)  
  
    df.iloc[int(row_index), int(column_index)] = value  
  
    if file_type == '.db':  
        conn = sqlite3.connect(full_filename)  
        table_name = get_sqlite_table_name(full_filename, conn)  
        conn.close()  
    else:  
        table_name = None  
  
    save_file(df, filename, file_type, table_name)  
  
def delete_col_or_row(filename: str, index: int, is_row: bool):  
    full_filename = os.path.join(output_directory, filename)  
    file_path = pathlib.Path(full_filename)  
    file_type = file_path.suffix  
    df = parse_file(full_filename, file_path.suffix)  
  
    if is_row:  
        df.drop([index], inplace=True)  
    else:  
        df.drop(df.columns[index], axis=1, inplace=True)  
  
    # Get table name if necessary  
    if file_type == '.db':  
        conn = sqlite3.connect(full_filename)  
        table_name = get_sqlite_table_name(full_filename, conn)  
        conn.close()  
    else:  
        table_name = None  
  
    save_file(df, filename, file_type, table_name)  
  
    return df
```