Курсовая работа

По дисциплине: «Визуальное программирование и человеко-машинное взаимодействие»

Вариант: 2

«Учет ремонтной мастерской»

Выполнил:

Cтудент 2 курса группы ИП-111  
Попов М.И.

Проверил:

Доцент кафедры ПМиК,

Мерзлякова Е.Ю

Кафедра прикладной математики и кибернетики

Сибирский государственный университет  
телекоммуникаций и информатики

Новосибирск, 2023

Министерство цифрового развития, связи  
и массовых коммуникаций Российской Федерации

Оглавление

1. [ВВЕДЕНИЕ 3](#_Toc154320383)
2. [Анализ задач и пользователей 4](#_Toc154320384)
3. [Выбор репрезентативных задач 5](#_Toc154320385)
4. [Заимствование 6](#_Toc154320386)
5. [Прототип интерфейса 7](#_Toc154320387)
6. [Реализация 9](#_Toc154320388)
7. [CWT Анализ 12](#_Toc154320389)
8. [GOMS Анализ 13](#_Toc154320390)
9. [Список источников 15](#_Toc154320391)

10. [Приложение 16](#_Toc154320392)

# **ВВЕДЕНИЕ**

В современном мире, где технологический прогресс неустанно взлетает ввысь, эффективное управление и автоматизация бизнес-процессов становятся ключевыми факторами успеха предприятий. Одной из важных составляющих данного аспекта является создание и использование программного обеспечения, специально адаптированного под нужды конкретной предметной области. В данном контексте, ремонтные мастерские выступают важным звеном в индустрии обслуживания техники, требуя надежных и современных инструментов для эффективного учета и управления своей деятельностью.

Цель настоящей курсовой работы заключается в разработке программы учета для ремонтной мастерской, с использованием популярного фреймворка Qt. Подобный программный продукт не только облегчит повседневные операции в мастерской, но и оптимизирует процессы учета, анализа и планирования, повышая общую эффективность деятельности предприятия.

В данном введении мы проведем обзор предметной области, рассмотрим основные аспекты управления и учета в ремонтных мастерских, а также выявим ключевые проблемы, которые могут быть решены с помощью создания специализированной программы. Помимо этого, мы также рассмотрим фреймворк Qt как инструмент для разработки приложений с графическим интерфейсом, обосновывая выбор данной технологии в контексте задачи создания программы учета для ремонтной мастерской.

Популярные приложения для учета задач для бизнеса с задачами ремонта существует не много, обычно для использует собственные разработки или учет используя excel таблицу или базы 1С, но можно отметить популярные зарубежные решения: HelloClient.ru - Одна из самых востребованных и доступных по цене программ для управления сервисным центром и ремонтной мастерской на сегодня. Мощная и при этом простая облачная платформа предназначена для того, чтобы упростить жизнь владельцу бизнеса и его сотрудникам; Ремонлайн - другая популярная программа для сервисных центров доступная на сегодня и предоставляющая пользователям большинство функций, которые можно ожидать от стандартной программы для ремонтных мастерских.

# **Анализ задач и пользователей**

Для варианта задания «Учет ремонтной мастерской» при рассмотрении заинтересованных лиц были выбраны люди, которые имеют опыт в управлении и командной разработке. Для рассмотрения задачи были выбраны мой друг который занимается ремонтом и изготовлений деталей из пластика и метала, и я который занимается ремонтом бытовой техники и изготовлений различной мебельных деталей.

1. Друг: возраст – 21 год, образование – среднее общее, студент НГТУ, навыки в выбранной сфере – опыт работы более 2 лет, навыки работы на компьютере – выше среднего, высокая степень владения токарным делом.
2. Я: возраст – 22 года, образование – среднее профессиональное, студент СибГУТИ, навыки в выбранной сфере – опыт работы около 1 гоад, навыки работы на компьютере – высокие, умеренный опыт в паяльном деле, мелкие навыки деревообработки.

Основные термины: заказ, работа, сроки, заявки.

Возможности существующих программ по схожей тематике: Учет задач, контроль выполнения, контроль сроков выполнения, учет клиента.

# **Выбор репрезентативных задач**

1. Создать интерфейс заполнения задачи.
2. Создать интерфейс выбор мастера на работу.
3. Добавление сроков выполнения задачи.
4. Контроль выполнения срока задачи.
5. Создание поля для дополнительной информации.
6. Создать поля для имени заказчика и его контактной информации
7. Добавление фильтров для поиска задач

# **Заимствование**

HelloClient.ru:

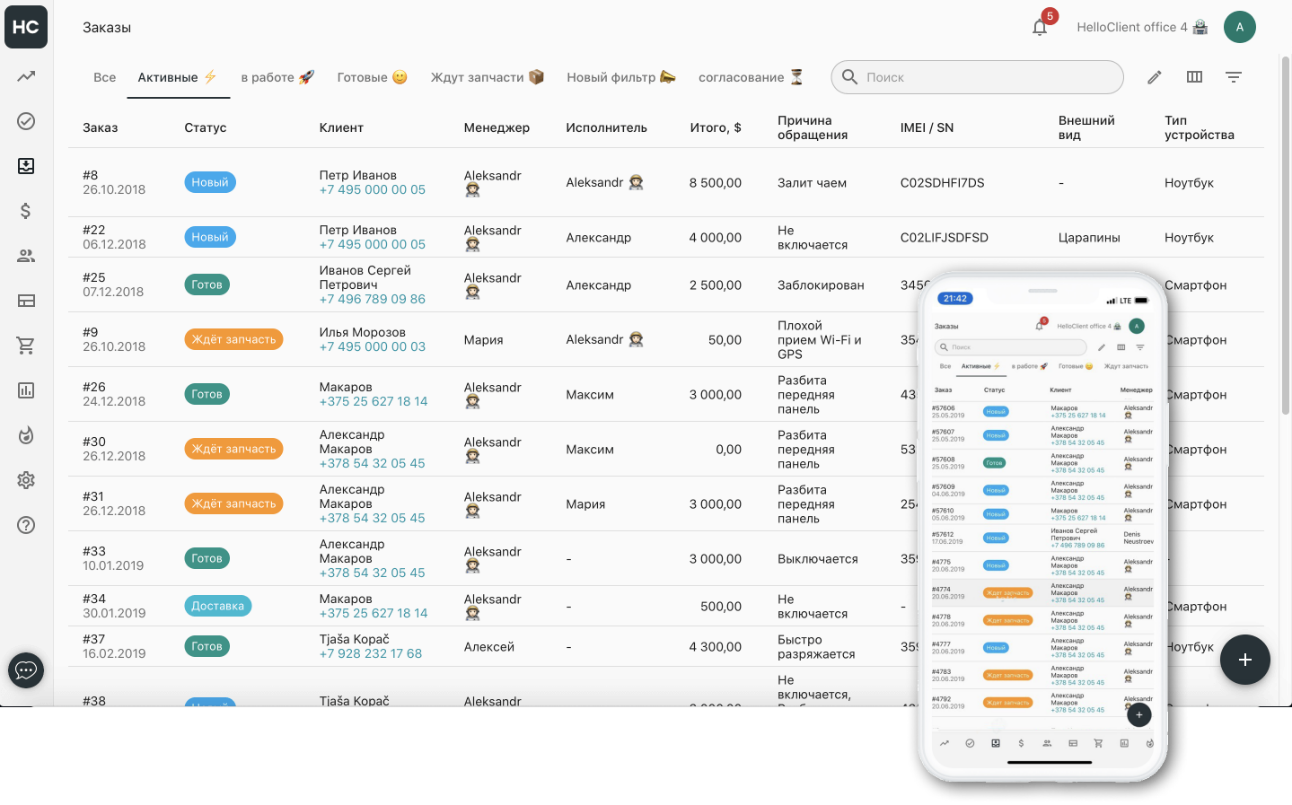


Рисунок 2. – главное окно приложения HelloClient.ru

Тут можно по заимствовать табличную верстку задач и типы отображения полей

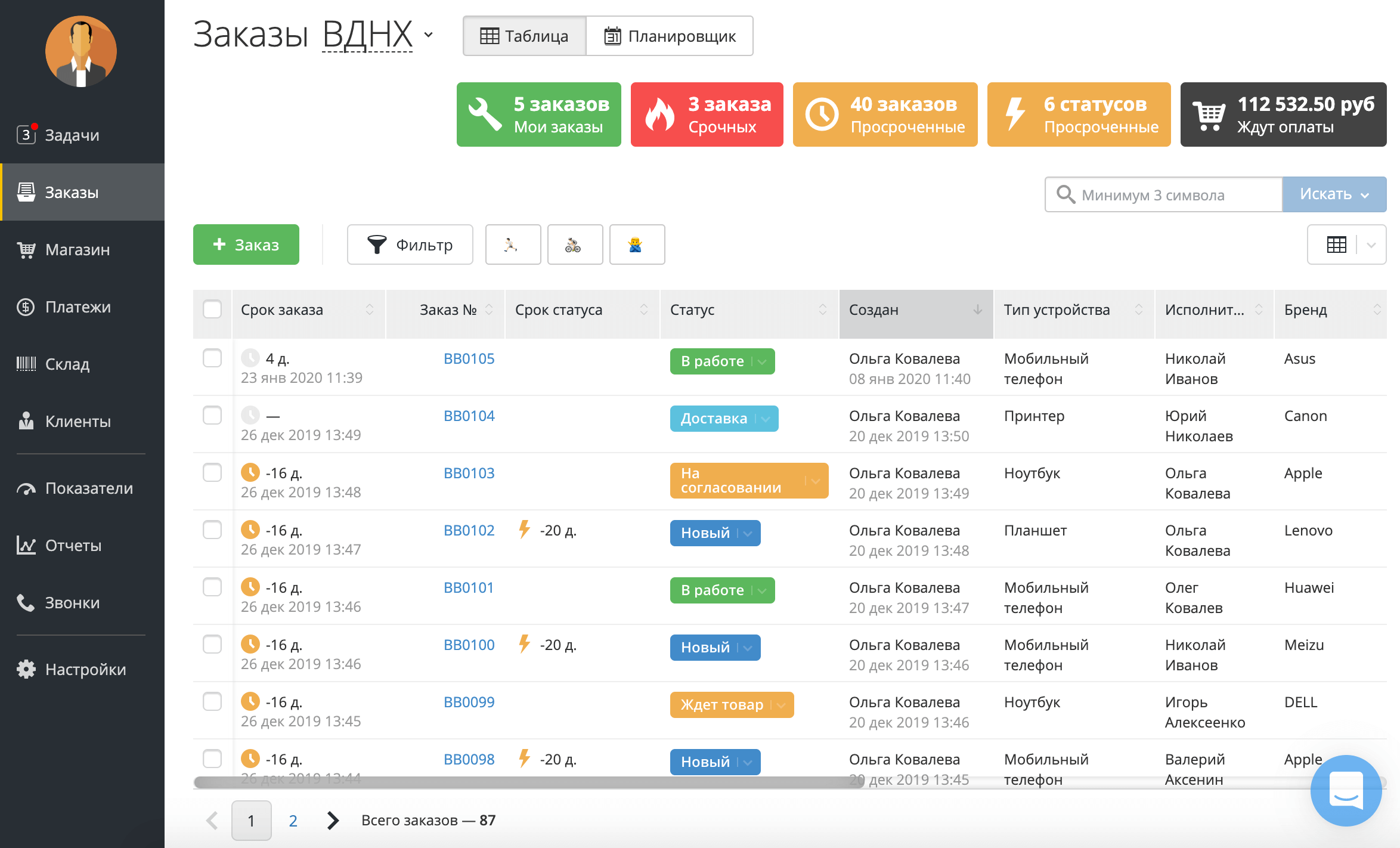
Ремонтлайн:  


Рисунок 2. – главное окно приложения Ремонтлайн

Отсюда можно взять примерное положение полей для общего вывода задач и счетчики для текущих задач.

# **Прототип интерфейса**

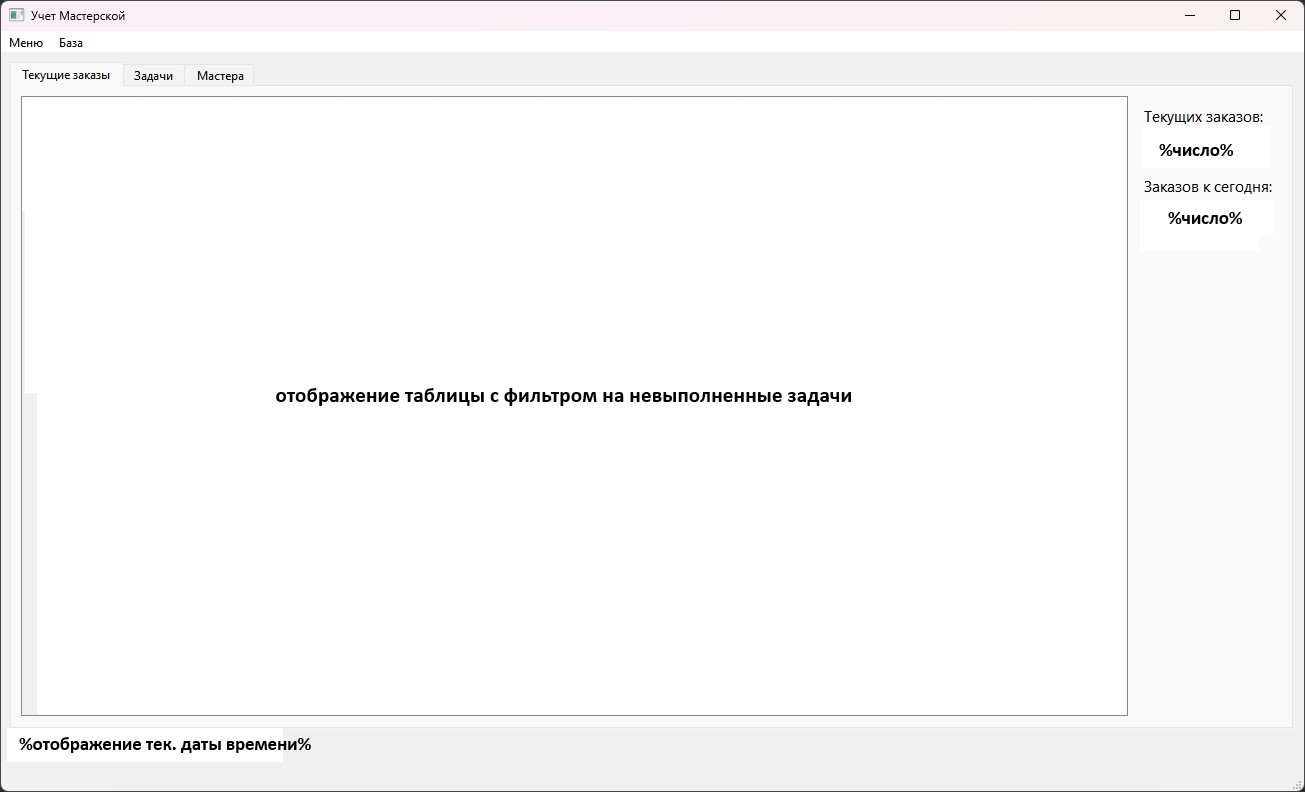


Рисунок 3. – Главное окно с открытой вкладкой текущих задач

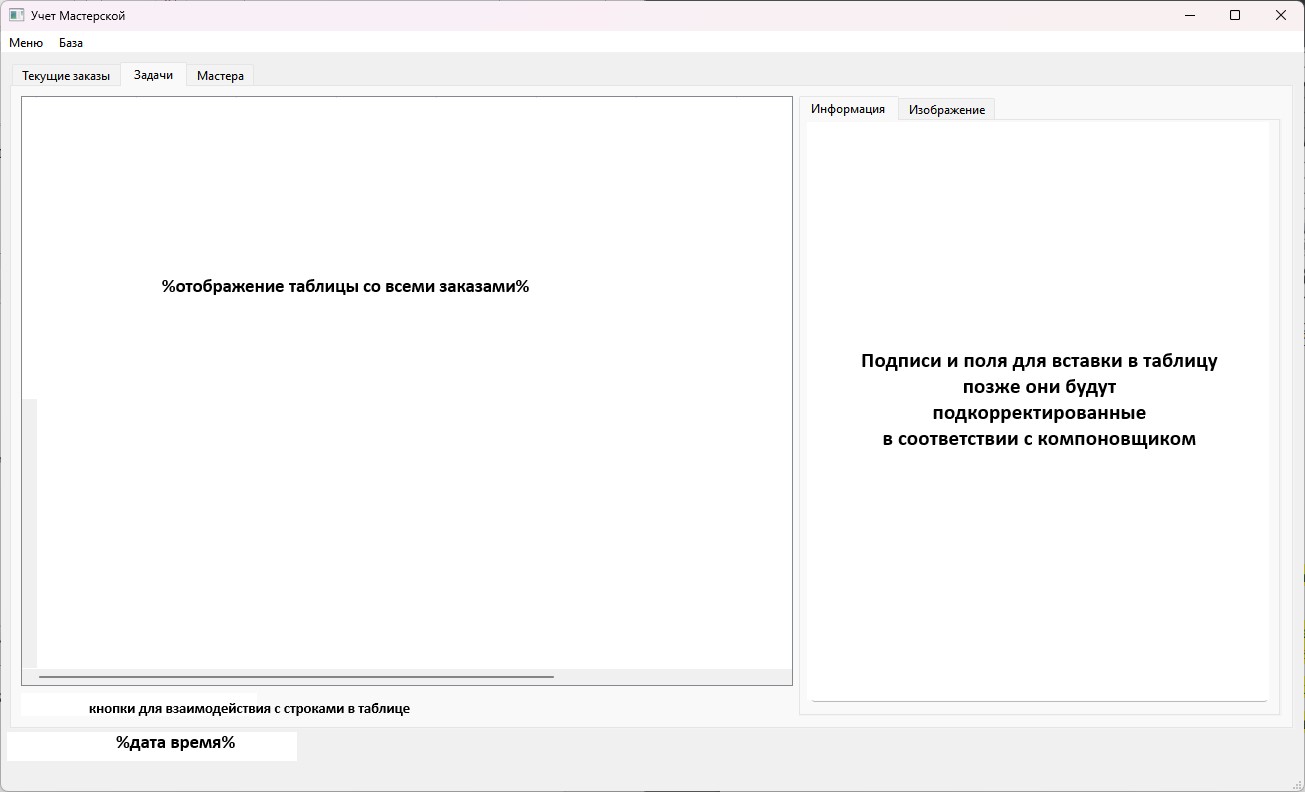


Рисунок 4. – Главное окно с открытой вкладкой всех задач, редактирование полей

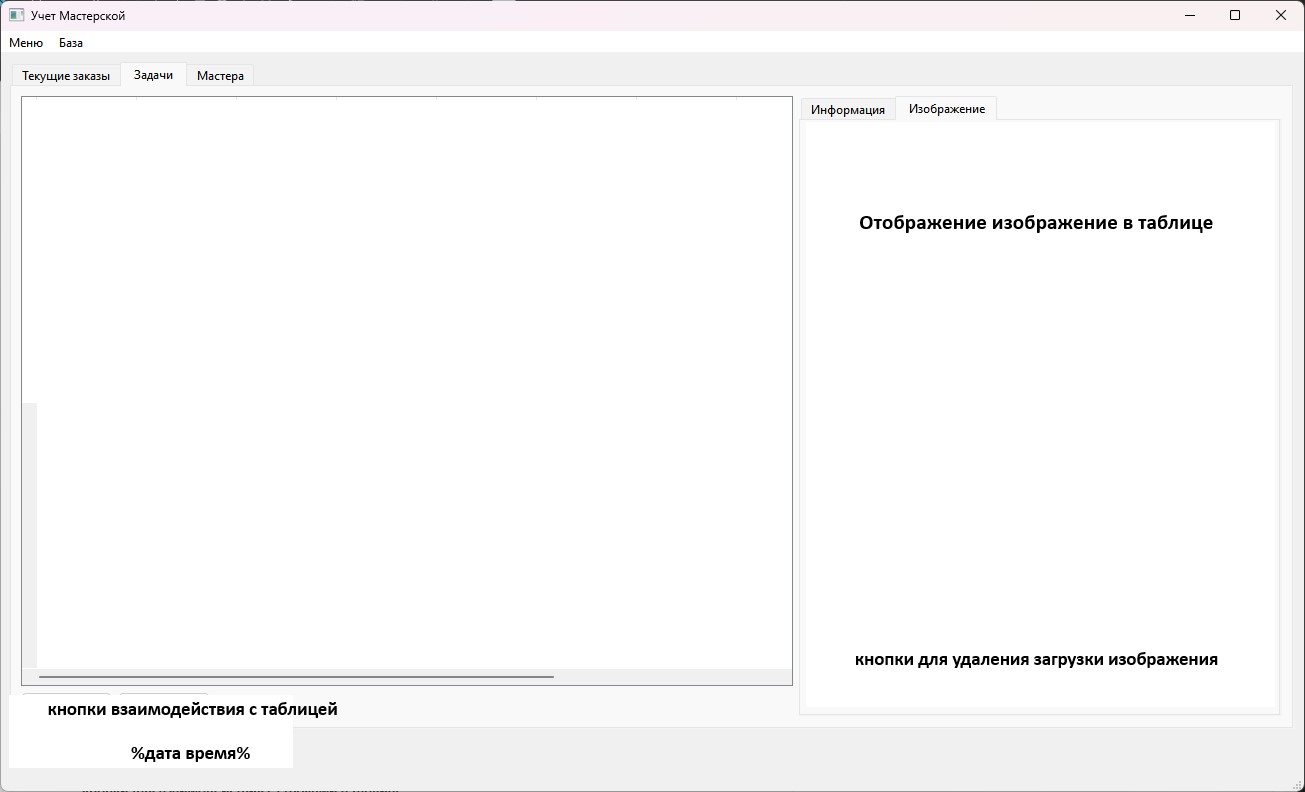


Рисунок 5. – Главное окно с открытой вкладкой всех задач, изменение изображения

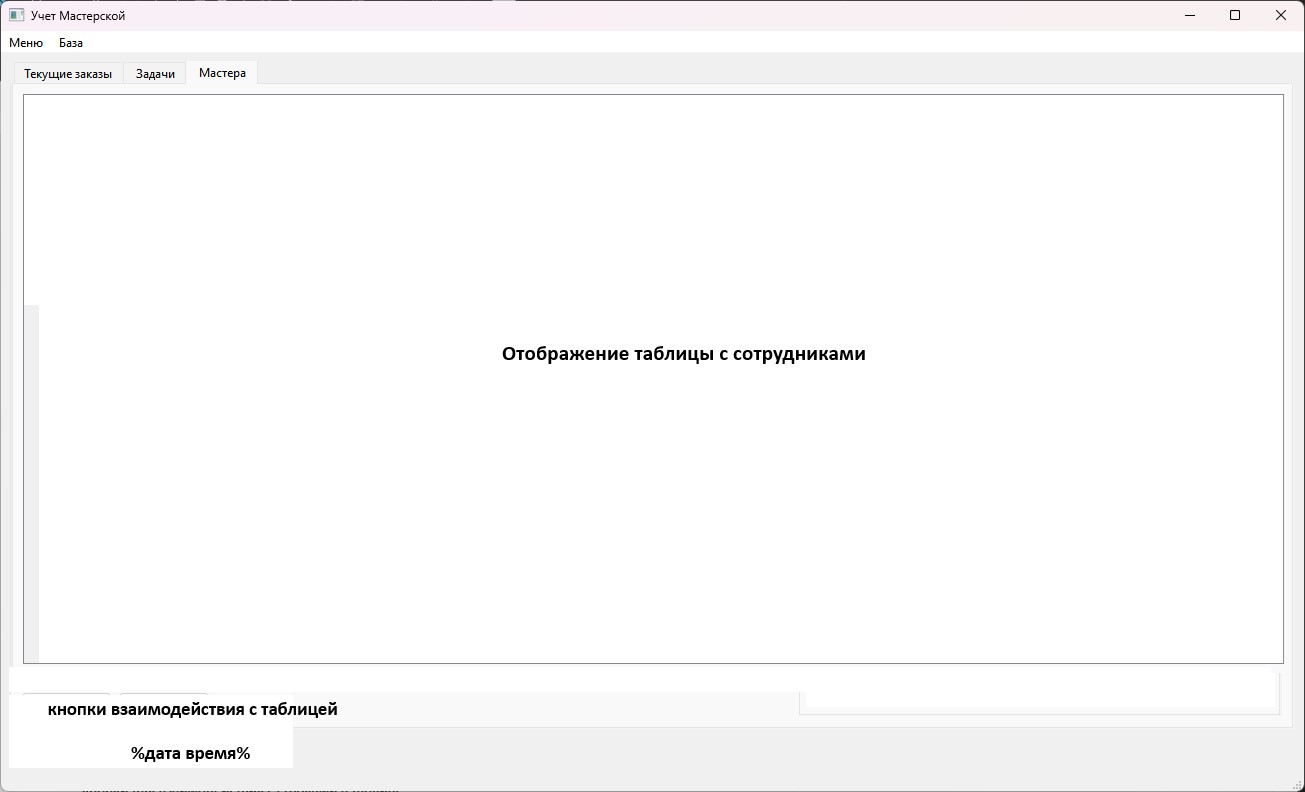


Рисунок 6. – Главное окно с открытой вкладкой сотрудников

Описание элементов интерфейса:

Главное окно будет содержать два лейбла в которых отображается тек. дата и время, и окно с вкладками:

* «Текущие задачи»
  + Содержит отображение таблицы по фильтру активных задач (рис.3)
  + Содержит в панели слева счетчики активных задач и задач к сдаче на сегодня
* «Задачи» Рису
  + Содержит отображение всех задач в таблице
  + Есть кнопки для добавления, удаления строк
  + Справа в панели две вкладки для редактирование полей и изображение в записи (рис. 4 и 5)
  + Также будет Action опция для вызова фильтра
* «Мастера»
  + Содержит таблицу мастеров (рис. 6)
  + Имеются кнопки для удаления и добавления новой строки

# **Реализация**

Приложения написано на Qt 6.6.0, содержит функционал открытие базы данных SQLLite, проверки её наличия, вывод текущей даты/времени, редактирование таблиц через виджет QTableView задач с указание полей и изображения, иметься проверка на правильность введения даты, проверка на опережение даты заявки, фильтрация полей через вызов action опции редактирование таблицы мастеров, кнопки добавления и удаления, счетчики текущих задач и задач на сегодня, выбор мастеров в comboBox и добавление их из таблицы.

Приложение состоит из окон: MainWindow.

Классы: MainWindow.

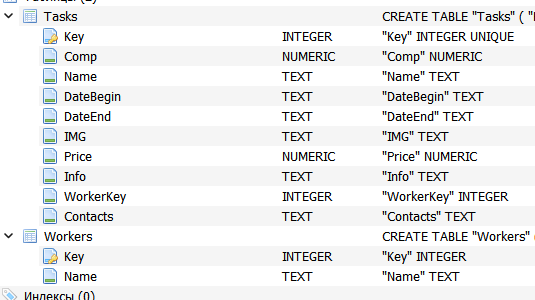
База данных представляет собой две таблицы Tasks и Workers c полями:  


Рисунок 7. – Поля в таблицах используемой в базе

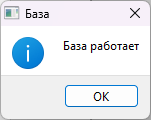


Рисунок 8. – Окно успешной загрузки базы данных.

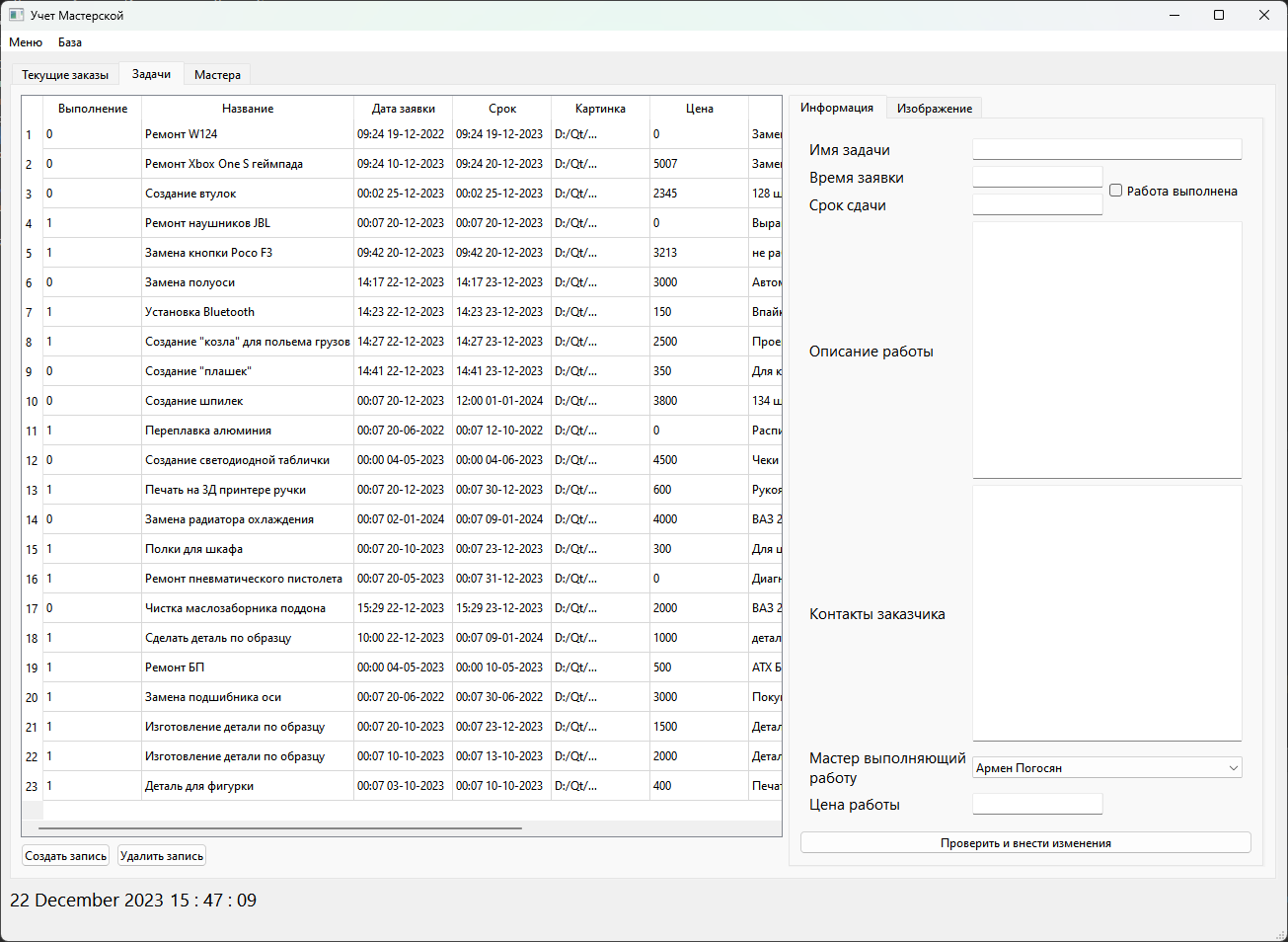


Рисунок 9. – Главное окно программы, вкладка «Задачи».

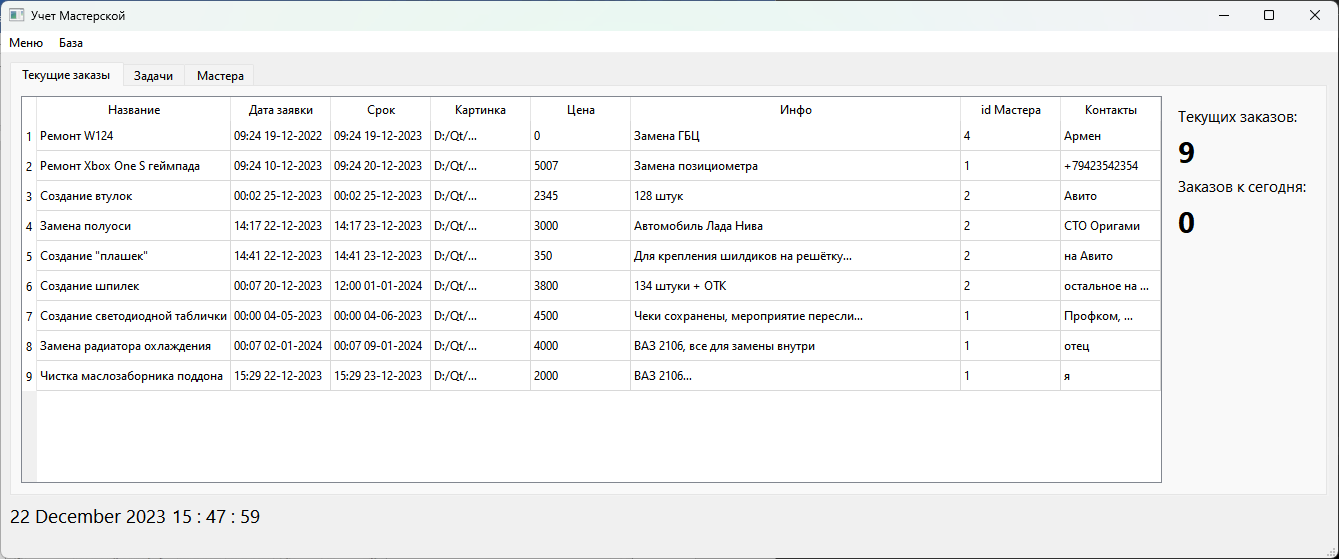


Рисунок 10. – Главное окно программы, вкладка «Текущие заказы».

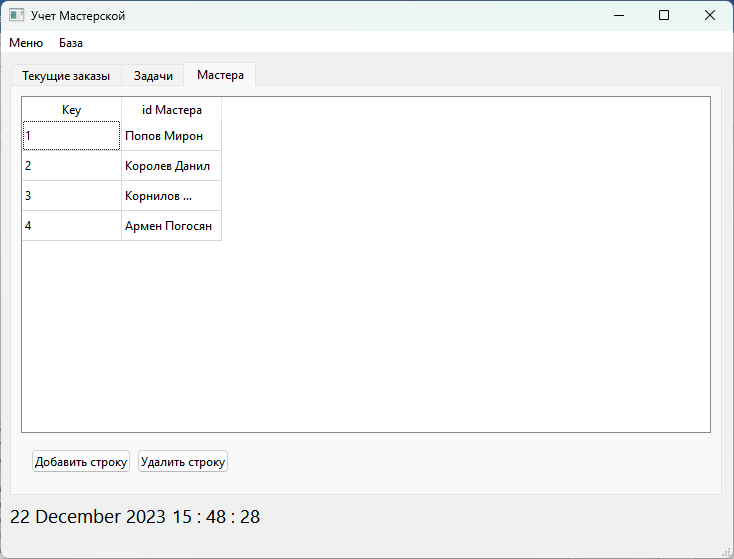


Рисунок 10. – Главное окно программы, вкладка «Мастера».

# **CWT Анализ**

**Удаление записи в базе данных:**

1. Запуск приложения

Человек запускает программу и удостоверяется что база данных успешно загружена

1. Открытие главного окна и вкладки «Задачи»

В главном окне необходимо перейти во вкладку «Задачи», нажав на её вкладку сверху

1. Выделение нужной строки для удаления

Пользователь выделяет нужную строку

1. Нажатие кнопки удаления

Внизу отображения таблицы есть кнопка для удаления текущей строки

**Фильтрация данных в таблице «Задачи»**

1. Запуск приложения

Человек запускает программу и удостоверяется что база данных успешно загружена

1. Открытие главного окна и вкладки «Задачи»

В главном окне необходимо перейти во вкладку «Задачи», нажав на её вкладку сверху

1. Поиск кнопки фильтра

Он находиться в action опции во вкладке базы

1. Открытие фильтра и выбор нужных полей

Необходимо указать необходимые пользователю поля для фильтрации

1. Нажатие на кнопку применить

Для отображения таблицы по заданному фильтру

# **GOMS Анализ**

Практически все интерфейсные взаимодействия в можно описать следующими операциями:

K – нажатие клавиши;

B – клик кнопкой мыши;

P – наведение указателя мыши;

R – ожидание ответной реакции компьютера;

H – перенос руки с клавиатуры на мышь или наоборот

D – проведение с помощью мыши прямой линии (например, выделение или прокрутка текста);

M – мыслительная подготовка (к осуществлению одной из перечисленных операций).

Разные пользователи выполняют указанные операции за разное время. Однако, GOMS исследует работу опытного пользователя. Многочисленные исследования выявили средние значения времени операций, выполняемых опытными пользователями.

K 0.2 с

B 0.2 с

P 1.1 с

H 0.4 с

M 1.35 с

**Открытие программы и редактирование записи**

1. Открытие программы и закрытия уведомления о подключенной базе

PBRMPB

1. Выбор таблицы «Задачи» и выбор строки для редактирования

PBMPB

1. Редактирование поля цена

PBH(4K)

1. Сохранение введённых данных

HPB

Итого: 6P+6B+1R+2M+2H+4K = 11,6 сек

**Открытие программы и удаление записи**

1. Открытие программы и закрытия уведомления о подключенной базе

PBRMPB

1. Выбор таблицы «Задачи» и выбор строки для редактирования

PBMPB

1. Нажатие на удаление записи

MB

Итого: 4P+5B+1R+3M = 9,75 сек

# **Список источников**

1. 5 лучших программ для сервисных центров и ремонтных мастерских в 2020 [Электронный ресурс] Программа для автоматизации бизнеса услуг и продаж. – URL: <https://helloclient.ru/5-best-crm-for-service-center> (дата обращения 22.12.23)
2. Qt Documentation [Электронный ресурс] // doc.qt.io - Qt Documentation – URL: <https://doc.qt.io/> (дата обращения 22.12.23)
3. LIKE (Transact-SQL) [Электронный ресурс] // learn.microsoft.com – Microsoft Learn: приобретение навыков, которые открывают путь к карьерному росту – URL: <https://learn.microsoft.com/ru-ru/sql/t-sql/language-elements/like-transact-sql?view=sql-server-ver16> (дата обращения 22.12.23)
4. Описание класса QSqlTableModel [Электронный ресурс] // doc.crossplatform.ru - Все о кроссплатформенном программировании | Документация – URL: <http://doc.crossplatform.ru/qt/4.3.2/qsqltablemodel.html> (дата обращения 22.12.23)

# **Приложение**

Mainwindow.h

|  |
| --- |
| #ifndef MAINWINDOW\_H #define MAINWINDOW\_H  #include <QMainWindow> #include <QFileInfo> #include <QMessageBox> #include <QDebug> #include <QTime> #include <QTimer> #include <QString> #include <QListWidget> #include <QFile> #include <QPixmap> #include <QGraphicsView> #include <QFileDialog>  #include <QtSql/QSql> #include <QtSql/QSqlQuery> #include <QtSql/QSqlError> #include <QtSql/QSqlDatabase> #include <QtSql/QSqlTableModel> #include <QtSql/QSqlRecord> #include <QtSql/QSqlIndex> #include <QtSql/QSqlField> #include <QtSql/QSqlRelation>  QT\_BEGIN\_NAMESPACE namespace Ui {class MainWindow;} QT\_END\_NAMESPACE class MainWindow : public QMainWindow{  Q\_OBJECT public:  MainWindow(QWidget \*parent = nullptr);  ~MainWindow();  Ui::MainWindow \*ui;  QSqlDatabase sqllite3 = QSqlDatabase::addDatabase("QSQLITE");  QSqlQuery query = QSqlQuery(sqllite3);  QSqlTableModel\* modelTasks;  QSqlTableModel \*modelWorkers;  QString filename;  TaskEdit task;  void loadTasks();  void loadWorkers();  void UpdateWorkers();  void loadTasks\_active();  QString GetWorker(int i);  QString GetWorker(QString string);  QSqlTableModel\* getModelWorkers(); private slots:  void on\_pushButton\_clicked();  void on\_pushButton\_2\_clicked();  void showTime();  void SetToEdit();  void on\_pushButton\_3\_clicked();  void on\_action\_triggered();  void on\_action\_2\_triggered();  void on\_tableView\_clicked(const QModelIndex &index);  void on\_pushButton\_4\_clicked();  void on\_pushButton\_7\_clicked();  void on\_pushButton\_6\_clicked();  void on\_action\_3\_triggered();  void on\_pushButton\_5\_clicked();  void on\_action\_4\_triggered();  void on\_pushButton\_9\_clicked();  void on\_pushButton\_8\_clicked();  void on\_tabWidget\_tabBarClicked(int index); signals:  void sendModel(QSqlTableModel\*); }; #endif // MAINWINDOW\_H |

Mainwindow.cpp

|  |
| --- |
| #include "mainwindow.h" #include "ui\_mainwindow.h"  MainWindow::MainWindow(QWidget \*parent) : QMainWindow(parent), ui(new Ui::MainWindow){  ui->setupUi(this);  ui->dockWidget->setVisible(false);  //загрузка базы  sqllite3.setDatabaseName("D:\\Documents\\mydatabase.sqlite");  if (!sqllite3.open()) QMessageBox::information(this, "База", "База не подключилась");  else QMessageBox::information(this, "База", "База работает");  //загрузка таблиц  modelTasks = new QSqlTableModel(this, sqllite3);  modelTasks->setTable("Tasks");  modelTasks->setEditStrategy(QSqlTableModel::OnFieldChange);  modelWorkers = new QSqlTableModel(this, sqllite3);  modelWorkers->setTable("Workers");   QDate dateToday = QDate::currentDate();  ui->label->setText(dateToday.toString("dd MMMM yyyy"));  QTimer \*timer = new QTimer(this);  connect(timer, SIGNAL(timeout()),this,SLOT(showTime()));  // запуск таймера по дате время  timer->start();  on\_action\_triggered(); } //получение модели workers QSqlTableModel\* MainWindow::getModelWorkers(){  return modelWorkers; } //обновление сотрадников void MainWindow::UpdateWorkers(){  int i = modelWorkers->rowCount();  ui->comboBox->clear();  ui->comboBox\_3->clear();  while(i--){  ui->comboBox->addItem(modelWorkers->record(i).value("Name").toString());  ui->comboBox\_3->addItem(modelWorkers->record(i).value("Name").toString());  }  ui->comboBox->addItem("");  ui->comboBox\_3->addItem(""); } //ltcnhernjhd MainWindow::~MainWindow(){  delete ui; } void MainWindow::SetToEdit(){  int selectedRow = ui->tableView->currentIndex().row();  qDebug() << selectedRow;  //заполняем пустыми  if (selectedRow < 0){  ui->lineEdit->setText(" ");  ui->lineEdit\_2->setText(" ");  ui->lineEdit\_3->setText(" ");  ui->checkBox->setChecked(0);  ui->plainTextEdit->setPlainText(" ");  ui->plainTextEdit\_2->setPlainText(" ");  ui->lineEdit\_4->setText(" ");  }  //заполняем панель слева данными  else {  ui->lineEdit->setText(modelTasks->record(selectedRow).value("Name").toString());  ui->lineEdit\_2->setText(modelTasks->record(selectedRow).value("DateBegin").toString());  ui->lineEdit\_3->setText(modelTasks->record(selectedRow).value("DateEnd").toString());  if (modelTasks->record(selectedRow).value("Comp").toString()=="1") ui->checkBox->setChecked(true);  else ui->checkBox->setChecked(false);   ui->plainTextEdit->setPlainText(modelTasks->record(selectedRow).value("Сontacts").toString());  ui->plainTextEdit\_2->setPlainText(modelTasks->record(selectedRow).value("Info").toString());  ui->lineEdit\_4->setText(modelTasks->record(selectedRow).value("Price").toString());   if (modelTasks->record(selectedRow).value("WorkerKey").toInt()!=0)  ui->comboBox->setCurrentText(GetWorker(modelTasks->record(selectedRow).value("WorkerKey").toInt()-1));  else ui->comboBox->setCurrentText(" ");   QPixmap img = (modelTasks->record(selectedRow).value("IMG").toString());   ui->label\_11->setPixmap(img);  } } //получение имени сотрудника по ключу QString MainWindow::GetWorker(int i){  return modelWorkers->record(i).value("Name").toString(); } //получение id сотрудника по имени QString MainWindow::GetWorker(QString string){  modelWorkers->setFilter("Name ='"+string+"'");  modelWorkers->select();  QString i = modelWorkers->record(0).value("Key").toString();  return i; } //обновление табл. сотрудники void MainWindow::loadWorkers(){  modelWorkers->select();  ui->tableView\_2->setModel(modelWorkers); } //загрузка таблицы задачи void MainWindow::loadTasks(){  modelTasks->select();  ui->tableView->setModel(modelTasks); } //загрузка таблицы с активными задачами void MainWindow::loadTasks\_active(){  modelTasks = new QSqlTableModel(this, sqllite3);  modelTasks->setTable("Tasks");  modelTasks->setFilter(QString("DateEnd LIKE '%" + QDate::currentDate().toString("dd-MM-yyyy") + "%'"));  modelTasks->select();   int dsd = modelTasks->rowCount();  ui->label\_14->setText(QString::number(dsd));   modelTasks->setFilter("Comp= 0");  modelTasks->select();  int ids = modelTasks->rowCount();  ui->label\_10->setText(QString::number(ids));  ui->tableView\_3->setModel(modelTasks); } //показ времени void MainWindow::showTime(){  QTime time = QTime::currentTime();  ui->label\_2->setText(time.toString("hh : mm : ss")); } //вставка void MainWindow::on\_pushButton\_clicked(){  qDebug() << "Вставка строки" ;  modelTasks->insertRows(modelTasks->rowCount(),1); } //удаление void MainWindow::on\_pushButton\_3\_clicked(){  int selectedRow = ui->tableView->currentIndex().row();  if (selectedRow >= 0){  qDebug() << "удал. строку" << modelTasks->removeRows(selectedRow,1);  }  else qDebug() << "нет выделенной строки"; } //загрузка и обновление таблиц с прем. столбцов и удаление лишних void MainWindow::on\_action\_triggered(){  loadWorkers();  modelWorkers->setHeaderData(1, Qt::Horizontal, "id Мастера");  modelWorkers->setHeaderData(2, Qt::Horizontal, "ФИО");   modelTasks->setFilter("");  loadTasks();  modelTasks->setHeaderData(1, Qt::Horizontal, "Выполнение");  modelTasks->setHeaderData(2, Qt::Horizontal, "Название");  modelTasks->setHeaderData(3, Qt::Horizontal, "Дата заявки");  modelTasks->setHeaderData(4, Qt::Horizontal, "Срок");  modelTasks->setHeaderData(5, Qt::Horizontal, "Картинка");  modelTasks->setHeaderData(6, Qt::Horizontal, "Цена");  modelTasks->setHeaderData(7, Qt::Horizontal, "Инфо");  modelTasks->setHeaderData(8, Qt::Horizontal, "id Мастера");  modelTasks->setHeaderData(9, Qt::Horizontal, "Контакты");  ui->tableView->hideColumn(0);   ui->tableView->horizontalHeader()->setSectionResizeMode(2, QHeaderView::ResizeToContents);  ui->tableView->horizontalHeader()->setSectionResizeMode(7, QHeaderView::Stretch);  ui->tableView->horizontalHeader()->setSectionResizeMode(9, QHeaderView::ResizeToContents);   loadTasks\_active();  modelTasks->setHeaderData(1, Qt::Horizontal, "Выполнение");  modelTasks->setHeaderData(2, Qt::Horizontal, "Название");  modelTasks->setHeaderData(3, Qt::Horizontal, "Дата заявки");  modelTasks->setHeaderData(4, Qt::Horizontal, "Срок");  modelTasks->setHeaderData(5, Qt::Horizontal, "Картинка");  modelTasks->setHeaderData(6, Qt::Horizontal, "Цена");  modelTasks->setHeaderData(7, Qt::Horizontal, "Инфо");  modelTasks->setHeaderData(8, Qt::Horizontal, "id Мастера");  modelTasks->setHeaderData(9, Qt::Horizontal, "Контакты");  ui->tableView\_3->hideColumn(0);  ui->tableView\_3->hideColumn(1);  ui->tableView\_3->horizontalHeader()->setSectionResizeMode(2, QHeaderView::ResizeToContents);  ui->tableView\_3->horizontalHeader()->setSectionResizeMode(7, QHeaderView::Stretch);   UpdateWorkers();   modelTasks->select(); } //обновление таблицы void MainWindow::on\_action\_2\_triggered(){  SetToEdit(); } //обновление таблицы void MainWindow::on\_tableView\_clicked(const QModelIndex &index){  SetToEdit(); } //загрузка изображения void MainWindow::on\_pushButton\_4\_clicked(){  QFileDialog dialog(this, "Выберите изображение");  dialog.setFileMode(QFileDialog::ExistingFile);  dialog.setNameFilter(tr("Изображения (\*.png \*.jpg)"));  if (dialog.exec()) {  filename = dialog.selectedFiles().first();  QPixmap pixmap(filename);  ui->label\_11->setPixmap(pixmap);  }  qDebug() <<"filename= "<< filename;  modelTasks->record(ui->tableView->currentIndex().row()).setValue("IMG",filename);   QSqlRecord rec = modelTasks->record(ui->tableView->currentIndex().row());  rec.setValue("IMG",filename);  modelTasks->setRecord(ui->tableView->currentIndex().row(),rec);   if (!modelTasks->submitAll()) {  qDebug() << modelTasks->lastError().text();  } } //заполение void MainWindow::on\_pushButton\_7\_clicked(){  qDebug() <<"cur ROW= "<< ui->tableView->currentIndex().row();  qDebug() <<"else= "<< ui->lineEdit->text();   QSqlRecord rec = modelTasks->record(ui->tableView->currentIndex().row());  //дата начала  QDateTime dateTime = QDateTime::fromString(ui->lineEdit\_2->text(), "hh:mm dd-MM-yyyy");  if(dateTime.isValid()) rec.setValue("DateBegin",ui->lineEdit\_2->text());  else {  QMessageBox::information(this, "Ошибка", "Неправильный формат Времени заявки [hh:mm dd-MM-yyyy] \nБудет вставлено сегодня");  rec.setValue("DateBegin",QDateTime::currentDateTime().toString("hh:mm dd-MM-yyyy"));  }  //дата окончания  QDateTime dateTime2 = QDateTime::fromString(ui->lineEdit\_3->text(), "hh:mm dd-MM-yyyy");  if(dateTime2.isValid()) rec.setValue("DateEnd",ui->lineEdit\_3->text());  else {  QMessageBox::information(this, "Ошибка", "Неправильный формат Срок сдачи [hh:mm dd-MM-yyyy] \nБудет вставлено завтра");  QDateTime chk = QDateTime::currentDateTime().addDays(1);  rec.setValue("DateEnd",chk.toString("hh:mm dd-MM-yyyy"));  }  //проверка на опережение  if (QDateTime::fromString(ui->lineEdit\_2->text(),"hh:mm dd-MM-yyyy")>QDateTime::fromString(ui->lineEdit\_3->text(), "hh:mm dd-MM-yyyy")){  QMessageBox::information(this, "Ошибка", "Дата сдачи стоит раньше даты времени заявки");  QVariant ds = ui->lineEdit\_2->text(); //begin  QVariant dfs = ui->lineEdit\_3->text(); //endf  rec.setValue("DateBegin",dfs);  rec.setValue("DateEnd",ds);  ui->lineEdit\_2->setText(dfs.toString());  ui->lineEdit\_3->setText(ds.toString());  }   if (ui->comboBox->currentText()!=""){  rec.setValue("WorkerKey",GetWorker(ui->comboBox->currentText()));  }   if (ui->checkBox->isChecked()) rec.setValue("Comp","1");  else rec.setValue("Comp","0");  //вставка записей  rec.setValue("Name",ui->lineEdit->text());  rec.setValue("Info",ui->plainTextEdit\_2->toPlainText());  rec.setValue("Сontacts",ui->plainTextEdit->toPlainText());  rec.setValue("Price",ui->lineEdit\_4->text());   modelTasks->setRecord(ui->tableView->currentIndex().row(),rec);   if (!modelTasks->submitAll()) {  qDebug() << modelTasks->lastError().text();  } } //вставка строки в сотрудники void MainWindow::on\_pushButton\_6\_clicked(){  qDebug() << "Вставка строки" ;  modelWorkers->insertRows(modelWorkers->rowCount(),1); } //удаление строки void MainWindow::on\_pushButton\_2\_clicked(){  int selectedRow = ui->tableView\_2->currentIndex().row();  if (selectedRow >= 0){  qDebug() << "удал. строку" << modelWorkers->removeRows(selectedRow,1);  }  else qDebug() << "нет выделенной строки"; } //установка пути до базы void MainWindow::on\_action\_3\_triggered(){  QFileDialog dialog(this, "Выберите путь до базы");  dialog.setFileMode(QFileDialog::ExistingFile);  dialog.setNameFilter(tr("База (\*.sqlite)"));  if (dialog.exec()) {  QString f2ilename = dialog.selectedFiles().first();  sqllite3.setDatabaseName(f2ilename);  } } //запись картинки в базу void MainWindow::on\_pushButton\_5\_clicked(){  modelTasks->record(ui->tableView->currentIndex().row()).setValue("IMG",filename);   QSqlRecord rec = modelTasks->record(ui->tableView->currentIndex().row());  rec.setValue("IMG"," ");  modelTasks->setRecord(ui->tableView->currentIndex().row(),rec);   if (!modelTasks->submitAll()) {  qDebug() << modelTasks->lastError().text();  } } //показать фильтр void MainWindow::on\_action\_4\_triggered(){  ui->dockWidget->setVisible(true); } //фильтр void MainWindow::on\_pushButton\_9\_clicked(){  QString str;  int and\_c = 0;  qDebug() <<"Filter active ";  //счетчик and  if (ui->lineEdit\_5->text()!="") {  if (and\_c!=0) {  str+=" AND ";  and\_c--;  }  str+="Name LIKE '"+ui->lineEdit\_5->text()+"%'";  and\_c++;  }  //дата начала  if (ui->lineEdit\_6->text()!="00:00 01-01-2000"){  if (QDateTime::fromString(ui->lineEdit\_6->text(),"hh:mm dd-MM-yyyy").isValid()){  if (and\_c!=0) {  str+=" AND ";  and\_c--;  }  str+="DateBegin LIKE '%"+QDateTime::fromString(ui->lineEdit\_6->text(),"hh:mm dd-MM-yyyy").toString("dd-MM-yyyy")+"%'";  and\_c++;  }  else {  QMessageBox::information(this, "Ошибка", "Неправильный формат Времени заявки [hh:mm dd-MM-yyyy]");  }  }  //дата конца  if (ui->lineEdit\_9->text()!="00:00 01-01-2000"){  if (QDateTime::fromString(ui->lineEdit\_9->text(),"hh:mm dd-MM-yyyy").isValid()){  if (and\_c!=0) {  str+=" AND ";  and\_c--;  }  str+="DateEnd LIKE '%"+QDateTime::fromString(ui->lineEdit\_9->text(),"hh:mm dd-MM-yyyy").toString("dd-MM-yyyy")+"%'";  and\_c++;  }  else {  QMessageBox::information(this, "Ошибка", "Неправильный формат Времени заявки [hh:mm dd-MM-yyyy]");  }  }  //сотрудник  if (ui->comboBox\_3->currentText()!=""){  if (and\_c!=0) {  str+=" AND ";  and\_c--;  }  GetWorker(ui->comboBox\_3->currentText());  str+="WorkerKey LIKE "+GetWorker(ui->comboBox\_3->currentText())+"";  and\_c++;  }  //цена  if (ui->lineEdit\_8->text()!="") {  if (and\_c!=0) {  str+=" AND ";  and\_c--;  }  str+="Price LIKE '"+ui->lineEdit\_8->text()+"%'";  and\_c++;  }  ui->lineEdit\_7->setText(str);  modelTasks->setFilter(str);  modelTasks->select();  ui->tableView->setModel(modelTasks);  }  //очистка фильтра void MainWindow::on\_pushButton\_8\_clicked(){  ui->lineEdit\_5->clear();  ui->lineEdit\_8->clear();  ui->comboBox\_3->setCurrentText("");  ui->lineEdit\_9->setText("00:00 01-01-2000");  ui->lineEdit\_6->setText("00:00 01-01-2000"); }  //обновление таблицы по нажатию на вкладку void MainWindow::on\_tabWidget\_tabBarClicked(int index){  on\_action\_triggered(); } |