У вас в этот раз гита не будет, так что начинаем с 4 пункта

1. Создаем репозиторий

2. Клонируем его к себе. В случае этого репозитория будет команда командой git clone https://github.com/rainovskyqt/demo-20240413\_repairshop.git

3. На его основе репозиторя создаем новый проект

4. Через pip устанавливаем PySide6

pip install PySide6

5. Создаем файл main.py. Для того что бы проверить, что все корректно установилось, заполним файл базовыми командами

from PySide6.QtWidgets import QApplication, QMainWindow  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication()  
 main\_window = QMainWindow()  
 main\_window.show()  
 app.exec()

Запустим скрипт, должно появиться пустое окно

Закоммитим изменения.

Если не стоит автоматическое добавление файлов к коммиту, то пропишем

git add main.py .

И сделаем комммит

git commit -a -m "Created empty main.py file"

Отправим на удаленный сервер

git push origin main

5. Создаем модуль database (правой кнопкой на папке src→new→Python Package)

6. В нем файл db\_namager.py

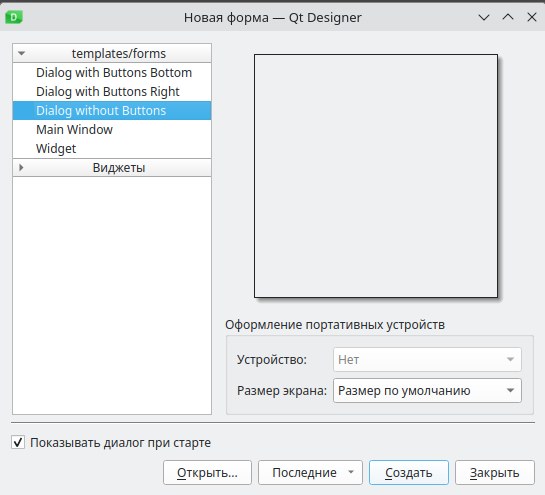
from PySide6.QtSql import QSqlDatabase, QSqlQuery  
import os  
  
  
class DBManager: # Создаем класс DBManager  
 def \_\_init\_\_(self, db\_path: str) -> None: # При создании он будет принимать путь к базе  
 self.db\_path = db\_path  
 print(db\_path, self.db\_path)  
  
 def check\_base(self): # Функция проверки наличия файла базы данных  
 exist = os.path.exists(self.db\_path)  
 print(f'Database exists = {exist}') # Контроль существования  
 return exist  
  
 def connect\_to\_base(self): # Функция подключения к базе данных  
 con = QSqlDatabase.addDatabase("QSQLITE")  
 con.setDatabaseName(self.db\_path)  
 print(con.databaseName()) # Вывод в консоль для проверки подключения  
 print(con.connectionName())  
 print(f'Database is open = {con.open()}') # Открываем базу данных  
 if con.lastError().isValid(): # Если есть ошибки то выводим их в консоль  
 print(f"Database error: {con.lastError().text()}")  
  
 @staticmethod  
 def execute\_file(file\_path: str):  
 query = QSqlQuery() # создаем новый запрос  
 with open(file\_path) as file: # открываем файл  
 rows = file.read().split(";") # Читаем файл и разбиваем его на строки по знаку ;  
 for row in rows:  
 query.exec(row) # Выполняем запросы  
  
 def create\_base(self, script\_tables\_path: str): # создание БД  
 self.execute\_file(script\_tables\_path)  
  
 def fill\_init\_data(self, script\_data\_path: str): # Заполение первоначальныи данными  
 self.execute\_file(script\_data\_path)

7. Создаем файл settings.ini

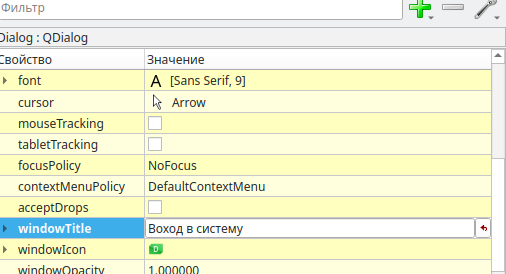
DB\_PATH = 'data.sqlite'' # Путь к файлу дфнных

8. В консоли запускаем pyside6-designer

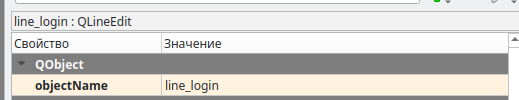
9. Выбираем диалог

10. Создаем окно авторизации

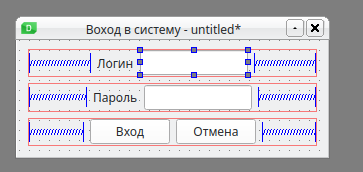
Задаем заголовок окна



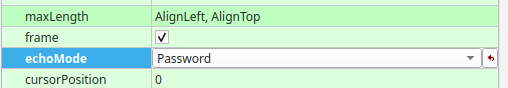
Не забываем давать осознаные именна виджетам



Форма выглядит примерно так

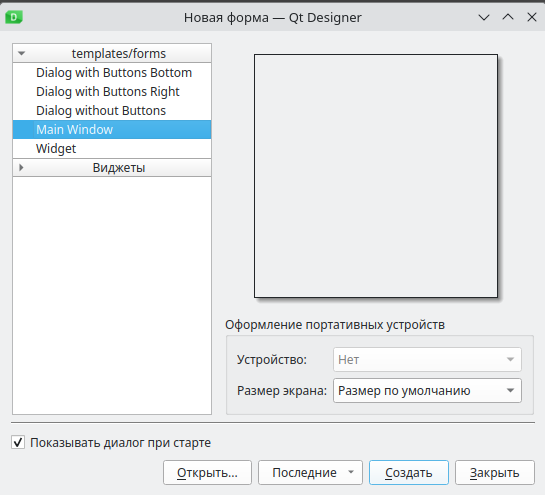


Для поля пароля обязательно установить echoMode = Password



9. Создаем каталог src/ui и сохраняем в него файл под именем loginform.ui

10. В designer создаем новый файл, но выбираем Main Window



Это будет главное окно

Пока оставляем его полностью пустым, созраняем в src/ui под именем mainwindow

Закрываем designer

Преобразуем главное окно и окно логина в .py файлы.

В консоли переходим в каталог src/ui и прописываем

pyside6-uic mainwindow.ui -o ui\_mainwindow.py

и

pyside6-uic loginform.ui -o ui\_loginform.py

В папке src/ui создаем файлы mainwindow.py и loginform.py

Для начала прописываем базовый шаблон в mainwindow.py

from PySide6.QtWidgets import QMainWindow  
from ui\_mainwindow import Ui\_MainWindow  
  
  
class MainWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super(MainWindow, self).\_\_init\_\_()  
 self.ui = Ui\_MainWindow()  
 self.ui.setupUi(self)

Изменяем main.py

Прописываем в нем наше окно аутентификации

from PySide6.QtWidgets import QApplication  
from ui.loginform import LoginForm  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication()  
 login\_form = LoginForm()  
 login\_form.show()  
 app.exec()

Запускаем, проверяем что работает

создаем src/database/models.py это файл с классами базы данных

добавляем user

class User:  
 def \_\_init\_\_(self, new\_id: int, name: str, post: int, post\_name: str = ""):  
 self.id = new\_id  
 self.name = name  
 self.post = post  
 self.post\_name = post\_name

Создаем query\_manager.py с классом QueryManager, этот класс будет хранить все запросы, патерн «Фасад». Добавляем функцию проверки логина

from PySide6.QtSql import QSqlQuery  
from db\_namager import DBManager  
  
  
class QueryManager:  
 def \_\_init\_\_(self, database: DBManager):  
 self.base = database  
  
 @staticmethod  
 def execute\_query(query\_row: str, params: tuple): # функция выполнения запроса  
 def execute\_query(query\_row: str, params: dict): # функция выполнения запроса  
 query = QSqlQuery()  
 query.prepare(query\_row) # Подготовка запроса из параметра query\_row  
  
 for key, value in params.items(): # Подстановка значений для запроса  
 query.bindValue(key, value)  
  
 query.exec() # Выполнение запроса  
  
 if query.lastError().isValid(): # В случае ошибки выводим сообщение  
 print(f'Query error: {query.lastError().text()}')  
  
 return query  
  
def check\_login(self, login: str, password: str):  
 query\_string = "SELECT id, name FROM users WHERE login = :login AND password = :password"  
 params = {":login": login, ":password": password}  
  
 answer = self.execute\_query(query\_string, params)  
 if answer.next():  
 user = User(answer.value("id"), answer.value("name"), answer.value("post"))  
 return user  
 else:  
 return 0

В loginform добавляем проверку логина главное окно

from PySide6.QtWidgets import QDialog, QMessageBox  
from ui.ui\_loginform import Ui\_dialog  
from database.db\_namager import DBManager  
from settings import DB\_PATH, TABLE\_SCRIPT\_PATH, DATA\_SCRIPT\_PATH  
from database.query\_manager import QueryManager  
from database.models import User  
from mainwindow import MainWindow  
  
  
class LoginForm(QDialog):

main\_window: MainWindow

def \_\_init\_\_(self):  
 super(LoginForm, self).\_\_init\_\_()  
 self.ui = Ui\_dialog()  
 self.ui.setupUi(self)  
 self.database = self.init\_base()  
 self.query = QueryManager(self.database)  
 self.init\_connections()  
  
 def init\_connections(self):  
 self.ui.btn\_cancel.clicked.connect(self.exit)  
 self.ui.btn\_ok.clicked.connect(self.login)  
  
 @staticmethod  
 def init\_base():  
 database = DBManager(DB\_PATH)  
 base\_exists = database.check\_base()  
 database.connect\_to\_base()  
 if not base\_exists:  
 database.create\_base(TABLE\_SCRIPT\_PATH)  
 database.fill\_init\_data(DATA\_SCRIPT\_PATH)  
 return database  
  
 def login(self):  
 user = self.query.check\_login(self.ui.line\_login.text(), self.ui.line\_password.text())  
 if not user:  
 self.login\_incorrect()  
 return  
 else:  
 self.start\_main\_window(user)  
  
 def login\_incorrect(self):  
 QMessageBox.critical(self, "Неудача", "Неверный логин или пароль")  
  
 def start\_main\_window(self, user: User):  
 self.main\_window = MainWindow(user)  
 self.main\_window.show()  
 self.close()  
  
 def exit(self):  
 self.close()

Добавляем функцию логина

После этого можно проверить, что если ввести значения логина и пароля admin и admin то откроется главня форма, иначе будет сообщение об ошибке

на этом окно аутинфикации завершено

открываем mainwindow.ui в desinger (в консоли пишем pyside6-designer)

опять конветрируем в .py файл

накидываем виджеты, не забываем их логично называть

Для надачала добавим фиджет отображения заявок(QTableView), кнопку добаления, редактирования и удаления заявки

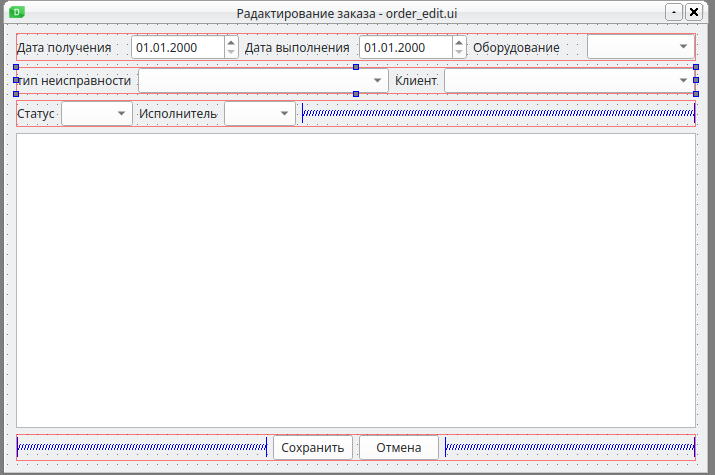
прписываем модели и их связи в файле mainwindow.py

rom PySide6.QtCore import QSortFilterProxyModel  
from PySide6.QtCore import Qt  
from PySide6.QtWidgets import QMainWindow  
from PySide6.QtSql import QSqlDatabase, QSqlRelationalTableModel, QSqlRelation  
from ui.ui\_mainwindow import Ui\_MainWindow  
from database.models import User  
  
  
class MainWindow(QMainWindow):  
 def \_\_init\_\_(self, user: User, login\_form):  
 super(MainWindow, self).\_\_init\_\_()  
 self.ui = Ui\_MainWindow()  
 self.ui.setupUi(self)  
 self.user = user  
 self.login\_form = login\_form  
 '''  
 При создании базы данных она создается одна на всю программу, но их может быть несколько, таким образом   
 мы получаем ссылку на глобальную базу, если в скобках указать имя подключения, то ссылка создастся на ту базу,  
 сейчас нам это не нужно, но что бы понимать для чего тут self.database = QSqlDatabase.database()   
 '''  
 self.database = QSqlDatabase.database()  
 self.sql\_model = QSqlRelationalTableModel(self, self.database)  
 self.proxy\_model = QSortFilterProxyModel(self)  
 self.proxy\_model.setSourceModel(self.sql\_model)  
 self.ui.tv\_orders.setModel(self.proxy\_model)  
 self.init\_connections()  
 self.set\_table()  
 self.select\_orders()  
  
 def exit(self):  
 self.login\_form.show()  
 self.close()  
  
 def set\_table(self): # устанавливаем связи с таблицами  
 # Устанавливаем таблицу из которой наша модель будет получать данные  
 self.sql\_model.setTable("orders")  
 # Устанавливаем связи с таблицами на которые ссылаются внешние ключи  
 # self.sql\_model.setRelation(номер колонки внешнего ключа, QSqlRelation("название внешней таблицы", "поле ключа", "поле котрое поставитьь из внешней таблицы"))  
 self.sql\_model.setRelation(3, QSqlRelation("equipment", "id", "name"))  
 self.sql\_model.setRelation(4, QSqlRelation("fault", "id", "name"))  
 self.sql\_model.setRelation(6, QSqlRelation("clients", "id", "name"))  
 self.sql\_model.setRelation(7, QSqlRelation("status", "id", "name"))  
 self.sql\_model.setRelation(8, QSqlRelation("users", "id", "name"))  
  
 def select\_orders(self):  
 self.sql\_model.select() # Вызов выборки из базы  
 # Установка заголовков таблицы показывающей данные  
 self.sql\_model.setHeaderData(0, Qt.Orientation.Horizontal, "№ заявки")  
 self.sql\_model.setHeaderData(1, Qt.Orientation.Horizontal, "Дата получения")  
 self.sql\_model.setHeaderData(2, Qt.Orientation.Horizontal, "Дака выполнения")  
 self.sql\_model.setHeaderData(3, Qt.Orientation.Horizontal, "Оборудование")  
 self.sql\_model.setHeaderData(4, Qt.Orientation.Horizontal, "Тип неисправности")  
 self.sql\_model.setHeaderData(5, Qt.Orientation.Horizontal, "Описание")  
 self.sql\_model.setHeaderData(6, Qt.Orientation.Horizontal, "Клиент")  
 self.sql\_model.setHeaderData(7, Qt.Orientation.Horizontal, "Статус")  
 self.sql\_model.setHeaderData(8, Qt.Orientation.Horizontal, "Исполнитель")  
 self.ui.tv\_orders.resizeColumnsToContents() # Установка столбцов по ширине данных  
  
 def init\_connections(self):  
 self.ui.btn\_exit.clicked.connect(self.exit)

На этом этапе у нас будут выводиться данные в таблицу

Сделаем добавление данных

В desinger создаем форму order\_edit и конвертируем его в .py файл



добавим удаление заказа, в v mainwindow

def delete\_record(self):  
 rec\_id = self.get\_current\_record\_id()  
 if not rec\_id:  
 return  
  
 answer = QMessageBox.question(self, "Удаление записи", f"Удалить заказ {rec\_id}?")  
 if answer == QMessageBox.StandardButton.No:  
 return  
  
 self.query\_manager.delete\_order(rec\_id)  
 self.select\_orders()

в функции init\_connection у класса mainwindow

self.ui.btn\_delete.clicked.connect(self.delete\_record)

у нас есть удаление записи

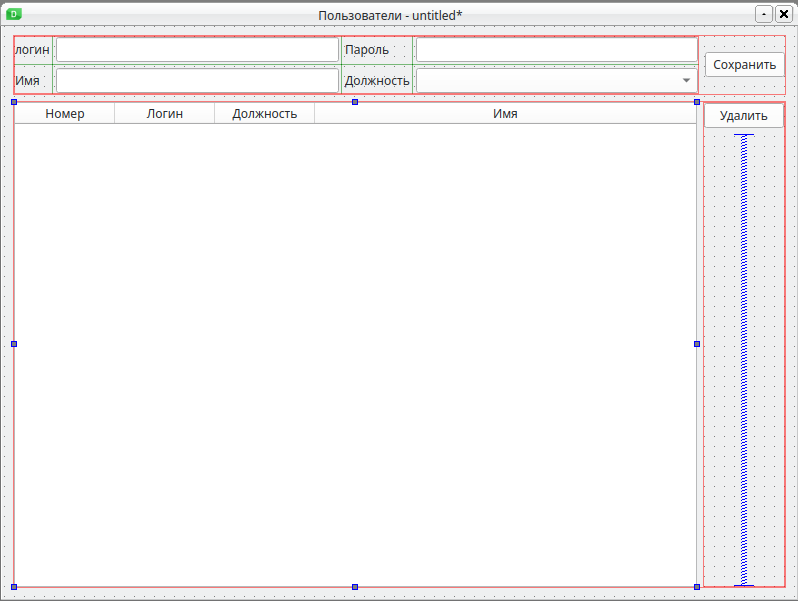
в database/models.py создаем класс

class Order:  
 def \_\_init\_\_(self, add\_date, resolve\_date, equipment\_id, fault\_id,  
 description, client\_id, status\_id, worker\_id, base\_id = 0):  
 self.add\_date = add\_date  
 self.resolve\_date = resolve\_date  
 self.equipment\_id = equipment\_id  
 self.fault\_id = fault\_id  
 self.description = description  
 self.client\_id = client\_id  
 self.status\_id = status\_id  
 self.worker\_id = worker\_id  
 self.base\_id = base\_id

Если осталось время

Реализуем функции добавления, редактировани и удалниея пользователей, т. е. Сокращенно CRUD

Откроем desinger и создадим окно пользователей, выберем опять Dialog whithout buttons



в QueryManager добавляем выборку пользователей