МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ

ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Прикладной информатики

БАКАЛАВРСКАЯ РАБОТА

На тему

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ ДЛЯ УЧЕТА ЗАТРАЧЕННОГО ВРЕМЕНИ И АВТОМАТИЗИРОВАННОЙ ГЕНЕРАЦИИ ОТЧЕТНЫХ ДОКУМЕНТОВ ДЛЯ УДАЛЕННЫХ СОТРУДНИКОВ

Исполнитель

Гончаров Василий Александрович

Руководитель

кандидат технических наук, доцент

Котиков Павел Евгеньевич

«К защите допускаю»

Заведующий кафедрой

кандидат технических наук

Слесарева Людмила Сергеевна

«___»_____20__г.

Санкт–Петербург

2016

Содержание

ВВЕДЕНИЕ

Прошло лишь менее 25 лет с того момента как Всемирная паутина стала общедоступной, а она уже прочно вошла в нашу жизнь. За эту четверть века был совершен уже не один скачок в области информационных технологий и, что говорить, технологические прорывы совершаются и по сей день. Общество потребления сыграло немаловажную роль в развитии технологий и на данный момент мы, вместо огромных вычислительных машин (что характерно, которые существуют и сейчас в виде суперкомпьютеров), помимо ставших уже привычными персональных компьютеров и ноутбуков, имеем компактные и более компактные устройства — ультрабуки, планшеты, смартфоны. Они, вкупе с доступом к Интернету, дают человеку больше возможностей в плане выбора своего места обитания. Ведь благодаря сети появляется возможность постоянно быть в так называемой информационной «зоне комфорта» – посещать знакомые ресурсы, заходить в привычные социальные сети, общаться со старыми знакомыми, тем самым отказываясь от необходимости быть постоянно в одной и той же географической точке. Помимо проживания, аналогичная ситуация картина сложилась и с рынком труда — теперь есть возможность работать на другой точке шара не меняя своего местоположения.

Актуальность данной работы состоит в том, что количество удаленных сотрудников на предприятиях постепенно растет. Согласно исследованию J’son & Partners Consulting, проведенному по заказу компании «Битрикс24», к 2020 году примерно 20% россиян перейдут на работу с удаленным доступом [1]. В связи с этим, наличие вспомогательного инструмента для учета времени и формирования документов значительно упростит процесс обмена документами между удаленными сотрудниками и предприятием, на котором они работают.

Цель данной работы — проектирование и разработка приложения, позволяющего автоматизировать процессы учета времени и формирования документов, для предоставления отчета о проделанной работе и её оплаты.

Для реализации поставленной цели в рамках данной работы необходимо выполнить следующие задачи: составить план предпроектного обследования, провести исследование компании и аналогов приложения, а также разработать проект приложения, которое будет обеспечивать базовый функционал: учет времени, его обработку и хранение, а также генерацию отчетных документов.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Составление плана предпроектного обследования

Главной задачей разрабатываемого программного обеспечения является учет рабочего времени, соответственно основными пользователями данного ПО будут удаленные сотрудники предприятий.

При составлении плана предпроектного обследования были выделены подразделения предприятия, которые могут помочь в обследовании, проконсультировав по ряду вопросов – отдел информационных технологий (ОИТ) и отдел кадров (ОК). Также в опросе участвуют руководители, в чьем подчинении находятся удаленные сотрудники (РУК).

В соответствии с компетенцией выделенных отделов был разработан список вопросов, который представлен в таблице 1.

Таблица 1 – Список вопросов для уточнения

Номер	Отдел	Вопрос
1	ОК	Количество удаленных сотрудников?
2	ОК	В чем заключается деятельность удаленных сотрудников?
3	ОК, РУК	Как проходит учет времени сейчас?
4	ОК, РУК	Какая информация должна находиться в отчете?
5	ОК, РУК	Как проходит округление часов?
6	ОК, РУК	Есть ли какие-либо планы, нормы, минимумы по рабочим часам для УС, регламент таких норм?
7	ОК, РУК	Необходим ли онлайн-учет или достаточно выгрузок вечером?
8	ОК, РУК	Какова периодичность отправки отчета?
9	ОК, РУК	Необходимо ли формировать групповые отчеты?
10	ОК, РУК	Необходимо ли автоматизированное формирование актов?
11	ОК, РУК	Как производится оплата работ удаленных сотрудников?
12	ОИТ	Как сейчас выполняется подключение по УД?
13	ОИТ	Возможно ли организовать хранение базы данных на серверах компании?
14	ОИТ	Выдаются ли удаленным сотрудникам корпоративные устройства (ноутбуки, планшеты)?

1.2 Составление плана интервьюирования и анкетирования сотрудников

После анализа первичной документации был разработан список вопросов (таблица 2) для удаленных сотрудников, необходимый для уточнения требований к проектируемому приложению.

Таблица 2 – Список вопросов для анкетирования удаленных сотрудников

Номер	Вопрос
1	Контроль за исполнением задач и корректностью указанных трудочасов. Как осуществляется сейчас?
2	Каким образом формируются отчеты? Их формат?
3	Сколько обычно затрачивается времени на формирование отчета?
4	Каким образом формируются акты?
5	Сколько времени уходит на формирование акта?
6	Как часто отправляются отчеты?
7	Как часто отправляются акты?
8	Какой формат у отправляемых документов?
9	Какие устройства используются при работе?
10	Какие технические характеристики у указанных устройств?
11	Какие операционные системы используются на данных устройствах?

По результатам анкетирования было выяснено, что на данный момент контроль корректности указанных трудочасов ведется на «доверии», но при возникновении подозрений у руководителя о завышении значения затраченных часов — может быть инициирована проверка, в ходе которой будет проведен анализ логов внутренних систем, отображающих длительность сессий пользователя.

Единый формат есть только для актов, отчеты сдаются по требованию руководителя в свободной форме. Отчет и акт отправляются 1 раз в месяц, отчет отображает рабочую активность в течение дня. Акт является основанием для оплаты. Формирование акта занимает 30 минут в месяц, ведение отчета — в среднем 20 минут в день, то есть, приблизительно, 10 часов в месяц.

1.3 Анализ технического и программного обеспечения

В ходе анализа были уточнены как программно-аппаратные характеристики офисных персональных компьютеров предприятия, так и программно-аппаратные конфигурации персональных компьютеров удаленных работников.

После анализа полученных данных была выбрана самая низкопроизводительная конфигурация ПК, которая отображена в таблице 3. Данная конфигурация будет принята при разработке как минимальная конфигурация, на которой гарантируется выполнение приложения.

Таблица 3 – Минимальная аппаратная конфигурация ПК

Аппаратная часть:	
Элемент	Значение
Процессор	Intel Pentium J2900
ОЗУ	2 Гб
HDD	250 Гб

Для разработки приложения из программной конфигурации необходимо знать только тип операционной системы, поэтому в таблице 4 отражен только этот параметр.

Таблица 4 – Программные конфигурации

Тип ПО	Значение
Операционная система	Microsoft Windows Server 2003 (терминальный доступ)
	Microsoft Windows 7
	Microsoft Windows 8
	Дистрибутивы ОС, базирующиеся на GNU/Linux (ядро 4.3.x и выше)

1.4 Исследование документации и внутренней отчетности предприятия

Для исследования документации и внутренней отчетности были запрошены примеры документов у удаленных сотрудников:

1. Отчет по затраченному времени.

Не имеет утвержденной формы и не является обязательным элементом, предоставляется по требованию непосредственного руководителя.

2. Акт сдачи-приема работ

Утвержден юридическим отделом, является приложением к договору возмездного оказания услуг с физическими лицами. Документ отправляется в форматах doc или pdf и содержит следующие поля для заполнения:

1. Дата.
2. Место составления документа.
3. Наименование организации.
4. Ф.И.О. лица, представляющего организацию (Заказчик).
5. Основание для работы Заказчика (доверенность организации, её номер и дата).
6. Ф.И.О. лица, оказавшего услуги (Исполнитель).
7. Основание для работы Исполнителя (договор возмездного оказания услуг с предприятием, его номер и дата).
8. Таблица, содержащая в себе перечисление типов работ и количество затраченных часов на каждый из указанных типов.
9. Место оказания услуг.
10. Сумма оплаты без учета налоговых сборов.

1.5 Разработка SADT-модели

Для уточнения требований к ПО, для чего необходимо описать бизнес-процесс, в котором будет задействовано разрабатываемое приложение.

На рисунке 1 отображена контекстная диаграмма в нотации IDEF0, отображающая бизнес-процесс по согласованию и оплате трудозатрат, который здесь назван как «Согласование рабочего времени за месяц»

Рисунок 1 Контекстная диаграмма «Согласование выполненных работ»

Как видно из контекстной диаграммы, удаленный сотрудник формирует документы исходя из следующих динамических данных — часы, затраченные на работу за отчетный период, связка полей «дата» и «комментарий», где указывается что и когда именно было выполнено, ставка, по которой рассчитывается заработная плата, а также конечная сумма.

Регламент сдачи отчетных документов оговорен в Договоре, заключенным между предприятием и удаленным сотрудником, а также в Трудовом кодексе Российской Федерации (ТК РФ). В последнем, также регламентированы иные аспекты работы и оплаты труда, которые не указаны в Договоре.

В процессе формирования документов помимо удаленного сотрудника также участвуют:

1. Непосредственный руководитель удаленного сотрудника, с которым согласовывается отчет по выполненной работе

2. Исполнительный директор, который подписывает акт, что дает основание для выплаты исполнителю указанной в нем суммы.

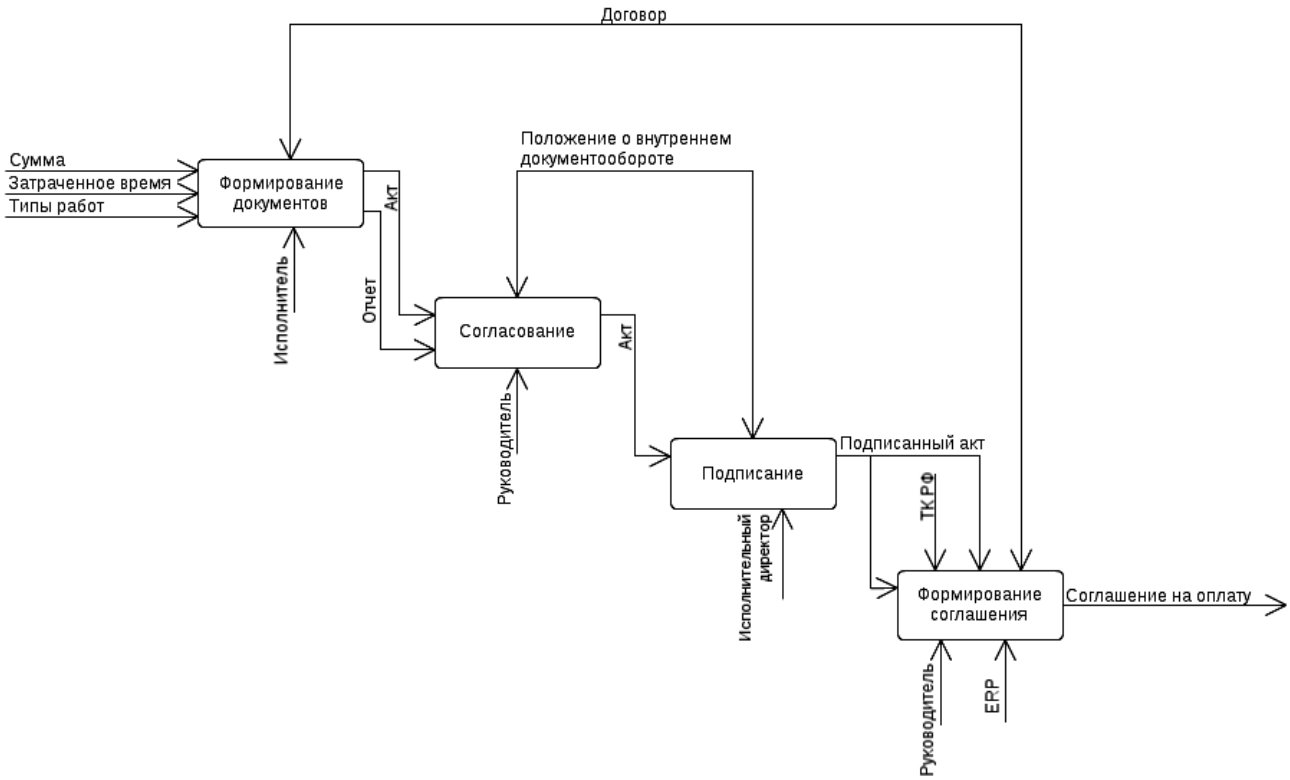


Рисунок 2 Декомпозиция диаграммы «Согласование выполненных работ» Модель AS IS

Процесс согласования документов и сроки регламентируются положением о внутреннем документообороте предприятия. Рассмотрим подробнее данный процесс, а также изменения, которые внесет внедрение разрабатываемого программного обеспечения — на рисунке 2 представлена декомпозиция описываемого процесса. Как можно увидеть из данной диаграммы, весь бизнес-процесс состоит из 4рех уровней:

Уровень 1: Формирование документов.

На данном этапе удаленный сотрудник, в соответствии с указанным в Договоре регламентом сдачи отчетных документов, при помощи любых имеющихся у него инструментов составляет в отчет, в котором указывает по датам количество часов и комментарии по дням, а также рассчитывает общую сумму по ставке. Также, на основе имеющейся формы, составляется акт за отчетный период, куда вносятся общие данные по работе с разбиением на категории и указанием суммы выплаты за выполненную работу.

Вместе с передачей документов руководителю процесс продвигается на следующий уровень..

Уровень 2: Согласование с непосредственным руководителем.

Руководитель просматривает отчет и акт, сверяет данные в документах. Если нет претензий по соотношению объема и качества выполненной работы, то он [руководитель] отправляет согласует акт и отправляет его на следующий уровень. Отчет по задачам остается у руководителя.

Уровень 3: Согласование с исполнительным директором.

Исполнительный директор ознакомляется с актом и, если также не имеет возражений, подписывает его и возвращает непосредственному руководителю сотрудника.

Уровень 4: Заявка на оплату.

На данном уровне согласованный и подписанный акт является также и основанием для выплаты указанной суммы предприятием сотруднику. Здесь руководитель оформляет в ERP заявку на выплату, которая передается ответственным лицам.

На этом бизнес процесс согласования можно считать завершенным, так как продвижение заявки в ERP не относится к процессу, который затрагивает разрабатываемое приложение.

Построив модели AS IS бизнес-процесса, следует также проанализировать, какие именно этапы будут затронуты при внедрении приложения.

На рисунке 3 предоставлена модель TO BE: декомпозиция диаграммы рисунка 1 с учетом внедрения разрабатываемого приложения.

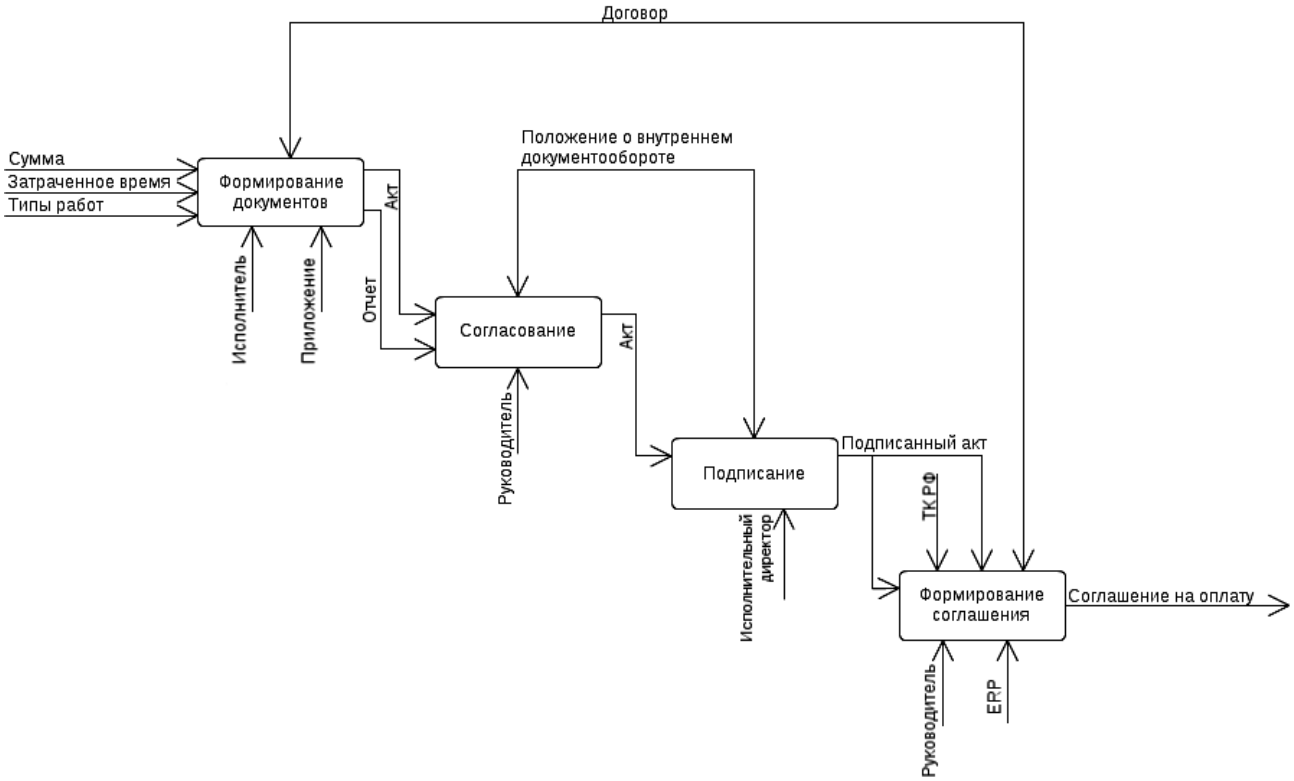


Рисунок 3 Декомпозиция процесса «Согласование выполненных работ». Модель TO BE

Как видно из схемы, представленной на рисунке выше, с внедрением приложения изменяется уровень «Формирование документов», на котором добавляется еще один механизм — приложение, которое будет генерировать отчетные документы.

1.6 Изучение аналогов приложения

Прямых аналогов разрабатываемое программное обеспечение не имеет, но есть целый ряд приложений, относящихся к системам управления проектами, в которых указано наличие учета трудозатрат по задачам и суммарно по проектам. Далее будут рассмотрены некоторые из них.

1. Redmine

Redmine — одно из самых распространенных серверных веб-приложений для управления проектами. Является бесплатным, свободно распространяемым приложением с открытым исходным кодом. Предоставляет следующий функционал:

- Гибкая система доступа;
- Система отслеживания ошибок;
- Диаграммы Ганта и календарь;
- Ведение новостей проекта, документов и управление файлами;
- Учет временных затрат;
- Настраиваемые произвольные поля.
- Легкая интеграция с системами управления версиями.
- Создание записей об ошибках на основе полученных писем;[2]

2. Wrike

Wrike - онлайн-инструмент для командной работы. Он позволяет пользователям планировать проекты, устанавливать приоритеты для задач и следить за графиком их выполнения, а также совместно работать со своими коллегами в режиме онлайн. Предоставляет следующий функционал:

- Иерархия проектов с неограниченным числом уровней.
- Выставление приоритетов задачам.
- Ведение диалогов в задачах.
- Табличное форматирование.
- Создание подзадач.
- Массовые опции по отношению к задачам.

- Отслеживание степени загрузки сотрудников.
- Учет затраченного времени.
- Отчётность и уведомления.
- Фильтры для задач. [3]

Как было уже ранее указано, данные приложения относятся к системам управления проектами, основным функционалом которых является никак не учет времени, а планирование задач, распределение ресурсов, документирование проектов и т.п. Весь заявленный учет времени систем управления проектами заключается в том, что пользователю необходимо внести в каждую задачу количество часов, которое он затратил на выполнение и, в дальнейшем, по указанному времени в задачах возможно сгенерировать выгрузку.

Но, в то же время в данные системы, как правило, достаточно сложно вносить ряд не планируемых краткосрочных задач, таких как, к примеру, консультация пользователей по телефону или почте — время заведения заявки сопоставимо со временем её выполнения, что увеличивает время реакции для всех остальных задач. Создание же одной общей задачи для типа работ может внести путаницу в статистику что, в конечном итоге, может заставить пользователя вручную вносить исправления в отчетные документы.

Проектируемое приложение позволяет частично закрыть проблему учета времени, тем самым становясь не конкурентом вышеуказанного типа приложений, а наоборот — вспомогательным инструментом.

1.7 Схема базы данных

Для хранения информации, фиксируемой приложением, используется база данных на основе SQLite[1]. Так как не исключается возможность

1

многопользовательского доступа к БД, структура базы изначально приведена к третьей нормальной форме. На рисунке 4 представлена текущая схема БД и отношения между таблицами.

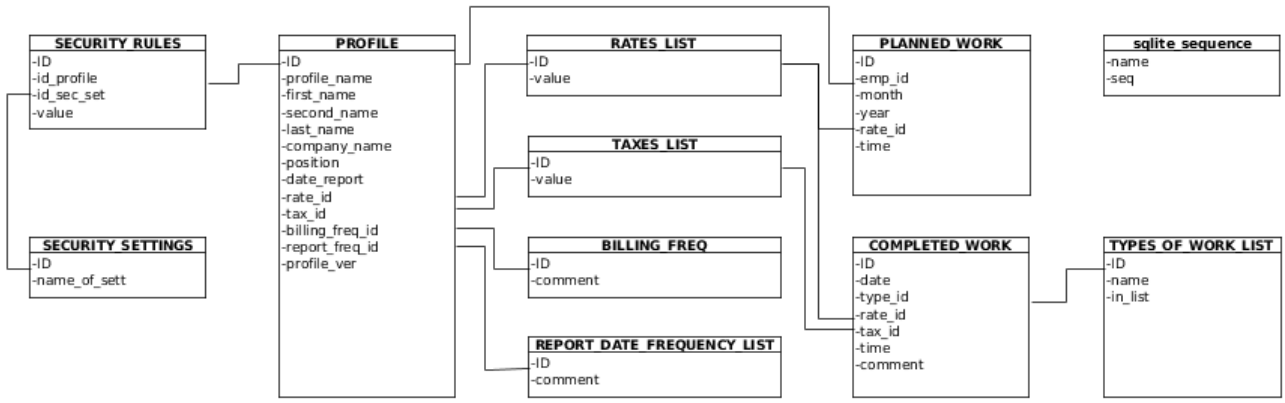


Рисунок 4 Схема базы данных

Ввиду приведения базы к третьей нормальной форме указанные таблицы можно разделить на три категории:

1. Основные (PROFILE, COMPLETED_WORK, PLANNED_WORK) — предназначены для хранения данных о пользователе, затраченном и запланированном времени.

2. Справочники (TYPES_OF_WORK_LIST, RATES_LIST, TAXES_LIST, BILLING_FREQ, REPORT_DATE_FREQUENCY_LIST) — содержат информацию о типах работ, налоговых ставках, ставках оплаты и оплачиваемых и отчетных периодах за все время работы.

3. Внутренние и технические таблицы (SECURITY_RULES, SECURITY_SETTINGS, sqlite_sequence) — более подробно эти таблицы рассмотрены в следующем разделе.

1.7.1 Список полей таблиц

1. PROFILE

SQLite — компактная встраиваемая реляционная база данных[4], исходный код которой является общественным достоянием, что, в свою очередь, позволяет использовать её без каких-либо ограничений[5].

- ID
- profile_name
- first_name
- second_name
- last_name
- company_name
- position
- date_report
- rate_id
- tax_id
- report_freq_id
- profile_ver

2. RATES_LIST
   - ID
   - value
3. TAXES_LIST
   - ID
   - value
4. BILLING_FREQ
   - ID
   - comment
5. REPORT_DATE_FREQUENCY_LIST
   - ID
   - comment

6. COMPLETED_WORK
   - ID
   - date

- type_id
- rate_id
- tax_id
- time
- comment

7. PLANNED_WORK
    - ID
    - emp_id
    - month
    - year
    - rate_id
    - time
8. SECURITY_RULES
    - ID
    - id_profile
    - id_sec_set
    - value
9. SECURITY_SETTINGS
    - ID
    - name_of_sett
10. TYPES_OF_WORK_LIST
    - ID
    - name
    - in_list

Также в структуре текущей базы данных присутствует дополнительная таблица, именованная как sqlite_sequence и содержащая следующие поля:

- name
- seq

Данная таблица является внутренней для всех баз данных развернутых на SQLite и используется в реализации автоматического инкремента для суррогатных ключей таблиц.

Каждая запись в sqlite_sequence соответствует одной таблице базы данных, которая использует для одного из своих полей свойство AUTOINCREMENT.

В таблице PROFILE в качестве ID предполагается использование табельного номера сотрудника на предприятии, что позволит однозначно идентифицировать его. Если значение в ID является суррогатным, то поиск производится по группе полей: фамилия, имя, отчество, должность, компания.

Таблицы SECURITY_RULES и SECURITY_SETTINGS созданы для последующей реализации разграничений доступа к полям профиля, в том случае, если изменения в поля будет вносить пользователь приложения. Более подробно данный момент описан в разделе .

1.8 Создание модели приложения

В данном разделе производится графическое описание проектируемого приложения на языке UML (Unified Modelling Language)[6]. На основании представленных ниже диаграмм в дальнейшем возможно проведение согласования с заказчиком требований, предъявляемых к проектируемому приложению, построение его архитектуры, а также написание технической документации к приложению.

1.8.1 Диаграмма прецедентов

На рисунке 5 представлена диаграмма прецедентов, описывающих варианты взаимодействия пользователя с разрабатываемым приложением.

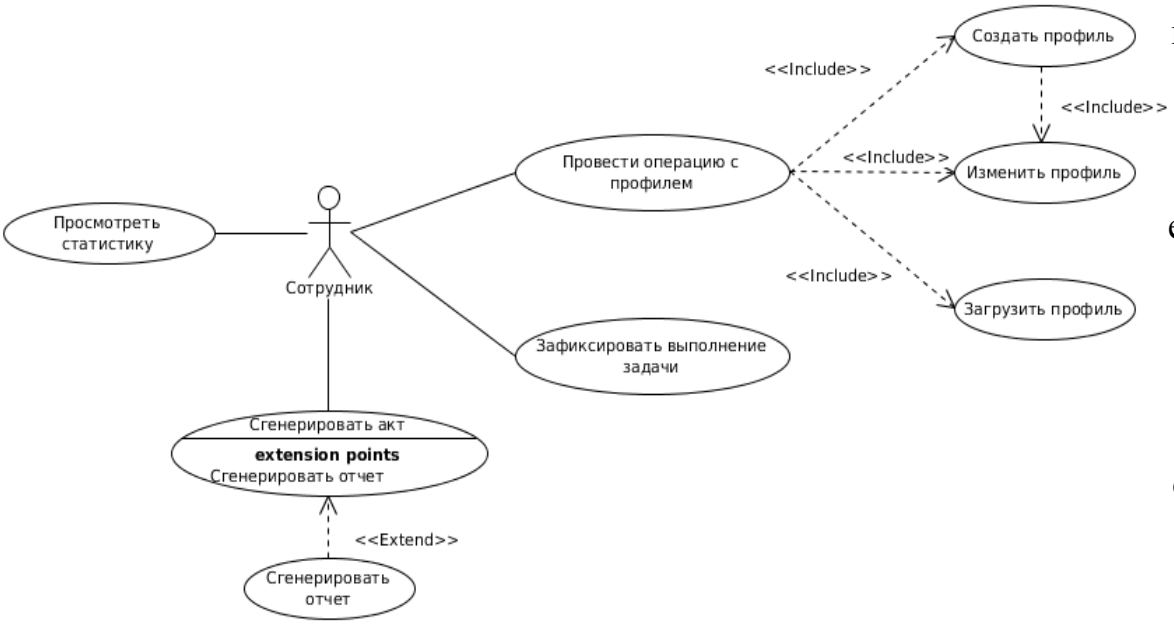


Рисунок 5 Диаграмма прецедентов

Описание прецедентов:

1. **Просмотреть статистику** — Сотрудник может узнать о количестве затраченных часов в день или отчетный период (по умолчанию — месяц), а также о сумме и сумме за вычетом налогов за данный период времени.

2. **Сгенерировать акт —** Сотрудник может сгенерировать акт по имеющемуся шаблону. Также, в дополнение к акту есть возможность сгенерировать отчет (также по заранее созданному шаблону).

3. **Зафиксировать выполнение задачи —** при выполнении данного прецедента Сотрудником выполняется ряд действий, в ходе которых к значению времени добавляется дополнительная информация и выполняется запись в базу данных.

4. **Провести операцию с профилем** — данный прецедент подразумевает под собой один или несколько следующих прецедентов:

- Создать профиль — Сотрудником заполняется основная информация и запускается генерация первичного профиля, после

чего открывается окно редактирования профиля. Более подробно данный процесс описан в .

- Изменить профиль — Сотрудник может изменить информацию в своем профиле и сохранить новую его версию. При этом на редактирование профиля наложены определенные ограничения на внесение изменений в определенные поля. Таблица ограничений внесена в базу и должна быть предварительно настроена руководителем удаленного работника или сотрудником, ответственным за безопасность данных.
- Загрузить профиль — В этом прецеденте выполняется загрузка профиля, а также подключение к базе данных с указанными в профиле данными. В случае с удаленной базой данных и удаленным учетом времени, предполагается что удаленный сотрудник не будет заниматься предварительным внесением всех необходимых данных для работы, а будет просто получать стартовый файл профиля, содержащий минимальный набор данных, позволяющий подключиться к базе данных, а остальной набор настроек для работы будет выгружен при первом успешном подключении к удаленной базе.

1.8.2 Диаграмма классов

На диаграмме, представленной на рисунке 6, описан минимальный набор классов, необходимых для реализации функционала приложения и связи между ними. Данная диаграмма разработана с концептуальной точки зрения, для её упрощения, классы представлены в виде модулей, в названии который указано их назначение. К переменным внутри модулей применено тоже правило. При разработке диаграмма классов должна быть изменена с учетом разрабатываемых функций и оформлена с точки зрения реализации.

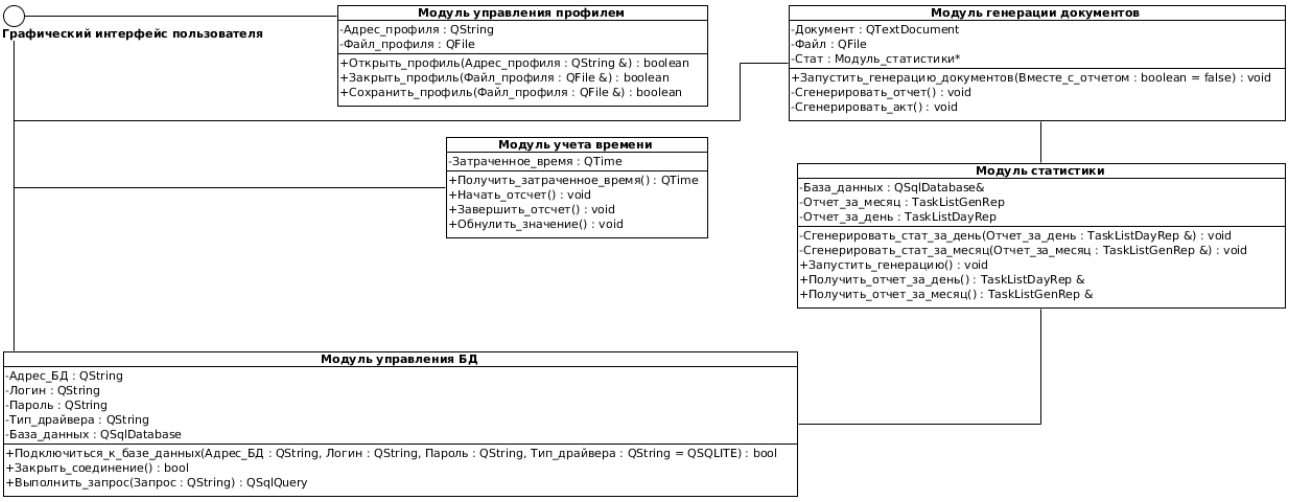


Рисунок 6 Диаграмма классов (концептуальная)

Основным на данном рисунке связующим звеном выступает интерфейс пользователя, который позволяет взаимодействовать с модулями посредством различных окон графического интерфейса.

Модуль учета времени отвечает за процесс контроля затраченного времени (управление таймером), а также за работу со значением таймера (инкремент, обнуление).

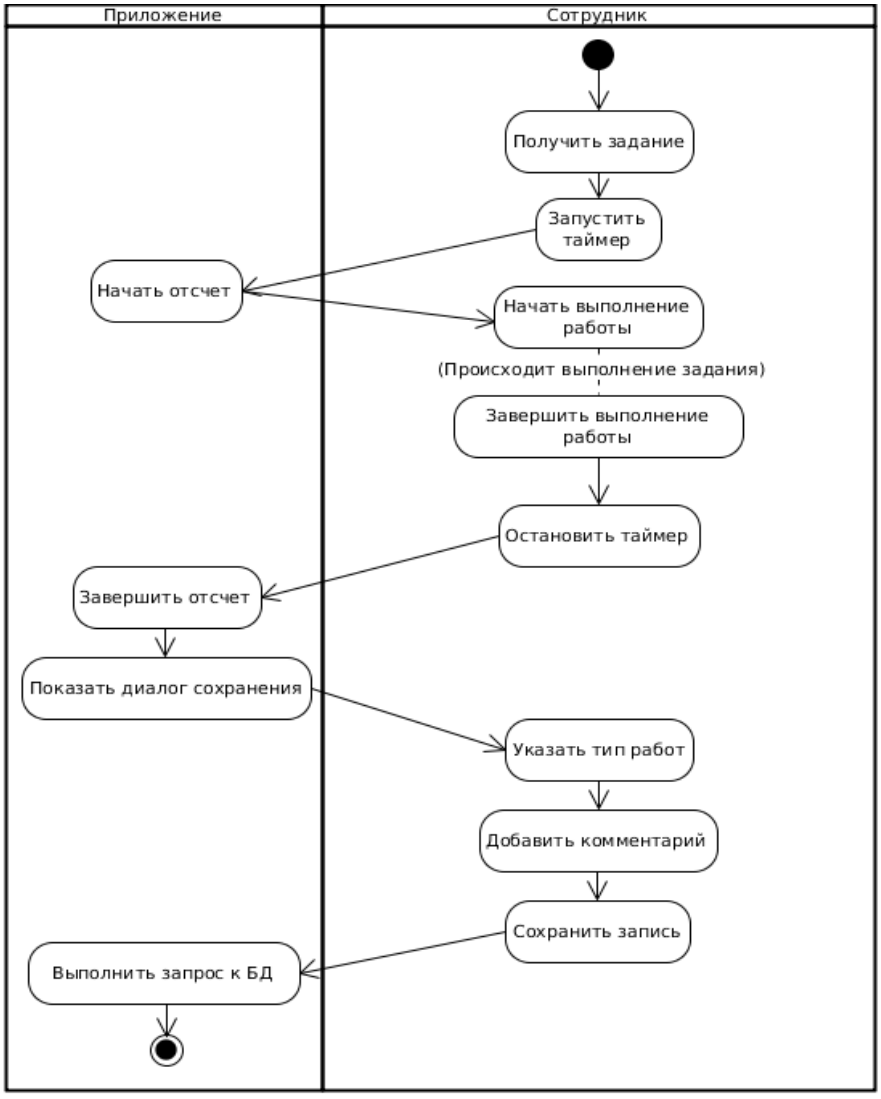
Модуль статистики отвечает за сбор данных, необходимых для заполнения шаблона, путем формирования запросов к базе данных, при помощи модуля управления базой данных, и последующей агрегации полученного запроса в установленные типы данных, формат которых приведен в разделе . Эти данные использует модуль генерации документов, роль которого состоит в

том, чтобы с помощью элемента QTextDocument[7] вносить собранные и оформленные данные в XML-структуру[8] шаблонов, заранее подготовленных пользователем.

1.8.3 Диаграмма деятельности

1.8.3.1 Прецедент «Зафиксировать выполнение задачи»

На рисунке 7 представлен алгоритм прецедента «Зафиксировать выполнение задачи».



Для данной диаграммы предполагается что уже сотрудником уже создан и загружен корректный профиль. Сотрудник, получив задание, запускает таймер и

начинает выполнение задачи. По факту выполнения задания сотрудник останавливает таймер, указывает дополнительную информацию — тип проведенных работ и комментарий (к примеру, это может быть номер задачи на портале) и сохраняет запись в базу данных (обращение к базе данных не отображено на диаграмме для сокращения избыточности).

Данный алгоритм является типовым и не включает в себя такие исключения как постановка задачи на паузу, отмену выполнения, сброс таймера и тому подобные — обработка перечисленных ситуаций должна быть реализована в ходе разработки.

1.8.3.2 Прецедент «Сгенерировать документ»

На рисунке 8 представлен типовой алгортим генерации документов.

Для данной диаграммы предполагается что пользователь уже подготовил шаблоны и разместил их в каталоге, а также инициировал процесс генерации

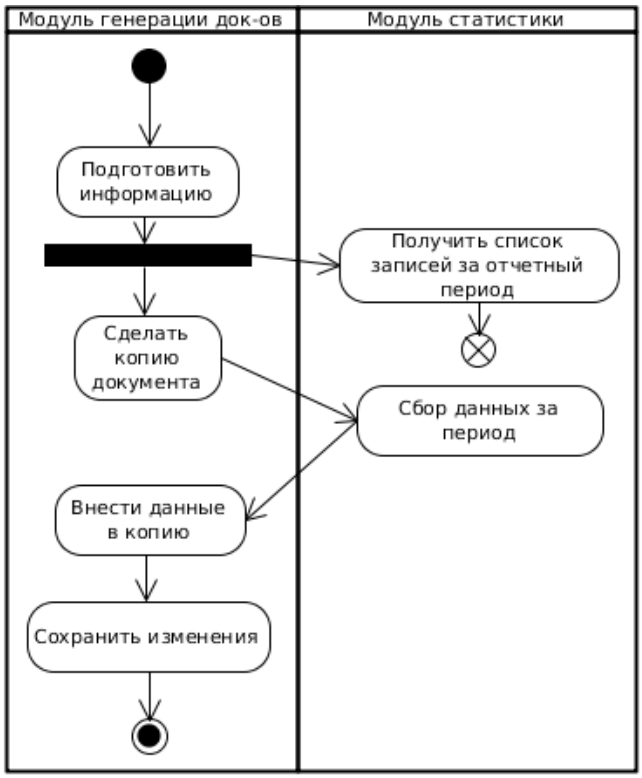


Рисунок 8 Процесс генерации документа

документа

При запуске формирования документа, модуль генерации обращается к модулю статистики, который, в свою очередь, запрашивает у базы данных информацию об операциях (обращение к базе данных не отображено на диаграмме для сокращения избыточности) за отчетный период (указывается в настройках профиля).

После получения записей за весь период, формируется представление данных за периоды, расположенные внутри отчетного периода. К примеру, это может представление за месяц — для акта или представление за каждый день внутри периода — для отчета.

Сгенерированное представление идет в копию шаблона, предоставленного пользователем заранее перед генерацией и оформленного в соответствии с правилами. Формат представления данных и правила оформления перечислены в разделе .

В диаграмме (рисунок 8) представлен алгоритм, в котором подразумевается, что данные не были собраны и формируется только один документ.

В случае формирования как акта, так и отчета — алгоритм для отчета будет аналогичен представленному, за исключением запроса к базе данных — так как выгрузка информации с базы уже была произведена, повторный запрос выполняться не будет. Также, ввиду различий в представлении информации, для акта будет использоваться другой шаблон документа.

1.8.4 Диаграмма компонентов

Диаграмма компонентов, представленная на рисунке 9, представляет собой отображение проектируемой на данный момент реализации приложения, а именно — однопользовательское приложение с локальной базой данных.

База данных, структура которой представлена в разделе , в данном примере также является однопользовательской, но в дальнейшем её возможно

развернуть на удаленном сервере, при необходимости промигрировав на СУБД (к примеру, MySQL), и внедрить в качестве многопользовательской базы данных.

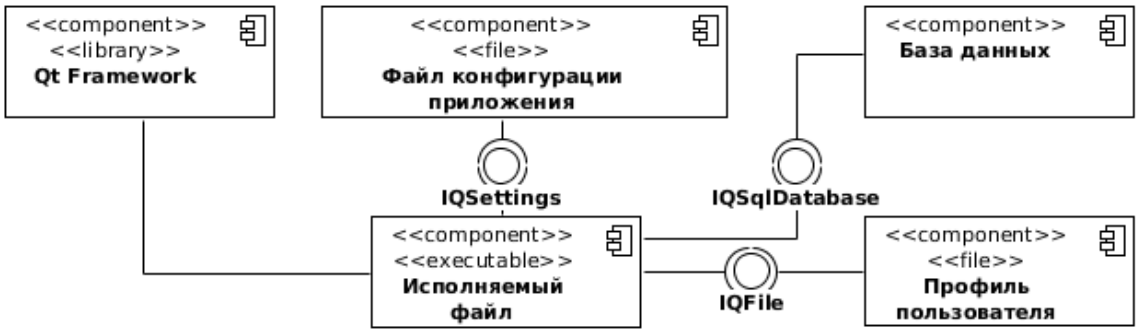


Рисунок 9 Диаграмма компонентов

Загрузка профиля пользователя и работа с файлом выполняется с помощью QFile[9], работа с данными в файле, ввиду того что сам профиль является XML-документом, о чем более подробно рассказано в разделе , выполняется с помощью следующих технологий: создание файла профиля — SAX (Simple API for XML)[10], а внесение изменений в структуру — с помощью QDom (QDomDocument)[11].

Подключение базы данных выполняется с помощью QSqlDatabase[12], которому передаются параметры (включая тип драйвера), необходимые для корректного соединения с базой данных.

Файл конфигурации приложения подключается только посредством QSettings[13], так как данный интерфейс самостоятельно выполняет запись и замену наименований настроек и их переменных, а также их форматирование в файле, без необходимости явно открывать его с помощью QFile.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

В данной части будет рассмотрена реализация основных компонентов разрабатываемого приложения, а именно — как сейчас ведется учет времени, как собирается статистика, в каком виде передается. Также будут описаны принципы работы с файлами: работа с профилем пользователя и генерацией документа.

2.1 Учет времени

Учет времени — главная функция в разрабатываемом приложении. В отличии от большинства продуктов на рынке, которые говоря об «учете времени» подразумевают форму ввода, в которую пользователь должен ввести количество затраченных часов, предварительно им же посчитанных, данное приложение будет самостоятельно вести учет времени и подготавливать полученные значения для выгрузки в базу данных, для чего используются штатные средства библиотеки Qt.

2.1.1 Схема работы и инструменты таймера

Учет времени ведется по стандартному алгоритму — при начале работы, сотрудник запускает таймер, с установленным интервалом (по умолчанию — 1 сек). Запущенный таймер через заданный интервал времени отправляет сигнал timeout(), который посредством использования механизмов сигналов и слотов Qt, связан со слотом, в котором переменной прибавляется 1 секунда.

Использование таймера для учета времени обусловлено тем, что измерение трудозатрат по локальному времени компьютера дает возможность пользователю указать некорректное время, затраченное на конкретную задачу, путем перевода часов на компьютере и последующем останове таймера.

Так как таймер, реализованный в QObject, имеет такие недостатки как необходимость реализации метода, принимающего события объекта, а также необходимость создания системы различий для таймеров, если в одном классе

их несколько, в приложении используется класс QTimer, являющийся наследником QObject и имеющим необходимые функции. Он объявлен в классе MainWindow, отвечающем за логику работы с графическим интерфейсом основного окна приложения.

2.1.2 Хранение информации о затраченном времени

Для хранения времени была разработан класс StopwatchController, представляющий собой обертку для объекта stopwatch класса QTime, в котором хранятся данные о затраченном времени.

Объект stopwatch объявлен приватным и основное взаимодействие с ним осуществляется через публичные методы:

```
void StopwatchController::inc_time()
```

Прибавляет 1 секунду к текущему значению stopwatch. Ввиду иммутабельности переменных в Qt используется конструкция вида:

```
*stopwatch = stopwatch->addSecs(1);
```

Данное выражение присваивает значение объекту по ссылке, равное текущему значению этого объекта + 1 секунда. Добавление секунды выполняется с помощью метода QTime::addSecs(int).

```
void StopwatchController::reset_timer()
```

Устанавливает значения таймера на стартовую позицию (00 ч., 00 м., 00 с., 000 мс.)

```
QString StopwatchController::get_string_time()
```

Возвращает значение таймера в текстовом формате, по умолчанию «hh:mm:ss», где hh — часы, mm — минуты, ss — секунды. Использование обертки позволяет скрыть объект от других классов, во избежание прямого воздействия на хранящееся в нем значение, такое как сброс или подмена, тем самым устанавливая дополнительную защиту от изменений.

2.1.3 Запись информации в базу данных

При нажатии на кнопку предназначенную для завершения работы над задаче, производится останов таймера и отображается окно сохранения данных,

в котором пользователю предлагается выбрать тип работ и оставить комментарий к задаче.

На основании полученных данных формируется запрос для базы данных, куда входит следующая информация:

1. Дата
2. ID типа работ
3. Время
4. Комментарий
5. ID текущей налоговой ставки
6. ID текущей ставки

Избыточность передачи данных (добавление к каждой записи ID из справочника) обусловена вероятностью смены этих данных в последующем и необходимо для корректного формирования статистики за прошлый период. О понятии «статистика» в данном контексте, а также о её структуре более подробно рассказано в разделе .

В связи с этим, предполагается, что записи из справочников не удаляются и их ID не изменяется (к примеру, путем перемены местами текстовых значений, относительно идентификатора), если они фигурируют, как минимум, в одной записи таблицы COMPLETED_WORK.

2.2 Сбор статистики

Под статистикой на данном этапе разработки приложения понимается набор данных, достаточный для отчетных документов, ниже перечислен их список:

1. Данные, необходимые для отчета за день:

- Дата
- Суммарное количество часов за дату
- Перечень выполненных задач за дату.

2. Данные, необходимые для отчета за месяц:

- Тип задач
- Суммарное количество часов для типа за месяц
- Перечень задач данного типа

На текущий момент, исходя из ранее проведенного опроса, предполагается, что формируется только два документа - подробный по дням и краткий, который включается в шаблон акта сдачи-приемки работ. Таким образом, определение дат генерации отчета осуществляется следующим образом: из поля date_report таблицы PROFILE базы данных берется дата сдачи отчета, а из даты, установленной на компьютере пользователя, берется месяц. На основе этих данных формируются две даты:

1. Дата предыдущего отчета: это день, указанный в параметре date_profile и месяц из текущей даты минус 1 день.
2. Дата текущего отчета: день, указанный в параметре date_profile, плюс 1 день и месяц из текущей даты.

Ниже рассмотрен пример формирования статистики сразу для двух документов — отчета и акта.

2.2.1 Структура хранения статистических данных

В приложении реализованы собственные типы переменных, предназначенных для сбора, передачи и хранения данных для отчетов.

Представлены они в виде структур-агрегаторов, для каждого типа документа: TaskListGenRep для акта и TaskListDayRep для отчета. Структуры содержат в себе только векторы, которые, в свою очередь, являются набором структур, имеющих тоже наименование, что и структуры-агрегаторы, но с приставкой Single, описание которых представлено ниже, в таблицах 5 и 6.

Таблица 5 — Структура TaskListGenRepSingle

Наименование	Тип	Комментарий
typeName	QString	Имя типа, к которому формируются задачи
tasksInType	QStringList	Перечень задач за месяц, относящийся к типу, указанному в typeName
totalForTypeTime	QString	Время, затраченное на выполнение задач данного типа, в формате «h:mm:ss»

Таблица 6 — Структура TaskListDayRepSingle

Наименование	Тип	Комментарий
date	QString	Имя типа, к которому формируются задачи
tasksInDay	QStringList	Перечень задач за день, относящийся к дате, указанной в date
totalForDayTime	QString	Время, затраченное на выполнение задач в указанную дату, в формате «h:mm:ss»

2.2.2 Алгоритм сбора статистики

При инициализации класса статистики, конструктору передаются две даты, представляющие диапазон, за который необходимо сформировать данные.

После проведения инициализации конструктором автоматически выполняется подготовка исходных данных, состоящая из нескольких этапов:

Этап 1 — Сбор первичной информации:

Вызывается функция, выполняющая запрос к базе данных, в котором собираются поля date, time, type_id, comment, за указанный диапазон дат и сохраняется в объект rangeQuery типа QSqlQuery[14].

Этап 2 — Подготовка структур для хранения данных:

В отдельные объекты QStringList собираются списки дат (объект daysInRange) и типов задач (объект typesInRange) с запроса, выполненного на первом этапе и хранящегося в объекте rangeQuery. Дублирующиеся записи в этих листах удаляются при помощи вспомогательного метода

QStringList::removeDuplicates(). Проводится инициализация векторов в структуре:

- Вектору, находящемуся в TaskListGenRep, задается размер, равный длине typesInRange.
- Вектору, находящемуся в TaskListDayRep, задается размер, равный длине daysInRange.

Этап 3 — Предварительное заполнение структур данных

На данном этапе производится перенос данных из ранее собранных объектов типа QStringList[15] — typesInRange и daysInRange в соответствующие объекты TaskListGenRepSingle (typeName) и TaskListDayRepSingle (date) внутри векторов TaskListGenRep и TaskListDayRep соответственно.

Этап 4 — Заполнение векторов данными

После того как подготовка выполнена, начинается этап сбора данных, выполняющийся следующим образом:

Для акта:

1. Запускается цикл, перебирающий все элементы вектора TaskListGenRep.
2. Запускается цикл, перебирающий все значения rangeQuery.
3. Если имеется совпадение между значением типа, хранящегося в конкретном элементе TaskListGenRepSingle и типа, указанного в конкретном элементе rangeQuery, то в tasksInType добавляется значение поля comment из rangeQuery.
4. Выполняется очистка дублей в tasksInType посредством метода QStringList::removeDuplicates().

Для отчета выполняются аналогичные действия:

1. Запускается цикл, перебирающий все элементы вектора TaskListDayRep.
2. Запускается цикл, перебирающий все значения rangeQuery.

3. Если имеется совпадение между значением даты, хранящегося в конкретном элементе TaskListDayRepSingle и даты, указанного в конкретном элементе rangeQuery, то в tasksInDat добавляется значение поля comment из rangeQuery.
4. Выполняется очистка дублей в tasksInDay посредством метода QStringList::removeDuplicates()

На данном этапе формирование запросов к базе данных значительно упростило бы обработку данных, снизив количество вычислительных операций, но при этом создало бы достаточно высокую нагрузку на базу данных, чем можно пренебречь в рамках использования локальной базы данных, но необходимо учитывать при взаимодействии с удаленной СУБД.

Этап 5 — Расчет дополнительных параметров

На этом этапе производится подсчет суммарного объема трудозатрат за месяц и для акта сдачи-приемки работ округляется до целого часа по следующему правилу: если в общей сумме трудозатрат >= 15 минут, то они округляются в большую сторону, иначе — в меньшую. После этого, на основе всех полученных данных, производится расчет суммы оплаты и суммы за вычетом налогов.

2.2.3 Пример обработки данных

В таблице 7 представлен пример входных данных, которые должны быть обработаны и подготовлены приложением перед непосредственной генерацией документа. Далее в таблицах 7-10 в названии в скобках указаны наименования таблиц базы данных разрабатываемого приложения, ранее рассмотренной в разделе .

Таблица 7 — Входные данные для обработки из таблицы работ (COMPLETED_WORK)

ID	DATE	TYPE_ID	TAX_ID	RATE_ID	TIME	COMMENT
1	2016-04-20	2	1	3	00:21:00	55252
2	2016-04-20	1	1	3	01:11:12	53841
3	2016-04-20	1	1	3	00:53:21	54312
4	2016-04-20	3	1	3	02:21:12	Почта
5	2016-04-21	1	1	3	03:12:59	53408
6	2016-04-21	2	1	3	02:00:11	55252
7	2016-04-21	2	1	3	00:13:05	55252
8	2016-04-21	3	1	3	01:33:10	Почта

В таблицах ниже указаны вспомогательные данные, такие как: информация о типах данных (табл. 8), информация о налоговых ставках (табл. 9), информация о ставках (табл. 10). В данных таблицах отображены только основные поля, задействованные в генерации документов.

Таблица 8 — Информация о типах данных для таблицы 7 (TYPES_OF_WORK_LIST)

ID	NAME
1	Техническая поддержка пользователей
2	Тестирование внедряемых функций
3	Консультирование пользователей

Таблица 9 — Информация о налоговых ставках (TAXES_LIST)

ID	VALUE
1	13.0
2	28.0

Таблица 10 — Информация о ставках (RATES_LIST)

ID	VALUE
1	115
2	127
3	285

В результате обработки данных согласно этапам 1-5, описанным в предыдущем разделе (), информация о работе представляется в таблицах для отчета (табл. 11) и акта приемки-сдачи работ (табл. 12).

Таблица 11 — Представление информации для отчета

Дата	Время	Задачи
2016-04-20	04:46:45	55252, 53841, 54312, Почта
2016-04-21	06:59:25	53408, 55252, Почта

Таблица 12 — Представление информации для акта сдачи-приемки работ

Тип	Время	Задачи
Техническая поддержка пользователей	6	53841, 54312, 53408
Тестирование внедряемых функций	3	55252
Консультирование пользователей	4	Почта

Дополнительно также вычисляются следующие данные:

- Количество отработанных часов за период формирования отчета и акта: 13
- Сумма оплаты: 3705
- Сумма оплаты минус налог: 3223.35

2.3 Генератор документов

Для поддержки генерации отчетов и актов разрабатываемое приложение должно поддерживать работу с форматами семейства Open Document Format, в частности — Open Document Text (*.odt). Выбор данного формата обусловлен наличием открытых спецификаций, отраженных в ГОСТ [18], а также поддержкой работы с данным форматом в библиотеке QTextDocument.

Заполнение документов производится на основе шаблонов, предоставленных пользователем, заранее подготовленные для генерации (установлено необходимое оформление, текст и т.п.)

На данный момент проектируется поддержка выгрузки отчетной информации только в табличном виде, примеры такой выгрузки представлены в таблицах 11 и 12. Сама выгрузка осуществляется при помощи библиотеки QTextDocument. Также, для лучшей совместимости генерируемого документа с различными офисными приложениями на различных операционных системных возможна генерация документа в формат PDF при помощи QPrinter[16].

2.3.1 Наименования переменных

Для подстановки данных в документ используются глобальные переменные документа, имеющие тип «Поле пользователя».

Тем самым, данный подход позволяет однозначно идентифицировать необходимые поля для заполнения.

Предполагается поддержка следующих полей:

- ${reportTablePoint} — Точка для начала генерации таблицы отчета.
- ${actTablePoint} — Точка для начала генерации таблицы акта.

Также, для генерации используются следующие дополнительные поля:

- ${totalTime} — Общие трудозатраты.
- ${totalAmount} — Общая сумма.
- ${totalAmountWT} — Общая сумма минус налог.
- ${dateDocGenitive} — Дата генерации документа.

2.3.2 Процесс заполнения документов

Этап 1 — Подготовка документа

В начале генерации приложение делает копию генерируемого документа и указывает в качестве названия дату генерации и тип документа, к примеру для отчета, созданного 2 февраля 2016 года наименование документа будет выглядеть следующим образом: *Report02022016.odt*.

Далее работа производится в скопированном документе — он открывается с помощью QFile, содержимое документа выгружается в QTextDocument.

Этап 2 — Подготовка данных

Процесс подготовки данных выделен в следующий этап, но при реализации, для уменьшения времени генерации документа, он выносится в отдельный поток.

На данном этапе производится заполнение векторов типа TaskListDayRep и TaskListGenRep при помощи модуля сбора статистики, механизм работы которого описан в разделе .

Так как на данный момент выгрузка статистических данных выгружается таблицей, создаются объекты типа QTextTable[17] в которые производится построчная выгрузка данных посредством добавления строк.

Этап 3 — Заполнение шаблона

В документе, созданном на первом этапе, осуществляется поиск тегов, ранее указанных в разделе . При нахождении тега в тексте, его значение считывается, после тега устанавливается курсор и выполняется его удаление. Вместо него размещается значение, соответствующее установленной ранее переменной. При достижении конца файла, он закрывается и на этом формирование документа считается завершенным.

2.4 Управление профилем

В приложении используются специальные конфигурационные файлы, предназначенные для упорядоченного хранения информации о пользователе.

Данный подход позволяет производить быструю настройку подключения к базе данных, а также выступать в роли кэша набора данных из базы, для минимизации обращений к ней. Тем самым файл профиля является дополнением к имеющейся базе данных приложения.

2.4.1 Структура профиля

Профиль представлен в виде XML-файла с расширением lttqt. Содержит в себе несколько разделов:

1. Информация о профиле — содержит теги, хранящие в себе информацию о наименовании профиля, его версии, табельный номер, а также адрес БД и параметры подключения к ней, такие как: тип драйвера, адрес базы и логин к ней. Пароль от базы данных вводится сотрудником при каждом открытии профиля.

2. Информация о пользователе — содержит краткую информацию о сотруднике, а именно — ФИО, должность, наименование компании.

2.4.2 Модуль создания профиля

Создание профиля выполняется в отдельном окне приложения, которое можно вызвать из главного окна. Окно создания профиля содержит минимальный набор полей, необходимый для корректного последующего запуска профиля, а именно — наименование профиля, табельный номер и параметры подключения к базе данных.

Дополнительно, если указано в настройках нового профиля, приложением генерируется база данных SQLite по указанному пользователем адресу в соответствии со структурой, указанной в разделе . Логин и пароль в данном случае не требуются, а ответственность за защиту и достоверность данных (налоговые ставки, ставки по оплате и другие параметры), находящихся в базе, возлагается на самого пользователя приложения.

После создания стартовой версии профиля и успешного подключения к БД пользователю предлагается ввести дополнительные данные самостоятельно, при условии, если в базе данных еще нет информации о данной учетной записи. Ввод дополнительных данных осуществляется через модуль изменения профиля, описанный в разделе .

2.4.3 Модуль изменения профиля

В окне изменения модуля, которое также возможно вызвать из главного окна приложения (при условии, что профиль уже загружен), у пользователя

появляется возможность отредактировать информацию о себе: указать фамилию, имя, отчество, а также свою должность и наименование компании.

Дополнительно к этому на отдельных вкладках окна изменения профиля, пользователю необходимо создать записи для типов работ, указать ставку по оплате и налоговую ставку.

После ввода всех необходимых данных пользователь должен сохранить изменения, тем самым обновив версию своего профиля.

Внесение дополнительной информации в профиль имеет смысл только для сотрудников, использующих локальную версию базы данных и/или использующих приложение для личных целей. Для сотрудников, подключающихся к удаленной базе данных, заполнение информации о пользователе в таблице должно осуществляться сотрудниками кадрового отдела, куратором работника (или непосредственным начальником) или IT-отделом.

2.4.4 Версионность профиля

Профиль пользователя поддерживает механизм версий, цель которых — ограничить возможность внесения изменений в профиль, если он редактируется со стороны предприятия — IT-отдела, непосредственного руководителя (куратора) или же кадрового отдела.

Перед сохранением изменений в профиле, приложение обращается к базе данных, выполняя поиск пользователя в таблице PROFILE для указанного ID. Если такой записи нет, то приложение выполняет поиск по набору полей: фамилия, имя, отчество, должность, компания.

При нахождении записи приложение сравнивает значение параметра profile_ver вышеупомянутой таблицы. Если значение данного параметра в базе данных больше, чем в профиле, то выполяется выгрузка данных из базы данных в профиль, значение параметра версии в профиле также актуализируется.

При создании профиля пользователем, ему присваивается версия **-1**. При создании профиля в базе данных, ей присваивается версия **0**, что позволяет производить синхронизацию при первом подключении к базе данных.

Если такой записи не найдено, то приложение проверяет наличие прав на выгрузку данных в таблицу PROFILE. Список доступных прав указан в таблице SECURITY_SETTINGS, а правила использования данных правил указаны в таблице SECURITY_RULES, где производится связывание ID пользователя с ID конкретного права и указание параметра для каждого такого правила из таблицы. В качестве параметра выступает значение числовое значение от 0 до 1, где 0 — ложь, 1 — истина. Если выгрузка в таблицу разрешена, то приложение вносит изменения в базу и обновляет значение параметров версий как в самом файле, так и в базе данных. Иначе — обновление не производится, тем самым блокируя изменение данных со стороны пользователя.

2.5 Стоимость разработки ПО

Расчет стоимости разработки программного обеспечения производится на основании установленной методики оценки трудоёмкости разработки программных средств[19] и состоит из нескольких этапов: определение количества строк кода, определение коэффициентов и расчета трудозатрат на основании полученных данных.

2.5.1 Определение количества строк кода

На таблице 13 представлен объем строк кода, необходимый для разработки функций программного обеспечения.

Таблица 13 — Количество строк кода

Наименование функции	Значение (стр.)
Реализация пользовательского интерфейса	
Реализация стандартного пользовательского интерфейса (многооконное приложение)	2500
Реализация взаимосвязей систем и компонентов	
Создание связей на основе встроенного механизма сигналов и слотов	50
Взаимодействие с БД/СУБД	
Создание и изменение схемы БД	70
Ведение базы данных (чтение)	10
Ведение базы данных (запись)	20
Реализация прикладных функций	
Статистическая обработка данных	400
Генерация документов	500

Суммарный объем кода разрабатываемого приложения составляет: 3650 строк.

2.5.2 Определение коэффициентов

В соответствии с предоставленными стандартами [19] для приложения были определены коэффициенты, представленные в таблице 14:

Таблица 14 — Перечень коэффициентов

Наименование	Краткое обозначение	Характеристика	Коэффициент
Сложность	Ксложн	Средний	1,0
Степень новизны	Кн	Является развитием определенного параметрического ряда систем, на прежнем типе ЭВМ/ОС	0,7
Надежность	Кнад	Высокий	1,1
Производительность	Кпроизв	Производительность должна обеспечивать приемлемое время отклика.	1,0
Информативность документации	Кдокум	Не учтены многие потребности жизненного цикла	0,81

Продолжение Таблицы 14

Наименование	Краткое обозначение	Характеристика	Коэффициент
Повторное использование компонентов	Кпик	На уровне проекта	1,0
Использование средств управления жизненным циклом	Кср.упр.жиз	Простые CASE средства, интеграция незначительна	1,09
Средство разработки	Кср.разр	C++, IBM-PC совместимый	1,0
Квалификация разработчика	Кквал	Очень низкий	1,62
Опыт разработки	Копыт	Средний (1 год)	1,0
Влияние сроков работ на трудоемкость	Кср	Продолжительность работ, по сравнению с номинальной — 85%	1,14
Нормы времени на разработку программного средства в зависимости от исходного объема	Норм	Объем: 4,0 тыс. строк. № нормы — 17.	302,0

2.5.3 Расчет трудозатрат

На основании данных, полученных в разделах и произведен расчет показателей, значения которых представлены в таблице 15:

Таблица 15 — Расчет показателей

Наименование	Краткое обозначение	Комментарий	Значение
Базовая трудоемкость	Тб	Норм * Ксложн	302,0
Поправочный коэффициент на качество	Ккач	Кнад * Кпроизв * Кдокум * Кпик	1,0
Общая трудоемкость	То	Тб * Кн * Ккач	211,4
Трудоемкость разработки с учетом конкретных условий	Тур	То * Кср.упр.жиз * Кср.разр.	230,4
Трудоемкость с учетом рейтинга разработчика	Тр	Тур * Кквал * Копыт	373,24

Также, вычислена трудоемкость каждой отдельной стадии разработки приложения. С учетом того, что разработка приложения ведется с применением CASE-средств, были получены следующие данные:

Стадия «Анализ разработки»: 149,2 чел.-дн.

Стадия «Проектирование»: 149,2 чел.-дн.

Стадия «Программирование»: 18,67 чел.-дн.

Стадия «Тестирование»: 37,32 чел.-дн.

Стадия «Внедрение»: 18,67 чел.-дн.

В общей сумме разработка приложения займет 373,06 чел.-дн.

Расчет стоимости в денежном эквиваленте не произведен по причине большого количества предложений на рынке компаний-разработчиков ПО, в связи с чем имеется широкий диапазон стоимостных значений.

ЗАКЛЮЧЕНИЕ

Целью данной работы является проектирование и разработка приложения, позволяющего упросить учет временных затрат и автоматизировать составление отчетных документов. Для реализации этой цели были поставлены и достигнуты следующие задачи: составление плана предпроектного обследования, исследование внутренних процессов компании, связанных с предметной областью работы и проектирование структуры и механизмов приложения. Для решения этих задач мною были составлены диаграмма прецедентов для описания необходимого функционала приложения, диаграмма классов — для составления его структуры. Также в виде диаграмм были описаны алгоритмы работы основных процессов приложения. Вместе с приложением было произведено проектирование и создание базы данных, создание структуры профиля и выполнен расчет трудозатрат, необходимых для разработки приложения.

На текущий момент в приложении реализованы следующие функции:

- Графический интерфейс пользователя.
- Создание профиля и выгрузка информации о нем в файл.
- Поддержка баз данных SQLite.
- Учет времени и запись в базу данных с выбором типа работ и указанием комментария.
- Формирование отчетной информации.

На рисунках 10, 11 и 12 представлен внешний вид текущей версии разрабатываемого приложения:

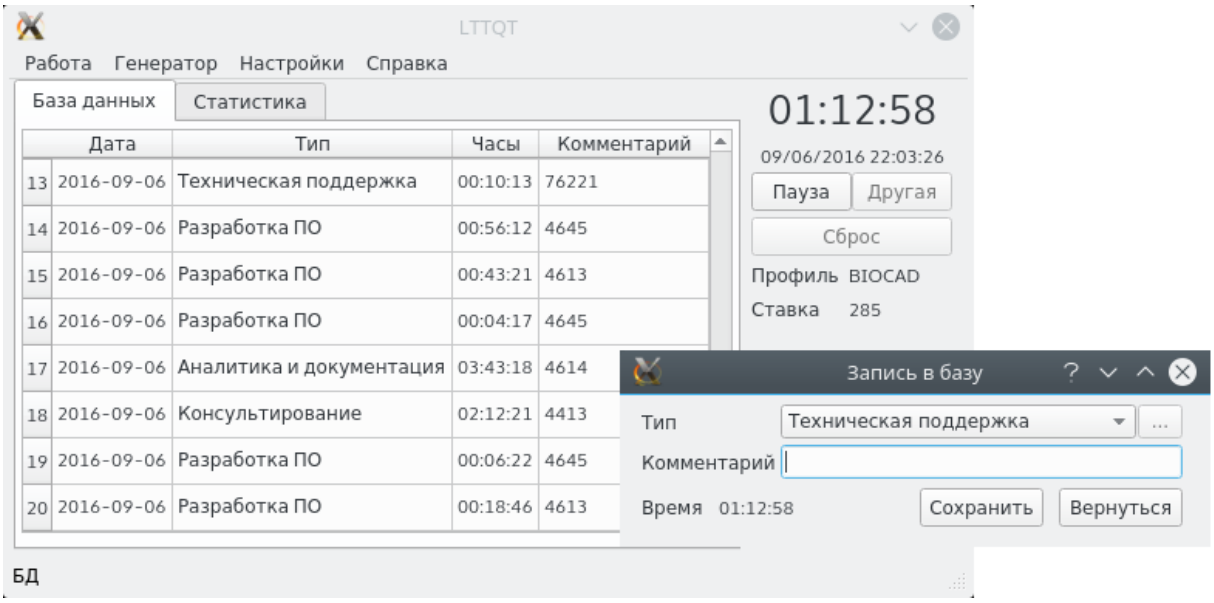


Рисунок 10 Главное окно приложения и диалог сохранения

На рисунке 11 дополнительно отражено наличие проверки на правильность ввода полей — те поля, что необходимы для заполнения, но не заполнены, отмечены красным

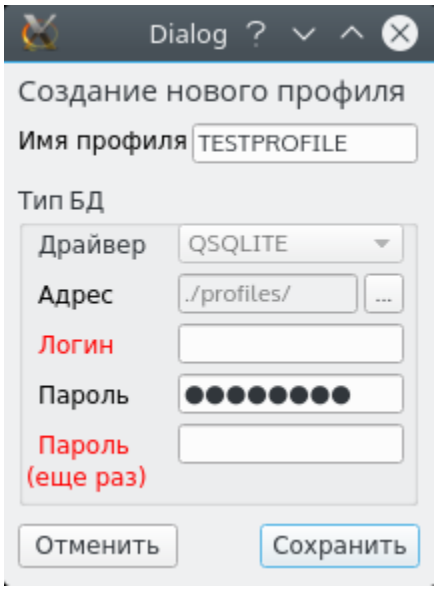


Рисунок 12 Создание профиля

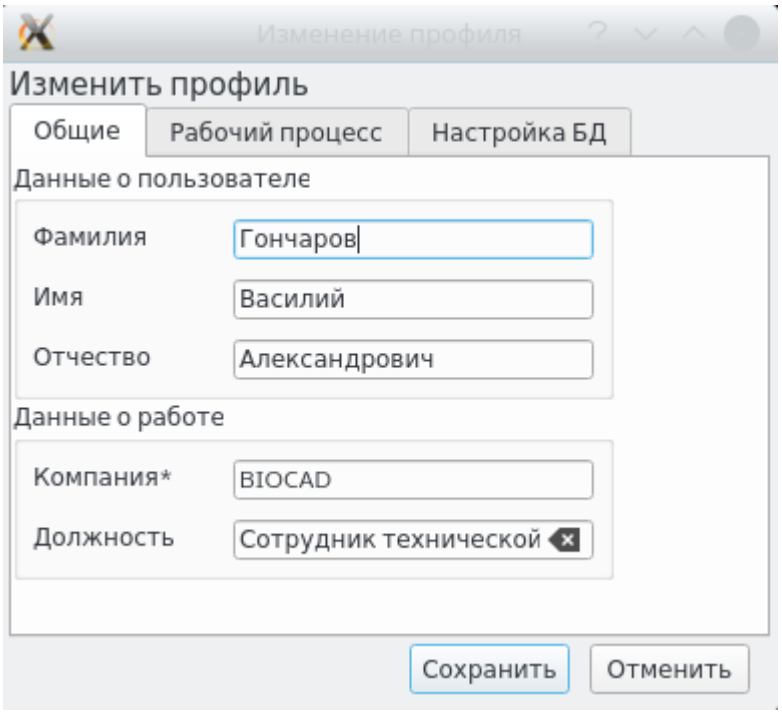


Рисунок 11 Изменение профиля

Версия программного обеспечения, представленная в данной работе, имеет лишь базовый функционал, который будет достаточен для удаленного сотрудника, использующего локальную базу данных. В дальнейшей перспективе возможно использование удаленных БД/СУБД (поддержка которых уже заложена в схему БД, о чем сказано в разделе), которые могут находиться, к примеру, на серверах в самом предприятии. Такой подход позволит централизованно собирать, хранить и обрабатывать информацию и использовать её для генерации различных отчетов, с участием одного или нескольких удаленных сотрудников. Также, переход на удаленные СУБД позволит внедрить следующие функции:

- Модуль генератора статистических данных. В этом модуле руководителям удаленных сотрудников, а также сотрудникам кадрового отдела будет предоставлена возможность просматривать сведения об активности сотрудников, используя фильтры по типам работ, задачам, датам и прочим свойствам.
- Модуль управления сотрудниками. В данном модуле планируется поддержка выгрузки сотрудников из имеющихся систем (ERP, KPI), а также возможность изменения данных о рабочем процессе для каждого пользователя, а именно: добавление/удаление типов работ, значений налога, ставок по оплате.
- Удаленная выгрузка шаблонов документов. Такая выгрузка позволит сотрудникам предприятия, ответственным за формы документов (к примеру это может быть юридический отдел) самостоятельно поддерживать шаблоны документов в актуальном состоянии, размещая их в определенном каталоге, из которого клиентская версия приложения будет запрашивать формы документов. Такая доработка предназначена для снижения вероятности ситуаций, в которых удаленные сотрудники

предоставляют акты сдачи-приемки выполненных работ в устаревшей форме.

- Интеграция с существующими СЭД и трекерами задач. Как было выяснено в анализе существующих систем (), проектируемое приложение будет дополнением к текущим системам учета времени и СЭД. В таком случае, возможно выполнение доработки для интеграции, при котором приложение сможет синхронизировать затраченное время с трекером (используя для идентификации, к примеру, номер задачи), а также автоматически выгружать в СЭД сгенерированный документ и запускать процесс согласования оплаты.
- Дополнение автогенерации другими тегами, которые могут понадобиться для генерации отчета или акта, а также предоставление других видов вывода отчетных данных, кроме табличного, примерный вид которого описан в разделе .

Учитывая тот факт, что распространение приложения, проект которого представлен в данной работе, предполагается под лицензией General Public License (GNU GPL) v.3[20], исходный код программы будет опубликован в открытом доступе. Исходя из этого, все вышеперечисленные варианты доработок приложения, а также любые другие дополнительные — могут быть выполнены самим предприятием по необходимости, а также выбирать любые инструменты для разработки. Единственное условие — соблюдение данной лицензии и «копилефта»[21] — принципа наследования прав, предоставляемых вышеуказанной лицензией.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Фролов, А. 20% россиян перейдут на удаленную работу к 2020 году — J'son & Partners Consulting [Электронный ресурс]: vc.ru. - Электронные текстовые данные .- 17.06.2015.- Режим доступа: https://vc.ru/n/udalenka-2020 . -
2. Redmine — Википедия [Электронный ресурс]: Википедия. - Электронные текстовые данные. - . - Режим доступа: https://ru.wikipedia.org/wiki/Redmine . -
3. Wrike — Википедия [Электронный ресурс]: Википедия. - Электронные текстовые данные. -. - Режим доступа: https://ru.wikipedia.org/wiki/Wrike . -
4. About SQLite [Электронный ресурс]: Официальный сайт SQLite. - Электронные текстовые данные. - . - Режим доступа: https://www.sqlite.org/about.html . -
5. SQLite Copyright [Электронный ресурс]: Официальный сайт SQLite . - Электронные текстовые данные.- .- Режим доступа: https://www.sqlite.org/copyright.html . -
6. UML [Электронный ресурс]: . - Википедия. - . - Режим доступа: https://ru.wikipedia.org/wiki/UML . -
7. QTextDocument Class [Электронный ресурс]: . - Qt Documentation . - . - Режим доступа: http://doc.qt.io/qt-5/qtextdocument.html . -
8. XML [Электронный ресурс]: . - Википедия. - . - Режим доступа: https://ru.wikipedia.org/wiki/XML . -
9. QFile Class [Электронный ресурс]: . - Qt Documentation . - . - Режим доступа: http://doc.qt.io/qt-5/qfile.html . -
10. The SAX interface [Электронный ресурс]: . - Qt Documentation . - . - Режим доступа: http://doc.qt.io/qt-5/xml-sax.html . -

11. QDomDocument Class [Электронный ресурс]: . - Qt Documentation . - . - Режим доступа: http://doc.qt.io/qt-5/qdomdocument.html . -
12. QSqlDatabase Class [Электронный ресурс]: . - Qt Documentation . - . - Режим доступа: http://doc.qt.io/qt-5/qsqldatabase.html . -
13. QSettings Class [Электронный ресурс]: . - Qt Documentation . - . - Режим доступа: http://doc.qt.io/qt-5/qsettings.html . -
14. QSQLQuery Class [Электронный ресурс]: . - Qt Documentation . - . - Режим доступа: http://doc.qt.io/qt-5/qsqlquery.html . -
15. QStringList Class [Электронный ресурс]: . - Qt Documentation . - . - Режим доступа: http://doc.qt.io/qt-5/qstringlist.html . -
16. QPrinter Class [Электронный ресурс]: . - Qt Documentation . - . - Режим доступа: http://doc.qt.io/qt-5/qprinter.html . -
17. QTextTable Class [Электронный ресурс]: . - Qt Documentation . - . - Режим доступа: http://doc.qt.io/qt-5/qtexttable.html . -
18. ГОСТ Р ИСО/МЭК 26300-2010: Информационная технология. Формат Open Document для офисных приложений (OpenDocument) v 1.0 . - Москва: Стандартинформ, 2011. - 893 с.
19. Котов С.Л. Разработка, стандартизация и сертификация программных средств и информационных технологий и систем [Текст]: учебное пособие / С.Л. Котов, Б.В. Палюх, С.Л. Федченко. - Тверь: ТГТУ, 2006. - 104 с.
20. The GNU General Public License v 3.0 [Электронный ресурс]: . - The GNU Project. - . - Режим доступа: http://www.gnu.org/licenses/gpl . -
21. Копилефт [Электронный ресурс]: . - Википедия. - . - Режим доступа: https://ru.wikipedia.org/wiki/Копилефт . -

Приложение 1. Исходный код класса MainWindow

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QPushButton>
#include <QTableView>
#include <QTimer>
#include <QDateTime>
#include <QFile>
#include <QSettings>
#include <QFileDialog>
#include <QMessageBox>
#include <QXmlStreamReader>
#include <QDebug>
#include <stopwatchcontroller.h>
#include <calcmodule.h>
#include <system_settings.h>
#include <settingswindow.h>
#include <editprofilewindow.h>
#include <savetimedialog.h>
#include <createprofilewindow.h>
#include <names_of_tables.h>
#include <stataggregator.h>
namespace Ui {
class MainWindow;
}
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
signals:
    void imChange();
private:
    Ui::MainWindow *ui;
    QTimer *stopwatch_timer;
    QTimer *current_datetime_timer;
    StopwatchController *stopwatch;
    QSettings *settings;
    QLabel *db_status;
    QSqlDatabase database;
    QFile profile_store;
    QSqlQueryModel *mainModel;
    CalcModule *clcmdl;
    QFile *profile;
     StatAggregator *st;
private slots:
    void initialization();
    void models_prepare();
    void refresh_view();
```

Приложение 1. Исходный код класса MainWindow

```
    void check_db_connection();
    void update_stopwatch_time();
    void update_current_datetime();
    void start_stop();
    void create_new_profile_window_open();
    void check_display();
    void settingswindow_open();
    void load_settings();
    void update_info_block();
    void reset_timer();
    void load_database(QString driver_type, QString db_addr,
QString login, QString passwd);
    void close_database();
    void read_profile(QString addr);
    void open_profile();
    void savetimedialog_open();
    void edit_current_profile_window_open();
    void on_menu_profile_open_triggered();
    void on_menu_profile_create_new_triggered();
    void on_menu_open_settings_triggered();
    void on_startstop_button_released();
    void on_menu_profile_change_save_triggered();
    void on_reset_button_released();
    void on_menu_close_profile_triggered();
    void on_STAG_TEST_clicked();
    void test_round_val();
    void test_add_hour();
    void test_getListOfTypesFromDB();
    void on_OBD_TEST_clicked();
    void test_openDB();
    void on_TCL_TEST_clicked();
    void test_calc_time();
};
#endif // MAINWINDOW_H
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    initialization();
    load_settings();
    check_db_connection();
    check_display();
}
void MainWindow::initialization() {
    settings = new QSettings(CONFIG_PATH, QSettings::IniFormat);
    mainModel = new QSqlQueryModel();
    stopwatch = new StopwatchController();
    stopwatch_timer = new QTimer();
    current_datetime_timer = new QTimer();
```

Приложение 1. Исходный код класса MainWindow

```
            db_status = new QLabel;
            clcmdl = new CalcModule();
            profile = new QFile();
            connect(this, SIGNAL(imChange()), this,
SLOT(check_db_connection()));
            connect(this, SIGNAL(imChange()), this,
SLOT(refresh_view()));
            connect(current_datetime_timer, SIGNAL(timeout()), this,
SLOT(update_current_datetime()));
            stopwatch_timer->setInterval(1000); // issue #6
            connect(stopwatch_timer, SIGNAL(timeout()), this,
SLOT(update_stopwatch_time())); // issue #6 // О б н о в л е н и е
            statusBar()->insertWidget(0, db_status, 0);
            db_status->setText("Б Д");
            ui->menu_profile_change_save->setEnabled(false);
            ui->startstop_button->setEnabled(false);
            ui->reset_button->setEnabled(false);
            ui->other_task_button->setEnabled(false);
            ui->db_view-
>setSelectionBehavior(QAbstractItemView::SelectRows);
            ui->db_view->horizontalHeader()->setStretchLastSection(true);
            ui->db_view-
>setSelectionMode(QAbstractItemView::SingleSelection);
        }
        void MainWindow::models_prepare()
        {
            mainModel->setQuery("SELECT date, (SELECT name FROM
"+QString(TB_TYPE_OF_WORK)+" WHERE id = type_id), time, comment FROM
"+QString(TB_CURRENT));
            mainModel->setHeaderData(0, Qt::Horizontal,
QObject::tr("Д а т а"));
            mainModel->setHeaderData(1, Qt::Horizontal,
QObject::tr("Т и п"));
            mainModel->setHeaderData(2, Qt::Horizontal,
QObject::tr("Ч а с ы"));
            mainModel->setHeaderData(3, Qt::Horizontal,
QObject::tr("К о м м е н т а р и й"));
            if (mainModel->lastError().isValid()) {
                qDebug() << "[MainWindow]:" << mainModel->lastError();
            }

        }
        void MainWindow::refresh_view()
        {
            if (database.isOpen())
            {
                ui->startstop_button->setEnabled(true);
            } else {
                mainModel->clear();
                ui->startstop_button->setEnabled(false);
            }
            mainModel->setQuery(mainModel->query().lastQuery());
```

Приложение 1. Исходный код класса MainWindow

```
        ui->db_view->resizeColumnsToContents();
    }
    void MainWindow::load_settings()
    {
        if (settings->value(CONFIG_SHOW_SYSTEM_CLOCK,
false).toBool()){
            ui->system_clock->setVisible(true);
            ui->stopwatch->setGeometry(480, 0, 131, 41);
            current_datetime_timer->start();
        } else {
            ui->system_clock->setVisible(false);
            ui->stopwatch->setGeometry(480, 15, 131, 41);
            current_datetime_timer->stop();
        }
    }
    void MainWindow::check_db_connection()
    {
        if (database.isOpen()) {
            qDebug() << "[MainWindow]: БД тут" << &database;
            db_status->setStyleSheet(MESSAGE_DB_IS_CONNECTED);
            ui->menu_profile_change_save->setEnabled(true);
        } else {
            db_status->setStyleSheet(MESSAGE_DB_ISNT_CONNECTED);
            ui->menu_profile_change_save->setEnabled(false);
            ui->startstop_button->setEnabled(false);
            ui->reset_button->setEnabled(false);
        }
    }
    void MainWindow::check_display()
    {
        if (stopwatch_timer->isActive()) {
            ui->startstop_button->setText("Пауза");
            ui->reset_button->setEnabled(false);
        } else {
            ui->startstop_button->setText("Старт");
            if (ui->stopwatch->text() != "00:00:00") {
                ui->reset_button->setEnabled(true);
            } else {
                ui->reset_button->setEnabled(false);
            }
        }
    }
    void MainWindow::update_stopwatch_time()
    {
        stopwatch->inc_time();
        ui->stopwatch->setText(stopwatch->get_string_time());
    }
    void MainWindow::update_current_datetime()
    {
        ui->system_clock-
>setText(QDateTime::currentDateTime().toString("dd/MM/yyyy hh:mm:ss"));
    }
```

Приложение 1. Исходный код класса MainWindow

```
void MainWindow::start_stop()
{
    if (stopwatch_timer->isActive()) {
        stopwatch_timer->stop();
        savetimedialog_open();
    } else {
        stopwatch_timer->start();
    }
    check_display();
}
void MainWindow::savetimedialog_open()
{
    SaveTimeDialog *dia = new SaveTimeDialog(stopwatch->get_string_time(), database);

    connect(dia, SIGNAL(good_transaction()), this, SLOT(refresh_view())); // issue #17
    if (dia->exec() == 0)
    {
        delete dia;
    }}
void MainWindow::settingswindow_open()
{
    SettingsWindow *set = new SettingsWindow();
    connect(set, SIGNAL(accepted()), this, SLOT(load_settings())); // issue #17
    if (set->exec() == 0)
    {
        delete set;
    }
}
void MainWindow::create_new_profile_window_open()
{
    CreateProfileWindow *create_new_profile_window = new CreateProfileWindow();
    if (create_new_profile_window->exec() == 0)
    {
        delete create_new_profile_window;
    }}
void MainWindow::edit_current_profile_window_open()
{
    EditProfileWindow *edit_profile_window = new EditProfileWindow(profile);
    if (edit_profile_window->exec() == 0)
    {
        delete edit_profile_window;
    }}
void MainWindow::reset_timer()
{
    QMessageBox confirm_reset;
    confirm_reset.setWindowTitle("В о п р о с !");
    confirm_reset.setIcon(QMessageBox::Question);
```

Приложение 1. Исходный код класса MainWindow

```
            confirm_reset.setText("Действительно сбросить таймер?");
            confirm_reset.setStandardButtons(QMessageBox::Yes | QMessageBox::No);
            confirm_reset.setDefaultButton(QMessageBox::No);
            confirm_reset.setEscapeButton(QMessageBox::No);
            int ret = confirm_reset.exec();
            if (ret == QMessageBox::Yes) {
                stopwatch->set_time(0,0,0,0);
                ui->stopwatch->setText(stopwatch->get_string_time());
                check_display();
            }}
        void MainWindow::open_profile()
        {
            qDebug() << "[MainWindow]: Начинаю открытие профиля";
            QString addr = QFileDialog::getOpenFileName(0, "Открыть профиль", PROFILE_DEFAULT_PROFILE_FILE_PLACEMENT, "*.lttqt");

            read_profile(addr);}
        void MainWindow::read_profile(QString addr)
        {
            qDebug() << "[MainWindow]: Пробую прочитать" << addr;
            profile->setFileName(addr);
            if (database.isOpen()) {
                close_database();}
            QXmlStreamReader xr;
            xr.setDevice(profile);
            xr.readNext();
            QString driver_type, db_addr, login, passwd;
            while (!xr.atEnd())
            {
                if (xr.isStartElement())
                {
                    if (xr.name() == PROFILE_DB_ADDR_ELEMENT)
                    {
                        foreach (const QXmlStreamAttribute &attr, xr.attributes())
                        {
                            if (attr.name() == PROFILE_DB_ADDR_ATTR_TYPEDB)
                            {driver_type = attr.value().toString();}
                            if (attr.name() == PROFILE_DB_ADDR_ATTR_LOGIN)
                            {login = attr.value().toString();}
                            if (attr.name() == PROFILE_DB_ADDR_ATTR_PASSWORD)
                            {passwd = attr.value().toString();}}
                        db_addr = xr.readElementText();
                    }}
```

Приложение 1. Исходный код класса MainWindow

```
            xr.readNext();}
        load_database(driver_type, db_addr, login, passwd);}
    void MainWindow::load_database(QString driver_type, QString
db_addr, QString login, QString passwd)
    {
        database = QSqlDatabase::addDatabase(driver_type);
        database.setDatabaseName(db_addr);
        if (database.open()) {
            emit imChange();
            models_prepare();
            ui->db_view->setModel(mainModel);
            ui->db_view->show();
            refresh_view();
        }}
    void MainWindow::close_database()
    {
        database.close();
        check_db_connection();
        refresh_view();
    }
    void MainWindow::on_startstop_button_released()
    {start_stop();}
    void MainWindow::on_menu_open_settings_triggered()
    {settingswindow_open();}
    void MainWindow::on_menu_profile_open_triggered()
    {open_profile();}
void MainWindow::on_menu_profile_create_new_triggered()
{create_new_profile_window_open();}
void MainWindow::on_menu_profile_change_save_triggered()
{edit_current_profile_window_open();}
void MainWindow::on_reset_button_released()
{reset_timer();}
void MainWindow::on_menu_close_profile_triggered()
{close_database();}
void MainWindow::on_STAG_TEST_clicked()
{    test_getListOfTypesFromDB();}
void MainWindow::on_OBD_TEST_clicked()
{test_openDB();}
void MainWindow::on_TCL_TEST_clicked()
{test_calc_time();}
void MainWindow::test_round_val(){
    stopwatch->set_time(1, 10, 5, 1);
    ui->stopwatch->setText(stopwatch->get_string_time());
    update_info_block();
}
void MainWindow::test_add_hour(){
    stopwatch->set_time(0, 45, 0, 0);
    ui->stopwatch->setText(stopwatch->get_string_time());
    update_info_block();
}
void MainWindow::test_getListOfTypesFromDB()
{
```

Приложение 1. Исходный код класса MainWindow

```
        read_profile("./chroot/profiles/lanman.lttqt");
        QDate date_start = QDate::fromString("2016-04-20",
CONFIG_DB_DATE_MASK);
        QDate date_end = QDate::fromString("2016-06-09",
CONFIG_DB_DATE_MASK);
        st = new StatAggregator(date_start, date_end);
        test_calc_time();
        delete st;
    }
    void MainWindow::test_calc_time()
    {
        st->getGeneralReport();
        qDebug() << "[MainWindow]: Всего времени за
период: " << QString::number(st->getTotalTime());
    }
    void MainWindow::test_openDB()
    {
        read_profile("./chroot/profiles/lanman.lttqt");
    }
    MainWindow::~MainWindow()
    {
        if (stopwatch_timer->isActive()) {
            stopwatch_timer->stop();
        }
        delete db_status;
        delete stopwatch;
        delete current_datetime_timer;
        delete clcmdl;
        delete profile;
        delete stopwatch_timer;
        delete mainModel;
        delete settings;
        delete ui;}
```

Приложение 2. Исходный код класса StatAggregator

```
#ifndef STATAGGREGATOR_H
#define STATAGGREGATOR_H
#include <QObject>
#include <QSqlRecord>
#include <QSqlQuery>
#include <QSqlError>
#include <QDebug>
#include <calcmodule.h>
#include <system_settings.h>
#include <calcmodule.h>
#include <generator_outtype.h>
#include <names_of_tables.h>
class StatAggregator : public QObject
{
    Q_OBJECT
public:
    explicit StatAggregator(QDate dstart, QDate dend, QObject
*parent = 0);
    ~StatAggregator();
    TaskListGenRep getGeneralReport();
    TaskListDayRep getDaysReport();
    int getTotalTime();
signals:
public slots:
private:
CalcModule clcmdl;
QDate dateStart;
QDate dateEnd;
QStringList typesInRange;
QStringList datesInRange;
QSqlQuery rangeQuery;
TaskListGenRep general_report;
TaskListDayRep days_report;
TaskListGenRep generateGeneralReport();
TaskListDayRep generateDaysReport();
void getGeneralTaskList();
void getDaysTasksList();
int getRangeDates();
QTime getTimeForType(QString &type);
QTime getTimeForDate(QString &date);
void loadQuery();
void loadRangeInfo();
void loadDatesAndTypesFromDB();
void loadTasksForType();
void loadTaskTypesFromList();
void loadTasksForDate();
void loadDatesFromList();
void calculateTypeTime();
void calculateDayTime();
int calculateTotalTime();
```

Приложение 2. Исходный код класса StatAggregator

```
        void prepareVectors();
        void sortNames();
        void resetPositionQuery();
    };
    #endif // STATAGGREGATOR_H
    #include "stataggregator.h"
    StatAggregator::StatAggregator(QDate dstart, QDate dend, QObject
*parent) : QObject(parent)
    {
        dateStart = dstart;
        dateEnd = dend;
        loadQuery();}
    TaskListGenRep StatAggregator::getGeneralReport()
    {return general_report;}
    TaskListDayRep StatAggregator::getDaysReport()
    {return days_report;}
    int StatAggregator::getTotalTime()
    {return calculateTotalTime();}
    int StatAggregator::getRangeDates()
    {
        int res = dateStart.daysTo(dateEnd);
        if (res == 0)
        {
            res = 1;
        }
        else if (res <= -1)
        {
            res = dateEnd.daysTo(dateStart);
        }
        return res;
    }
    void StatAggregator::getGeneralTaskList()
    {
        loadTaskTypesFromList();
        loadTasksForType();
        calculateTypeTime();
    }
    void StatAggregator::getDaysTasksList()
    {
        loadDatesFromList();
        loadTasksForDate();
        calculateDayTime();
    }
    QTime StatAggregator::getTimeForType(QString &type)
    {
        resetPositionQuery();
        QTime tmp_time;
        tmp_time.setHMS(0, 0, 0);
        do
        {
            if (rangeQuery.record().value(2).toString() == type)
            {
```

Приложение 2. Исходный код класса StatAggregator

```
                clcmdl.appendTime(tmp_time,
rangeQuery.record().value(1).toTime());
            }
        } while (rangeQuery.next());
        return tmp_time;
    }
    QTime StatAggregator::getTimeForDate(QString &date)
    {
        resetPositionQuery();

        QTime tmp_time;
        tmp_time.setHMS(0, 0, 0);
        do
        {
            if (rangeQuery.record().value(0).toString() == date)
            {
                clcmdl.appendTime(tmp_time,
rangeQuery.record().value(1).toTime());
            }
        } while (rangeQuery.next());
        return tmp_time;
    }
    void StatAggregator::loadQuery()
    {
        loadRangeInfo(); // Запрос в БД
        loadDatesAndTypesFromDB();
        general_report.grri.resize(typesInRange.size());
        days_report.drri.resize(datesInRange.size());
        prepareVectors();
    }
    void StatAggregator::loadRangeInfo()
    {
        rangeQuery.prepare("SELECT date, time, (SELECT name FROM
"+QString(TB_TYPE_OF_WORK)+" WHERE id = type_id), comment, rate_id,
tax_id FROM "+QString(TB_CURRENT)+" WHERE date BETWEEN :dateStart AND
:dateEnd");
        rangeQuery.bindValue(":dateStart", dateStart);
        rangeQuery.bindValue(":dateEnd", dateEnd);
        if (!rangeQuery.exec())
        {
            qDebug() << "[StatAggregator]: Ошибка в
запросе?";
            qDebug() << "[StatAggregator]:" <<
rangeQuery.lastError();
            qDebug() << "[StatAggregator]:" <<
rangeQuery.lastQuery();
        } else {
            qDebug() << "[StatAggregator]:" <<
rangeQuery.lastQuery();
        }}
    void StatAggregator::loadTasksForType()
    {
```

Приложение 2. Исходный код класса StatAggregator

```
        for (int i = 0; i < general_report.grri.size(); ++i)
        {
            resetPositionQuery();
            do
            {
                if (rangeQuery.record().value(2).toString() ==
general_report.grri[i].typeName) {
general_report.grri[i].tasksInType.append(rangeQuery.record().value(3).t
oString());}
            } while (rangeQuery.next());}
        for (int i = 0; i < general_report.grri.size(); ++i)
        {
            general_report.grri[i].tasksInType.removeDuplicates();
        }
    }

    void StatAggregator::loadTasksForDate()
    {
       for (int i = 0; i < days_report.drri.size(); ++i)
        {
            resetPositionQuery();
            do{
                if (rangeQuery.record().value(0).toString() ==
days_report.drri[i].date) {
days_report.drri[i].tasksInDay.append(rangeQuery.record().value(3).toStr
ing());}} while (rangeQuery.next());}
        for (int i = 0; i < days_report.drri.size(); ++i)
        {
            days_report.drri[i].tasksInDay.removeDuplicates();
        }}
    void StatAggregator::loadDatesAndTypesFromDB()
    {
        resetPositionQuery();
        do {
           typesInRange.insert(typesInRange.size(),
rangeQuery.value(2).toString());
           datesInRange.insert(datesInRange.size(),
rangeQuery.value(0).toString());
        } while (rangeQuery.next());
        typesInRange.removeDuplicates();
        datesInRange.removeDuplicates();
    }
    void StatAggregator::loadTaskTypesFromList()
    {
        for (int i = 0; i < typesInRange.size(); ++i)
        {
            general_report.grri[i].typeName = typesInRange.value(i);
        }}
    void StatAggregator::loadDatesFromList()
    {
        for (int i = 0; i < datesInRange.size(); ++i)
        {
```

Приложение 2. Исходный код класса StatAggregator

```
            days_report.drri[i].date = datesInRange.value(i);
        }}
    int StatAggregator::calculateTotalTime()
    {
        QTime totalTime;
        totalTime.setHMS(0, 0, 0);
        for (int i = 0; i < general_report.grri.size(); ++i)
        {
            clcmdl.appendTime(totalTime,
QTime::fromString(general_report.grri[i].totalForTypeTime,
CONFIG_STRUCT_TIME_FORMAT));
        }
        return clcmdl.getRoundedTime(totalTime);
    }
    void StatAggregator::calculateTypeTime()
    {
       for (int i = 0; i < general_report.grri.size(); ++i)
        {
            general_report.grri[i].totalForTypeTime =
getTimeForType(general_report.grri[i].typeName).toString(CONFIG_STRUCT_T
IME_FORMAT);
        }
    }
    void StatAggregator::calculateDayTime()
    {
         for (int i = 0; i < days_report.drri.size(); ++i)
        {
            days_report.drri[i].totalForDayTime =
getTimeForDate(days_report.drri[i].date).toString(CONFIG_STRUCT_TIME_FOR
MAT);}}
    void StatAggregator::prepareVectors()
    {
        getGeneralTaskList();
        getDaysTasksList();
    }
    void StatAggregator::resetPositionQuery()
    {
        if (!rangeQuery.at() == 0)
        {
            rangeQuery.first();
        }}
    StatAggregator::~StatAggregator()
    {
        days_report.drri.clear();
        days_report.drri.squeeze();
        general_report.grri.clear();
        general_report.grri.squeeze();
        typesInRange.clear();
    }
```

Приложение 3. Исходный код класса StopwatchController

```
#indef STOPWATCH_AUTO_H
#define STOPWATCH_AUTO_H
#pragma once
#include <QString>
#include <QTime>
#include <QDebug>
class StopwatchController
{
public:
    StopwatchController();
    ~StopwatchController();
    void inc_time();
    void set_time(int, int, int, int);
    QString get_string_time();
    int get_seconds();
    int get_minutes();
    int get_hour();
private:
    QTime *stopwatch;
};
#endif // STOPWATCHAUTO_H
#include "stopwatchcontroller.h"
StopwatchController::StopwatchController()
{
    stopwatch = new QTime;
    stopwatch->setHMS(0, 0, 0); }
void StopwatchController::inc_time()
{*stopwatch = stopwatch->addSecs(1);}
QString StopwatchController::get_string_time()
{return stopwatch->toString("hh:mm:ss");}
int StopwatchController::get_seconds()
{return stopwatch->second();}
int StopwatchController::get_minutes()
{return stopwatch->minute();}
int StopwatchController::get_hour()
{return stopwatch->hour();}
void StopwatchController::set_time(int h, int m, int s, int ms)
{stopwatch->setHMS(h, m, s, ms);}
StopwatchController::~StopwatchController()
{delete stopwatch;}
```

Приложение 4. Исходный код класса CalcModule

```
#indef CALCMODULE_H
#define CALCMODULE_H
#include <QDebug>
#include <QDate>
#include "system_settings.h"
#include "stopwatchcontroller.h"
class CalcModule
{
public:
    CalcModule();
    ~CalcModule();
    int getRoundedTime(QTime &resT);
    void appendTime(QTime &t1, QTime t2);
    int getTimeInSec(StopwatchController &stopwatch);
    int getHourFromTime(int);
    int getMinutesFromTime(int);
    int getSecondsFromTime(int);
    int getRangeDates(QString startDate, QString endDate);
    QString getTimeFromSecToString(int);
};
#endif // CALCMODULE_H
#include "calcmodule.h"
CalcModule::CalcModule()
{}
int CalcModule::getRoundedTime(QTime &resT)
{
    int h = resT.hour();
    int m = resT.minute();
    int s = resT.second();
    if (s >= 30)
    { m += 1; s = 0;}
    if (m >= 10)
    {h += 1;m = 0;}
    return h;
}
void CalcModule::appendTime(QTime &t1, QTime t2)
{
    int h = t1.hour() + t2.hour();
    int m = t1.minute() + t2.minute();
    int s = t1.second() + t2.second();
    if (s >= 60)
    {m += 1;s -= 60;}
    if (m >= 60)
    {h += 1;m -= 60;}
    t1.setHMS(h, m, s);}
int CalcModule::getTimeInSec(StopwatchController& stopwatch)
{
    int hour = stopwatch.get_hour();
    int min = stopwatch.get_minutes();
    int sec = stopwatch.get_seconds();
```

Приложение 5. Исходный код класса CalcModule

```
            sec += (hour*60*60) + (min*60);
            return sec;}
        int CalcModule::getHourFromTime(int time)
        {return time / 3600;}
        int CalcModule::getMinutesFromTime(int time)
        {return (time - getHourFromTime(time)*3600) / 60;}
        int CalcModule::getSecondsFromTime(int time)
        {return time - getHourFromTime(time)*3600 -
getMinutesFromTime(time)*60;}
        QString CalcModule::getTimeFromSecToString(int time){return
QString::number(getHourFromTime(time))+":"+QString::number(getMinutesFro
mTime(time))+":"+QString::number(getSecondsFromTime(time));}
        CalcModule::~CalcModule(){}
```

Приложение 5. Исходный код класса CreateProfileWindow

```
#indef CREATEPROFILEWINDOW_H
#define CREATEPROFILEWINDOW_H
#include <QDialog>
#include <QDebug>
#include <QFile>
#include <QMessageBox>
#include <QXmlStreamWriter>
#include "system_settings.h"
#include "xml_files_headers.h"
namespace Ui {class CreateProfileWindow;}
class CreateProfileWindow : public QDialog
{    Q_OBJECT
public:
    explicit CreateProfileWindow(QWidget *parent = 0);
    ~CreateProfileWindow();
private slots:
    void on_cancel_button_clicked();
    void on_save_button_clicked();
private:
    Ui::CreateProfileWindow *ui;
    void generate_file();
    int confirm_cancel();
    void creating_nfo();
    QString generate_name_for_db(QString profile_name);
    void general_check_before_generate();
    void state_change(bool state, QLabel &obj);
    bool check_for_profile_name(QString &filename);
    bool check_for_correct_passes();
    bool check_for_correct_fields_at_all()};
#endif // CREATEPROFILEWINDOW_H
#include "createprofilewindow.h"
#include "ui_createprofilewindow.h"
CreateProfileWindow::CreateProfileWindow(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::CreateProfileWindow)
{    ui->setupUi(this);
    ui->addr_input-
>setText(PROFILE_DEFAULT_PROFILE_FILE_PLACEMENT);
    ui->addr_input->setEnabled(false);}
void CreateProfileWindow::generate_file()
{    QFile fl;
    QXmlStreamWriter xw;
    fl.setFileName(PROFILE_DEFAULT_PROFILE_FILE_PLACEMENT+ui-
>profile_input->text()+PROFILE_FILEEXT);
    QString tmp = fl.fileName();
    bool isReadyForGenerate = true;
    if (!fl.open(QIODevice::WriteOnly))
    {
        isReadyForGenerate = false;
    }
```

Приложение 5. Исходный код класса CreateProfileWindow

```
        if (isReadyForGenerate)
        {
            xw.setDevice(&fl);
            xw.setAutoFormatting(true);
            xw.writeStartDocument();
            xw.writeStartElement(HEADER_PROFILE_BLOCK);
            xw.writeStartElement(PROFILE_NAME_ELEMENT);
                xw.writeCharacters(ui->profile_input->text());
            xw.writeEndElement();
            xw.writeStartElement(PROFILE_VER_ELEMENT);
                xw.writeCharacters("-1");
            xw.writeEndElement();
            xw.writeStartElement(PROFILE_DB_ADDR_ELEMENT);
                xw.writeAttribute(PROFILE_DB_ADDR_ATTR_TYPEDB, ui->driver_selecter->currentText());
                xw.writeAttribute(PROFILE_DB_ADDR_ATTR_LOGIN, ui->login_input->text());
                xw.writeAttribute(PROFILE_DB_ADDR_ATTR_PASSWORD, ui->pass_input->text());
                if (ui->driver_selecter->currentText() == "QSQLITE")
                {xw.writeCharacters(generate_name_for_db(ui->profile_input->text()));
                } else {xw.writeCharacters(ui->addr_input->text());}
            xw.writeEndElement();
            xw.writeStartElement(HEADER_PROFILE_USER_INFO);
                xw.writeStartElement(PROFILE_USER_INFO_FIRST_NAME);
                xw.writeEndElement();
                xw.writeStartElement(PROFILE_USER_INFO_LAST_NAME);
                xw.writeEndElement();
                xw.writeStartElement(PROFILE_USER_INFO_MIDDLE_NAME);
                xw.writeEndElement();
            xw.writeEndElement();
            xw.writeEndDocument();
            fl.close();
            creating_nfo();
        }
    }
    QString CreateProfileWindow::generate_name_for_db(QString profile_name){
        return PROFILE_DEFAULT_DATABASE_PLACEMENT+profile_name+PROFILE_DBEXT;
    }
    void CreateProfileWindow::general_check_before_generate()
    {
        if (!check_for_correct_fields_at_all())
        {
        } else {
            if (!check_for_correct_passes())
            {
                QMessageBox err_window;
```

Приложение 5. Исходный код класса CreateProfileWindow

```
                    err_window.setWindowTitle("Пароли не
совпадают");
err_window.setIcon(QMessageBox::Warning);
                    err_window.setText("Введенные
пароли не совпадают");
err_window.setInformativeText("Проверьте ввод и
попробуйте снова");
err_window.setStandardButtons(QMessageBox::Ok);
                    err_window.setDefaultButton(QMessageBox::Ok);
                    err_window.exec();
                } else {generate_file();}}}
    bool CreateProfileWindow::check_for_correct_passes()
    {
        if (ui->pass_input->text() != ui->pass_again_input->text())
        {return false;} else {return true;}}
    bool CreateProfileWindow::check_for_correct_fields_at_all()
    {bool stat = true;
        if (ui->profile_input->text() == "")
        {
            stat = false;
            state_change(false, *ui->profile_name);
        } else {
            state_change(true, *ui->profile_name);}
        if (ui->login_input->text() == "")
        {
            stat = false;
            state_change(false, *ui->login_name);
        } else {
            state_change(true, *ui->login_name);
        }
        if (ui->addr_input->text() == "")
        {
            stat = false;
            state_change(false, *ui->addr_name);
        } else {
            state_change(true, *ui->addr_name);
        }
        if (ui->pass_input->text() == "")
        {
            stat = false;
            state_change(false, *ui->pass_name);
        } else  {
            state_change(true, *ui->pass_name);
        }
        if (ui->pass_again_input->text() == "")
        {
            stat = false;
            state_change(false, *ui->passwd_again_name);
            state_change(false, *ui->passwd_again_name_tip);
        } else {
            state_change(true, *ui->passwd_again_name);
            state_change(true, *ui->passwd_again_name_tip);
```

Приложение 5. Исходный код класса CreateProfileWindow

```
        }
        return stat;
    }
    void CreateProfileWindow::state_change(bool state, QLabel &obj)
    {
        if (state)
        {
            obj.setStyleSheet(MESSAGE_GENERAL_COLOR);
        } else {
            obj.setStyleSheet(MESSAGE_GENERAL_ERR);
        }}
    bool CreateProfileWindow::check_for_profile_name(QString
&filename)
    {
        QFile t;
        t.setFileName(filename);
        if (t.exists())
        {
            QMessageBox err_window;
            err_window.setWindowTitle("Ошибка!");
            err_window.setIcon(QMessageBox::Warning);
            err_window.setText("Файл "+ui->profile_input-
>text()+" уже есть в "+ui->addr_input->text());
            err_window.setInformativeText("Выберите
другое имя и попробуйте снова");
            err_window.setStandardButtons(QMessageBox::Ok);
            err_window.setDefaultButton(QMessageBox::Ok);
            err_window.exec();
            return false;
        } else {return true;}}
    int CreateProfileWindow::confirm_cancel()
    {
        QMessageBox really;
        really.setWindowTitle("Вопрос!");
        really.setIcon(QMessageBox::Question);
        really.setText("Создание профиля в
процессе");
        really.setInformativeText("Действительно
хотите прервать его?");
        really.setStandardButtons(QMessageBox::Yes |
QMessageBox::No);
        really.setDefaultButton(QMessageBox::No);
        really.setEscapeButton(QMessageBox::No);
          int ret = really.exec();
          if (ret == QMessageBox::Yes) {
              CreateProfileWindow::close();
              return 0;
          }
          else{}
      return 0;
    }
    void CreateProfileWindow::creating_nfo()
```

Приложение 5. Исходный код класса CreateProfileWindow

```
{
    QMessageBox info_window;
    info_window.setWindowTitle("Создание профиля");
    info_window.setIcon(QMessageBox::Information);
    info_window.setText("Статус профиля");
    info_window.setInformativeText("Успешно создан");
    info_window.setStandardButtons(QMessageBox::Ok);
    info_window.setDefaultButton(QMessageBox::Ok);
    info_window.exec();
    CreateProfileWindow::close();
}
void CreateProfileWindow::on_cancel_button_clicked()
{
    confirm_cancel();
}
void CreateProfileWindow::on_save_button_clicked()
{
    general_check_before_generate();;
}
CreateProfileWindow::~CreateProfileWindow()
{
    delete ui;
}
```

Приложение 6. Исходный код класса SaveTimeDialog

```
#indef SAVETIMEDIALOG_H
#define SAVETIMEDIALOG_H
#include <QDialog>
#include <QDebug>
#include <QSqlQueryModel>
#include <QSqlQuery>
#include <QSortFilterProxyModel>
#include <QSqlRecord>
#include <QSqlError>
#include "databaseutils.h"
#include "calcmodule.h"
#include "names_of_tables.h"
namespace Ui {
class SaveTimeDialog;
}
class SaveTimeDialog : public QDialog
{
    Q_OBJECT
public:
    explicit SaveTimeDialog(QString texttime, QSqlDatabase &db,
QWidget *parent = 0);
    ~SaveTimeDialog();
signals:
    void good_transaction();
private slots:
    void on_add_to_db_button_clicked();
    void on_cancel_save_button_clicked();
private:
    Ui::SaveTimeDialog *ui;
    DatabaseUtils *db_view;
    QSqlQueryModel model;
    QSqlQuery query;
    QSqlDatabase ptr_db;
    QString session_time;
    void init_models();
    void flood_my_combos();
    int add_to_db();};
#endif // SAVETIMEDIALOG_H
#include "savetimedialog.h"
#include "ui_savetimedialog.h"
SaveTimeDialog::SaveTimeDialog(QString texttime, QSqlDatabase
&db, QWidget *parent) :
    QDialog(parent),
    ui(new Ui::SaveTimeDialog)
{    ui->setupUi(this);
    ptr_db = db;
    session_time = texttime;
    ui->time_val->setText(session_time);
    init_models();}
void SaveTimeDialog::init_models()
```

Приложение 6. Исходный код класса SaveTimeDialog

```
{
    model.setQuery("SELECT name FROM "+QString(TB_TYPE_OF_WORK)+" "+QString(TB_TYPE_OF_WORK)+" WHERE name='"+ui->type_selecter->currentText()+"'");
    if (model.lastError().isValid()) {
        qDebug() << "[SaveTimeDialog]: " << model.lastError();
    }
    flood_my_combos();}
void SaveTimeDialog::flood_my_combos()
{
    ui->type_selecter->setModel(&model);
    ui->type_selecter->setModelColumn(0);}
int SaveTimeDialog::add_to_db()
{   QDate date = QDate::currentDate();
    QSqlQuery query(ptr_db);
    QSqlQuery tmp_query(ptr_db);
    tmp_query.exec("SELECT tax_id FROM "+QString(TB_PROFILE));
    tmp_query.next();
    int tax_id = tmp_query.value(0).toInt();
    tmp_query.exec("SELECT rate_id FROM "+QString(TB_PROFILE));
    tmp_query.next();
    int rate_id = tmp_query.value(0).toInt();
    query.prepare("INSERT INTO "+QString(TB_CURRENT)+" (date, type_id, time, comment, tax_id, rate_id) VALUES (:date, :type_id, :time, :comment, :tax_id, :rate_id)");
    query.bindValue(":date", date);
    query.bindValue(":type_id", tmp_query.exec("SELECT id FROM "+QString(TB_TYPE_OF_WORK)+" WHERE name='"+ui->type_selecter->currentText()+"'"));
    query.bindValue(":tax_id", tax_id);
    query.bindValue(":rate_id", rate_id);
    query.bindValue(":time", session_time);
    query.bindValue(":comment", ui->comment_val->text());
    if (query.exec())
    {   qDebug() << query.lastQuery();
        emit good_transaction();
        SaveTimeDialog::close();
        return 0;
    } else {
        qDebug() << query.lastError();
        return 0;}
    return 1;}
void SaveTimeDialog::on_add_to_db_button_clicked()
{add_to_db();}
void SaveTimeDialog::on_cancel_save_button_clicked()
{SaveTimeDialog::close();}
SaveTimeDialog::~SaveTimeDialog()
{delete ui;}
```

```
#include <mainwindow.h>
#include <QApplication>
#include <QLockFile>
#include <QDir>
#include <QMessageBox>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QString tmpDir = QDir::tempPath();
       QLockFile lockFile(tmpDir + "/.<uniq id>.lock");
       if(!lockFile.tryLock(100)){
           QMessageBox msgBox;
           msgBox.setIcon(QMessageBox::Warning);
           msgBox.setText("Приложение уже запущено.¥n"
                          "Разрешено запускать только один экземпляр приложения.");
           msgBox.exec();
           return 1;}
    QApplication::setOrganizationName("LNCorp");
    QApplication::setApplicationName("LTTQT");
    MainWindow w;
    w.show();
    return a.exec();
}
```