# Сборник Python файлов

## ../mainwindow.py

import os  
import sqlite3  
import sys  
from pathlib import Path  
  
from PySide6.QtCore import QFile, QTextStream  
from PySide6.QtSql import QSqlDatabase  
from PySide6.QtWidgets import (  
 QApplication,  
 QDialog,  
 QMainWindow,  
 QMessageBox,  
 QTabWidget,  
)  
  
from create\_db import create\_db  
from accept\_app.accept\_invoice\_registry import AcceptInvoiceRegistry  
from accept\_app.accept\_invoice\_viewer import AcceptInvoiceWidget  
from accept\_app.accept\_request\_registry import AcceptRequestRegistry  
from accept\_app.accept\_request\_viewer import AcceptRequestWidget  
from contracts\_app.contract\_widget import ContractWidget  
from contracts\_app.contracts\_registry import ContractRegistry  
from interfaces.ui\_auth\_dialog import Ui\_Auth\_dialog  
from interfaces.ui\_mainwindow import Ui\_MainWindow  
from invoice\_app.invoice\_registry\_widget import InvoiceRegistryWidget  
from invoice\_app.invoice\_widget import InvoiceWidget  
from reports.reports\_widget import ReportWidget  
from requests\_app.request\_registry\_widget import RequestRegistryWidget  
from requests\_app.request\_widget import RequestWidget  
from users.user\_registry\_widget import UserRegistryWidget  
from users.user\_widget import UserWidget  
from vender\_app.vender\_registry import VenderRegistry  
from vender\_app.vender\_widget import VenderWidget  
  
BASE\_DIR = Path(\_\_file\_\_).resolve().parent  
  
  
class AuthDialog(QDialog):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.ui = Ui\_Auth\_dialog()  
 self.ui.setupUi(self)  
  
 self.ui.buttonBox.accepted.disconnect()  
 self.ui.buttonBox.accepted.connect(self.set\_current\_user)  
  
 def check\_login\_and\_password(self, login, password):  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 user = cur.execute("""  
 SELECT id, login, password, purchaser   
 FROM Users   
 WHERE login=? AND password=?;  
 """, (login, password)).fetchone()  
 con.close()  
 return user  
  
 def set\_current\_user(self):  
 login = self.ui.lineEdit.text()  
 password = self.ui.lineEdit\_2.text()  
 user = self.check\_login\_and\_password(login, password)  
 if not user:  
 QMessageBox.warning(self, "Предупреждение", "Неверный логин или пароль")  
 return  
 self.user = login  
 self.purchaser = user[3]  
 self.id = user[0]  
 self.accept()  
  
  
class MainWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.database\_file = os.path.join(BASE\_DIR, "database.db")  
 if not os.path.isfile(self.database\_file):  
 create\_db(self.database\_file)  
 dialog = AuthDialog(self)  
 if dialog.exec() == QDialog.Accepted:  
 self.user\_login = dialog.user  
 self.purchaser = dialog.purchaser  
 self.user\_id = dialog.id  
 else:  
 sys.exit()  
 self.db = QSqlDatabase.addDatabase("QSQLITE")  
 self.db.setDatabaseName("database.db")  
 self.db.open()  
  
 self.ui = Ui\_MainWindow()  
 self.ui.setupUi(self)  
 self.status\_bar = self.statusBar()  
  
 if not self.purchaser:  
 self.ui.groupBox\_3.hide()  
 self.ui.groupBox\_4.hide()  
 self.ui.create\_invoice\_btn.hide()  
 self.ui.invoice\_registry\_btn.hide()  
 self.ui.contract\_registry\_btn.hide()  
  
 self.tab\_widget = QTabWidget()  
 self.tab\_widget.setTabBarAutoHide(False)  
 self.tab\_widget.setTabsClosable(True)  
 self.ui.horizontalLayout.addWidget(self.tab\_widget)  
  
 self.tab\_widget.tabCloseRequested.connect(lambda i: self.tab\_widget.removeTab(i))  
 self.ui.create\_request\_btn.clicked.connect(self.open\_request\_creation)  
 self.ui.request\_registry\_btn.clicked.connect(self.open\_request\_registry)  
 self.ui.user\_registry\_btn.clicked.connect(self.open\_user\_registry)  
 self.ui.accept\_request\_btn.clicked.connect(self.open\_accept\_request\_registry)  
 self.ui.create\_invoice\_btn.clicked.connect(self.open\_invoice\_creation)  
 self.ui.invoice\_registry\_btn.clicked.connect(self.open\_invoice\_registry)  
 self.ui.accept\_invoice\_btn.clicked.connect(self.open\_accept\_invoice\_registry)  
 self.ui.reports\_btn.clicked.connect(self.open\_reports\_widget)  
 self.ui.contract\_registry\_btn.clicked.connect(self.open\_contract\_registry)  
 self.ui.vendor\_btn.clicked.connect(self.open\_vender\_registry)  
 self.ui.exit\_btn.clicked.connect(self.exit\_app)  
  
 def exit\_app(self):  
 ans = QMessageBox.warning(self, "Предупреждение", "Вы хотите выйти из приложения?", QMessageBox.Yes | QMessageBox.No)  
 if ans == QMessageBox.Yes:  
 sys.exit()  
  
 def open\_request\_creation(self):  
 request\_creation\_widget = RequestWidget(self)  
 self.tab\_widget.addTab(request\_creation\_widget, "Создание заявки")  
 self.tab\_widget.setCurrentWidget(request\_creation\_widget)  
  
 def open\_request\_registry(self):  
 request\_registry\_widget = RequestRegistryWidget(self)  
 self.tab\_widget.addTab(request\_registry\_widget, "Реестр заявок")  
 self.tab\_widget.setCurrentWidget(request\_registry\_widget)  
  
 def open\_request\_creation\_with\_data(self, request\_id):  
 request\_creation\_widget = RequestWidget(self)  
 request\_creation\_widget.load\_request\_data(request\_id)  
 self.tab\_widget.addTab(request\_creation\_widget, f"Заявка {request\_id}")  
 self.tab\_widget.setCurrentWidget(request\_creation\_widget)  
  
 def open\_user\_creation(self):  
 user\_creation\_widget = UserWidget(self)  
 self.tab\_widget.addTab(user\_creation\_widget, "Создание пользователя")  
 self.tab\_widget.setCurrentWidget(user\_creation\_widget)  
  
 def open\_user\_creation\_with\_data(self, user\_id):  
 user\_creation\_widget = UserWidget(self)  
 user\_creation\_widget.load\_data(user\_id)  
 self.tab\_widget.addTab(user\_creation\_widget, f"Пользователь {user\_id}")  
 self.tab\_widget.setCurrentWidget(user\_creation\_widget)  
  
 def open\_user\_registry(self):  
 request\_registry\_widget = UserRegistryWidget(self)  
 self.tab\_widget.addTab(request\_registry\_widget, "Реестр пользователей")  
 self.tab\_widget.setCurrentWidget(request\_registry\_widget)  
  
 def close\_current\_tab(self):  
 self.tab\_widget.removeTab(self.tab\_widget.currentIndex())  
  
 def open\_accept\_request\_registry(self):  
 accept\_request\_registry\_widget = AcceptRequestRegistry(self)  
 self.tab\_widget.addTab(accept\_request\_registry\_widget, "Согласовать заявки")  
 self.tab\_widget.setCurrentWidget(accept\_request\_registry\_widget)  
  
 def open\_invoice\_registry(self):  
 w = InvoiceRegistryWidget(self)  
 self.tab\_widget.addTab(w, "Реестр счетов")  
 self.tab\_widget.setCurrentWidget(w)  
  
 def open\_invoice\_creation(self):  
 w = InvoiceWidget(self)  
 self.tab\_widget.addTab(w, "Создание счета")  
 self.tab\_widget.setCurrentWidget(w)  
   
 def open\_invoice\_creation\_with\_data(self, invoice\_id):  
 w = InvoiceWidget(self)  
 w.load\_invoice\_data(invoice\_id)  
 self.tab\_widget.addTab(w, f"Счет {invoice\_id}")  
 self.tab\_widget.setCurrentWidget(w)  
  
 def open\_accept\_invoice\_registry(self):  
 w = AcceptInvoiceRegistry(self)  
 self.tab\_widget.addTab(w, "Согласовать счета")  
 self.tab\_widget.setCurrentWidget(w)  
  
 def open\_reports\_widget(self):  
 w = ReportWidget(self)  
 self.tab\_widget.addTab(w, "Отчеты")  
 self.tab\_widget.setCurrentWidget(w)  
  
 def open\_contract\_registry(self):  
 w = ContractRegistry(self)  
 self.tab\_widget.addTab(w, "Реестр договоров")  
 self.tab\_widget.setCurrentWidget(w)  
  
 def open\_contract\_creation(self):  
 w = ContractWidget(self)  
 self.tab\_widget.addTab(w, "Создание договора")  
 self.tab\_widget.setCurrentWidget(w)  
  
 def open\_contract\_with\_data(self, contract\_id):  
 w = ContractWidget(self, contract\_id)  
 self.tab\_widget.addTab(w, f"Договор {contract\_id}")  
 self.tab\_widget.setCurrentWidget(w)  
  
 def open\_vender\_registry(self):  
 w = VenderRegistry(self)  
 self.tab\_widget.addTab(w, "Реестр контрагентов")  
 self.tab\_widget.setCurrentWidget(w)  
  
 def open\_vender\_creation(self):  
 w = VenderWidget(self)  
 self.tab\_widget.addTab(w, "Создание контрагента")  
 self.tab\_widget.setCurrentWidget(w)  
  
 def open\_vender\_with\_data(self, vender\_id):  
 w = VenderWidget(self, vender\_id)  
 self.tab\_widget.addTab(w, f"Контрагент {vender\_id}")  
 self.tab\_widget.setCurrentWidget(w)  
  
 def open\_accept\_request\_viewer(self, request\_id):  
 w = AcceptRequestWidget(self, request\_id)  
 self.tab\_widget.addTab(w, f"Согласование Заявки {request\_id}")  
 self.tab\_widget.setCurrentWidget(w)  
  
 def open\_accept\_invoice\_viewer(self, invoice\_id):  
 w = AcceptInvoiceWidget(self, invoice\_id)  
 self.tab\_widget.addTab(w, f"Согласование Счета {invoice\_id}")  
 self.tab\_widget.setCurrentWidget(w)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 file = QFile("style.qss")  
 file.open(QFile.ReadOnly | QFile.Text)  
 qss = QTextStream(file)  
 app.setStyleSheet(qss.readAll())  
  
 ex = MainWindow()  
 ex.showFullScreen()  
 sys.exit(app.exec())

## ../create\_db.py

import sqlite3  
  
  
def create\_db(filepath):  
 con = sqlite3.connect(filepath)  
  
 cur = con.cursor()  
  
 cur.executescript("""  
 CREATE TABLE Access\_rights (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 invoice BOOLEAN,  
 requests BOOLEAN  
 );  
  
 CREATE TABLE Users (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 first\_name VARCHAR,  
 second\_name VARCHAR,  
 third\_name VARCHAR,  
 position VARCHAR,  
 login VARCHAR UNIQUE,  
 password VARCHAR,  
 access\_right\_id INTEGER,  
 FOREIGN KEY (access\_right\_id) REFERENCES Access\_rights(id)  
 );  
  
 CREATE TABLE Vendor (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name VARCHAR,  
 address VARCHAR  
 );  
  
 CREATE TABLE Vendor\_managers (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 first\_name VARCHAR,  
 second\_name VARCHAR,  
 third\_name VARCHAR,  
 position VARCHAR,  
 vender\_id INTEGER,  
 FOREIGN KEY (vender\_id) REFERENCES Vendor(id)  
 );  
  
 CREATE TABLE Nomenclature\_category (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name VARCHAR,  
 parent\_category\_id INTEGER,  
 FOREIGN KEY (parent\_category\_id) REFERENCES Nomenclature\_category(id)  
 );  
  
 CREATE TABLE Nomenclature (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name VARCHAR,  
 unit VARCHAR,  
 category\_id INTEGER,  
 FOREIGN KEY (category\_id) REFERENCES Nomenclature\_category(id)  
 );  
  
 CREATE TABLE Vendor\_nomenclature (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name VARCHAR,  
 nomenclature\_id INTEGER,  
 FOREIGN KEY (nomenclature\_id) REFERENCES Nomenclature(id)  
 );  
  
 CREATE TABLE Request\_category (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name VARCHAR  
 );  
  
 CREATE TABLE Requests (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 description VARCHAR,  
 created\_at DATE,  
 status VARCHAR,  
 category\_id INTEGER,  
 initiator\_id INTEGER,  
 FOREIGN KEY (category\_id) REFERENCES Request\_category(id),  
 FOREIGN KEY (initiator\_id) REFERENCES Users(id)  
 );  
  
 CREATE TABLE Request\_items (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 amount DECIMAL,  
 request\_id INTEGER,  
 item\_id INTEGER,  
 FOREIGN KEY (request\_id) REFERENCES Requests(id),  
 FOREIGN KEY (item\_id) REFERENCES Nomenclature(id)  
 );  
  
 CREATE TABLE Invoice (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 invoice\_name VARCHAR,  
 description VARCHAR,  
 created\_at DATE,  
 vender\_manager\_id INTEGER,  
 purchaser\_id INTEGER,  
 FOREIGN KEY (vender\_manager\_id) REFERENCES Vendor\_managers(id),  
 FOREIGN KEY (purchaser\_id) REFERENCES Users(id)  
 );  
  
 CREATE TABLE Invoice\_items (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 amount DECIMAL,  
 invoice\_id INTEGER,  
 vender\_item\_id INTEGER,  
 request\_item\_id INTEGER,  
 FOREIGN KEY (invoice\_id) REFERENCES Invoice(id),  
 FOREIGN KEY (vender\_item\_id) REFERENCES Vendor\_nomenclature(id),  
 FOREIGN KEY (request\_item\_id) REFERENCES Request\_items(id)  
 );  
  
 CREATE TABLE Approval\_templates (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name VARCHAR,  
 description VARCHAR  
 );  
  
 CREATE TABLE Template\_approval\_stages (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 stage\_order INTEGER,  
 acceptor\_id INTEGER,  
 template\_id INTEGER,  
 FOREIGN KEY (acceptor\_id) REFERENCES Users(id),  
 FOREIGN KEY (template\_id) REFERENCES Approval\_templates(id)  
 );  
  
 CREATE TABLE Request\_approvals\_stages (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 approval\_status VARCHAR,  
 approved\_at DATE,  
 comment VARCHAR,  
 stage\_order INTEGER,  
 request\_id INTEGER,  
 template\_stage\_id INTEGER,  
 acceptor\_id INTEGER,  
 FOREIGN KEY (request\_id) REFERENCES Requests(id),  
 FOREIGN KEY (template\_stage\_id) REFERENCES Template\_approval\_stages(id),  
 FOREIGN KEY (acceptor\_id) REFERENCES Users(id)  
 );  
  
 CREATE TABLE Invoice\_approvals\_stages (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 approval\_status VARCHAR,  
 approved\_at DATE,  
 comment VARCHAR,  
 stage\_order INTEGER,  
 invoice\_id INTEGER,  
 template\_stage\_id INTEGER,  
 acceptor\_id INTEGER,  
 FOREIGN KEY (invoice\_id) REFERENCES Invoice(id),  
 FOREIGN KEY (template\_stage\_id) REFERENCES Template\_approval\_stages(id),  
 FOREIGN KEY (acceptor\_id) REFERENCES Users(id)  
 );  
 """)  
  
 cur.execute("""  
 INSERT INTO Invoice(invoice\_name) VALUES ("Счет1"), ("Счет2"), ("Счет3");  
 """)  
  
 con.commit()  
 con.close()

## ../invoice\_app/invoice\_registry\_widget.py

import sqlite3  
  
from PySide6.QtCore import Qt  
from PySide6.QtSql import QSqlRelation, QSqlRelationalTableModel  
from PySide6.QtWidgets import QTableView, QWidget  
  
from interfaces.ui\_invoice\_registry import Ui\_Invoice\_registry  
from utils.models import DateDelegate  
  
  
class InvoiceRegistryWidget(QWidget):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.ui = Ui\_Invoice\_registry()  
 self.ui.setupUi(self)  
  
 self.model = QSqlRelationalTableModel()  
 self.model.setTable("Invoice")  
 self.model.setRelation(5, QSqlRelation("Users", "id", "login"))  
 self.model.select()  
  
 self.ui.invoice\_list.setModel(self.model)  
 self.ui.invoice\_list.setEditTriggers(QTableView.EditTriggers.NoEditTriggers)  
 self.ui.invoice\_list.setSelectionBehavior(QTableView.SelectionBehavior.SelectRows)  
 self.ui.invoice\_list.hideColumn(0)  
 self.ui.invoice\_list.hideColumn(6)  
 self.ui.invoice\_list.hideColumn(7)  
 self.model.setHeaderData(1, Qt.Horizontal, "Номер")  
 self.model.setHeaderData(2, Qt.Horizontal, "Описание")  
 self.model.setHeaderData(3, Qt.Horizontal, "Дата создания")  
 self.model.setHeaderData(4, Qt.Horizontal, "Статус")  
 self.model.setHeaderData(5, Qt.Horizontal, "Закупщик")  
 self.model.setHeaderData(6, Qt.Horizontal, "Контрагент")  
 self.model.setHeaderData(7, Qt.Horizontal, "Файл")  
 self.model.setHeaderData(8, Qt.Horizontal, "Сумма")  
 date\_delegate = DateDelegate()  
 self.ui.invoice\_list.setItemDelegateForColumn(3, date\_delegate)  
 self.ui.invoice\_list.resizeColumnsToContents()  
  
 self.setup\_vender\_combobox()  
  
 self.ui.invoice\_list.doubleClicked.connect(self.open\_invoice\_details)  
 self.ui.close\_btn.clicked.connect(parent.close\_current\_tab)  
 self.ui.refresh\_btn.clicked.connect(self.refresh\_values)  
 self.ui.create\_btn.clicked.connect(parent.open\_invoice\_creation)  
 self.ui.vender\_combo\_box.currentTextChanged.connect(self.filter\_list)  
 self.ui.comboBox.currentTextChanged.connect(self.filter\_list)  
  
 def filter\_list(self):  
 vender\_name = self.ui.vender\_combo\_box.currentText()  
 status\_name = self.ui.comboBox.currentText()  
  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
  
 if vender\_name == "-" and status\_name == "-":  
 self.model.setFilter("")  
 return  
 elif status\_name == "-":  
 vender\_id = cur.execute("SELECT id FROM Vendor WHERE name=?", (vender\_name,)).fetchone()[0]  
 self.model.setFilter(f"vender\_id={vender\_id}")  
 elif vender\_name == "-":  
 self.model.setFilter(f"status='{status\_name}'")  
 else:  
 vender\_id = cur.execute("SELECT id FROM Vendor WHERE name=?", (vender\_name,)).fetchone()[0]  
 self.model.setFilter(f"vender\_id={vender\_id} AND status='{status\_name}'")  
  
 con.close()  
 self.refresh\_values()  
  
 def setup\_vender\_combobox(self):  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 categories = cur.execute("SELECT name FROM Vendor").fetchall()  
 con.close()  
 self.ui.vender\_combo\_box.clear()  
 categories = list(map(lambda x: x[0], categories))  
 self.ui.vender\_combo\_box.insertItems(0, categories)  
 self.ui.vender\_combo\_box.insertItem(0, "-")  
 self.ui.vender\_combo\_box.setCurrentIndex(0)  
  
 def open\_invoice\_details(self, index):  
 invoice\_id = self.ui.invoice\_list.model().data(self.ui.invoice\_list.model().index(index.row(), 0))  
 self.parent.open\_invoice\_creation\_with\_data(invoice\_id)  
  
 def refresh\_values(self):  
 self.model.select()  
 self.ui.invoice\_list.reset()

## ../invoice\_app/invoice\_widget.py

import sqlite3  
import datetime as dt  
  
from PySide6.QtWidgets import QWidget, QDialog, QTableWidgetItem, QMessageBox  
  
from interfaces.ui\_create\_invoice import Ui\_Invoice  
from nomenclature.nomenclature\_dialog import NomenclatureDialog  
from accept\_app.accept\_dialog import AcceptDialog  
from vender\_app.vender\_dialog import VenderDialog  
from contracts\_app.contracts\_dialog import ContractDialog  
from files\_app.file\_dialog import FileDialog  
  
  
class InvoiceWidget(QWidget):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.id = None  
 self.contract\_id = None  
 self.ui = Ui\_Invoice()  
 self.ui.setupUi(self)  
 self.ui.delete\_btn.hide()  
 self.ui.send\_btn.hide()  
 self.ui.accept\_viewer\_btn.hide()  
 self.ui.date\_lbl.setText("--.--.----")  
 self.ui.status\_lbl.setText("---------")  
 self.ui.tableWidget.hideColumn(0)  
  
 self.ui.add\_nomenclature\_btn.clicked.connect(self.open\_nomenclature\_dialog)  
 self.ui.delete\_nomenclature\_btn.clicked.connect(self.delete\_nomenclature\_row)  
 self.ui.save\_btn.clicked.connect(self.save\_invoice)  
 self.ui.close\_btn.clicked.connect(parent.close\_current\_tab)  
 self.ui.delete\_btn.clicked.connect(self.delete\_invoice)  
 self.ui.send\_btn.clicked.connect(self.open\_accept\_dialog)  
 self.ui.vendor\_btn.clicked.connect(self.open\_vender\_dialog)  
 self.ui.contract\_btn.clicked.connect(self.open\_contract\_dialog)  
 self.ui.file\_btn.clicked.connect(self.open\_file\_dialog)  
 self.ui.accept\_viewer\_btn.clicked.connect(lambda: parent.open\_accept\_invoice\_viewer(self.id))  
 self.ui.tableWidget.itemChanged.connect(self.calculate\_and\_display\_sum)  
  
 def open\_vender\_dialog(self):  
 dialog = VenderDialog(self.parent)  
 if dialog.exec() == QDialog.Accepted and dialog.vender:  
 self.ui.vendor\_lbl.setText(dialog.vender)  
 self.contract\_id = None  
 self.ui.contract\_lbl.clear()  
  
 def open\_contract\_dialog(self):  
 vender\_name = self.ui.vendor\_lbl.text()  
 if not vender\_name:  
 QMessageBox.warning(self, "Предупреждение", "Вы не добавили контрагента")  
 return  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 vender\_id = cur.execute("SELECT id FROM Vendor WHERE name = ?;", (vender\_name,)).fetchone()[0]  
 dialog = ContractDialog(self.parent, vender\_id)  
 if dialog.exec() == QDialog.Accepted:  
 self.contract\_id = dialog.contract\_id  
 self.ui.contract\_lbl.setText(dialog.contract\_name)  
  
 def open\_nomenclature\_dialog(self):  
 dialog = NomenclatureDialog(self.parent)  
 if dialog.exec() == QDialog.Accepted:  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 nomenclature = cur.execute("SELECT \* FROM Nomenclature WHERE id=?", (dialog.nomenclature\_id,)).fetchone()  
 con.close()  
  
 rows = self.ui.tableWidget.rowCount()  
 self.ui.tableWidget.setRowCount(rows + 1)  
 self.ui.tableWidget.setItem(rows, 0, QTableWidgetItem(str(nomenclature[0])))  
 self.ui.tableWidget.setItem(rows, 1, QTableWidgetItem(nomenclature[1]))  
 self.ui.tableWidget.setItem(rows, 2, QTableWidgetItem(nomenclature[2]))  
  
 def delete\_nomenclature\_row(self):  
 self.ui.tableWidget.removeRow(self.ui.tableWidget.currentRow())  
 self.calculate\_and\_display\_sum()  
  
 def set\_is\_created(self, id):  
 self.ui.save\_btn.clicked.disconnect()  
 self.ui.save\_btn.clicked.connect(self.update\_invoice)  
 self.id = id  
 self.ui.delete\_btn.show()  
 self.ui.send\_btn.show()  
 self.ui.accept\_viewer\_btn.show()  
  
 def set\_uneditable\_fields(self, date, status):  
 self.ui.date\_lbl.setText(date.split(".")[0])  
 self.ui.status\_lbl.setText(status)  
  
 def check\_and\_return\_editable\_fields(self):  
 vender\_name = self.ui.vendor\_lbl.text()  
 name = self.ui.name\_edit.text()  
 description = self.ui.description\_text.toPlainText()  
 rows = self.ui.tableWidget.rowCount()  
  
 # Извлечение значений из столбца 'Количество'  
 amounts = []  
 prices = []  
 for row in range(rows):  
 item = self.ui.tableWidget.item(row, 3)  
 if item and item.text().strip():  
 try:  
 amounts.append(float(item.text().strip()))  
 except ValueError:  
 QMessageBox.warning(self, "Ошибка", f"Некорректное значение в строке {row + 1}, столбец 'Количество'")  
 return None, None, None, None, None, None, None  
 else:  
 QMessageBox.warning(self, "Ошибка", f"Отсутствует значение в строке {row + 1}, столбец 'Количество'")  
 return None, None, None, None, None, None, None  
 price = self.ui.tableWidget.item(row, 4)  
 if price and price.text().strip():  
 try:  
 prices.append(float(price.text().strip()))  
 except ValueError:  
 QMessageBox.warning(self, "Ошибка", f"Некорректное значение в строке {row + 1}, столбец 'Цена'")  
 return None, None, None, None, None, None, None  
 else:  
 QMessageBox.warning(self, "Ошибка", f"Отсутствует значение в строке {row + 1}, столбец 'Цена'")  
 return None, None, None, None, None, None, None  
  
 item\_ids = [self.ui.tableWidget.item(row, 0).text() for row in range(rows)]  
  
  
 if not name:  
 QMessageBox.warning(self, "Предупреждение", "Вы не заполнили номер счета")  
 elif not vender\_name:  
 QMessageBox.warning(self, "Предупреждение", "Вы не добавили контрагента")  
 elif not description:  
 QMessageBox.warning(self, "Предупреждение", "Вы не заполнили описание")  
 elif not rows:  
 QMessageBox.warning(self, "Предупреждение", "Вы не добавили номенклатуру")  
 elif not self.contract\_id:  
 QMessageBox.warning(self, "Предупреждение", "Вы не добавили договор")  
 else:  
 return name, description, rows, amounts, item\_ids, prices  
  
 return None, None, None, None, None, None, None  
  
 def save\_invoice(self):  
 name, description, rows, amounts, item\_ids, prices = self.check\_and\_return\_editable\_fields()  
 if not (name and description and rows and amounts and prices):  
 return  
 amounts = list(amounts)  
   
 created\_at = dt.datetime.now()  
 status = "Не согласовано"  
 user\_id = self.parent.user\_id  
  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 try:  
 cur.execute("""  
 INSERT INTO Invoice(invoice\_name, description, created\_at, status, purchaser\_id, contract\_id)   
 VALUES (?, ?, ?, ?, ?, ?);  
 """,   
 (name, description, created\_at, status, user\_id, self.contract\_id))  
 except sqlite3.Error as err:  
 QMessageBox.critical(self, "Неизвестная ошибка", str(err))  
   
 invoice\_id = cur.lastrowid  
 invoice\_ids = [str(invoice\_id)] \* rows  
  
 cur.executemany("INSERT INTO Invoice\_items(amount, invoice\_id, item\_id, price) VALUES (?, ?, ?, ?);",   
 zip(amounts, invoice\_ids, item\_ids, prices))  
 con.commit()  
 con.close()  
  
 self.parent.tab\_widget.setTabText(self.parent.tab\_widget.currentIndex(), f"Счет {invoice\_id}")  
 self.set\_is\_created(invoice\_id)  
 self.set\_uneditable\_fields(str(created\_at), status)  
 self.parent.status\_bar.showMessage("Счет успешно сохранен", 3000)  
  
 def update\_invoice(self):  
 name, description, rows, amounts, item\_ids, prices = self.check\_and\_return\_editable\_fields()  
 amounts = list(amounts)  
 if not (name and description and rows and amounts, prices):  
 return  
   
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 try:  
 cur.execute("""  
 UPDATE Invoice   
 SET invoice\_name=?, description=?, contract\_id=?   
 WHERE id=?;  
 """, (name, description, self.contract\_id, self.id))  
 except sqlite3.OperationalError as err:  
 QMessageBox.critical(self, "Неизвестная ошибка", str(err))  
   
 invoice\_ids = [str(self.id)] \* rows  
  
 cur.execute("DELETE FROM Invoice\_items WHERE invoice\_id=?", (self.id,))  
 cur.executemany("INSERT INTO Invoice\_items(amount, invoice\_id, item\_id, price) VALUES (?, ?, ?, ?);",   
 zip(amounts, invoice\_ids, item\_ids, prices))  
 con.commit()  
 con.close()  
 self.parent.status\_bar.showMessage("Счет успешно сохранен", 3000)  
  
  
 def load\_invoice\_data(self, invoice\_id):  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
   
 # Получаем основную информацию о счете  
 invoice = cur.execute("""  
 SELECT  
 Invoice.invoice\_name,  
 Invoice.description,   
 Invoice.created\_at,   
 Invoice.status,   
 Contracts.id,  
 Contracts.number,   
 Contracts.date,   
 Vendor.name,  
 Invoice.items\_sum  
 FROM Invoice   
 LEFT JOIN Contracts ON Invoice.contract\_id = Contracts.id  
 LEFT JOIN Vendor ON Contracts.vender\_id = Vendor.id  
 WHERE Invoice.id = ?;  
 """, (invoice\_id,)).fetchone()  
 if invoice:  
 self.ui.name\_edit.setText(invoice[0])  
 self.ui.description\_text.setPlainText(invoice[1])  
 self.ui.vendor\_lbl.setText(invoice[7])  
 self.set\_uneditable\_fields(str(invoice[2]), invoice[3])  
 self.contract\_id = invoice[4]  
 self.ui.contract\_lbl.setText(invoice[5] + " от " + str(invoice[6]))  
 self.ui.sum\_lcd.display(invoice[8]) # Заполняем итоговую сумму  
  
 # Очищаем таблицу и добавляем связанные позиции заявки  
 self.ui.tableWidget.setRowCount(0)  
 items = cur.execute("""  
 SELECT item\_id, amount, price   
 FROM Invoice\_items   
 WHERE invoice\_id = ?;  
 """, (invoice\_id,)).fetchall()  
 for item\_id, amount, price in items:  
 nomenclature = cur.execute("SELECT name, unit FROM Nomenclature WHERE id = ?", (item\_id,)).fetchone()  
 if nomenclature:  
 row = self.ui.tableWidget.rowCount()  
 self.ui.tableWidget.setRowCount(row + 1)  
 self.ui.tableWidget.setItem(row, 0, QTableWidgetItem(str(item\_id)))  
 self.ui.tableWidget.setItem(row, 1, QTableWidgetItem(nomenclature[0]))  
 self.ui.tableWidget.setItem(row, 2, QTableWidgetItem(nomenclature[1]))  
 self.ui.tableWidget.setItem(row, 3, QTableWidgetItem(str(amount)))  
 self.ui.tableWidget.setItem(row, 4, QTableWidgetItem(str(price)))  
  
 con.close()  
 self.set\_is\_created(invoice\_id)  
   
 con.close()  
 self.set\_is\_created(invoice\_id)  
  
 def delete\_invoice(self):  
 res = QMessageBox.warning(self, "Предупреждение", "Вы уверены, что хотите удалить объект?", QMessageBox.Yes, QMessageBox.No)  
 if res == QMessageBox.No:  
 return  
  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 cur.execute("DELETE FROM Invoice\_items WHERE invoice\_id=?", (self.id,))  
 cur.execute("DELETE FROM Invoice WHERE id=?", (self.id,))  
 con.commit()  
 con.close()  
  
 self.parent.close\_current\_tab()  
 self.parent.status\_bar.showMessage("Счет успешно удален", 3000)  
  
 def open\_accept\_dialog(self):  
 if not self.id:  
 ans = QMessageBox.warning(self, "Предупреждение", "Для продолжения необходимо сохранить объект")  
 if ans == QMessageBox.Ok:  
 self.save\_invoice()  
 else:  
 return  
 dialog = AcceptDialog(self.parent)  
 if dialog.exec() == QDialog.Accepted:  
 approval\_status = "Не согласовано"  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
  
 max\_stage = cur.execute("SELECT MAX(stage\_order) FROM Invoice\_approvals\_stages WHERE invoice\_id=?;", (self.id,)).fetchone()[0]  
 if not max\_stage:  
 max\_stage = 0  
 else:  
 max\_stage = int(max\_stage) + 1  
  
 for stage\_order, acceptor\_id in enumerate(dialog.accepted\_users):  
 cur.execute("""  
 INSERT INTO Invoice\_approvals\_stages(approval\_status, stage\_order, invoice\_id, acceptor\_id)   
 VALUES (?, ?, ?, ?)  
 """, (approval\_status, max\_stage + stage\_order if dialog.is\_step\_by\_step else 1, self.id, acceptor\_id))  
  
 con.commit()  
 con.close()  
 if not dialog.accepted\_users:  
 self.parent.status\_bar.showMessage("Не выбраны согласованты", 3000)  
 else:  
 self.parent.status\_bar.showMessage("Заявка успешно отправлена на согласование", 3000)  
  
 def open\_file\_dialog(self):  
 if not self.id:  
 QMessageBox.warning(  
 self,   
 "Предупреждение",   
 "Для продолжения необходимо сохранить счет"  
 )  
 return  
 dialog = FileDialog(self.parent, self.id)  
 dialog.exec()  
  
 def calculate\_and\_display\_sum(self):  
 total\_sum = 0  
 rows = self.ui.tableWidget.rowCount()  
 for row in range(rows):  
 price\_item = self.ui.tableWidget.item(row, 4)  
 amount\_item = self.ui.tableWidget.item(row, 3)  
 if price\_item and amount\_item:  
 try:  
 price = float(price\_item.text())  
 amount = float(amount\_item.text())  
 total\_sum += price \* amount  
 except ValueError:  
 continue  
 self.ui.sum\_lcd.display(total\_sum)

## ../contracts\_app/contracts\_registry.py

import sqlite3  
  
from PySide6.QtWidgets import QMessageBox, QTableView, QTableWidgetItem, QWidget  
  
from interfaces.ui\_contract\_registry import Ui\_ContractRegistry  
from utils.models import DateDelegate  
  
  
class ContractRegistry(QWidget):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.ui = Ui\_ContractRegistry()  
 self.ui.setupUi(self)  
 self.ui.tableWidget.setEditTriggers(QTableView.EditTriggers.NoEditTriggers)  
 self.ui.tableWidget.setSelectionBehavior(QTableView.SelectionBehavior.SelectRows)  
 self.ui.tableWidget.hideColumn(0)  
 date\_delegate = DateDelegate()  
 self.ui.tableWidget.setItemDelegateForColumn(2, date\_delegate)  
 self.ui.tableWidget.resizeColumnsToContents()  
  
 self.update\_list()  
  
 self.ui.tableWidget.doubleClicked.connect(self.open\_contract)  
 self.ui.close\_btn.clicked.connect(parent.close\_current\_tab)  
 self.ui.refresh\_btn.clicked.connect(self.update\_list)  
 self.ui.create\_btn.clicked.connect(parent.open\_contract\_creation)  
  
 def update\_list(self):  
 self.ui.tableWidget.clearContents()  
 self.ui.tableWidget.setRowCount(0)  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 contracts = cur.execute("SELECT Contracts.id, Contracts.number, Contracts.date, Vendor.name FROM Contracts LEFT JOIN Vendor ON Contracts.vender\_id = Vendor.id;").fetchall()  
 con.close()  
 self.ui.tableWidget.setRowCount(len(contracts))  
 for row, contract in enumerate(contracts):  
 self.ui.tableWidget.setItem(row, 0, QTableWidgetItem(str(contract[0])))  
 self.ui.tableWidget.setItem(row, 1, QTableWidgetItem(str(contract[1])))  
 self.ui.tableWidget.setItem(row, 2, QTableWidgetItem(str(contract[2])))  
 self.ui.tableWidget.setItem(row, 3, QTableWidgetItem(str(contract[3])))  
 self.ui.tableWidget.resizeColumnsToContents()  
  
 def open\_contract(self, index):  
 contract\_id\_item = self.ui.tableWidget.item(index.row(), 0)  
 if contract\_id\_item is not None:  
 try:  
 contract\_id = int(contract\_id\_item.text())  
 self.parent.open\_contract\_with\_data(contract\_id)  
 except ValueError:  
 QMessageBox.warning(self, "Ошибка", "Некорректный идентификатор контракта.")  
 else:  
 QMessageBox.warning(self, "Ошибка", "Не удалось получить идентификатор контракта.")

## ../contracts\_app/contract\_widget.py

import sqlite3  
  
from PySide6.QtCore import QDate, QUrl  
from PySide6.QtGui import QDesktopServices  
from PySide6.QtWidgets import QDialog, QFileDialog, QMessageBox, QWidget  
  
from interfaces.ui\_contract import Ui\_Contract  
from vender\_app.vender\_dialog import VenderDialog  
  
  
class ContractWidget(QWidget):  
 def \_\_init\_\_(self, parent, contract\_id=None):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.contract\_id = contract\_id  
 self.ui = Ui\_Contract()  
 self.ui.setupUi(self)  
 self.ui.delete\_btn.hide()  
  
 # Подключение кнопок  
 self.ui.vendor\_btn.clicked.connect(self.open\_vender\_dialog)  
 self.ui.attach\_file.clicked.connect(self.attach\_file)  
 self.ui.delete\_file.clicked.connect(self.delete\_file)  
 self.ui.open\_file.clicked.connect(self.open\_file)  
 self.ui.save\_btn.clicked.connect(self.save\_contract)  
 self.ui.delete\_btn.clicked.connect(self.delete\_contract)  
 self.ui.close\_btn.clicked.connect(parent.close\_current\_tab)  
  
 if self.contract\_id:  
 self.load\_data()  
 self.ui.delete\_btn.show()  
  
 def load\_data(self):  
 """Загружает данные контракта из базы для редактирования."""  
 with sqlite3.connect(self.parent.database\_file) as conn:  
 cursor = conn.cursor()  
 cursor.execute("SELECT number, date, vender\_id, file\_path FROM Contracts WHERE id = ?", (self.contract\_id,))  
 contract = cursor.fetchone()  
 if contract:  
 self.ui.number\_edit.setText(contract[0])  
 self.ui.date\_edit.setDate(QDate.fromString(contract[1], "yyyy-MM-dd"))  
 vendor\_id = contract[2]  
 file\_path = contract[3]  
   
 # Получение названия поставщика  
 cursor.execute("SELECT name FROM Vendor WHERE id = ?", (vendor\_id,))  
 vendor = cursor.fetchone()  
 if vendor:  
 self.ui.vendor\_lbl.setText(vendor[0])  
   
 self.ui.filename\_lbl.setText(file\_path if file\_path else "")  
  
 def save\_contract(self):  
 number = self.ui.number\_edit.text()  
 date = self.ui.date\_edit.date().toString("yyyy-MM-dd")  
 vendor\_name = self.ui.vendor\_lbl.text()  
 file\_path = self.ui.filename\_lbl.text()  
  
 if not number:  
 QMessageBox.warning(self, "Ошибка", "Вы не заполнили номер договора")  
 return  
 if not vendor\_name:  
 QMessageBox.warning(self, "Ошибка", "Вы не добавили контрагента")  
 return  
 if not date:  
 QMessageBox.warning(self, "Ошибка", "Вы не заполнили дату договора")  
 return  
  
 with sqlite3.connect(self.parent.database\_file) as conn:  
 cursor = conn.cursor()  
 cursor.execute("SELECT id FROM Vendor WHERE name = ?", (vendor\_name,))  
 vendor = cursor.fetchone()  
 if not vendor:  
 QMessageBox.warning(self, "Ошибка", "Выбранный поставщик не найден.")  
 return  
 vendor\_id = vendor[0]  
  
 if self.contract\_id:  
 cursor.execute("""  
 UPDATE Contracts  
 SET number = ?, date = ?, vender\_id = ?, file\_path = ?  
 WHERE id = ?  
 """, (number, date, vendor\_id, file\_path, self.contract\_id))  
 else:  
 cursor.execute("""  
 INSERT INTO Contracts (number, date, vender\_id, file\_path)  
 VALUES (?, ?, ?, ?)  
 """, (number, date, vendor\_id, file\_path))  
 self.ui.delete\_btn.hide()  
 conn.commit()  
  
 QMessageBox.information(self, "Успех", "Контракт успешно сохранен.")  
  
 def delete\_contract(self):  
 """Удаляет контракт из базы данных."""  
 if not self.contract\_id:  
 QMessageBox.warning(self, "Ошибка", "Контракт не найден.")  
 return  
  
 with sqlite3.connect(self.parent.database\_file) as conn:  
 cursor = conn.cursor()  
 cursor.execute("DELETE FROM Contracts WHERE id = ?", (self.contract\_id,))  
 conn.commit()  
  
 QMessageBox.information(self, "Успех", "Контракт успешно удален.")  
 self.parent.close\_current\_tab()  
  
 def attach\_file(self):  
 """Прикрепляет файл к контракту."""  
 file\_path, \_ = QFileDialog.getOpenFileName(self, "Выбрать файл")  
 if file\_path:  
 self.ui.filename\_lbl.setText(file\_path)  
  
 def delete\_file(self):  
 """Удаляет прикрепленный файл."""  
 self.ui.filename\_lbl.setText("")  
  
 def open\_file(self):  
 """Открывает прикрепленный файл."""  
 file\_path = self.ui.filename\_lbl.text()  
 if file\_path:  
 QDesktopServices.openUrl(QUrl.fromLocalFile(file\_path))  
 else:  
 QMessageBox.warning(self, "Ошибка", "Файл не прикреплен.")  
  
 def open\_vender\_dialog(self):  
 """Открывает диалог выбора поставщика."""  
 ans = QMessageBox.warning(  
 self,   
 "Предупреждение",   
 "Если изменить контрагента в контракте, то он изменится во всех связанных счетах",   
 QMessageBox.Ok,  
 QMessageBox.Cancel,  
 )  
 if ans == QMessageBox.Cancel:  
 return  
 dialog = VenderDialog(self.parent)  
 if dialog.exec() == QDialog.Accepted:  
 self.ui.vendor\_lbl.setText(dialog.vender)

## ../contracts\_app/contracts\_dialog.py

import sqlite3  
  
from PySide6.QtWidgets import QDialog, QTableWidgetItem, QMessageBox, QTableView  
  
from interfaces.ui\_contract\_dialog import Ui\_Dialog  
from utils.models import DateDelegate  
  
  
class ContractDialog(QDialog):  
 def \_\_init\_\_(self, parent, vender\_id):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.vender\_id = vender\_id  
 self.ui = Ui\_Dialog()  
 self.ui.setupUi(self)  
 self.ui.tableWidget.setEditTriggers(QTableView.EditTriggers.NoEditTriggers)  
 self.ui.tableWidget.setSelectionBehavior(QTableView.SelectionBehavior.SelectRows)  
 self.ui.tableWidget.hideColumn(0)  
 date\_delegate = DateDelegate()  
 self.ui.tableWidget.setItemDelegateForColumn(2, date\_delegate)  
 self.ui.tableWidget.resizeColumnsToContents()  
  
 self.update\_list()  
  
 self.ui.pushButton.clicked.connect(self.add\_contract)  
 self.ui.buttonBox.accepted.disconnect()  
 self.ui.buttonBox.accepted.connect(self.select\_contract)  
  
 def update\_list(self):  
 self.ui.tableWidget.clearContents()  
 self.ui.tableWidget.setRowCount(0)  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 contracts = cur.execute("SELECT id, number, date FROM Contracts WHERE vender\_id=?;", (self.vender\_id,)).fetchall()  
 con.close()  
 self.ui.tableWidget.setRowCount(len(contracts))  
 for row, contract in enumerate(contracts):  
 self.ui.tableWidget.setItem(row, 0, QTableWidgetItem(str(contract[0])))  
 self.ui.tableWidget.setItem(row, 1, QTableWidgetItem(str(contract[1])))  
 self.ui.tableWidget.setItem(row, 2, QTableWidgetItem(str(contract[2])))  
 self.ui.tableWidget.resizeColumnsToContents()  
  
 def select\_contract(self):  
 if len(self.ui.tableWidget.selectedItems()) != 1:  
 QMessageBox.warning(self, "Предупреждение", "Выберете один договор")  
 return  
 self.contract\_id = self.ui.tableWidget.item(self.ui.tableWidget.currentRow(), 0).text()  
 contract\_number = self.ui.tableWidget.item(self.ui.tableWidget.currentRow(), 1).text()  
 contract\_date = self.ui.tableWidget.item(self.ui.tableWidget.currentRow(), 2).text()  
 self.contract\_name = contract\_number + " от " + contract\_date  
 self.accept()  
  
 def add\_contract(self):  
 number = self.ui.lineEdit.text()  
 date = self.ui.dateEdit.date().toPython()  
 if number and date:  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 cur.execute("INSERT INTO Contracts(number, date, vender\_id) VALUES (?, ?, ?)", (number, date, self.vender\_id))  
 con.commit()  
 con.close()  
 self.update\_list()

## ../utils/models.py

import sqlite3  
from PySide6.QtWidgets import QStyledItemDelegate, QPushButton, QHBoxLayout, QLineEdit, QWidget, QDialog, QTableWidgetItem  
from PySide6.QtCore import QDateTime, Qt  
  
from nomenclature.nomenclature\_dialog import NomenclatureDialog  
  
  
class DateDelegate(QStyledItemDelegate):  
 def displayText(self, value, locale):  
 if isinstance(value, str):  
 # Обрезаем строку до секунд, чтобы исключить доли секунды  
 value = value.split(".")[0]  
 # Преобразуем строку в QDateTime, используя соответствующий формат  
 date\_time = QDateTime.fromString(value, "yyyy-MM-dd HH:mm:ss")  
 # Проверяем, удалось ли преобразование  
 if date\_time.isValid():  
 # Возвращаем только дату  
 return date\_time.date().toString("yyyy-MM-dd")  
 return value  
   
  
class ReadOnlyDelegate(QStyledItemDelegate):  
 def createEditor(self, parent, option, index):  
 return None  
  
  
class EditableDelegate(QStyledItemDelegate):  
 def createEditor(self, parent, option, index):  
 # Создаём кастомный редактор с кнопками  
 editor = QWidget(parent)  
 layout = QHBoxLayout(editor)  
 layout.setContentsMargins(0, 0, 0, 0)  
  
 # Поле для ввода текста  
 line\_edit = QLineEdit(editor)  
 layout.addWidget(line\_edit)  
  
 # Кнопка открытия окна  
 open\_button = QPushButton("■", editor)  
 open\_button.setFixedSize(20, 20)  
 layout.addWidget(open\_button)  
  
 # Привязка событий  
 open\_button.clicked.connect(lambda: self.open\_nomenclature\_window\_with\_index(index))  
  
 return editor  
  
 def open\_nomenclature\_window\_with\_index(self, index):  
 self.open\_nomenclature\_window(index)  
  
 def setEditorData(self, editor, index):  
 line\_edit = editor.findChild(QLineEdit)  
 line\_edit.setText(index.data(Qt.DisplayRole))  
  
 def setModelData(self, editor, model, index):  
 line\_edit = editor.findChild(QLineEdit)  
 model.setData(index, line\_edit.text(), Qt.EditRole)  
  
 def show\_dropdown(self, line\_edit):  
 # Здесь вы можете сделать запрос в базу данных и создать выпадающий список  
 print(f"Показываем выпадающий список для подстроки: {line\_edit.text()}")  
  
 def open\_nomenclature\_window(self, index):  
 dialog = NomenclatureDialog(self.parent.parent)  
 if dialog.exec() == QDialog.Accepted:  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 nomenclature = cur.execute("SELECT \* FROM Nomenclature WHERE id=?", (dialog.nomenclature\_id,)).fetchone()  
 con.close()  
  
 if nomenclature:  
 # Получаем доступ к таблице  
 table = self.parent.ui.tableWidget  
  
 # Определяем строку, в которой находится делегат  
 row = index.row()  
  
 # Устанавливаем значения в ячейки  
 table.setItem(row, 0, QTableWidgetItem(str(nomenclature[0]))) # ID  
 table.setItem(row, 1, QTableWidgetItem(nomenclature[1])) # Наименование  
 table.setItem(row, 2, QTableWidgetItem(nomenclature[2])) # Другие данные

## ../accept\_app/accept\_dialog.py

from PySide6.QtGui import QStandardItem, QStandardItemModel  
from PySide6.QtSql import QSqlRelationalTableModel  
from PySide6.QtWidgets import QDialog, QMessageBox, QTableView  
  
from interfaces.ui\_accept\_dialog import Ui\_Accept\_dialog  
  
  
class AcceptDialog(QDialog):  
 def \_\_init\_\_(self, parent):  
 self.parent = parent  
 super().\_\_init\_\_()  
 self.ui = Ui\_Accept\_dialog()  
 self.ui.setupUi(self)  
  
 # Модель для таблицы пользователей  
 self.user\_model = QSqlRelationalTableModel()  
 self.user\_model.setTable("Users")  
 self.user\_model.select()  
  
 self.ui.user\_table.setModel(self.user\_model)  
 self.ui.user\_table.setEditTriggers(QTableView.EditTriggers.NoEditTriggers)  
 self.ui.user\_table.setSelectionBehavior(QTableView.SelectionBehavior.SelectRows)  
 self.ui.user\_table.hideColumn(0)  
 self.ui.user\_table.hideColumn(5)  
 self.ui.user\_table.hideColumn(6)  
 self.ui.user\_table.hideColumn(7)  
 self.ui.user\_table.resizeColumnsToContents()  
  
 # Модель для таблицы выбранных пользователей  
 self.accept\_model = QStandardItemModel(0, 4, self)  
 self.accept\_model.setHorizontalHeaderLabels(["ID", "Имя", "Фамилия", "Отчество"])  
 self.ui.accept\_table.setModel(self.accept\_model)  
 self.ui.accept\_table.setEditTriggers(QTableView.EditTriggers.NoEditTriggers)  
 self.ui.accept\_table.setSelectionBehavior(QTableView.SelectionBehavior.SelectRows)  
 self.ui.accept\_table.hideColumn(0)  
 self.ui.accept\_table.resizeColumnsToContents()  
  
 # События для кнопок  
 self.ui.add\_btn.clicked.connect(self.add\_user)  
 self.ui.remove\_btn.clicked.connect(self.remove\_user)  
 self.ui.up\_btn.clicked.connect(self.move\_up)  
 self.ui.down\_btn.clicked.connect(self.move\_down)  
  
 # Событие для подтверждения выбора  
 self.ui.buttonBox.accepted.disconnect()  
 self.ui.buttonBox.accepted.connect(self.select\_accept\_users)  
  
 def add\_user(self):  
 """Перенос пользователя из user\_table в accept\_table"""  
 selected\_rows = self.ui.user\_table.selectionModel().selectedRows()  
  
 if not selected\_rows:  
 QMessageBox.warning(self, "Ошибка", "Выберите хотя бы одного пользователя.")  
 return  
  
 # Перенос выбранных пользователей в таблицу accept\_table  
 for index in selected\_rows:  
 row = index.row()  
 user\_id = self.user\_model.data(self.user\_model.index(row, 0))  
 first\_name = self.user\_model.data(self.user\_model.index(row, 1))  
 second\_name = self.user\_model.data(self.user\_model.index(row, 2))  
 third\_name = self.user\_model.data(self.user\_model.index(row, 3))  
  
 # Проверка на наличие в accept\_table  
 if self.find\_in\_accept\_table(user\_id):  
 continue  
  
 # Добавление в accept\_table  
 self.add\_to\_accept\_table(user\_id, first\_name, second\_name, third\_name)  
 self.ui.accept\_table.resizeColumnsToContents()  
  
 def remove\_user(self):  
 """Удаление пользователя из accept\_table"""  
 selected\_rows = self.ui.accept\_table.selectionModel().selectedRows()  
  
 if not selected\_rows:  
 QMessageBox.warning(self, "Ошибка", "Выберите хотя бы одного пользователя для удаления.")  
 return  
  
 # Удаление пользователя из accept\_table  
 for index in sorted(selected\_rows, reverse=True):  
 self.accept\_model.removeRow(index.row())  
  
 def add\_to\_accept\_table(self, user\_id, first\_name, second\_name, third\_name):  
 """Добавление пользователя в accept\_table."""  
 id\_item = QStandardItem(str(user\_id))  
 first\_name\_item = QStandardItem(first\_name)  
 second\_name\_item = QStandardItem(second\_name)  
 third\_name\_item = QStandardItem(third\_name)  
  
 self.accept\_model.appendRow([id\_item, first\_name\_item, second\_name\_item, third\_name\_item])  
  
 def find\_in\_accept\_table(self, user\_id):  
 """Проверяет, есть ли пользователь с данным ID в accept\_table"""  
 for row in range(self.accept\_model.rowCount()):  
 if self.accept\_model.item(row, 0).text() == str(user\_id):  
 return True  
 return False  
  
 def move\_up(self):  
 """Перемещает выделенную строку вверх"""  
 selected\_rows = self.ui.accept\_table.selectionModel().selectedRows()  
  
 if len(selected\_rows) != 1:  
 QMessageBox.warning(self, "Ошибка", "Выберите одну строку для перемещения.")  
 return  
  
 row = selected\_rows[0].row()  
 if row == 0:  
 return  
  
 self.swap\_rows(row, row - 1)  
  
 def move\_down(self):  
 """Перемещает выделенную строку вниз"""  
 selected\_rows = self.ui.accept\_table.selectionModel().selectedRows()  
  
 if len(selected\_rows) != 1:  
 QMessageBox.warning(self, "Ошибка", "Выберите одну строку для перемещения.")  
 return  
  
 row = selected\_rows[0].row()  
 if row == self.accept\_model.rowCount() - 1:  
 return  
  
 self.swap\_rows(row, row + 1)  
  
 def swap\_rows(self, row1, row2):  
 """Меняет местами две строки в модели accept\_table"""  
 items\_row1 = [self.accept\_model.item(row1, col).text() for col in range(self.accept\_model.columnCount())]  
 items\_row2 = [self.accept\_model.item(row2, col).text() for col in range(self.accept\_model.columnCount())]  
  
 # Заменяем строки, создавая новые элементы  
 for col in range(self.accept\_model.columnCount()):  
 self.accept\_model.setItem(row1, col, QStandardItem(items\_row2[col]))  
 self.accept\_model.setItem(row2, col, QStandardItem(items\_row1[col]))  
  
 # Обновляем выделение  
 self.ui.accept\_table.selectRow(row2)  
  
 def select\_accept\_users(self):  
 """Сохраняет выбранных пользователей из accept\_table"""  
 self.is\_step\_by\_step = True if self.ui.type\_combo\_box.currentText() == "По очереди" else False  
 selected\_user\_ids = []  
 for row in range(self.accept\_model.rowCount()):  
 user\_id = self.accept\_model.item(row, 0).text()  
 selected\_user\_ids.append(int(user\_id))  
 self.accepted\_users = selected\_user\_ids  
 if not self.accepted\_users:  
 QMessageBox.warning(  
 self,  
 "Предупреждение",  
 "Выберете пользователей для согласования"  
 )  
 return  
 self.accept()

## ../accept\_app/accept\_request\_viewer.py

import sqlite3  
  
from PySide6.QtCore import Qt  
from PySide6.QtSql import QSqlRelation, QSqlRelationalTableModel  
from PySide6.QtWidgets import QMessageBox, QTableView, QWidget, QDialog  
  
from accept\_app.accept\_dialog import AcceptDialog  
from interfaces.ui\_accept\_viewer import Ui\_AcceptViewer  
from utils.models import ReadOnlyDelegate  
  
  
class AcceptRequestWidget(QWidget):  
 def \_\_init\_\_(self, parent, request\_id):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.request\_id = request\_id  
 self.ui = Ui\_AcceptViewer()  
 self.ui.setupUi(self)  
  
 # Настройка модели  
 self.model = QSqlRelationalTableModel(self)  
 self.model.setTable("Request\_approvals\_stages")  
 self.ui.tableView.setEditTriggers(QTableView.EditTriggers.NoEditTriggers)  
 self.model.setRelation(7, QSqlRelation("Users", "id", "first\_name || ' ' || second\_name || ' ' || third\_name"))  
  
 # Задаем заголовки таблицы  
 self.model.setHeaderData(1, Qt.Horizontal, "Статус")  
 self.model.setHeaderData(2, Qt.Horizontal, "Дата утверждения")  
 self.model.setHeaderData(3, Qt.Horizontal, "Комментарий")  
 self.model.setHeaderData(4, Qt.Horizontal, "Порядок этапа")  
 self.model.setHeaderData(7, Qt.Horizontal, "ФИО")  
  
 # Фильтрация по ID заявки  
 self.model.setFilter(f"request\_id = {self.request\_id}")  
 self.model.select()  
  
 # Настройка таблицы  
 self.ui.tableView.setModel(self.model)  
 self.ui.tableView.hideColumn(0) # Скрываем ID записи  
 self.ui.tableView.hideColumn(5) # Скрываем ID заявки  
 self.ui.tableView.hideColumn(6) # Скрываем template\_stage\_id  
 self.ui.tableView.resizeColumnsToContents()  
  
 # Устанавливаем запрет редактирования поля "Статус"  
 self.ui.tableView.setItemDelegateForColumn(1, ReadOnlyDelegate())  
  
 # Привязка кнопок  
 self.ui.save\_btn.clicked.connect(self.save\_accept)  
 self.ui.add\_btn.clicked.connect(self.add\_accept\_row)  
 self.ui.remove\_btn.clicked.connect(self.remove\_accept\_row)  
 self.ui.cancel\_btn.clicked.connect(self.revert\_contacts)  
 self.ui.close\_btn.clicked.connect(parent.close\_current\_tab)  
  
 def revert\_contacts(self):  
 self.model.revertAll()  
 self.model.select()  
  
 def save\_accept(self):  
 if self.model.submitAll():  
 QMessageBox.information(self, "Сохранение", "Изменения успешно сохранены.")  
 else:  
 QMessageBox.critical(self, "Ошибка", "Не удалось сохранить изменения.")  
 self.model.select()  
  
 def add\_accept\_row(self):  
 dialog = AcceptDialog(self.parent)  
 if dialog.exec() == QDialog.Accepted:  
 approval\_status = "Не согласовано"  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
  
 max\_stage = cur.execute("SELECT MAX(stage\_order) FROM Request\_approvals\_stages WHERE request\_id=?;", (self.request\_id,)).fetchone()[0]  
 if not max\_stage:  
 max\_stage = 0  
 else:  
 max\_stage = int(max\_stage) + 1  
  
 for stage\_order, acceptor\_id in enumerate(dialog.accepted\_users):  
 cur.execute("""  
 INSERT INTO Request\_approvals\_stages(approval\_status, stage\_order, request\_id, acceptor\_id)   
 VALUES (?, ?, ?, ?)  
 """, (approval\_status, max\_stage + stage\_order if dialog.is\_step\_by\_step else 1, self.request\_id, acceptor\_id))  
  
 con.commit()  
 con.close()  
 self.model.select()  
  
 def remove\_accept\_row(self):  
 selected\_rows = self.ui.tableView.selectionModel().selectedIndexes()  
 if not selected\_rows:  
 QMessageBox.warning(self, "Ошибка", "Не выбрана строка для удаления.")  
 return  
  
 confirm = QMessageBox.question(  
 self, "Подтверждение удаления",  
 "Вы уверены, что хотите удалить выбранные строки?",  
 QMessageBox.Yes | QMessageBox.No  
 )  
 if confirm == QMessageBox.Yes:  
 for index in reversed(selected\_rows): # Удаляем строки в обратном порядке  
 self.model.removeRow(index.row())

## ../accept\_app/accept\_invoice\_viewer.py

import sqlite3  
  
from PySide6.QtCore import Qt  
from PySide6.QtSql import QSqlRelation, QSqlRelationalTableModel  
from PySide6.QtWidgets import QMessageBox, QTableView, QWidget, QDialog  
  
from accept\_app.accept\_dialog import AcceptDialog  
from interfaces.ui\_accept\_viewer import Ui\_AcceptViewer  
from utils.models import ReadOnlyDelegate  
  
  
class AcceptInvoiceWidget(QWidget):  
 def \_\_init\_\_(self, parent, invoice\_id):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.invoice\_id = invoice\_id  
 self.ui = Ui\_AcceptViewer()  
 self.ui.setupUi(self)  
  
 # Настройка модели  
 self.model = QSqlRelationalTableModel(self)  
 self.model.setTable("Invoice\_approvals\_stages")  
 self.ui.tableView.setEditTriggers(QTableView.EditTriggers.NoEditTriggers)  
 self.model.setRelation(7, QSqlRelation("Users", "id", "first\_name || ' ' || second\_name || ' ' || third\_name"))  
  
 # Задаем заголовки таблицы  
 self.model.setHeaderData(1, Qt.Horizontal, "Статус")  
 self.model.setHeaderData(2, Qt.Horizontal, "Дата утверждения")  
 self.model.setHeaderData(3, Qt.Horizontal, "Комментарий")  
 self.model.setHeaderData(4, Qt.Horizontal, "Порядок этапа")  
 self.model.setHeaderData(7, Qt.Horizontal, "ФИО")  
  
 # Фильтрация по ID заявки  
 self.model.setFilter(f"invoice\_id = {self.invoice\_id}")  
 self.model.select()  
  
 # Настройка таблицы  
 self.ui.tableView.setModel(self.model)  
 self.ui.tableView.hideColumn(0) # Скрываем ID записи  
 self.ui.tableView.hideColumn(5) # Скрываем ID заявки  
 self.ui.tableView.hideColumn(6) # Скрываем template\_stage\_id  
 self.ui.tableView.resizeColumnsToContents()  
  
 # Устанавливаем запрет редактирования поля "Статус"  
 self.ui.tableView.setItemDelegateForColumn(1, ReadOnlyDelegate())  
  
 # Привязка кнопок  
 self.ui.save\_btn.clicked.connect(self.save\_accept)  
 self.ui.add\_btn.clicked.connect(self.add\_accept\_row)  
 self.ui.remove\_btn.clicked.connect(self.remove\_accept\_row)  
 self.ui.cancel\_btn.clicked.connect(self.revert\_contacts)  
 self.ui.close\_btn.clicked.connect(parent.close\_current\_tab)  
  
 def revert\_contacts(self):  
 self.model.revertAll()  
 self.model.select()  
  
 def save\_accept(self):  
 if self.model.submitAll():  
 QMessageBox.information(self, "Сохранение", "Изменения успешно сохранены.")  
 else:  
 QMessageBox.critical(self, "Ошибка", "Не удалось сохранить изменения.")  
 self.model.select()  
  
 def add\_accept\_row(self):  
 dialog = AcceptDialog(self.parent)  
 if dialog.exec() == QDialog.Accepted:  
 approval\_status = "Не согласовано"  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
  
 max\_stage = cur.execute("SELECT MAX(stage\_order) FROM Invoice\_approvals\_stages WHERE invoice\_id=?;", (self.invoice\_id,)).fetchone()[0]  
 if not max\_stage:  
 max\_stage = 0  
 else:  
 max\_stage = int(max\_stage) + 1  
  
 for stage\_order, acceptor\_id in enumerate(dialog.accepted\_users):  
 cur.execute("""  
 INSERT INTO Invoice\_approvals\_stages(approval\_status, stage\_order, invoice\_id, acceptor\_id)   
 VALUES (?, ?, ?, ?)  
 """, (approval\_status, max\_stage + stage\_order if dialog.is\_step\_by\_step else 1, self.invoice\_id, acceptor\_id))  
  
 con.commit()  
 con.close()  
 self.model.select()  
  
 def remove\_accept\_row(self):  
 selected\_rows = self.ui.tableView.selectionModel().selectedIndexes()  
 if not selected\_rows:  
 QMessageBox.warning(self, "Ошибка", "Не выбрана строка для удаления.")  
 return  
  
 confirm = QMessageBox.question(  
 self, "Подтверждение удаления",  
 "Вы уверены, что хотите удалить выбранные строки?",  
 QMessageBox.Yes | QMessageBox.No  
 )  
 if confirm == QMessageBox.Yes:  
 for index in reversed(selected\_rows): # Удаляем строки в обратном порядке  
 self.model.removeRow(index.row())

## ../accept\_app/accept\_invoice\_registry.py

import datetime as dt  
import sqlite3  
  
from PySide6.QtCore import Qt  
from PySide6.QtSql import QSqlRelation, QSqlRelationalTableModel  
from PySide6.QtWidgets import QMessageBox, QTableView, QWidget  
  
from interfaces.ui\_accept\_requests\_registry import Ui\_Accept\_requests\_registry  
from utils.models import DateDelegate  
  
  
class AcceptInvoiceRegistry(QWidget):  
 def \_\_init\_\_(self, parent):  
 # Инициализация виджета для обработки заявок на согласование  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.ui = Ui\_Accept\_requests\_registry()  
 self.ui.setupUi(self)  
  
 # Настройка модели таблицы для отображения заявок  
 self.model = QSqlRelationalTableModel()  
 self.model.setTable("Invoice")  
 self.model.setRelation(5, QSqlRelation("Users", "id", "login"))  
  
 # Загрузка списка заявок, доступных для согласования  
 try:  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 query = """  
 SELECT DISTINCT r1.invoice\_id  
 FROM Invoice\_approvals\_stages AS r1  
 WHERE r1.acceptor\_id = ?   
 AND r1.approval\_status = 'Не согласовано'  
 AND NOT EXISTS (  
 SELECT 1   
 FROM Invoice\_approvals\_stages AS r2  
 WHERE r2.invoice\_id = r1.invoice\_id   
 AND r2.stage\_order < r1.stage\_order   
 AND r2.approval\_status = 'Не согласовано'  
 );  
 """  
 invoice\_ids = cur.execute(query, (self.parent.user\_id,)).fetchall()  
 except sqlite3.DatabaseError as e:  
 # Обработка ошибок при работе с базой данных  
 QMessageBox.critical(self, "Ошибка базы данных", f"Не удалось загрузить данные: {e}")  
 invoice\_ids = []  
 finally:  
 con.close()  
  
 # Установка фильтра для отображения заявок текущего пользователя  
 self.model.setFilter(f"Invoice.id IN ({', '.join(map(lambda x: str(x[0]), invoice\_ids))})")  
 self.model.select()  
  
 # Настройка отображения таблицы заявок  
 self.ui.request\_list.setModel(self.model)  
 self.ui.request\_list.setEditTriggers(QTableView.EditTriggers.NoEditTriggers)  
 self.ui.request\_list.setSelectionBehavior(QTableView.SelectionBehavior.SelectRows)  
 self.ui.request\_list.hideColumn(0)  
 self.ui.request\_list.hideColumn(6)  
 self.ui.request\_list.hideColumn(7)  
 self.model.setHeaderData(1, Qt.Horizontal, "Номер")  
 self.model.setHeaderData(2, Qt.Horizontal, "Описание")  
 self.model.setHeaderData(3, Qt.Horizontal, "Дата создания")  
 self.model.setHeaderData(4, Qt.Horizontal, "Статус")  
 self.model.setHeaderData(5, Qt.Horizontal, "Закупщик")  
 self.model.setHeaderData(6, Qt.Horizontal, "Контрагент")  
 self.model.setHeaderData(7, Qt.Horizontal, "Файл")  
 self.model.setHeaderData(8, Qt.Horizontal, "Сумма")  
 date\_delegate = DateDelegate()  
 self.ui.request\_list.setItemDelegateForColumn(3, date\_delegate)  
 self.ui.request\_list.resizeColumnsToContents()  
  
 # Привязка событий к элементам интерфейса  
 self.ui.request\_list.doubleClicked.connect(self.load\_invoice)  
 self.ui.accept\_btn.clicked.connect(self.accept\_invoice)  
 self.ui.reject\_btn.clicked.connect(self.reject\_invoice)  
 self.ui.refresh\_btn.clicked.connect(self.refresh\_list)  
 self.ui.close\_btn.clicked.connect(parent.close\_current\_tab)  
  
 def refresh\_list(self):  
 """Обновление списка заявок"""  
 self.model.select()  
 self.ui.request\_list.reset()  
  
 def load\_invoice(self):  
 """Загрузка данных о выбранной заявке"""  
 selected\_rows = self.ui.request\_list.selectionModel().selectedRows()  
 if len(selected\_rows) != 1:  
 QMessageBox.warning(self, "Ошибка выбора", "Выберите только одну заявку.")  
 return  
 invoice\_id = self.model.data(selected\_rows[0])  
 if not invoice\_id:  
 QMessageBox.warning(self, "Ошибка", "Идентификатор заявки не найден.")  
 return  
 self.parent.open\_invoice\_creation\_with\_data(invoice\_id)  
 self.parent.status\_bar.showMessage("Заявка успешно выбрана", 3000)  
  
 def check\_to\_change\_invoice\_status(self):  
 """Проверка и изменение статуса заявки на основе этапов согласования"""  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 invoice\_ids = cur.execute("SELECT DISTINCT invoice\_id FROM Invoice\_approvals\_stages").fetchall()  
 invoice\_ids = map(lambda x: x[0], invoice\_ids)  
 for id in invoice\_ids:  
 statuss = cur.execute("SELECT approval\_status FROM Invoice\_approvals\_stages WHERE invoice\_id=?;", (id,)).fetchall()  
 if not statuss:  
 QMessageBox.warning(self, "Ошибка", f"Статусы для заявки {id} не найдены.")  
 continue  
 statuss = list(map(lambda x: x[0], statuss))  
 is\_acepted = list(map(lambda x: x != "Не согласовано", statuss))  
 if all(is\_acepted):  
 if "Отклонено" in list(statuss):  
 cur.execute("UPDATE Invoice SET status=? WHERE id=?;", ("Отклонено", id))  
 self.parent.status\_bar.showMessage("Статус заявки изменен на 'Отклонено'", 3000)  
 else:  
 cur.execute("UPDATE Invoice SET status=? WHERE id=?;", ("Согласовано", id))  
 self.parent.status\_bar.showMessage("Статус заявки изменен на 'Согласовано'", 3000)  
 cur.execute("DELETE FROM Invoice\_approvals\_stages WHERE invoice\_id=?;", (id,))  
 con.commit()  
 con.close()  
  
 def accept\_invoice(self):  
 """Согласование выбранной заявки"""  
 selected\_rows = self.ui.request\_list.selectionModel().selectedRows()  
 if len(selected\_rows) != 1:  
 QMessageBox.warning(self, "Ошибка выбора", "Выберите только одну заявку.")  
 return  
 approved\_at = dt.datetime.now()  
 comment = self.ui.lineEdit.text()  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 invoice\_id = self.model.data(selected\_rows[0])  
 cur.execute("""  
 UPDATE Invoice\_approvals\_stages  
 SET comment=?, approval\_status=?, approved\_at=?  
 WHERE acceptor\_id=? AND invoice\_id=?;  
 """, (comment, "Согласовано", approved\_at, self.parent.user\_id, invoice\_id))  
 con.commit()  
 con.close()  
 self.refresh\_list()  
 self.check\_to\_change\_invoice\_status()  
 self.parent.status\_bar.showMessage("Заявка согласована", 3000)  
  
 def reject\_invoice(self):  
 """Отклонение выбранной заявки"""  
 selected\_rows = self.ui.request\_list.selectionModel().selectedRows()  
 if len(selected\_rows) != 1:  
 QMessageBox.warning(self, "Ошибка выбора", "Выберите только одну заявку.")  
 return  
 approved\_at = dt.datetime.now()  
 comment = self.ui.lineEdit.text()  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 invoice\_id = self.model.data(selected\_rows[0])  
 cur.execute("""  
 UPDATE Invoice\_approvals\_stages  
 SET comment=?, approval\_status=?, approved\_at=?  
 WHERE acceptor\_id=? AND invoice\_id=?;  
 """, (comment, "Отклонено", approved\_at, self.parent.user\_id, invoice\_id))  
 con.commit()  
 con.close()  
 self.refresh\_list()  
 self.check\_to\_change\_invoice\_status()  
 self.parent.status\_bar.showMessage("Заявка отклонена", 3000)

## ../accept\_app/accept\_request\_registry.py

import datetime as dt  
import sqlite3  
  
from PySide6.QtSql import QSqlRelation, QSqlRelationalTableModel  
from PySide6.QtWidgets import QTableView, QWidget  
  
from interfaces.ui\_accept\_requests\_registry import Ui\_Accept\_requests\_registry  
  
from utils.models import DateDelegate  
  
  
class AcceptRequestRegistry(QWidget):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.ui = Ui\_Accept\_requests\_registry()  
 self.ui.setupUi(self)  
  
 self.model = QSqlRelationalTableModel()  
 self.model.setTable("Requests")  
 self.model.setRelation(4, QSqlRelation("Request\_category", "id", "name"))  
 self.model.setRelation(5, QSqlRelation("Users", "id", "login"))  
  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 query = """  
 SELECT DISTINCT r1.request\_id  
 FROM Request\_approvals\_stages AS r1  
 WHERE r1.acceptor\_id = ?   
 AND r1.approval\_status = 'Не согласовано'  
 AND NOT EXISTS (  
 SELECT 1   
 FROM Request\_approvals\_stages AS r2  
 WHERE r2.request\_id = r1.request\_id   
 AND r2.stage\_order < r1.stage\_order   
 AND r2.approval\_status = 'Не согласовано'  
 );  
 """  
 request\_ids = cur.execute(query, (self.parent.user\_id,)).fetchall()  
 con.close()  
  
 self.model.setFilter(f"Requests.id IN ({', '.join(map(lambda x: str(x[0]), request\_ids))})")  
 self.model.select()  
  
 self.ui.request\_list.setModel(self.model)  
 self.ui.request\_list.setEditTriggers(QTableView.EditTriggers.NoEditTriggers)  
 self.ui.request\_list.setSelectionBehavior(QTableView.SelectionBehavior.SelectRows)  
 self.ui.request\_list.hideColumn(0)  
 self.ui.request\_list.hideColumn(0)  
 date\_delegate = DateDelegate()  
 self.ui.request\_list.setItemDelegateForColumn(2, date\_delegate)  
 self.ui.request\_list.resizeColumnsToContents()  
  
 self.ui.request\_list.doubleClicked.connect(self.load\_request)  
 self.ui.accept\_btn.clicked.connect(self.accept\_request)  
 self.ui.reject\_btn.clicked.connect(self.reject\_request)  
 self.ui.refresh\_btn.clicked.connect(self.refresh\_list)  
 self.ui.close\_btn.clicked.connect(parent.close\_current\_tab)  
  
 def refresh\_list(self):  
 self.model.select()  
 self.ui.request\_list.reset()  
  
 def load\_request(self):  
 selected\_rows = self.ui.request\_list.selectionModel().selectedRows()  
 if len(selected\_rows) > 1:  
 return  
 request\_id = self.model.data(selected\_rows[0])  
 self.parent.open\_request\_creation\_with\_data(request\_id)  
  
 self.parent.status\_bar.showMessage("Заявка успешно выбрана", 3000)  
  
 def check\_to\_change\_request\_status(self):  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 request\_ids = cur.execute("SELECT DISTINCT request\_id FROM Request\_approvals\_stages").fetchall()  
 request\_ids = map(lambda x: x[0], request\_ids)  
 for id in request\_ids:  
 statuss = cur.execute("SELECT approval\_status FROM Request\_approvals\_stages WHERE request\_id=?;", (id,)).fetchall()  
 statuss = list(map(lambda x: x[0], statuss))  
 is\_acepted = list(map(lambda x: x != "Не согласовано", statuss))  
 if all(is\_acepted):  
 if "Отклонено" in list(statuss):  
 cur.execute("UPDATE Requests SET status=? WHERE id=?;", ("Отклонено", id))  
 self.parent.status\_bar.showMessage("Статус заявки изменен на 'Отклонено'", 3000)  
 else:  
 cur.execute("UPDATE Requests SET status=? WHERE id=?;", ("Согласовано", id))  
 self.parent.status\_bar.showMessage("Статус заявки изменен на 'Согласовано'", 3000)  
 cur.execute("DELETE FROM Request\_approvals\_stages WHERE request\_id=?;", (id,))  
 con.commit()  
 con.close()  
  
 def accept\_request(self):  
 approved\_at = dt.datetime.now()  
 comment = self.ui.lineEdit.text()  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 request\_id = self.model.data(self.ui.request\_list.selectionModel().selectedRows()[0])  
 cur.execute("""  
 UPDATE Request\_approvals\_stages  
 SET comment=?, approval\_status=?, approved\_at=?  
 WHERE acceptor\_id=? AND request\_id=?;  
 """, (comment, "Согласовано", approved\_at, self.parent.user\_id, request\_id))  
 con.commit()  
 con.close()  
 self.refresh\_list()  
 self.check\_to\_change\_request\_status()  
  
 self.parent.status\_bar.showMessage("Заявка согласована", 3000)  
  
 def reject\_request(self):  
 approved\_at = dt.datetime.now()  
 comment = self.ui.lineEdit.text()  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 request\_id = self.model.data(self.ui.request\_list.selectionModel().selectedRows()[0])  
 cur.execute("""  
 UPDATE Request\_approvals\_stages   
 SET comment=?, approval\_status=? approved\_at=?  
 WHERE acceptor\_id=? AND request\_id=?;  
 """, (comment, "Отклонено", approved\_at, self.parent.user\_id, request\_id))  
 con.commit()  
 con.close()  
 self.refresh\_list()  
 self.check\_to\_change\_request\_status()  
  
 self.parent.status\_bar.showMessage("Заявка отклонена", 3000)

## ../users/user\_registry\_widget.py

from PySide6.QtWidgets import QWidget, QTableView  
from PySide6.QtSql import QSqlRelationalTableModel  
  
from interfaces.ui\_user\_registry\_widget import Ui\_User\_registry  
  
  
class UserRegistryWidget(QWidget):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.ui = Ui\_User\_registry()  
 self.ui.setupUi(self)  
  
 self.model = QSqlRelationalTableModel()  
 self.model.setTable("Users")  
 self.model.select()  
  
 self.ui.user\_list.setModel(self.model)  
 self.ui.user\_list.setEditTriggers(QTableView.EditTriggers.NoEditTriggers)  
 self.ui.user\_list.setSelectionBehavior(QTableView.SelectionBehavior.SelectRows)  
 self.ui.user\_list.hideColumn(0)  
 self.ui.user\_list.hideColumn(6)  
 self.ui.user\_list.resizeColumnsToContents()  
  
 self.ui.user\_list.doubleClicked.connect(self.open\_user\_details)  
 self.ui.close\_btn.clicked.connect(parent.close\_current\_tab)  
 self.ui.refresh\_btn.clicked.connect(self.refresh\_values)  
 self.ui.create\_btn.clicked.connect(parent.open\_user\_creation)  
  
 def refresh\_values(self):  
 self.model.select()  
 self.ui.user\_list.reset()  
  
 def open\_user\_details(self, index):  
 user\_id = self.ui.user\_list.model().data(self.ui.user\_list.model().index(index.row(), 0))  
 self.parent.open\_user\_creation\_with\_data(user\_id)

## ../users/user\_widget.py

import sqlite3  
  
from PySide6.QtWidgets import QWidget, QMessageBox  
  
from interfaces.ui\_user\_widget import Ui\_User\_widget  
  
  
class UserWidget(QWidget):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.ui = Ui\_User\_widget()  
 self.ui.setupUi(self)  
 self.ui.delete\_btn.hide()  
 self.ui.change\_password\_frame.hide()  
  
 self.ui.save\_btn.clicked.connect(self.save\_user)  
 self.ui.close\_btn.clicked.connect(parent.close\_current\_tab)  
 self.ui.delete\_btn.clicked.connect(self.delete\_request)  
 self.ui.chenge\_password\_btn.clicked.connect(self.change\_password)  
  
 def set\_is\_created(self):  
 self.ui.save\_btn.clicked.disconnect()  
 self.ui.save\_btn.clicked.connect(self.update\_request)  
 self.ui.delete\_btn.show()  
 self.ui.change\_password\_frame.show()  
 self.ui.login\_edit.setDisabled(True)  
 self.ui.password\_edit.setDisabled(True)  
  
 def check\_and\_return\_editable\_fields(self):  
 first\_name = self.ui.first\_name\_edit.text()  
 second\_name = self.ui.second\_name\_edit.text()  
 third\_name = self.ui.third\_name\_edit.text()  
 position = self.ui.position\_edit.text()  
 login = self.ui.login\_edit.text()  
 if not login:  
 QMessageBox.warning(self, "Предупреждение", "Необходимо заполнить поле Логин")  
 return  
 password = self.ui.password\_edit.text()  
 if not password:  
 QMessageBox.warning(self, "Предупреждение", "Необходимо заполнить поле Пароль")  
 return  
 return first\_name, second\_name, third\_name, position, login, password  
  
 def save\_user(self):  
 first\_name, second\_name, third\_name, position, login, password = self.check\_and\_return\_editable\_fields()  
 access\_rights = self.ui.access\_rights\_combobox.currentText()  
 purchaser = access\_rights == "Закупщик"  
  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 try:  
 cur.execute("INSERT INTO Users(login, password, first\_name, second\_name, third\_name, position, purchaser) VALUES (?, ?, ?, ?, ?, ?, ?);",   
 (login, password, first\_name, second\_name, third\_name, position, purchaser))  
 except sqlite3.Error as err:  
 QMessageBox.critical(self, "Неизвестная ошибка", str(err))  
   
 user\_id = cur.lastrowid  
 con.commit()  
 con.close()  
  
 self.parent.tab\_widget.setTabText(self.parent.tab\_widget.currentIndex(), f"Пользователь {user\_id}")  
 self.ui.id\_ldl.setText(str(user\_id))  
 self.set\_is\_created()  
  
 self.parent.status\_bar.showMessage("Пользователь сохранен", 3000)  
  
 def update\_request(self):  
 first\_name, second\_name, third\_name, position, login, password = self.check\_and\_return\_editable\_fields()  
 user\_id = self.ui.id\_ldl.text()  
 access\_rights = self.ui.access\_rights\_combobox.currentText()  
 purchaser = access\_rights == "Закупщик"  
  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 try:  
 cur.execute("UPDATE Users SET first\_name=?, second\_name=?, third\_name=?, position=?, purchaser=? WHERE id=?;", (first\_name, second\_name, third\_name, position, purchaser, user\_id))  
 except sqlite3.Error as err:  
 QMessageBox.critical(self, "Неизвестная ошибка", str(err))  
 con.commit()  
 con.close()  
  
 self.parent.status\_bar.showMessage("Пользователь сохранен", 3000)  
  
 def load\_data(self, user\_id):  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
   
 user = cur.execute("SELECT first\_name, second\_name, third\_name, position, login, password, purchaser FROM Users WHERE id=?;", (str(user\_id),)).fetchone()  
 if user:  
 self.ui.first\_name\_edit.setText(user[0])  
 self.ui.second\_name\_edit.setText(user[1])  
 self.ui.third\_name\_edit.setText(user[2])  
 self.ui.position\_edit.setText(user[3])  
 self.ui.login\_edit.setText(user[4])  
 self.ui.password\_edit.setText(user[5])  
 if user[6]:  
 self.ui.access\_rights\_combobox.setCurrentIndex(1)  
 else:  
 self.ui.access\_rights\_combobox.setCurrentIndex(0)  
  
 con.close()  
 self.set\_is\_created()  
 self.ui.id\_ldl.setText(str(user\_id))  
  
 def delete\_request(self):  
 res = QMessageBox.warning(self, "Предупреждение", "Вы уверены, что хотите удалить объект?", QMessageBox.Yes, QMessageBox.No)  
 if res == QMessageBox.No:  
 return  
 user\_id = self.ui.id\_ldl.text()  
  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 cur.execute("DELETE FROM USERS WHERE id=?", (user\_id,))  
 con.commit()  
 con.close()  
  
 self.parent.close\_current\_tab()  
  
 self.parent.status\_bar.showMessage("Пользователь удален", 3000)  
  
 def change\_password(self):  
 password1 = self.ui.new\_password1\_edit.text()  
 password2 = self.ui.new\_password2\_edit.text()  
 if password1 != password2:  
 QMessageBox.warning(self, "Предупреждение", "Введенные пароли не совпадают")  
 return  
 user\_id = self.ui.id\_ldl.text()  
  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 try:  
 cur.execute("UPDATE Users SET password=? WHERE id=?;", (password1, user\_id))  
 except sqlite3.Error as err:  
 QMessageBox.critical(self, "Неизвестная ошибка", str(err))  
 con.commit()  
 con.close()  
 QMessageBox.information(self, "Завершено успешно", "Пароль изменен")

## ../vender\_app/vender\_registry.py

import sqlite3  
  
from PySide6.QtWidgets import QMessageBox, QTableView, QTableWidgetItem, QWidget  
  
from interfaces.ui\_vender\_registry import Ui\_VenderRegistry  
  
  
class VenderRegistry(QWidget):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.ui = Ui\_VenderRegistry()  
 self.ui.setupUi(self)  
 self.ui.tableWidget.setEditTriggers(QTableView.EditTriggers.NoEditTriggers)  
 self.ui.tableWidget.setSelectionBehavior(QTableView.SelectionBehavior.SelectRows)  
 self.ui.tableWidget.hideColumn(0)  
 self.ui.tableWidget.resizeColumnsToContents()  
  
 self.update\_list()  
  
 self.ui.tableWidget.doubleClicked.connect(self.open\_vender)  
 self.ui.close\_btn.clicked.connect(parent.close\_current\_tab)  
 self.ui.refresh\_btn.clicked.connect(self.update\_list)  
 self.ui.create\_btn.clicked.connect(parent.open\_vender\_creation)  
  
 def update\_list(self):  
 self.ui.tableWidget.clearContents()  
 self.ui.tableWidget.setRowCount(0)  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 contracts = cur.execute("SELECT id, name, address FROM Vendor;").fetchall()  
 con.close()  
 self.ui.tableWidget.setRowCount(len(contracts))  
 for row, contract in enumerate(contracts):  
 self.ui.tableWidget.setItem(row, 0, QTableWidgetItem(str(contract[0])))  
 self.ui.tableWidget.setItem(row, 1, QTableWidgetItem(str(contract[1])))  
 self.ui.tableWidget.setItem(row, 2, QTableWidgetItem(str(contract[2])))  
 self.ui.tableWidget.resizeColumnsToContents()  
  
 def open\_vender(self, index):  
 vender\_id\_item = self.ui.tableWidget.item(index.row(), 0)  
 if vender\_id\_item is not None:  
 try:  
 vender\_id = int(vender\_id\_item.text())  
 self.parent.open\_vender\_with\_data(vender\_id)  
 except ValueError:  
 QMessageBox.warning(self, "Ошибка", "Некорректный идентификатор.")  
 else:  
 QMessageBox.warning(self, "Ошибка", "Не удалось получить идентификатор.")

## ../vender\_app/vender\_dialog.py

import sqlite3  
  
from PySide6.QtWidgets import QDialog, QListWidgetItem, QMessageBox  
  
from interfaces.ui\_vender\_dialog import Ui\_Vender\_dialog  
  
  
class VenderDialog(QDialog):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.ui = Ui\_Vender\_dialog()  
 self.ui.setupUi(self)  
  
 self.update\_list()  
  
 self.ui.add\_btn.clicked.connect(self.add\_vender)  
 self.ui.buttonBox.accepted.disconnect()  
 self.ui.buttonBox.accepted.connect(self.select\_vender)  
  
 def update\_list(self):  
 self.ui.listWidget.clear()  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 venders = cur.execute("SELECT id, name FROM Vendor;").fetchall()  
 con.close()  
 for vender in venders:  
 QListWidgetItem(vender[1], self.ui.listWidget)  
  
 def select\_vender(self):  
 rows = self.ui.listWidget.selectedItems()  
 if len(rows) != 1:  
 QMessageBox.warning(self, "Предупреждение", "Выберите одного контрагента")  
 return  
 self.vender = self.ui.listWidget.currentItem().text()  
 self.accept()  
  
 def add\_vender(self):  
 name = self.ui.name\_edit.text()  
 if name:  
 try:  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 cur.execute("INSERT INTO Vendor(name) VALUES (?)", (name,))  
 con.commit()  
 con.close()  
 self.update\_list()  
 except sqlite3.IntegrityError:  
 QMessageBox.critical(self, "Ошибка", "Наименование уже присутствует в базе")

## ../vender\_app/vender\_widget.py

import sqlite3  
  
from PySide6.QtCore import Qt  
from PySide6.QtSql import QSqlRelationalTableModel, QSqlTableModel  
from PySide6.QtWidgets import QMessageBox, QWidget  
  
from interfaces.ui\_vender import Ui\_Form  
  
  
class VenderWidget(QWidget):  
 def \_\_init\_\_(self, parent, vender\_id=None):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.vender\_id = vender\_id  
 self.ui = Ui\_Form()  
 self.ui.setupUi(self)  
  
 # Настройка начального состояния кнопки удаления  
 self.ui.delete\_btn.setDisabled(self.vender\_id is None)  
  
 # Настройка модели  
 self.model = QSqlRelationalTableModel(self)  
 self.model.setTable("Vendor\_managers")  
 self.model.setEditStrategy(QSqlTableModel.OnManualSubmit)  
  
 # Задаем заголовки таблицы  
 self.model.setHeaderData(1, Qt.Horizontal, "Имя")  
 self.model.setHeaderData(2, Qt.Horizontal, "Фамилия")  
 self.model.setHeaderData(3, Qt.Horizontal, "Отчество")  
 self.model.setHeaderData(4, Qt.Horizontal, "Должность")  
 self.model.setHeaderData(5, Qt.Horizontal, "Номер телефона")  
  
 # Фильтрация по ID поставщика  
 if self.vender\_id is not None:  
 self.model.setFilter(f"vender\_id = {self.vender\_id}")  
 self.load\_vendor\_data()  
 else:  
 self.model.setFilter("vender\_id = -1")  
  
 self.model.select()  
  
 # Настройка таблицы  
 self.ui.tableView.setModel(self.model)  
 self.ui.tableView.hideColumn(0)  
 self.ui.tableView.hideColumn(6)  
 self.ui.tableView.resizeColumnsToContents()  
  
 # Привязка кнопок  
 self.ui.save\_btn.clicked.connect(self.save\_vendor)  
 self.ui.delete\_btn.clicked.connect(self.delete\_vendor)  
 self.ui.add\_btn.clicked.connect(self.add\_manager\_row)  
 self.ui.remove\_btn.clicked.connect(self.remove\_manager\_row)  
 self.ui.cancel\_btn.clicked.connect(self.revert\_contacts)  
  
 def revert\_contacts(self):  
 self.model.revertAll()  
 self.model.select()  
  
 def load\_vendor\_data(self):  
 """Загрузка данных контрагента по ID."""  
 try:  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 cur.execute("SELECT name, address FROM Vendor WHERE id = ?", (self.vender\_id,))  
 result = cur.fetchone()  
 if result:  
 name, address = result  
 self.ui.name\_edit.setText(name)  
 self.ui.address\_edit.setText(address)  
 else:  
 QMessageBox.warning(self, "Ошибка", "Контрагент с указанным ID не найден.")  
 except sqlite3.DatabaseError as e:  
 QMessageBox.critical(self, "Ошибка базы данных", f"Не удалось загрузить данные контрагента: {e}")  
 finally:  
 con.close()  
  
 def save\_vendor(self):  
 name = self.ui.name\_edit.text().strip()  
 address = self.ui.address\_edit.text().strip()  
  
 if not name:  
 QMessageBox.warning(self, "Ошибка", "Вы не заполнили Наименование контрагента")  
 return  
  
 try:  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
  
 if self.vender\_id is None:  
 cur.execute("INSERT INTO Vendor (name, address) VALUES (?, ?)", (name, address))  
 self.vender\_id = cur.lastrowid  
 QMessageBox.information(self, "Успех", "Поставщик успешно создан.")  
 else:  
 cur.execute("UPDATE Vendor SET name = ?, address = ? WHERE id = ?", (name, address, self.vender\_id))  
 QMessageBox.information(self, "Успех", "Данные поставщика успешно обновлены.")  
  
 con.commit()  
 self.model.submitAll()  
  
 self.model.setFilter(f"vender\_id = {self.vender\_id}")  
 self.model.select()  
 self.ui.delete\_btn.setDisabled(False)  
  
 except sqlite3.DatabaseError as e:  
 QMessageBox.critical(self, "Ошибка базы данных", f"Не удалось сохранить данные: {e}")  
 finally:  
 con.close()  
  
 def delete\_vendor(self):  
 """Удаление поставщика."""  
 confirm = QMessageBox.question(  
 self, "Подтверждение удаления",  
 "Вы уверены, что хотите удалить этого поставщика и всех его менеджеров?",  
 QMessageBox.Yes | QMessageBox.No  
 )  
  
 if confirm == QMessageBox.Yes:  
 try:  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 cur.execute("DELETE FROM Vendor\_managers WHERE vender\_id = ?", (self.vender\_id,))  
 cur.execute("DELETE FROM Vendor WHERE id = ?", (self.vender\_id,))  
 con.commit()  
 QMessageBox.information(self, "Успех", "Поставщик успешно удален.")  
 self.parent.close\_current\_tab()  
 except sqlite3.DatabaseError as e:  
 QMessageBox.critical(self, "Ошибка базы данных", f"Не удалось удалить поставщика: {e}")  
 finally:  
 con.close()  
  
 def add\_manager\_row(self):  
 """Добавление строки для нового менеджера."""  
 if self.vender\_id is None:  
 QMessageBox.warning(self, "Ошибка", "Сначала сохраните поставщика.")  
 return  
  
 row = self.model.rowCount()  
 self.model.insertRow(row)  
 self.model.setData(self.model.index(row, 6), self.vender\_id) # Установка vendor\_id для новой строки  
 self.ui.tableView.scrollToBottom()  
  
 def remove\_manager\_row(self):  
 """Удаление выбранной строки менеджера."""  
 selected\_rows = self.ui.tableView.selectionModel().selectedIndexes()  
 if not selected\_rows:  
 QMessageBox.warning(self, "Ошибка", "Не выбрана строка для удаления.")  
 return  
   
 rows\_to\_remove = [index.row() for index in selected\_rows]  
 for row in rows\_to\_remove:  
 self.model.removeRow(row)

## ../nomenclature/nomenclature\_dialog.py

import sqlite3  
  
from PySide6.QtWidgets import QDialog, QTableWidgetItem, QTableView, QMessageBox  
  
from interfaces.ui\_nomenclature\_dialog import Ui\_Nomenclature\_dialog  
  
  
class NomenclatureDialog(QDialog):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.ui = Ui\_Nomenclature\_dialog()  
 self.ui.setupUi(self)  
 self.update\_list()  
  
 self.ui.tableWidget.hideColumn(0)  
 self.ui.tableWidget.setEditTriggers(QTableView.EditTriggers.NoEditTriggers)  
  
 self.ui.pushButton.clicked.connect(self.add\_nomenclature)  
 self.ui.buttonBox.accepted.connect(self.select\_nomenclature)  
  
 def update\_list(self):  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 nomenclatures = cur.execute("SELECT \* FROM Nomenclature").fetchall()  
 con.close()  
  
 self.ui.tableWidget.setRowCount(len(nomenclatures))  
 for i, nomenclature in enumerate(nomenclatures):  
 self.ui.tableWidget.setItem(i, 0, QTableWidgetItem(str(nomenclature[0])))  
 self.ui.tableWidget.setItem(i, 1, QTableWidgetItem(nomenclature[1]))  
 self.ui.tableWidget.setItem(i, 2, QTableWidgetItem(nomenclature[2]))  
  
 def add\_nomenclature(self):  
 new\_name = self.ui.lineEdit.text().strip()  
 new\_unit = self.ui.lineEdit\_2.text().strip()  
 if new\_name and new\_unit:  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 cur.execute("INSERT INTO Nomenclature (name, unit) VALUES (?, ?)", (new\_name, new\_unit))  
 con.commit()  
 con.close()  
 self.update\_list()  
 self.ui.lineEdit.clear()  
 self.ui.lineEdit\_2.clear()  
  
 def select\_nomenclature(self):  
 items = self.ui.tableWidget.selectedItems()  
 if len(items) != 1:  
 QMessageBox.warning(self, "Предупреждение", "Выберете одну позицию")  
 return  
 row = items[0].row()  
 nomenclature\_id = self.ui.tableWidget.item(row, 0).text()  
 if nomenclature\_id:  
 self.nomenclature\_id = nomenclature\_id  
 self.accept()

## ../requests\_app/request\_widget.py

import sqlite3  
import datetime as dt  
  
from PySide6.QtWidgets import QWidget, QDialog, QTableWidgetItem, QMessageBox  
  
from interfaces.ui\_create\_request import Ui\_Request  
from nomenclature.nomenclature\_dialog import NomenclatureDialog  
from requests\_app.request\_category\_dialog import RequestCategoryDialog  
from accept\_app.accept\_dialog import AcceptDialog  
  
  
class RequestWidget(QWidget):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.id = None  
 self.ui = Ui\_Request()  
 self.ui.setupUi(self)  
 self.ui.delete\_btn.hide()  
 self.ui.mark\_done\_btn.hide()  
 self.ui.send\_btn.hide()  
 self.ui.accept\_viewer\_btn.hide()  
 self.ui.date\_lbl.setText("--.--.----")  
 self.ui.status\_lbl.setText("---------")  
 self.setup\_category\_combobox()  
  
 self.ui.tableWidget.hideColumn(0)  
  
 if not self.parent.purchaser:  
 self.ui.category\_dialog\_btn.hide()  
 self.ui.mark\_done\_btn.hide()  
  
 self.ui.add\_nomenclature\_btn.clicked.connect(self.open\_nomenclature\_dialog)  
 self.ui.delete\_nomenclature\_btn.clicked.connect(self.delete\_nomenclature\_row)  
 self.ui.save\_btn.clicked.connect(self.save\_request)  
 self.ui.close\_btn.clicked.connect(parent.close\_current\_tab)  
 self.ui.delete\_btn.clicked.connect(self.delete\_request)  
 self.ui.category\_dialog\_btn.clicked.connect(self.open\_category\_dialog)  
 self.ui.send\_btn.clicked.connect(self.open\_accept\_dialog)  
 self.ui.mark\_done\_btn.clicked.connect(self.mark\_done)  
 self.ui.accept\_viewer\_btn.clicked.connect(lambda: parent.open\_accept\_request\_viewer(self.id))  
  
 def setup\_category\_combobox(self):  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 categories = cur.execute("SELECT name FROM Request\_category").fetchall()  
 con.close()  
 self.ui.category\_combobox.clear()  
 categories = list(map(lambda x: x[0], categories))  
 self.ui.category\_combobox.insertItems(0, categories)  
  
 def open\_category\_dialog(self):  
 dialog = RequestCategoryDialog(self.parent)  
 if dialog.exec() == QDialog.Accepted:  
 self.setup\_category\_combobox()  
 if dialog.category:  
 self.ui.category\_combobox.setCurrentText(dialog.category)  
  
 def open\_nomenclature\_dialog(self):  
 dialog = NomenclatureDialog(self.parent)  
 if dialog.exec() == QDialog.Accepted:  
 if dialog.nomenclature\_id:  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 nomenclature = cur.execute("SELECT \* FROM Nomenclature WHERE id=?",  
 (dialog.nomenclature\_id,)).fetchone()  
 con.close()  
  
 rows = self.ui.tableWidget.rowCount()  
 self.ui.tableWidget.setRowCount(rows + 1)  
 self.ui.tableWidget.setItem(rows, 0, QTableWidgetItem(str(nomenclature[0])))  
 self.ui.tableWidget.setItem(rows, 1, QTableWidgetItem(nomenclature[1]))  
 self.ui.tableWidget.setItem(rows, 2, QTableWidgetItem(nomenclature[2]))  
  
 def delete\_nomenclature\_row(self):  
 self.ui.tableWidget.removeRow(self.ui.tableWidget.currentRow())  
  
 def set\_is\_created(self, id):  
 self.ui.save\_btn.clicked.disconnect()  
 self.ui.save\_btn.clicked.connect(self.update\_request)  
 self.id = id  
 self.ui.delete\_btn.show()  
 self.ui.send\_btn.show()  
 self.ui.accept\_viewer\_btn.show()  
 if self.parent.purchaser:  
 self.ui.mark\_done\_btn.show()  
  
 def set\_uneditable\_fields(self, date, status):  
 self.ui.date\_lbl.setText(date.split(".")[0])  
 self.ui.status\_lbl.setText(status)  
  
 def check\_and\_return\_editable\_fields(self):  
 description = self.ui.description\_text.toPlainText()  
 rows = self.ui.tableWidget.rowCount()  
 amounts = [self.ui.tableWidget.item(row, 3) for row in range(rows)]  
 item\_ids = None  
  
 if not description:  
 QMessageBox.warning(self, "Предупреждение", "Вы не заполнили описание")  
 elif not rows:  
 QMessageBox.warning(self, "Предупреждение", "Вы не добавили номенклатуру")  
 elif not all(amounts):  
 QMessageBox.warning(self, "Предупреждение", "Вы не указали `Количество` у некоторых позиций")  
 else:  
 amounts = map(lambda item: item.text(), amounts)  
 item\_ids = [self.ui.tableWidget.item(row, 0).text() for row in range(rows)]  
  
 return description, rows, amounts, item\_ids  
  
 def save\_request(self):  
 description, rows, amounts, item\_ids = self.check\_and\_return\_editable\_fields()  
 if not (description and rows and all(amounts)):  
 return  
 category\_name = self.ui.category\_combobox.currentText()  
  
 created\_at = dt.datetime.now()  
 status = "Не согласовано"  
 user\_id = self.parent.user\_id  
  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 category\_id = cur.execute("SELECT id FROM Request\_category WHERE name=?", (category\_name,)).fetchone()[0]  
 try:  
 cur.execute(  
 "INSERT INTO Requests(description, created\_at, status, category\_id, initiator\_id) VALUES (?, ?, ?, ?, ?);",  
 (description, created\_at, status, category\_id, user\_id))  
 except sqlite3.Error as err:  
 QMessageBox.critical(self, "Неизвестная ошибка", str(err))  
  
 request\_id = cur.lastrowid  
 request\_ids = [str(request\_id)] \* rows  
  
 cur.executemany("INSERT INTO Request\_items(amount, request\_id, item\_id) VALUES (?, ?, ?);",  
 zip(amounts, request\_ids, item\_ids))  
 con.commit()  
 con.close()  
  
 self.parent.tab\_widget.setTabText(self.parent.tab\_widget.currentIndex(), f"Заявка {request\_id}")  
 self.set\_is\_created(request\_id)  
 self.set\_uneditable\_fields(str(created\_at), status)  
 self.parent.status\_bar.showMessage("Заявка успешно сохранена", 3000)  
  
 def update\_request(self):  
 description, rows, amounts, item\_ids = self.check\_and\_return\_editable\_fields()  
 if not (description and rows and amounts):  
 return  
 category\_name = self.ui.category\_combobox.currentText()  
  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 category\_id = cur.execute("SELECT id FROM Request\_category WHERE name=?", (category\_name,)).fetchone()[0]  
 try:  
 cur.execute("UPDATE Requests SET description=?, category\_id=? WHERE id=?;",  
 (description, category\_id, self.id))  
 except sqlite3.OperationalError as err:  
 QMessageBox.critical(self, "Неизвестная ошибка", str(err))  
  
 request\_ids = [str(self.id)] \* rows  
  
 cur.execute("DELETE FROM Request\_items WHERE request\_id=?", (self.id,))  
 cur.executemany("INSERT INTO Request\_items(amount, request\_id, item\_id) VALUES (?, ?, ?);",  
 zip(amounts, request\_ids, item\_ids))  
 con.commit()  
 con.close()  
 self.parent.status\_bar.showMessage("Заявка успешно сохранена", 3000)  
  
 def load\_request\_data(self, request\_id):  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
  
 # Получаем основную информацию о заявке  
 request = cur.execute("""  
 SELECT   
 Requests.description,   
 Requests.created\_at,   
 Requests.status,   
 Request\_category.name   
 FROM Requests   
 LEFT JOIN Request\_category ON Requests.category\_id=Request\_category.id   
 WHERE Requests.id = ?;""", (request\_id,)).fetchone()  
 if request:  
 self.ui.description\_text.setPlainText(request[0])  
 self.ui.category\_combobox.setCurrentText(request[3])  
 self.set\_uneditable\_fields(str(request[1]), request[2])  
  
 # Очищаем таблицу и добавляем связанные позиции заявки  
 self.ui.tableWidget.setRowCount(0)  
 items = cur.execute("SELECT item\_id, amount FROM Request\_items WHERE request\_id = ?", (request\_id,)).fetchall()  
 for item\_id, amount in items:  
 nomenclature = cur.execute("SELECT name, unit FROM Nomenclature WHERE id = ?", (item\_id,)).fetchone()  
 if nomenclature:  
 row = self.ui.tableWidget.rowCount()  
 self.ui.tableWidget.setRowCount(row + 1)  
 self.ui.tableWidget.setItem(row, 0, QTableWidgetItem(str(item\_id)))  
 self.ui.tableWidget.setItem(row, 1, QTableWidgetItem(nomenclature[0]))  
 self.ui.tableWidget.setItem(row, 2, QTableWidgetItem(nomenclature[1]))  
 self.ui.tableWidget.setItem(row, 3, QTableWidgetItem(str(amount)))  
  
 con.close()  
 self.set\_is\_created(request\_id)  
  
 def delete\_request(self):  
 res = QMessageBox.warning(self, "Предупреждение", "Вы уверены, что хотите удалить объект?", QMessageBox.Yes,  
 QMessageBox.No)  
 if res == QMessageBox.No:  
 return  
  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 cur.execute("DELETE FROM Request\_items WHERE request\_id=?", (self.id,))  
 cur.execute("DELETE FROM Requests WHERE id=?", (self.id,))  
 con.commit()  
 con.close()  
  
 self.parent.close\_current\_tab()  
 self.parent.status\_bar.showMessage("Заявка успешно удалена", 3000)  
  
 def open\_accept\_dialog(self):  
 if not self.id:  
 ans = QMessageBox.warning(self, "Предупреждение", "Для продолжения необходимо сохранить объект")  
 if ans == QMessageBox.Ok:  
 self.save\_request()  
 else:  
 return  
 dialog = AcceptDialog(self.parent)  
 if dialog.exec() == QDialog.Accepted:  
 approval\_status = "Не согласовано"  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
  
 max\_stage = cur.execute("SELECT MAX(stage\_order) FROM Request\_approvals\_stages WHERE request\_id=?;",  
 (self.id,)).fetchone()[0]  
 if not max\_stage:  
 max\_stage = 0  
 else:  
 max\_stage = int(max\_stage) + 1  
  
 for stage\_order, acceptor\_id in enumerate(dialog.accepted\_users):  
 cur.execute("""  
 INSERT INTO Request\_approvals\_stages(approval\_status, stage\_order, request\_id, acceptor\_id)   
 VALUES (?, ?, ?, ?)  
 """, (  
 approval\_status, max\_stage + stage\_order if dialog.is\_step\_by\_step else max\_stage, self.id,  
 acceptor\_id))  
  
 con.commit()  
 con.close()  
 if not dialog.accepted\_users:  
 self.parent.status\_bar.showMessage("Не выбраны согласованты", 3000)  
 else:  
 self.parent.status\_bar.showMessage("Заявка успешно отправлена на согласование", 3000)  
  
 def mark\_done(self):  
 ans = QMessageBox(self, "Предупреждение", 'Вы уверены, что хотите установить статус заявки "Выполнено?"')  
 if ans != QMessageBox.Ok:  
 return  
 self.save\_request()  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 cur.execute("UPDATE Requests SET status=? WHERE id=?;", ("Выполнено", self.id))  
 con.commit()  
 con.close()  
  
 self.parent.status\_bar.showMessage("Заявка помечена как Выполнена", 3000)

## ../requests\_app/request\_registry\_widget.py

import sqlite3  
  
from PySide6.QtWidgets import QWidget, QTableView  
from PySide6.QtSql import QSqlRelationalTableModel, QSqlRelation  
  
from interfaces.ui\_request\_registry import Ui\_Request\_registry  
from utils.models import DateDelegate  
  
  
class RequestRegistryWidget(QWidget):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.ui = Ui\_Request\_registry()  
 self.ui.setupUi(self)  
  
 self.model = QSqlRelationalTableModel()  
 self.model.setTable("Requests")  
 self.model.setRelation(4, QSqlRelation("Request\_category", "id", "name"))  
 self.model.setRelation(5, QSqlRelation("Users", "id", "login"))  
 if not self.parent.purchaser:  
 self.model.setFilter(f"initiator\_id={self.parent.user\_id}")  
 self.model.select()  
  
 self.ui.request\_list.setModel(self.model)  
 self.ui.request\_list.setEditTriggers(QTableView.EditTriggers.NoEditTriggers)  
 self.ui.request\_list.setSelectionBehavior(QTableView.SelectionBehavior.SelectRows)  
 self.ui.request\_list.hideColumn(0)  
 self.ui.request\_list.hideColumn(0)  
 date\_delegate = DateDelegate()  
 self.ui.request\_list.setItemDelegateForColumn(2, date\_delegate)  
 self.ui.request\_list.resizeColumnsToContents()  
  
 self.setup\_category\_combobox()  
  
 self.ui.request\_list.doubleClicked.connect(self.open\_request\_details)  
 self.ui.close\_btn.clicked.connect(parent.close\_current\_tab)  
 self.ui.refresh\_btn.clicked.connect(self.refresh\_values)  
 self.ui.create\_btn.clicked.connect(parent.open\_request\_creation)  
 self.ui.category\_combobox.currentTextChanged.connect(self.filter\_list)  
 self.ui.comboBox.currentTextChanged.connect(self.filter\_list)  
  
 def filter\_list(self):  
 category\_name = self.ui.category\_combobox.currentText()  
 status\_name = self.ui.comboBox.currentText()  
  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
  
 if category\_name == "-" and status\_name == "-":  
 if not self.parent.purchaser:  
 self.model.setFilter(f"initiator\_id={self.parent.user\_id}")  
 else:  
 self.model.setFilter("")  
 return  
 elif status\_name == "-":  
 category\_id = cur.execute("SELECT id FROM Request\_category WHERE name=?", (category\_name,)).fetchone()[0]  
 if not self.parent.purchaser:  
 self.model.setFilter(f"category\_id={category\_id} AND initiator\_id={self.parent.user\_id}")  
 else:  
 self.model.setFilter(f"category\_id={category\_id}")  
 elif category\_name == "-":  
 if not self.parent.purchaser:  
 self.model.setFilter(f"status='{status\_name}' AND initiator\_id={self.parent.user\_id}")  
 else:  
 self.model.setFilter(f"status='{status\_name}'")  
 else:  
 category\_id = cur.execute("SELECT id FROM Request\_category WHERE name=?", (category\_name,)).fetchone()[0]  
 if not self.parent.purchaser:  
 self.model.setFilter(f"category\_id={category\_id} AND status='{status\_name}' AND initiator\_id={self.parent.user\_id}")  
 else:  
 self.model.setFilter(f"category\_id={category\_id} AND status='{status\_name}'")  
  
 con.close()  
 self.refresh\_values()  
  
 def setup\_category\_combobox(self):  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 categories = cur.execute("SELECT name FROM Request\_category").fetchall()  
 con.close()  
 self.ui.category\_combobox.clear()  
 categories = list(map(lambda x: x[0], categories))  
 self.ui.category\_combobox.insertItems(0, categories)  
 self.ui.category\_combobox.insertItem(0, "-")  
 self.ui.category\_combobox.setCurrentIndex(0)  
  
 def open\_request\_details(self, index):  
 # Получаем ID заявки из модели, например, из первой колонки  
 request\_id = self.ui.request\_list.model().data(self.ui.request\_list.model().index(index.row(), 0))  
 self.parent.open\_request\_creation\_with\_data(request\_id)  
  
 def refresh\_values(self):  
 self.model.select()  
 self.ui.request\_list.reset()

## ../requests\_app/request\_category\_dialog.py

import sqlite3  
  
from PySide6.QtWidgets import QDialog, QListWidgetItem, QMessageBox  
  
from interfaces.ui\_request\_category\_dialog import Ui\_Request\_category  
  
  
class RequestCategoryDialog(QDialog):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.ui = Ui\_Request\_category()  
 self.ui.setupUi(self)  
  
 self.update\_list()  
  
 self.ui.add\_btn.clicked.connect(self.add\_category)  
 self.ui.buttonBox.accepted.connect(self.select\_category)  
  
 def update\_list(self):  
 self.ui.category\_list.clear()  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 categories = cur.execute("SELECT id, name FROM Request\_category;").fetchall()  
 con.close()  
 for category in categories:  
 QListWidgetItem(category[1], self.ui.category\_list)  
  
 def select\_category(self):  
 categories = self.ui.category\_list.selectedItems()  
 if len(categories) != 1:  
 QMessageBox.warning(self, "Предупреждение", "Выберете одну категорию")  
 return  
 self.category = categories[0].text()  
 self.accept()  
  
 def add\_category(self):  
 name = self.ui.name\_edit.text()  
 if name:  
 try:  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 cur.execute("INSERT INTO Request\_category(name) VALUES (?)", (name,))  
 con.commit()  
 con.close()  
 self.update\_list()  
 except sqlite3.IntegrityError as err:  
 QMessageBox.critical(self, "Ошибка", "Наименование уже присутствует в базе")  
 print(err)

## ../files\_app/file\_dialog.py

import sqlite3  
  
from PySide6.QtWidgets import QDialog, QFileDialog, QMessageBox  
from PySide6.QtCore import QUrl  
from PySide6.QtGui import QDesktopServices  
  
from interfaces.ui\_file\_dialog import Ui\_FileDialog  
  
  
class FileDialog(QDialog):  
 def \_\_init\_\_(self, parent, invoice\_id):  
 super().\_\_init\_\_()  
 self.parent = parent  
 self.invoice\_id = invoice\_id  
 self.ui = Ui\_FileDialog()  
 self.ui.setupUi(self)  
  
 # Подключение сигналов к слотам  
 self.ui.attach\_file.clicked.connect(self.attach\_file)  
 self.ui.delete\_file.clicked.connect(self.delete\_file)  
 self.ui.open\_file.clicked.connect(self.open\_file)  
 self.ui.buttonBox.accepted.connect(self.accept)  
  
 self.load\_file\_path()  
  
 def load\_file\_path(self):  
 """Загружает текущий путь к файлу из базы данных."""  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 cur.execute("SELECT file\_path FROM Invoice WHERE id = ?", (self.invoice\_id,))  
 result = cur.fetchone()  
 con.close()  
  
 if result and result[0]:  
 self.ui.filename\_lbl.setText(result[0])  
 else:  
 self.ui.filename\_lbl.setText("Файл не прикреплен.")  
  
 def attach\_file(self):  
 """Прикрепляет новый файл к счету и сохраняет его в базе."""  
 file\_path, \_ = QFileDialog.getOpenFileName(self, "Выбрать файл")  
 if file\_path:  
 # Сохраняем путь в базе данных  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 cur.execute("UPDATE Invoice SET file\_path = ? WHERE id = ?", (file\_path, self.invoice\_id))  
 con.commit()  
 con.close()  
  
 # Обновляем отображение  
 self.ui.filename\_lbl.setText(file\_path)  
 QMessageBox.information(self, "Успех", "Файл успешно прикреплен.")  
  
 def delete\_file(self):  
 """Удаляет прикрепленный файл из базы данных."""  
 confirmation = QMessageBox.question(  
 self, "Удалить файл", "Вы уверены, что хотите удалить прикрепленный файл?",  
 QMessageBox.Yes | QMessageBox.No  
 )  
 if confirmation == QMessageBox.Yes:  
 # Удаляем путь из базы данных  
 con = sqlite3.connect(self.parent.database\_file)  
 cur = con.cursor()  
 cur.execute("UPDATE Invoice SET file\_path = NULL WHERE id = ?", (self.invoice\_id,))  
 con.commit()  
 con.close()  
  
 # Обновляем отображение  
 self.ui.filename\_lbl.setText("Файл не прикреплен.")  
 QMessageBox.information(self, "Успех", "Файл успешно удален.")  
  
 def open\_file(self):  
 """Открывает прикрепленный файл через системное приложение."""  
 file\_path = self.ui.filename\_lbl.text()  
 if file\_path and file\_path != "Файл не прикреплен.":  
 QDesktopServices.openUrl(QUrl.fromLocalFile(file\_path))  
 else:  
 QMessageBox.warning(self, "Ошибка", "Файл не прикреплен.")

## ../reports/reports\_widget.py

import sqlite3  
from PySide6.QtWidgets import QVBoxLayout, QWidget, QTableWidget, QTableWidgetItem, QPushButton  
from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas  
from matplotlib.figure import Figure  
  
  
class ReportWidget(QWidget):  
 def \_\_init\_\_(self, parent):  
 self.parent = parent  
 super().\_\_init\_\_()  
 self.setWindowTitle("Отчеты")  
  
 # Buttons to switch between reports  
 self.layout = QVBoxLayout(self)  
 self.buttons = {  
 "Объем заявок по месяцам": Report1,  
 "Топ-10 поставщиков": Report2,  
 "Статистика по категориям": Report3,  
 "Рейтинг сотрудников": Report6,  
 }  
 for name, report\_class in self.buttons.items():  
 btn = QPushButton(name)  
 btn.clicked.connect(lambda checked, cls=report\_class, \_parent=self.parent: self.show\_report(cls, \_parent))  
 self.layout.addWidget(btn)  
  
 self.current\_widget = None  
  
 def show\_report(self, report\_class, parent):  
 if self.current\_widget:  
 self.current\_widget.deleteLater()  
 self.current\_widget = report\_class(parent)  
 self.layout.addWidget(self.current\_widget)  
  
  
# Report 1: Объем заявок по месяцам  
class Report1(QWidget):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 layout = QVBoxLayout(self)  
 conn = sqlite3.connect(parent.database\_file)  
 cursor = conn.cursor()  
 cursor.execute("SELECT strftime('%Y-%m', created\_at) AS month, COUNT(\*) FROM Requests GROUP BY month")  
 data = cursor.fetchall()  
 conn.close()  
  
 months, counts = zip(\*data) if data else ([], [])  
 fig = Figure(figsize=(5, 3))  
 ax = fig.add\_subplot(111)  
 ax.bar(months, counts)  
 ax.set\_title("Объем заявок по месяцам")  
 ax.set\_xlabel("Месяц")  
 ax.set\_ylabel("Количество заявок")  
  
 canvas = FigureCanvas(fig)  
 layout.addWidget(canvas)  
  
  
# Report 2: Топ-10 поставщиков  
class Report2(QWidget):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 layout = QVBoxLayout(self)  
 conn = sqlite3.connect(parent.database\_file)  
 cursor = conn.cursor()  
 cursor.execute("""  
 SELECT Vendor.name, COUNT(\*)   
 FROM Invoice  
 LEFT JOIN Contracts ON Invoice.contract\_id = Contracts.id  
 LEFT JOIN Vendor ON Contracts.vender\_id = Vendor.id  
 GROUP BY Vendor.id  
 ORDER BY COUNT(\*) DESC  
 LIMIT 10  
 """)  
 data = cursor.fetchall()  
 conn.close()  
  
 table = QTableWidget(len(data), 2)  
 table.setHorizontalHeaderLabels(["Поставщик", "Количество счетов"])  
 for row, (supplier, count) in enumerate(data):  
 table.setItem(row, 0, QTableWidgetItem(supplier))  
 table.setItem(row, 1, QTableWidgetItem(str(count)))  
 table.resizeColumnsToContents()  
 layout.addWidget(table)  
  
  
# Report 3: Статистика по категориям  
class Report3(QWidget):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 layout = QVBoxLayout(self)  
 conn = sqlite3.connect(parent.database\_file)  
 cursor = conn.cursor()  
 cursor.execute("""  
 SELECT Request\_category.name, COUNT(\*)   
 FROM Requests   
 LEFT JOIN Request\_category ON Requests.category\_id = Request\_category.id  
 GROUP BY Requests.category\_id""")  
 data = cursor.fetchall()  
 conn.close()  
  
 categories, counts = zip(\*data) if data else ([], [])  
 fig = Figure(figsize=(5, 3))  
 ax = fig.add\_subplot(111)  
 ax.pie(counts, labels=categories, autopct="%1.1f%%")  
 ax.set\_title("Статистика по категориям")  
  
 canvas = FigureCanvas(fig)  
 layout.addWidget(canvas)  
  
  
# Report 6: Рейтинг сотрудников  
class Report6(QWidget):  
 def \_\_init\_\_(self, parent):  
 super().\_\_init\_\_()  
 layout = QVBoxLayout(self)  
 conn = sqlite3.connect(parent.database\_file)  
 cursor = conn.cursor()  
 cursor.execute("""  
 SELECT Users.login, COUNT(\*)   
 FROM Requests   
 LEFT JOIN Users ON Requests.initiator\_id = Users.id  
 GROUP BY Users.id   
 ORDER BY COUNT(\*) DESC  
 """)  
 data = cursor.fetchall()  
 conn.close()  
  
 table = QTableWidget(len(data), 2)  
 table.setHorizontalHeaderLabels(["Сотрудник", "Количество заявок"])  
 for row, (employee, count) in enumerate(data):  
 table.setItem(row, 0, QTableWidgetItem(employee))  
 table.setItem(row, 1, QTableWidgetItem(str(count)))  
 layout.addWidget(table)