МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Политехнический институт: Заочный
Кафедра «Системы автоматического управления»

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой

д.т.н., профессор

_____/ В.И. Ширяев

«____» _____ 2020 г.

Разработка на ОС Android автоматизированной системы организации труда, графиков планового ремонта и оценки качества выполняемых работ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ – 09.03.01.2020.030.00 ПЗ ВКР

Руководитель работы

доцент каф. САУ, к.т.н.

_____/ В.О. Чернецкий

«____» _____ 2020 г.

Автор работы

студент группы ПЗ-597

_____/ М.В. Приказчиков

«____» _____ 2020 г.

Нормоконтролер

доцент каф. САУ, к.т.н.

_____/ В.О. Чернецкий

«____» _____ 2020 г.

Челябинск 2020

АННОТАЦИЯ

Приказчиков М.В. Разработка на ОС Android автоматизированной системы организации труда, графиков планового ремонта и оценки качества выполняемых работ. – Челябинск: ЮУрГУ, ПИ: Заочный; 2020, 48 с., 44 ил., библиогр. список – 14 наим., 14 листов слайдов презентации ф.А4, 1 прил.

В данной выпускной квалификационной работе рассматривается процесс создания программного обеспечения на ОС Android для автоматизации системы организации труда, графиков планового ремонта и оценка качества выполняемых работ.

Цель работы – разработка автоматизированной системы организации труда, графиков планового ремонта и оценка качества выполняемых работ на участке контрольно-измерительных приборов и автоматики (КИПиА).

В дальнейшем планируется доработка программы, путем включения новых возможностей (част, уведомления на смартфон, оценка трудоемкости робот).

Также, выполнено описание комплекса программ, полученных в результате выполнения дипломного проекта.

В заключительной части сделаны выводы по эффективности разработки и полученных результатов от внедрения данного комплекса систем.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Цель работы: разработка автоматизированной системы организации труда, графиков планового ремонта и оценка качества выполняемых работ на участке контрольно-измерительных приборов и автоматики (далее КИПиА).

Приложение должно иметь функционал, который бы отвечал всем требованиям для работы с нормативными документами и управлением персоналом через мобильное приложение. Это позволило бы качественнее планировать предшествующий рабочий процесс с постановкой задач для работников на кратковременные и долговременные сроки и объективно оценивать качество выполненных работ.

Также, программа должна поддерживать ведение базы данных графика планового ремонта, сотрудников, нормативных документов и архивацию чата, обеспечивать правильное взаимодействие их в проекте, что позволит удобно анализировать любые статистические данные и динамику графиков планового ремонта. Структура базы данных должна обеспечить эффективность и гибкость запросов к ней и целостность хранимой в ней информации.

Приложение должно обладать интуитивным дружественным интерфейсом для пользователей с разной возрастной группой и квалификацией.

Перед разработчиком стоит задача создания автоматизированной информационной системы, с возможностью модификации функций в зависимости от набора задач конечного пользователя. Функциональность приложения направлена на получение оптимального программного решения от разработчика, которое позволяет эффективно использовать приложение в конечном релизе.

Для достижения всех этих задач нужно:

- изучить документооборот и структуру участка КИПиА;
- разработать базу данных;
- выбрать систему управления базами данных, язык программирования и среду разработки;
- разработать приложение на ОС Android;
- протестировать созданное приложение.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Актуальность работы

Актуальность данного проекта заключается в том, чтобы проводить контроль качества выполняемых работ, графиков планового ремонта и с наименьшими затратами времени организовывать распорядок дня работников.

Организация труда является одной из основополагающих столпов, без которых не может работать ни одно предприятие. Это касается не только качества выпускаемой продукции, но и правильного управления персоналом. На старых заводах постоянно оптимизируют количество рабочих мест, в частности, сокращают штат сотрудников и инженерно-технического персонала, что приводит к ухудшению качества обслуживания ремонтных участков.

Рассмотрим участок КИПиА на примере рисунка 1.1, в него входит один начальник участка, один бригадир и две бригады по два слесаря КИПиА. Бригадир раздает распоряжения, данные ему начальником участка, следит за выполнением поручений и графиком плановых работ на участке КИПиА, создает план отправки приборов в ремонт.

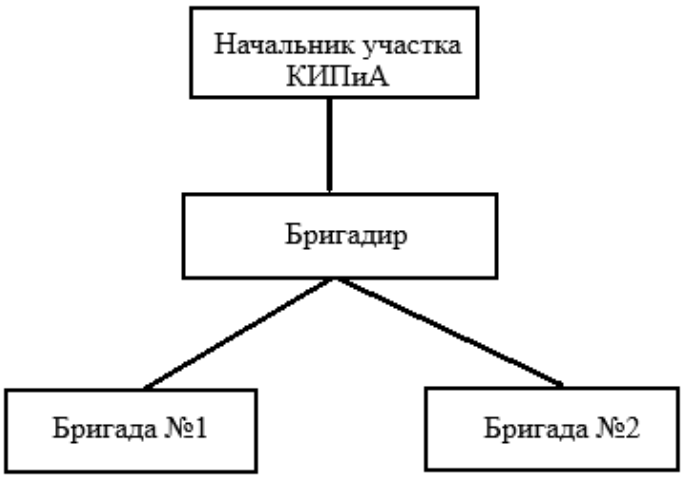

Рисунок 1.1 – Структурная схема подразделения

При сокращении бригадира, резко падает качество контроля над поставленными задачами.

Поскольку в обязанности начальника участка, также, входит оформление заявок на автомобиль, накладных, проведение инструктажей, разработка должностных инструкций, участие в ежедневных собраниях и обход оборудования на наличие нарушений по охране труда, имеет смысл замены бригадира на удобное приложение для дистанционного управления через мобильное устройство, за счет которому он будет отдавать распоряжения и контролировать статус выполняемых заданий.

1.2 Особенности постановки задач на платформе android

Организация труда — это форма, в которой реализуются экономические результаты трудовой деятельности.

В условиях рыночной экономики возрастает значение различных факторов, которые воздействуют на эффективность производства. Среди факторов эффективности немалое значение имеет организация труда. Даже самое современное оборудование и высокопроизводительная техника не дают желаемого результата при низкой организации их обслуживания. И наоборот, при правильном подходе организации труда можно получить высокий результат. Организация труда должна включать в себя:

- организацию рабочего времени на один день
- соблюдение графика плановых работ
- составление накладных
- нормы человеко-часов на день.

При внедрении мобильного приложения улучшается контроль за исполнением приказов и распоряжений, и соответственно:

- увеличивается производительность труда
- соблюдаются графики планового ремонта
- происходит своевременная отправка оборудования в ремонт
- имеется наглядная возможность наблюдения загруженности участка и регуляция нагрузки персонала в целом и отдельного рабочего.
- представляется возможность реально увидеть, как и чем работник был занят за последний месяц
- премирование работников за производственные результаты становится более объективным
- снижаются нагрузки на начальника участка
- появляется возможность дистанционной постановки задач
- более удобное наблюдение за выполнением распоряжений.

1.3 Архитектура взаимодействия

Объект исследования состоит из группы смартфонов и персонального компьютера, где компьютер представляет собой центр разработки приложения.

В структуру смартфона входит сенсорный экран, который обладает достаточным ресурсным потенциалом с высокой производительностью для комфортной и безотказной работы приложения.

Смартфон на прямую взаимодействует с пользователем, использующим данное приложение для быстрой и удаленной работы.

Важно при разработке программы учесть совместимость операционной системы Android всех смартфонов, на которых будет установлено данное приложение [1].

Персональный компьютер является высокопроизводительным средством по разработке приложения и загрузки его на смартфон. Сама реляционная база данных будет хранится на удаленном сервере, и связь с ним будет через интернет для создания удаленного доступа к ней.

Загрузка приложения с компьютера на смартфон будет осуществляться через USB-кабель, который позволяет переносить приложение с компьютера на телефон. Возможны и другие интерфейсы для передачи данных на смартфон, но для данной работы эти решения рассматриваться не будут.

Рекомендуемые системные требования с рабочей средой разработки Visual Studio 2019 net framework xamarin взятые с сайта разработчика:

- Windows 10 версии 1703 и выше: Домашняя, Pro, для образовательных учреждений и Корпоративная (выпуски LTSC и S не поддерживаются)
- Процессор с тактовой частотой не ниже 1,8 ГГц. Рекомендуется использовать как минимум двухъядерный процессор.
- 2 ГБ ОЗУ; рекомендуется 8 ГБ ОЗУ (минимум 2,5 ГБ при выполнении на виртуальной машине)
- Место на жестком диске: до 210 ГБ (минимум 800 МБ) свободного места в зависимости от установленных компонентов; обычно для установки требуется от 20 до 50 ГБ свободного места.
- Скорость жесткого диска: для повышения производительности установите Windows и Visual Studio на твердотельный накопитель (SSD)
- Видеоадаптер с минимальным разрешением 720p (1280 на 720 пикселей); для оптимальной работы Visual Studio рекомендуется разрешение WXGA (1366 на 768 пикселей) или более высокое.

Эти параметры указаны производителем как оптимальные требования для комфортной работы в Visual Studio.

1.4 Анализ аналогичных приложений

Подобные приложения делятся на три типа [2]:

- Приложение глобального управлением фирмы и их сотрудниками
- Приложение в виде почты
- Узконаправленные приложения, которых нет в открытом доступе

Приведем несколько примеров таких приложений:

Приложение Yaware.TimeTaracker

Приложение Yaware.TimeTaracker пример приложения глобального управлением фирмы, где включена вся структура производства рисунок 1.2.

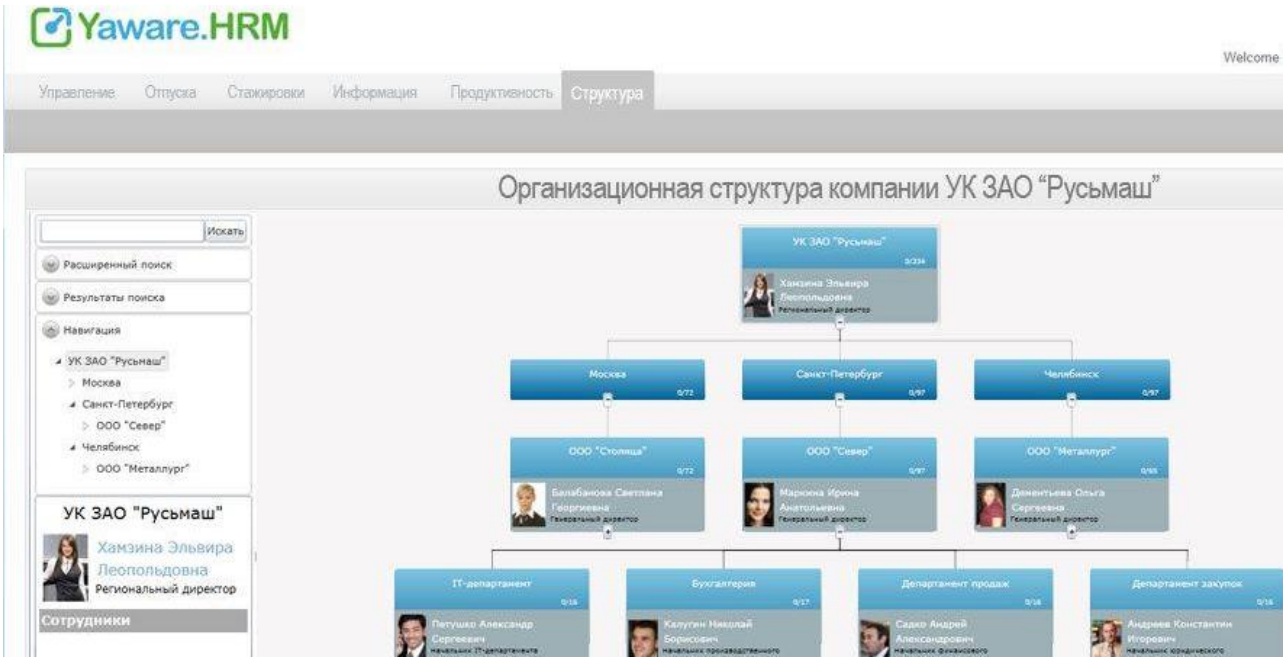

Рисунок 1.2 – Снимок экрана компьютера

Особенности приложения:

Управление персоналом:

- Есть Календарь отпусков
- Есть Профили работников
- Есть Производительность сотрудника
- Есть Примечания сотрудника
- Есть Хранение файлов

Число сотрудников Неограниченно:

- Есть Уведомления
- Есть Управление зарплатой

Набор персонала:

- Есть Размещение вакансий
- Есть Оценки
- Есть Портал самообслуживания
- Есть Инструменты совместной работы
- Есть Управление контактами / CRM
- Есть Планирование собеседований

Достоинства:

Приложение позволяет составлять иерархические ступени, мониторинг рабочего времени, набор персонала. Хорошо подходит для сети офисов, поскольку есть возможность оценки трудовой активности в разных регионах страны, и все это отображается на смартфоне рисунок 1.3

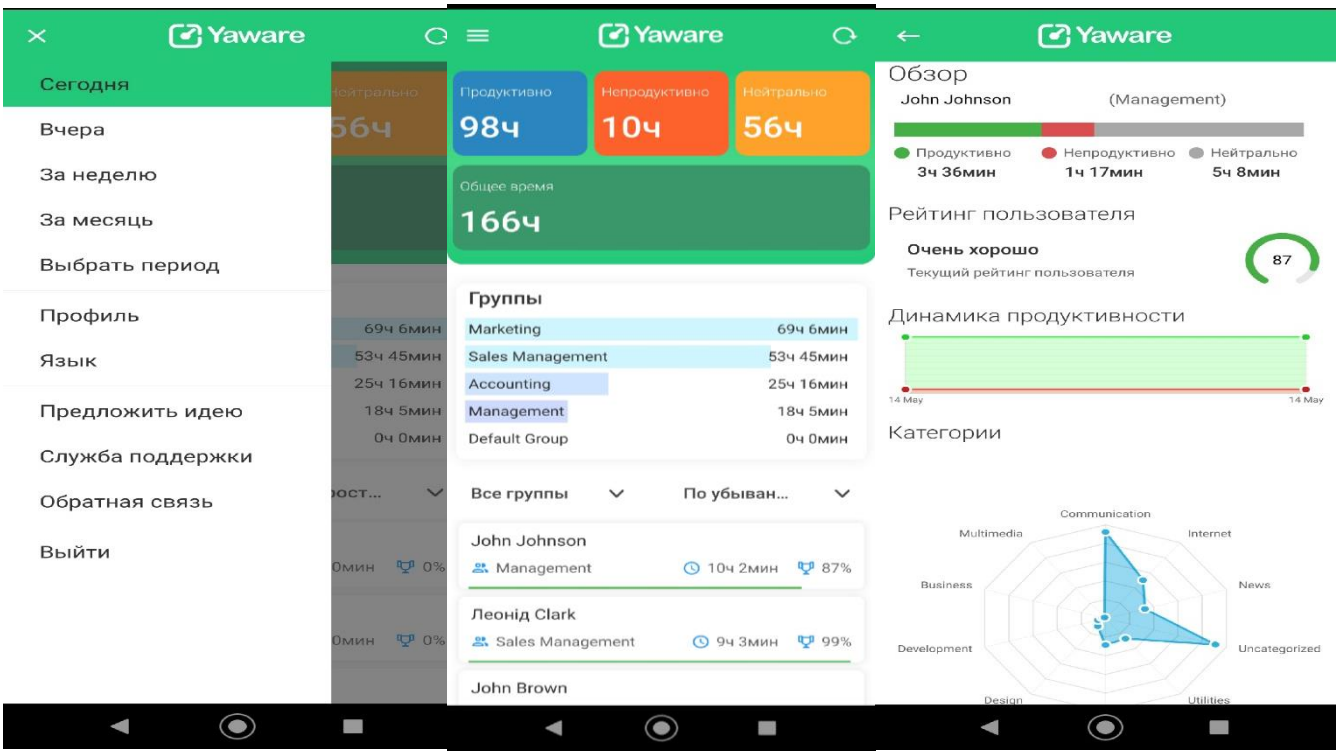

Рисунок 1.3 – Скриншоты мобильного приложения

Недостатки:

Приложение платное, излишняя многофункциональность, то есть разработчики предлагают полное администрирование всех функций компании через облачное хранение данных тем самым ставят под удар информационную безопасность компании. Все коммерческие данные должны хранится локально для обеспечения должного сокрытия.

IBM Notes Traveler

Мощное кроссплатформенное семейство программных продуктов, предназначенное для создания корпоративной инфраструктуры, обеспечивает доступ не только с смартфона, но и с компьютера. IBM Notes Traveler обеспечивает автоматическую двустороннюю синхронизацию данных IBM Notes и Domino по сети с использованием широкого спектра мобильных устройств [2].

IBM Notes Traveler имеет серверный компонент, развернутый в корпоративной сети компании рисунок 1.4.

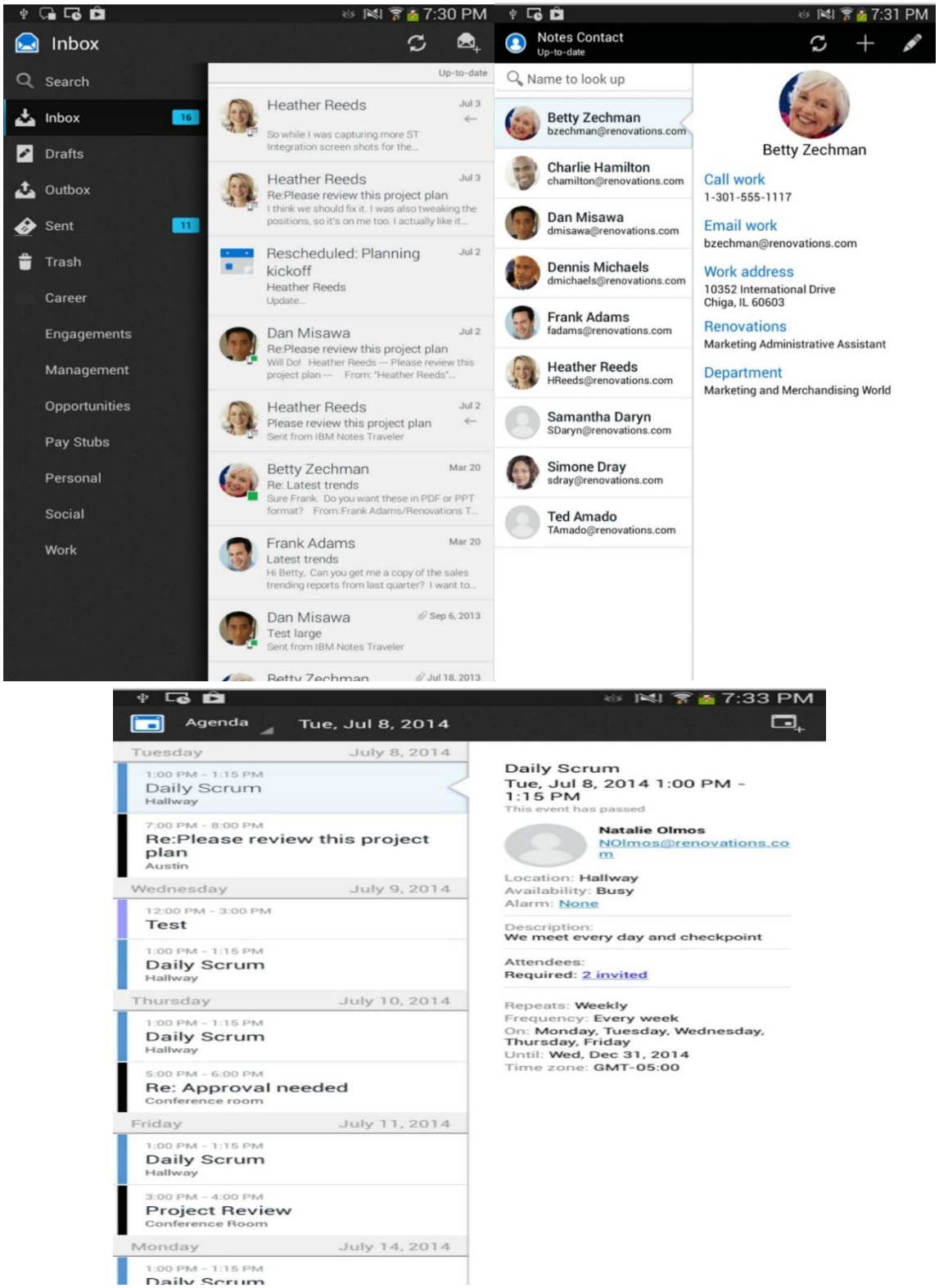

Рисунок 1.4 – Скриншоты мобильного приложения IBM Notes Traveler

Особенности IBM Notes Traveler:

- отправка сообщений;
- чтение, составление, ответ, пересылка, файлов;
- составление и чтение зашифрованной и подписи почты;

- управление почтовыми папками;
- добавление виджета на главный экран;
- календарь;
- просмотр повестки дня, недели и месяца;
- управление приглашениями, собраниями, встречами, событиями дня, юбилеями, напоминаниями;
- добавление виджета на главный экран;
- контакты;
- просмотр адресной книги Notes контакты;
- поиск контактов с сервера;
- просмотр по сроку, приоритету и завершенному статусу задания;
- создание пользовательских категорий;
- добавление виджета на домашний экран;
- настройка количества данных для синхронизации и как часто;
- использование SMS-уведомлений для экономии заряда батареи;
- управление политикой безопасности;
- автономная поддержка;
- данные, хранящиеся на устройстве, зашифрованы.

Преимущества приложения:

Хорошая защищённость корпоративной почты, можно развернуть свой локальный сервер для обеспечения информационной безопасности. Хороший почтовый сервис и органайзер.

Недостатки:

Не обеспечивает должного контроля над рабочим персоналом.

Существуют узконаправленные приложения, которые разрабатывают на заказ в IT фирмах. Они полностью предоставляют авторский проект по собранной информации от заказчика и полное его сопровождение.

Одна из таких фирм Woxapp которая профессионально занимаются созданием приложений. Этапы разработки мобильного приложения:

- идея продукта и бизнес-экспертиза;
- проведение интервью и предварительный анализ рынка;
- повторный контакт с клиентом;
- оценка проекта и приложения;
- подписания договора;
- создание прототипа;
- дизайн продукта;

- разработка технического задания;
- программирование и тестирование приложения;
- ретроспектива;
- публикация приложения;
- сопровождение проекта.

Выводы по первой главе

Существует множество мобильных приложений с разными функциональными возможностями, но ни одно из них полностью не удовлетворяет разных заказчиков.

Рассмотренные здесь приложения хорошо подходят для офисов или небольших производств. Но с их помощью очень сложно обеспечить автоматизированную работу предприятия, так как они направленны на общие потребности рынка и не учитывают индивидуальные особенности фирмы.

Данные исследования показывают необходимость создания персонального приложения для описанного нами предприятия, для того, чтобы соответствовать их потребностям. Таким образом мы добьёмся автоматизации участка КИПиА.

2 ВЫБОР СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ И ЯЗЫКА ПРОГРАММИРОВАНИЯ

2.1 Анализ и выбор системы управления базами данных для мобильного приложения на базе ОС Android

Прежде чем остановить свой выбор на какой-либо конкретной системой управления базами данных (далее СУБД).

СУБД – это комплекс программных и языковых средств, предоставляющих возможность создавать и управлять базами данных, система дает возможность хранения, обеспечивает единство и предоставляет безопасность при хранении данных [3].

Для разработки данного мобильного приложение требуется выбрать СУБД, это необходимо, для того чтобы приложение имело возможность работать с ней группе пользователей, а также имелась возможность добавления, редактирования, хранение и удаления данных.

Выбор СУБД был остановлен на следующих вариантов с учетом их особенностей и применимости к данной программе.

СУБД делится по типу хранимых данных таки как [3]:

- Иерархические – используется древовидное предоставление данных
- Сетевые – является расширенным вариантом иерархической модели
- Реляционные – предоставляет связь с системами баз данных
- Объектно-ориентированное СУБД – представляет базу данных как объект
- Объектно-реляционная СУБД – содержит в себе часть и возможности предыдущих двух пунктов

Для данного приложения лучше всего подходит реляционная СУБД. Далее будут рассмотрены только реляционные СУБД. СУБД, поддерживающими реляционные модели данных являются:

- PostgreSQ;
- Oracle;
- MS SQL Server;
- SQLite;
- MySQL.

PostgreSQL

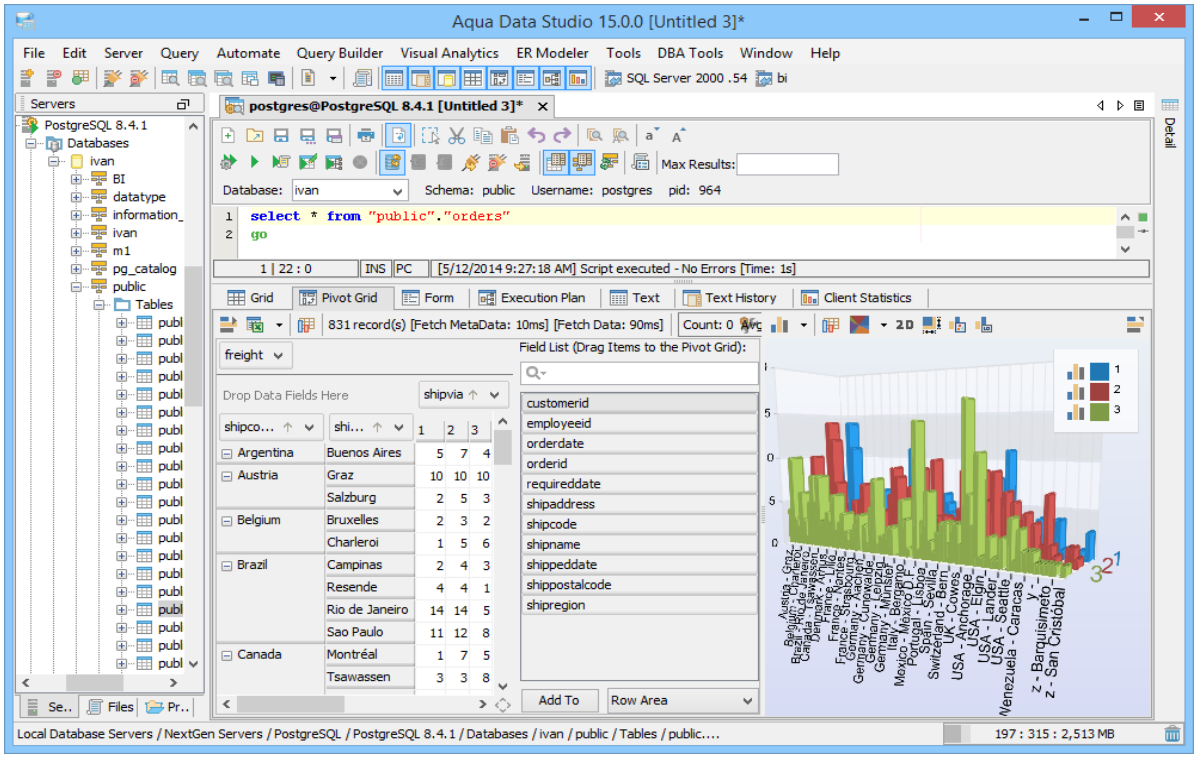

Рисунок 2.5 – Среда разработки PostgreSQL

PostgreSQL – составляет конкуренцию на рынке СУБД с открытым кодом боле тридцати лет рисунок 2.5. Язык PostgreSQL основан на SQL поддерживает большую часть из возможностей стандартного SQL [4].

PostgreSQL имеет ряд особенностей, такие как:

- высокая производительная и надежная система записи и копирования данных;
- максимальный размер таблицы составляет 32 Тбайт;
- максимальный размер поля составляет 1 Гбайт;
- максимальное количество записей в таблице ограничено ее размерами;
- размер базы данных не имеет ограничений;
- максимум полей в записи 250–1600, в зависимости от типов полей;
- индексов в таблице не имеет ограничений;
- расширяемость.

Oracle

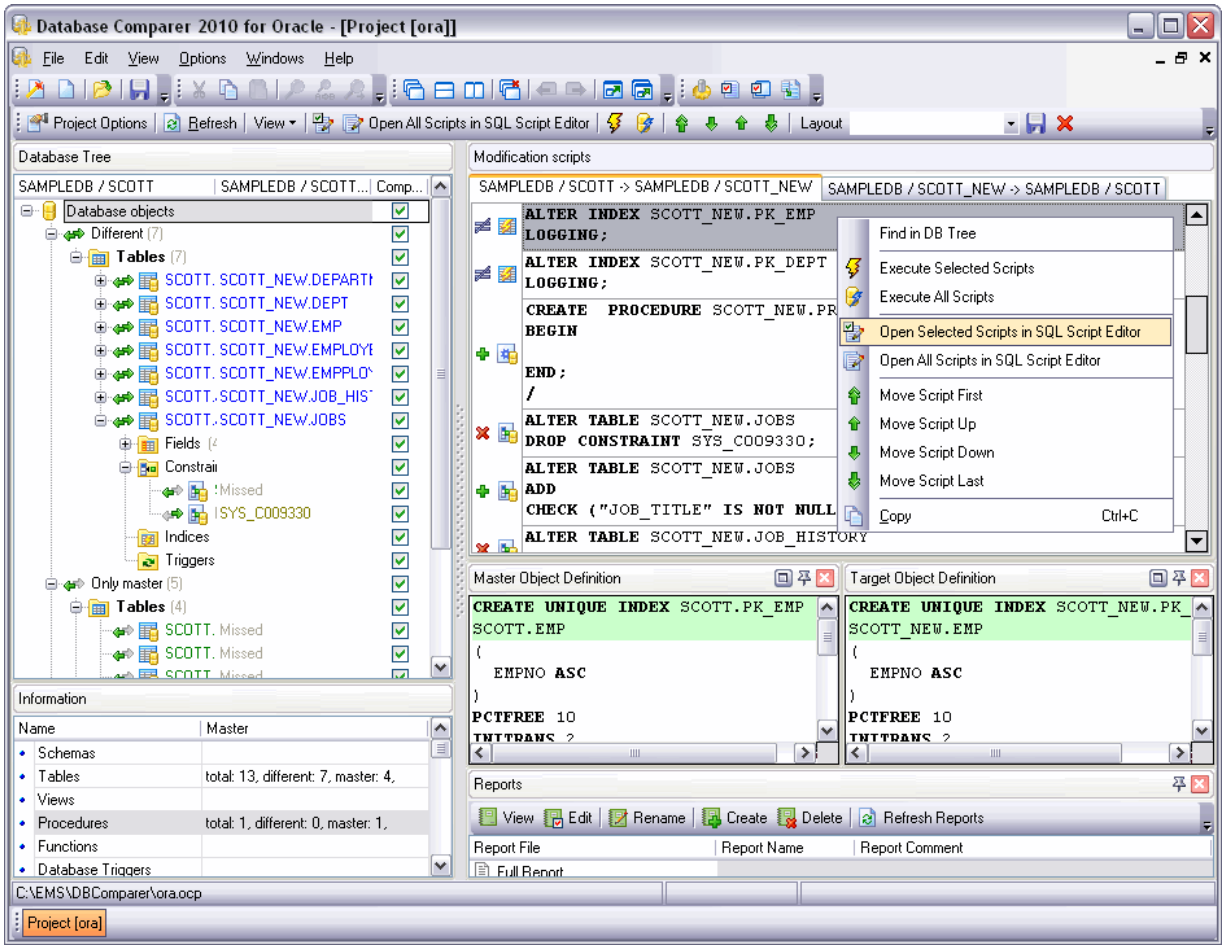

Рисунок 2.6 – Среда разработки Oracle

Oracle – эта система также, как и PostgreSQL поддерживает объектно-реляционною модель данных, а также реализует объектно-ориентированный подход рисунок 2.6.

Главные особенности Oracle:

- Real Application Cluster (RAC) один экземпляр базы данных может работать на нескольких узлах, обеспечивает управление нагрузки и свободно увеличивать или уменьшать систему в случае необходимости;
- Automatic Storage Management (ASM) позволяет автоматизировать и распределять между имеющимися ресурсами данные систем хранения
- производительность. Oracle позволяет автоматически управлять уровнями сервиса;
- самоуправление. Специальные механизмы Oracle позволяют самостоятельно перераспределять нагрузку на систему, оптимизировать и испровлять SQL-запросы, выявлять и предотвращать ошибки;

- большие базы данных. Допустимый размер экземпляра базы данных Oracle может достигать 8 экзабайт;
- недорогие серверные системы.

MS SQL Server

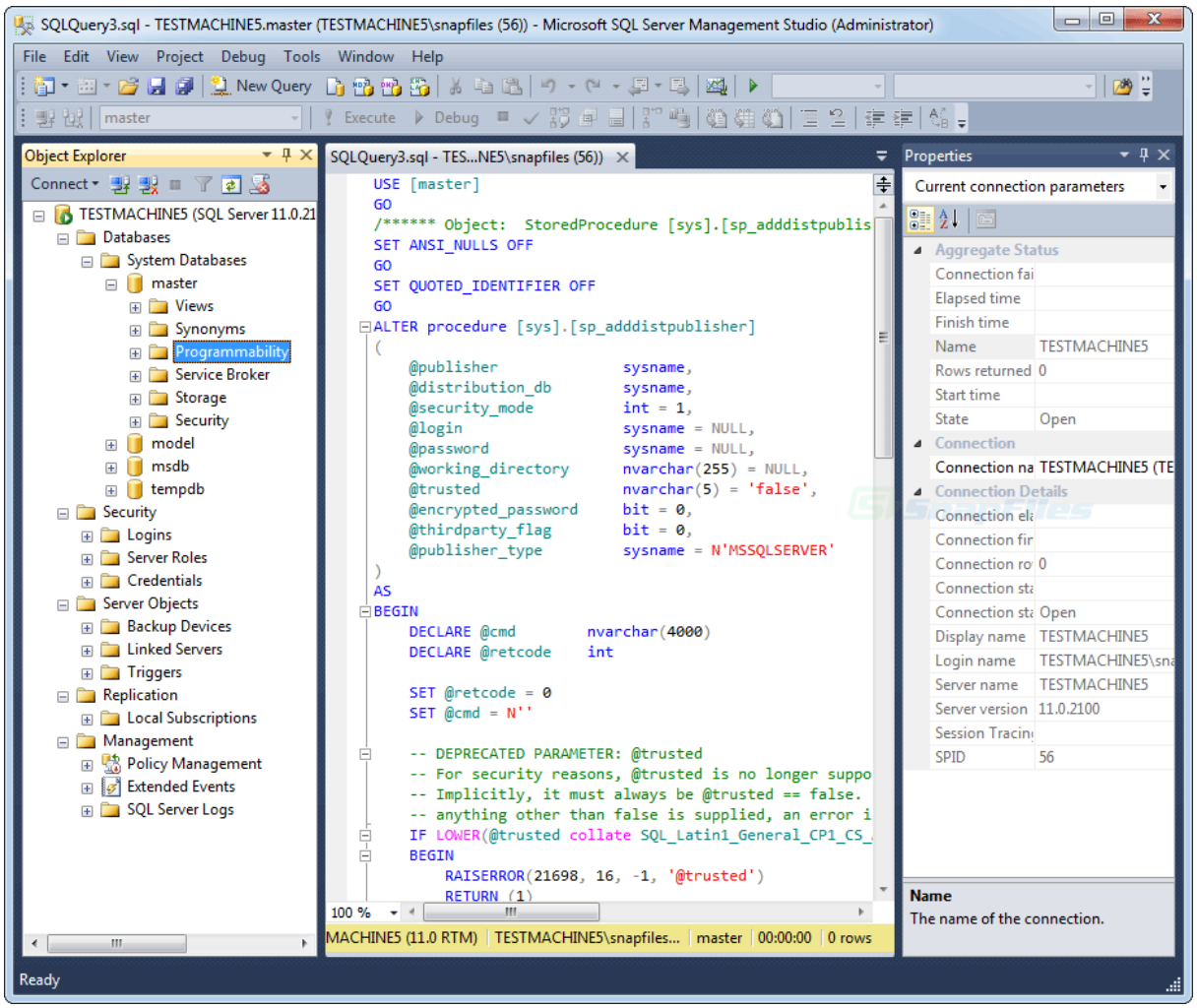

Рисунок 2.7 – Среда разработки MS SQL Server

MS SQL Server СУБД разработанная компанией Microsoft использующая язык запросов Transact-SQL имеет закрытый код рисунок 2.7.

Основные характеристики:

- поддержка большого количества пользователей;
- кроссплатформенность;
- расширяемость;
- язык T-SQL;
- парольное сжатие данных и восстановление базы данных;
- создание копий данных;
- распределение запросов.

SQLite

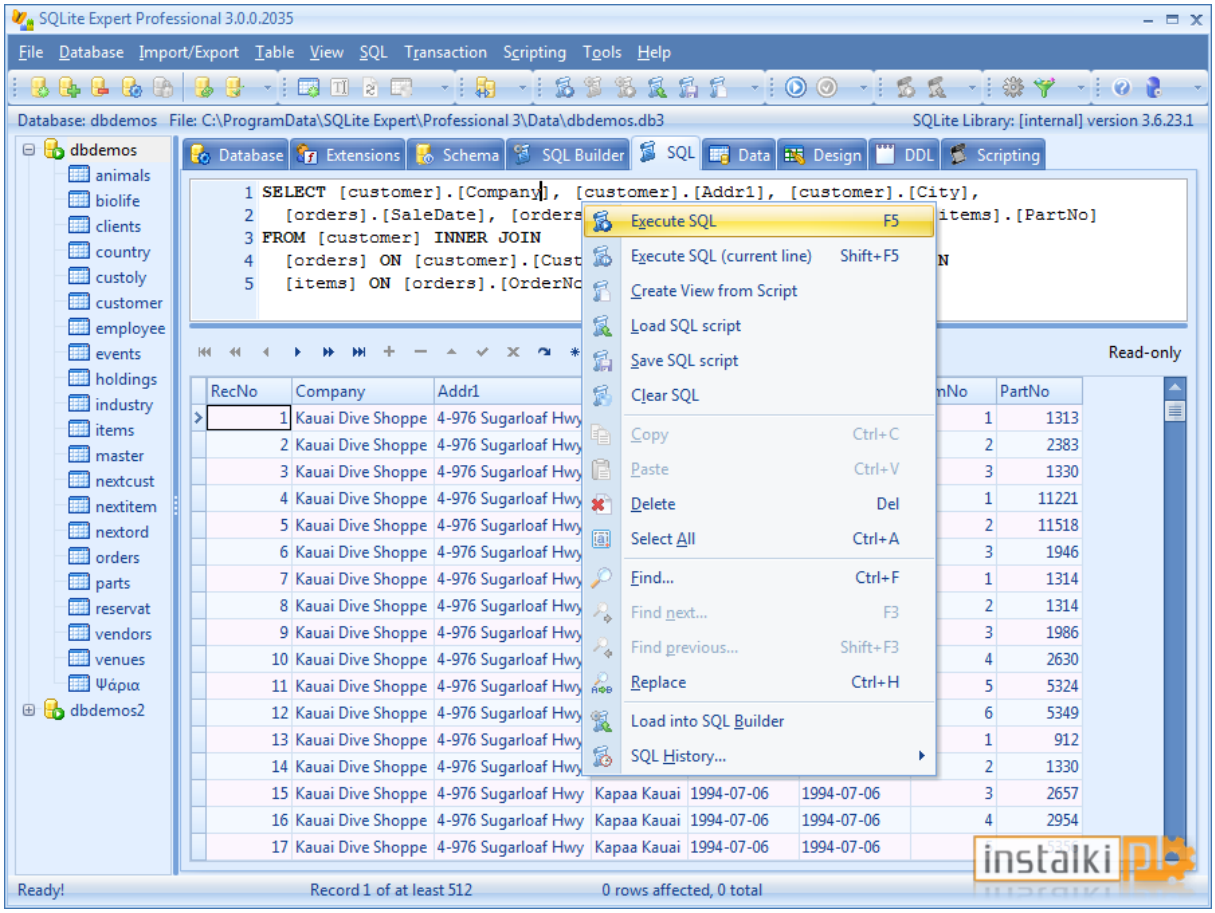

Рисунок 2.8 – Среда разработки SQLite

SQLite имеет открытый исходный код, компактная встраиваемая СУБД, то есть не использует модель клиент-сервер, а является частью программы рисунок 2.8.

При сборке проекта, библиотеки SQLLite имеет ограничения, которые при необходимости можно увеличивать.

Список ограничений:

- длина строки недолжна превышать 1 000 000 000 байт;
- максимальное количество столбцов 2000;
- длинна SQL-выражения не должна превышать 1 000 000 000 байт;
- максимальный размер базы данных 140ТБ;
- количество таблиц в выражениях с JOIN не должно превышать 64.

MySQL

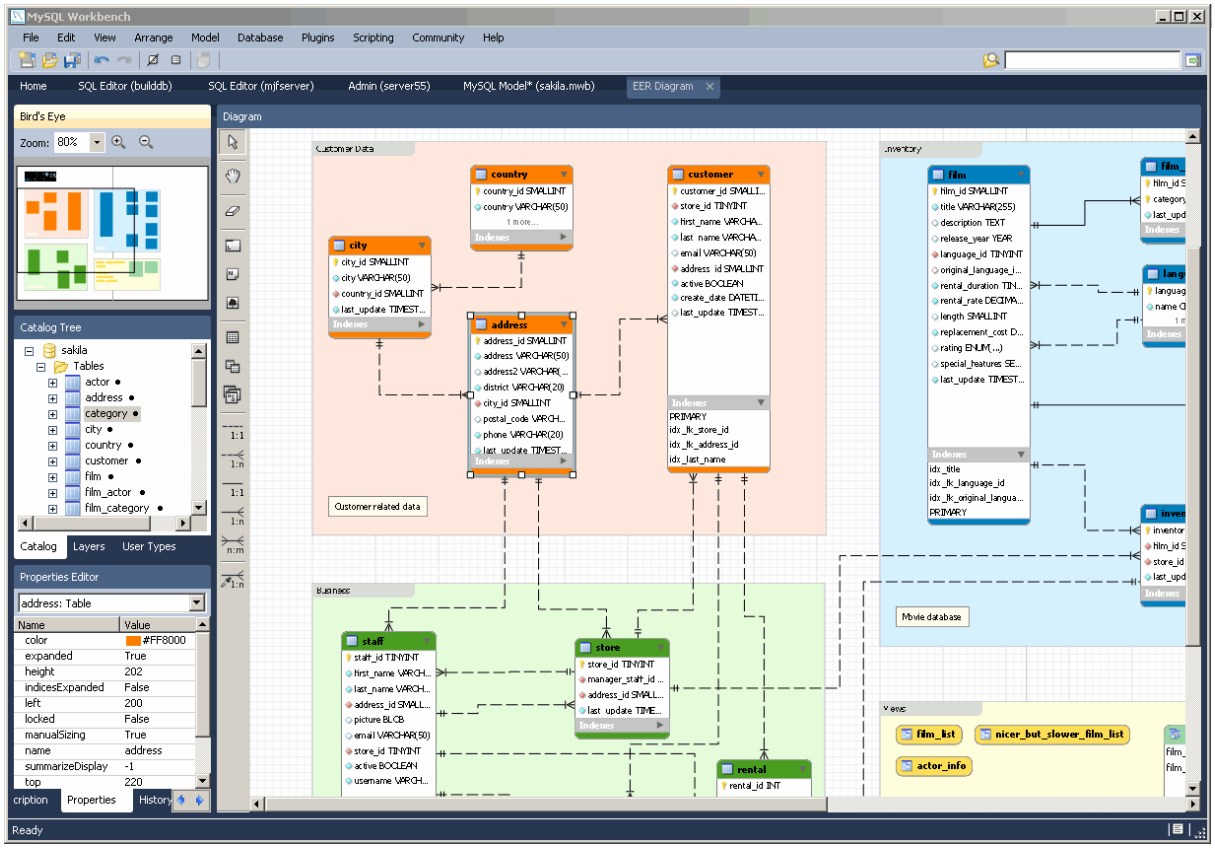

Рисунок 2.9 – Среда разработки MySQL

MySQL реляционная база данных сочетает в себе простоту и гибкость, рисунок 2.9 [5].

Основные особенности:

- программное обеспечения с открытым кодом;
- поддерживает много поточность SQL – сервер, является системой клиент сервер;
- максимальный размер таблицы составляет 8 миллионов терабайт;
- безопасность поддерживает разграничением доступа;
- кроссплатформенный.

Из выше перечисленного следует что каждая из этих СУБД имеет свои сильные стороны. Поскольку, все СУБД отвечают необходимым запросам выбор был сделан в пользу, SQLite основываясь на:

- удобстве работы с запросами к базе данных;
- кроссплатформенности;
- открытом исходном кодом;
- возможность группового использования.

2.2 Выбор языка программирования

Поскольку мы определись с выбором СУБД, следует выбрать с помощью какого языка программирования будет написана программа.

Язык программирования: это искусственные языки, предназначенные для записи алгоритмов, исполнение которых передается ЭВМ.

Языки программирования подразделяются на высокоуровневые и низкоуровневые. Отличие высокоуровневых от низкоуровневых заключается в то что, чем выше уровень программирования, тем дальше он удален от машинного кода и тем более он удобен и быстр в использование. В связи с этим выбор будет производиться только среди высокоуровневых языков программирования [6].

Ассортимент высокоуровневых языков программирования очень велик. Наш выбор будет происходить из 5 наиболее подходящих языков.

Python

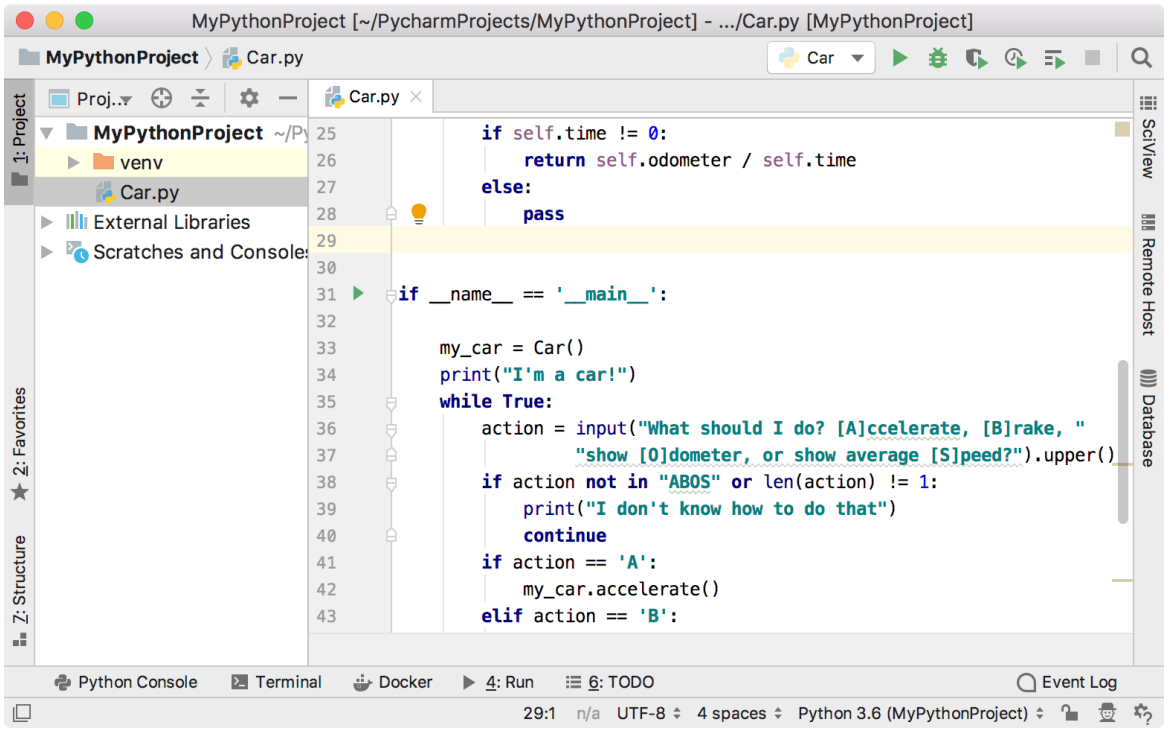

Рисунок 2.10 – Демонстрация кода Python

Python – ориентированный на превышение чтения и записи программного кода рисунок 2.10. Его особенностями являются [7]:

- свободный и открытый;
- имеет возможность переводить код с других языков программирования;
- многочисленные библиотеки;
- встраиваемый;
- объектно-ориентированный.

C#

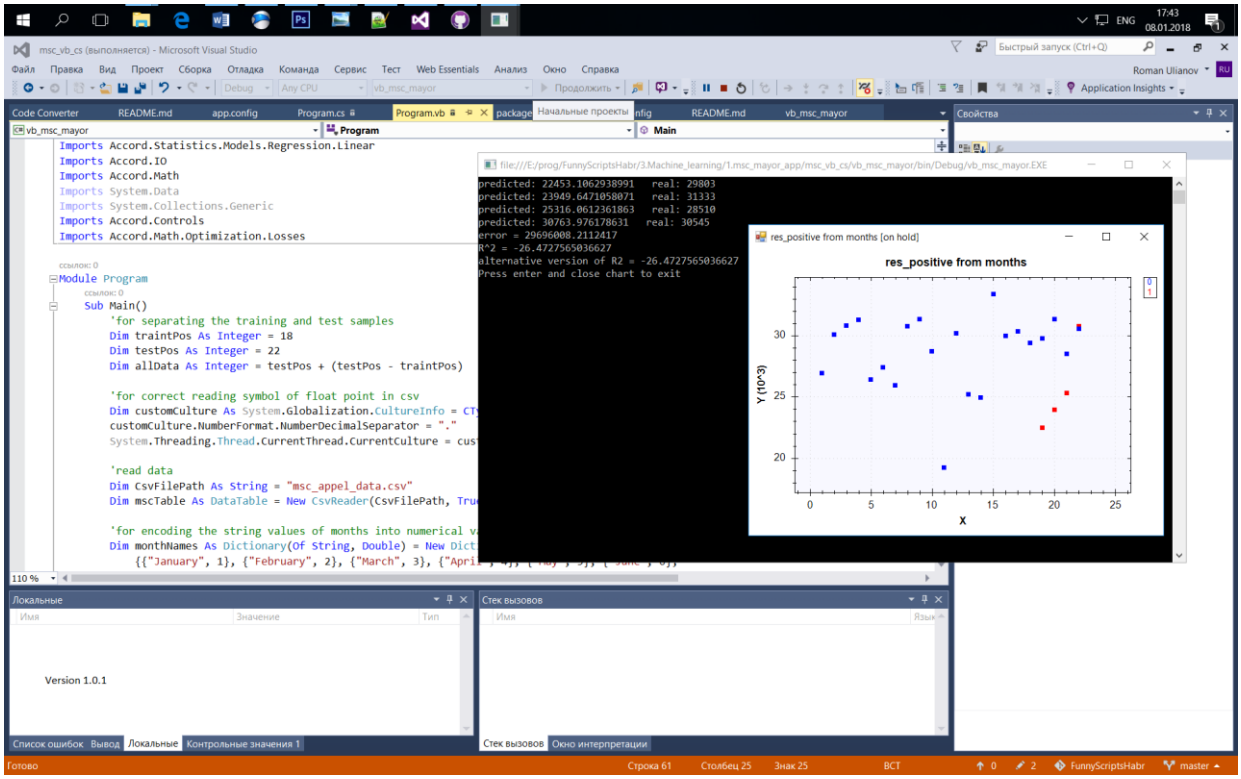

Рисунок 2.11 – Демонстрация кода C#

C# - имеет C подобный синтаксис, наиболее похож на C++ и Java рисунок 2.11 [8].

- Надежен и элегантен;
- вобрал в себя лучшее из современных языков программирования Java, C++, Visual Basic и т.п.;
- мощный объектно-ориентированный язык программирования;
- быстрее иных языков программирования работает с .Net Framework.

JavaScript.

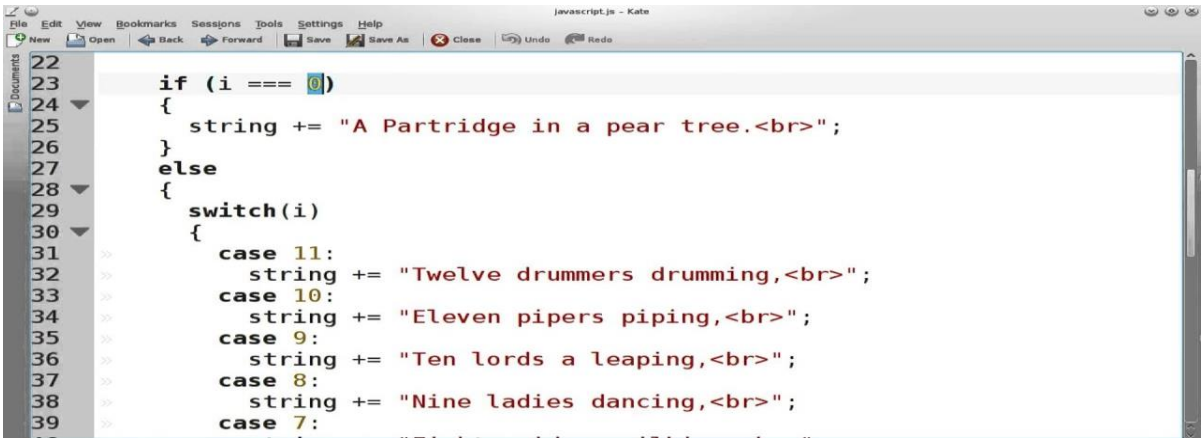

Рисунок 2.12 – Демонстрация кода JavaScript

JavaScript – язык сценарии и скриптов. Предназначенный для небольших клиент-сервер приложений в сети Internet рисунок 2.12 [6].

- Язык сильно зависит от регистра;
- слабый контроль типов, используемых данных;
- встраиваемые описания кода;
- объектно-ориентированный.

Go

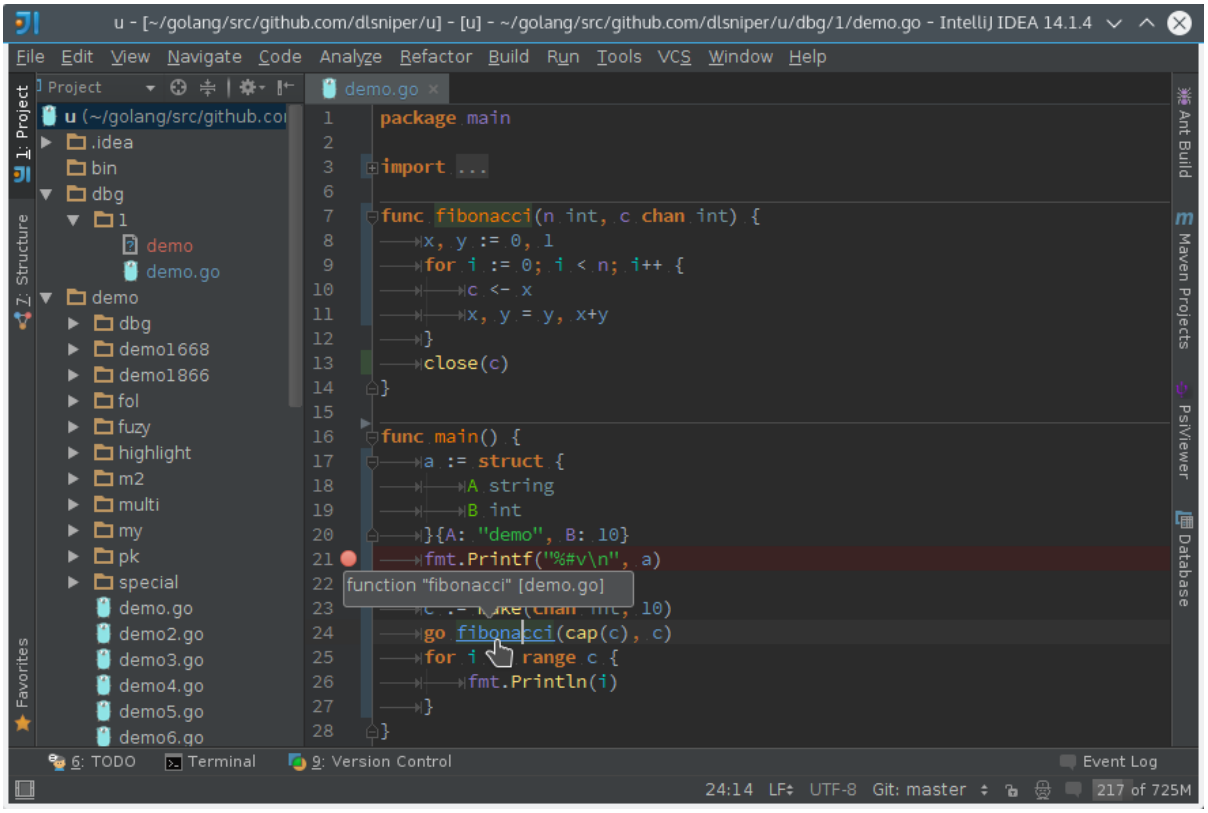

Рисунок 2.13 – Демонстрация кода Go

Go (так же Golang) – Язык разработанный компанией Google для использования в операционных системах FreeBSD, OpenBSD, Linux, macOS, Windows, DragonFly BSD, Plan 9, Solaris, Android, AIX рисунок 2.13.

- Групповая разработка;
- легок в написание кода в связи с чем увеличивается скорость написания текста;
- содержит инструменты, позволяющие сортировать код;
- строгий контроль форматирования кода.

При выборе языка программирования следует учитывать необходимые для написания программы особенности. В разработке программы принимает участие один человек, нет необходимости использовать языки программирования, направленные на групповую разработку приложений. Разрабатываемая программа

будет использовать библиотеки .Net Framework, удаленное подключение в БД. Будет целесообразнее использовать C#.

2.3 Выбор среды разработки

После выбора языка встает необходимость выбора среды разработки. Поскольку выбор сред разработки очень велик, мы провели выбор среди наиболее удобных сред разработки. В их число вошли: Eclipse, Project Rider, Visual Studio, MonoDevelop.

Eclipse

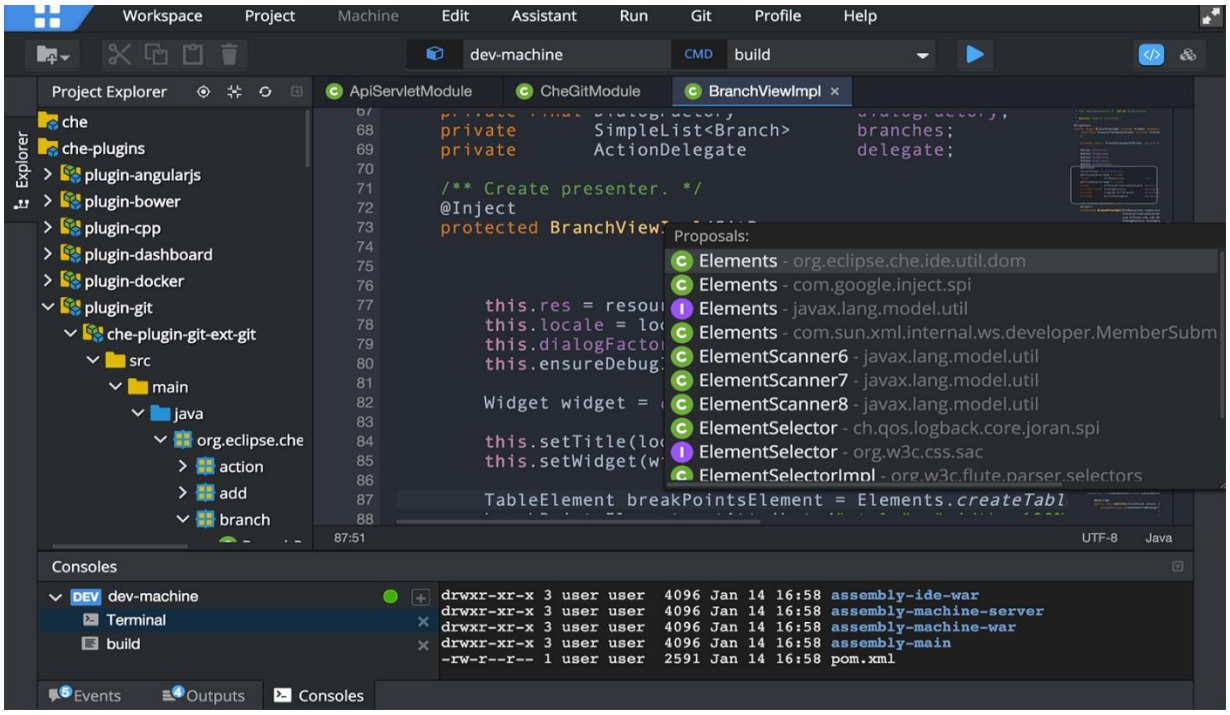

Рисунок 2.14 – Среды разработки Eclipse

Eclipse – среда разработки свободно интегрированных модульных кроссплатформенных приложений рисунок 2.14.

- Свободная интеграция собственных модулей;
- имеются подключаемый модуль для групповой разработки;
- имеется модуль проверки ошибок Bugzilla.

Project Rider

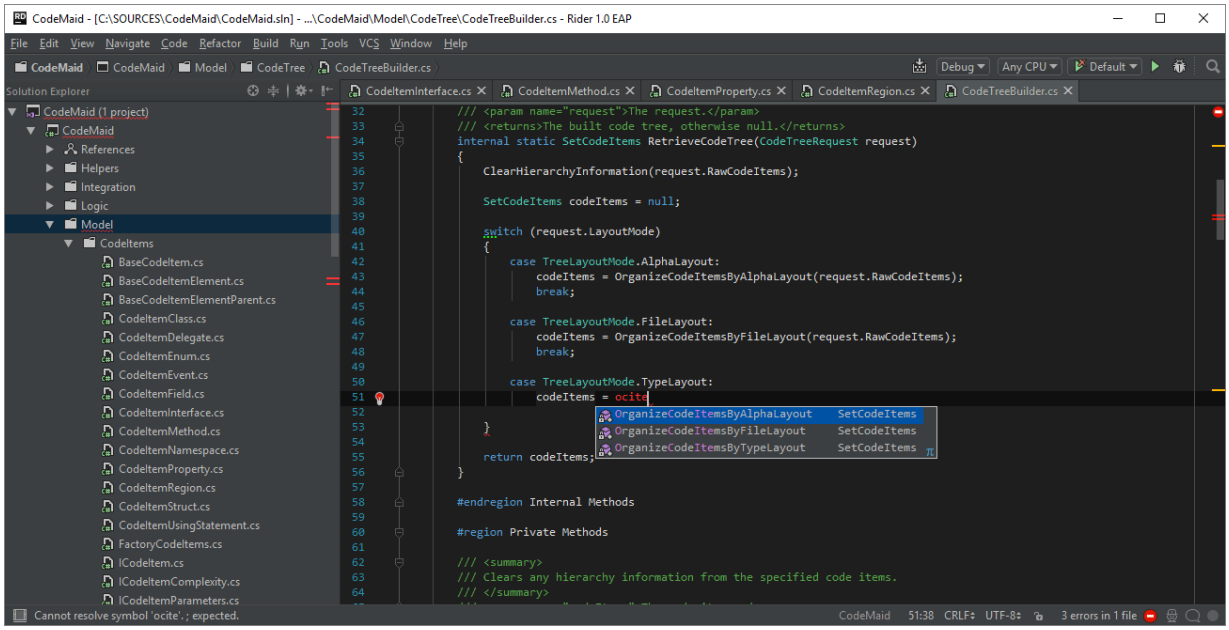

Рисунок 2.16 – Среды разработки Project Rider

Project Rider – кроссплатформенная среда разработки на платформе .Net рисунок 2.16.

- Кроссплатформенность;
- система контроля версий;
- совместимость с Visual Studio.

Visual Studio

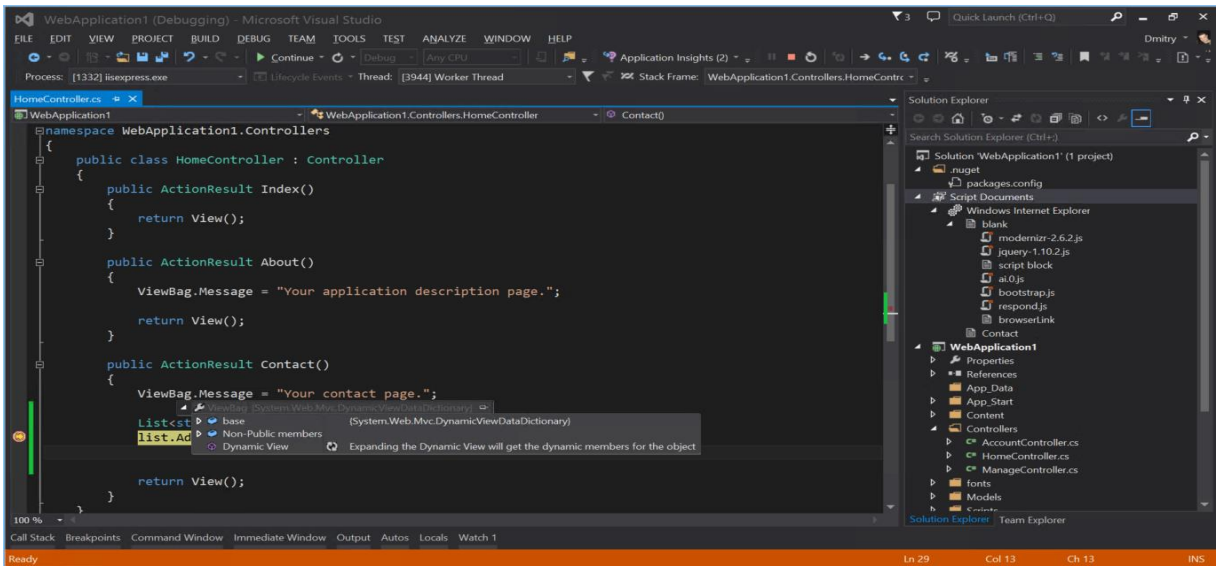

Рисунок 2.17 – Среды разработки Visual Studio

Visual Studio – интегрированная среда разработки программного обеспечения и ряд других инструментальных средств рисунок 2.17.

- Встроенный контроль за выполнением многопоточного кода;
- запись происходящего во время отладки;
- наличие бесплатной редакции Community.

MonoDevelop

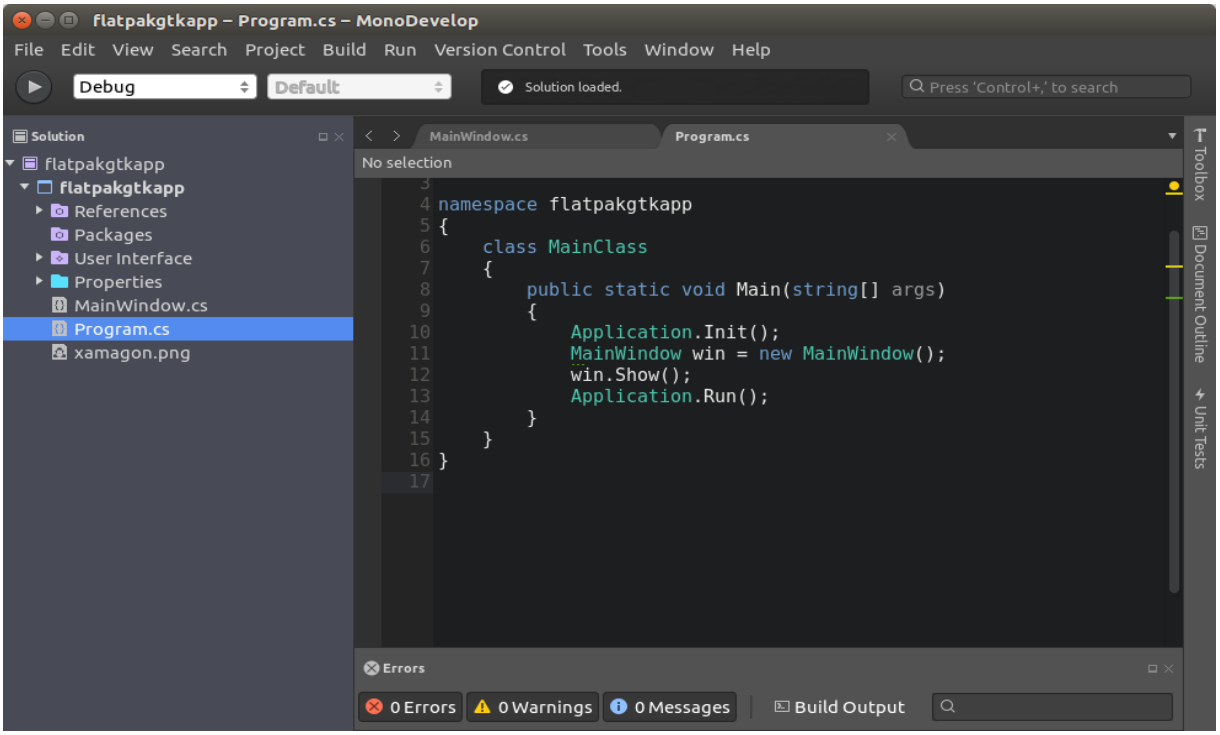

Рисунок 2.17 – Среды разработки MonoDevelop

MonoDevelop – является частью проекта Mono рисунок 2.17 [9].

- Визуальный конструктор форм (GTK#);
- автоматическое создание бинарных пакетов и архивов по завершению компиляции;
- множество стандартных шаблонов.

Из предложенных вариантов сред разработок наиболее перспективной выглядит Visual Studio, так как она содержит в себе возможности всех остальных предложенных вариантов, а также, имеет ряд функций, отсутствующих у прочих сред разработки. Из серьезных недостатков Visual Studio имеет только большую потребность в ресурсах компьютера.

2.4 Выбор систем управления контроля версий

Системы управления контроля версий служит для того, чтобы вести любые проекты, связанные с разработкой программного обеспечения [10]. Она позволяет

хранить локально или облачно изменения, внесенные в проект, чтобы в случае неудачного внедрения кода можно было откатить код обратно к рабочей версии. Также, она очень хорошо подходит для группы разработчиков тем, что изменения в коде можно заливать в удаленный репозитории, то есть каждый из группы разработчиков может писать отдельный модуль, а потом соединять в едино все модули это показано на рисунке 2.19.

Рисунок 2.19 – Схема работы системы контроля версии

Существует не мало систем управления контроля версий, но в основном большое распространения получили Git и Mercurial, о которых дальше и пойдет речь.

Git

Git бесплатная программа систем контроля версий с открытым исходным кодом является кроссплатформенным так, что его можно использовать на Windows, Mac OS X и Linux/Unix. Легок в освоении, хорошо подходит для больших и малых групп разработчиков. С его освоением проблем нет, так как по нему очень много литературы и бесплатных курсов в YouTube, где подробно рассказывается, как им пользоваться.

Работать с Git можно как через консольное приложение, так и через графическое приложение. Для начинающих разработчиков подойдет графическое приложение, так как оно интуитивно понятно. Одно из таких графических приложений TortoiseGit показан на рисунке 2.20

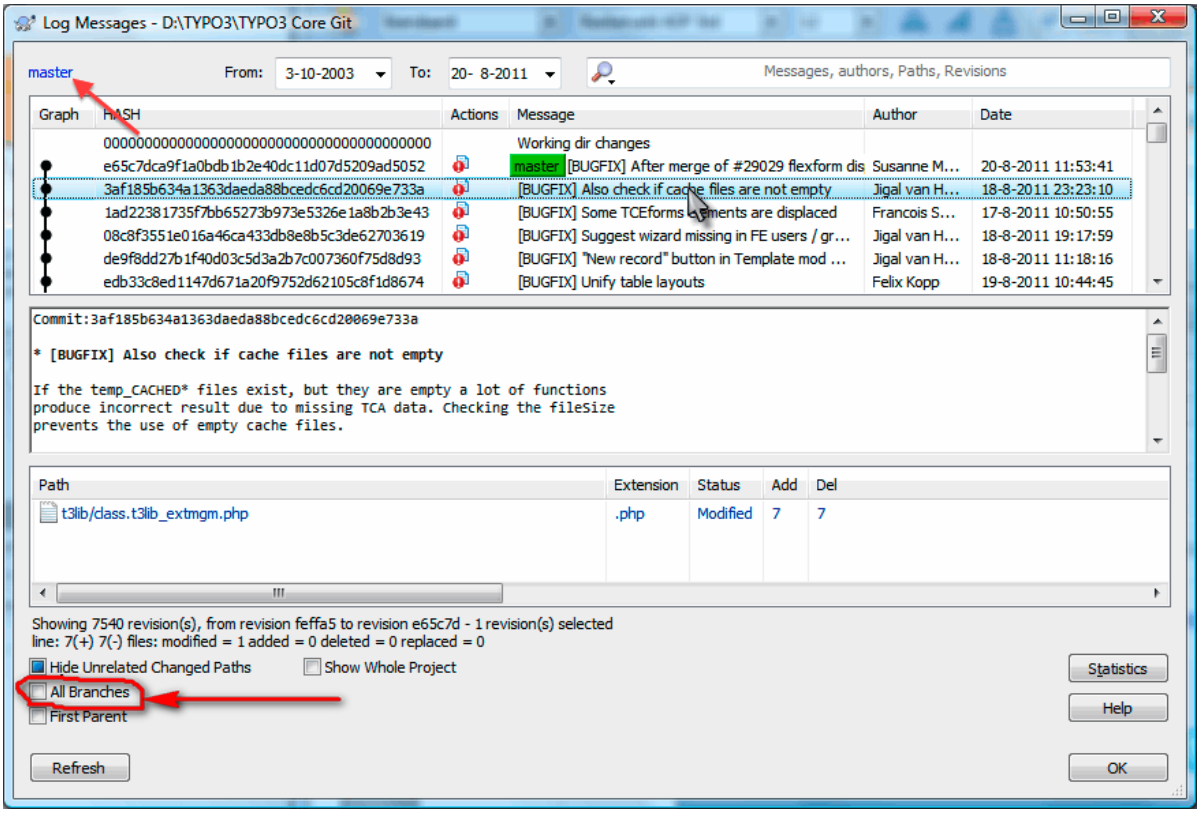

Рисунок 2.20 – Графический интерфейс TortoiseGit

В приложении TortoiseGit наглядно показано, как осуществляется введение контроля версии кода. Тут видно, какое последнее изменение было внесено, кто является автором этого модуля, дату и время изменения и последнюю действующую версию.

Тоже самое касаемо и консольной версии Git, где все изменения видны, но только в консольном виде для получения информации и заливки изменений в репозитории нужно писать команды в консольном приложении рисунок 2.21.

```
$ git status
On branch ver-1.0
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        test-file-for-ver-1.0.php

no changes added to commit (use "git add" and/or "git commit -a")

Nikita_Sp@Nikita_SpN MINGW64 /c/WebServers/home/lesson/www (ver-1.0)
$ git add .

Nikita_Sp@Nikita_SpN MINGW64 /c/WebServers/home/lesson/www (ver-1.0)
$ git commit -m "Test file for branch was added and readme modificated"
[ver-1.0 14d0325] Test file for branch was added and readme modificated
 2 files changed, 4 insertions(+), 1 deletion(-)
 create mode 100644 test-file-for-ver-1.0.php

Nikita_Sp@Nikita_SpN MINGW64 /c/WebServers/home/lesson/www (ver-1.0)
$ git checkout master
Switched to branch 'master'

Nikita_Sp@Nikita_SpN MINGW64 /c/WebServers/home/lesson/www (master)
$
```

Рисунок 2.21 – Пример консольного приложения Git

Преимущества Git в том, что очень много репозиториев поддерживают его, например, такой репозиторий как BitBucket, который не так давно отказался поддерживать Mercurial. Из-за это многим компаниям пришлось делать выбор: менять репозиторий или выбирать Git и переносить.

Mercurial

Mercurial точно также, как и Git, является системой контроля версий и больших отличий от Git не имеет.

Mercurial кроссплатформенный имеет открытый исходный код, подходит для разработок в больших и маленьких проектов. Удобный и дружественный интерфейс интуитивно понятен для любого разработчика.

У Mercurial есть два типа версий: консольная версия и графическая.

Графическая версия Mercurial TortoiseHg

На рисунке 2.22 показан интерфейс TortoiseHg. Здесь можно вести несколько проектов, также, есть основная линия и неосновные линии. То есть, можно увидеть, когда и какие изменения были внесены.

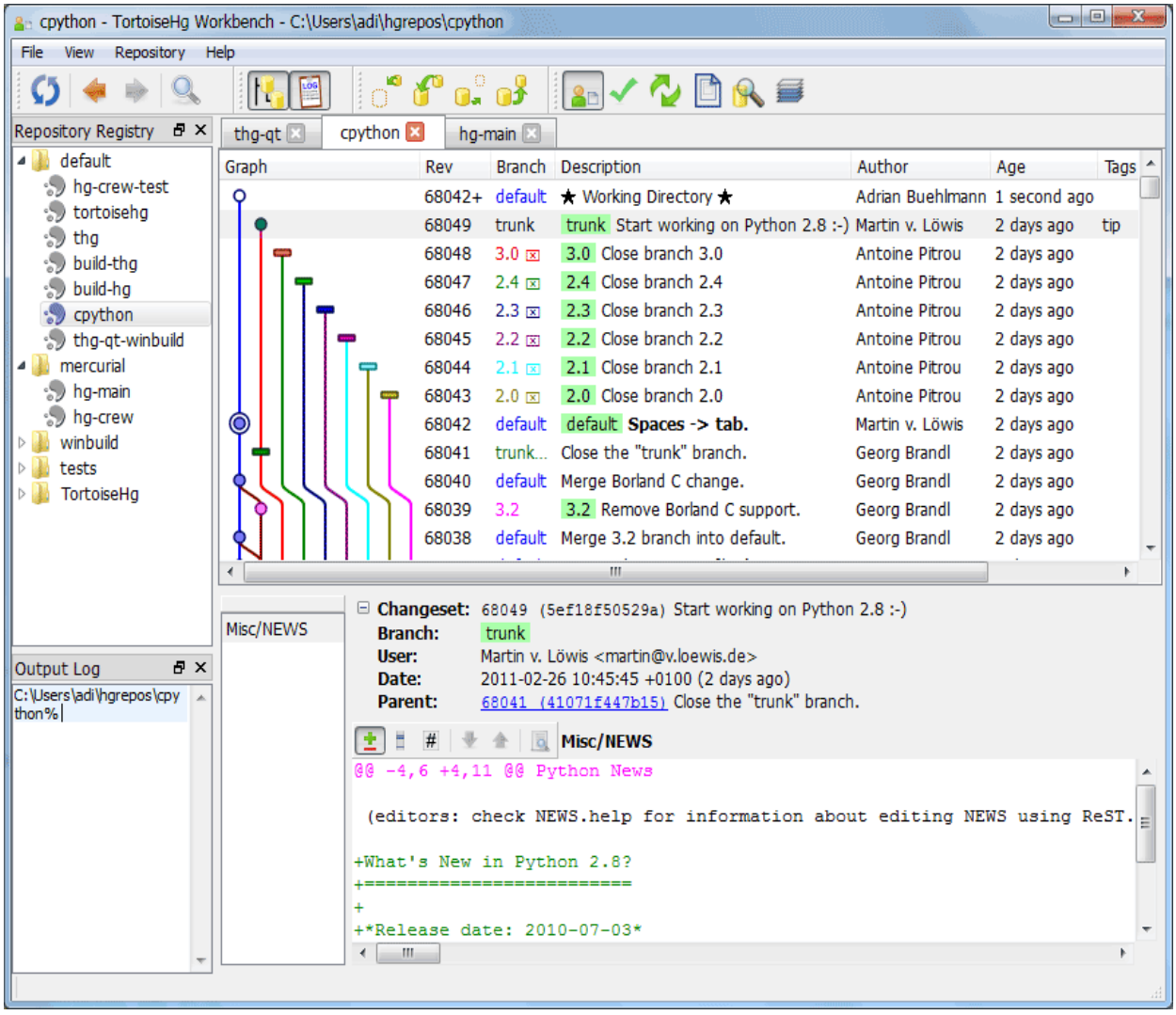

Рисунок 2.22 – Графический интерфейс TortoiseHg

Консольное приложение Mercurial показан на рисунке 2.23

```
output of hg log:
changeset:   2:70eb0ca9d264
tag:         tip
user:        Mr. Johnson
date:        Sun Nov 20 11:20:00 2011 +0100
summary:     Copy and move.

changeset:   1:487d7a20ccbc
user:        Mr. Johnson
date:        Sun Nov 20 11:11:00 2011 +0100
summary:     Say Hello World, not just Hello.

changeset:   0:a5ecbf5799c8
user:        Mr. Johnson
date:        Sun Nov 20 11:00:00 2011 +0100
summary:     Initial commit.
```

Рисунок 2.23 – Консольное приложение Mercurial

Выше были рассмотрены два приложения контроля версий Git и Mercurial, большой разницы между ними нет, кроме одной, очень важной, особенности, то что Git поддерживают большинство репозиториев.

Так что, можно остановиться на выборе Git. Так как, данное приложение является одним из самых распространённых и его поддерживает практически все репозитории.

Выводы по второй главе

Из рассмотренных вариантов был сделан объективный выбор. То что разработка приложения будет на языке C#, а средой разработки станет Visual Studio 2019 с использование СУБД SQLite так как это оптимальный вариант для решения поставленной задачи.

Приложения контроля пакетов версий в данной работе применятся не будут так как это излишне, но в больших проектах без этого является обязательным условием.

3 СОЗДАНИЕ ПРОГРАММЫ ПОД ANDROID ОС

3.1 Структура приложения на языке C#

Для разработки приложения на языке C# была использована подключаемая библиотека Xamarin, обеспечивающая кроссплатформенность.

Структурная схема отображена на рисунке 3.1

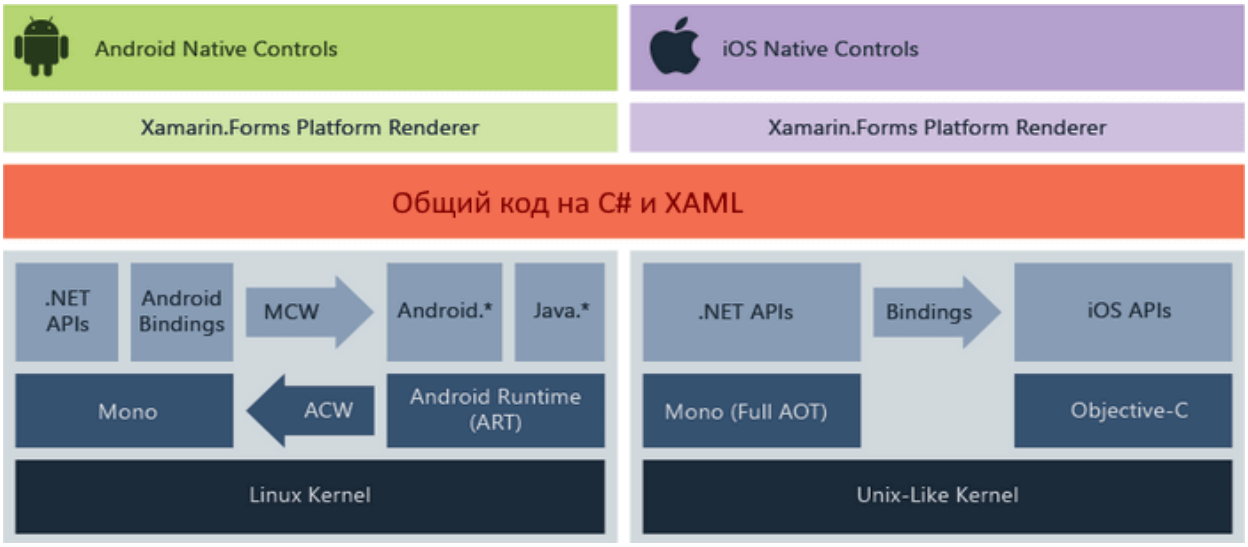

Рисунок 3.1 – Структурная схема работы приложения на платформе Xamarin

Xamarin работает на любой платформе Framework Mono, который предоставляет аналог .NET Framework, но с открытым исходным кодом. Mono работает на любых платформах Linix, MacOS и Windows.

Каждая отдельная платформа Xamarin использует ряд субплатформ. Такие как:

- Xamarin.Android – библиотеки предназначенная для разработки на ОС Android;
- Xamarin.iOS – библиотеки предназначенная для разработки на ОС iOS.

Эти субплатформы играют важную роль, с их помощью приложения отправляют запросы на устройства под управлением ОС Android или iOS.

Благодаря этому, можно писать общий код для всех платформ.

Структура приложения использует шаблон для написания кода MVVM(Model-View-ViewModel). Эта структура позволяет разделить код на две части: визуальная(Frontend) и аппаратная(Backend) рисунок 3.2

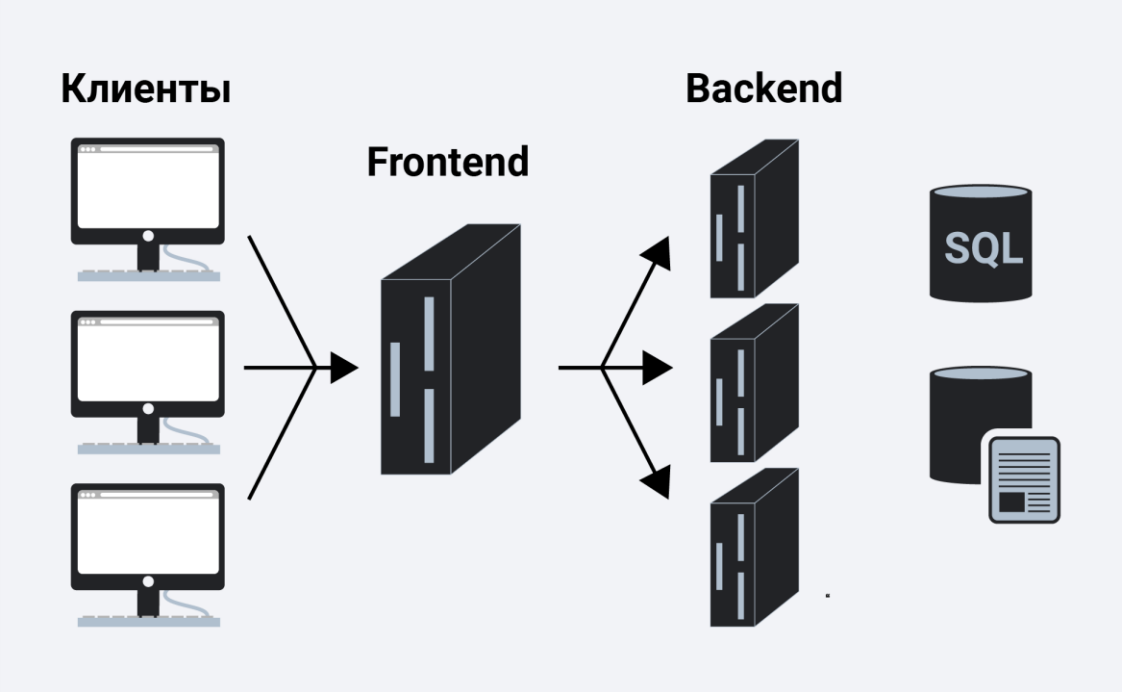

Рисунок 3.2 – Графическое представление frontend и backend

Первая часть структуры – визуальная, то есть, пользователи любой программы первым делом видят работу дизайнера. В его работу входит:

- пользовательский интерфейс;
- работа с языком разметки XML;
- графический дизайн;
- анимация на страницах;
- оптимизация графики.

Для написания скоростного интерфейса требуется разработчик графического дизайна. Он следит за скоростью пользовательского интерфейса, то есть, чтобы страница не загружалась слишком долго [11].

Вторая часть структуры backend или функциональная часть программы. Backend – это функциональная часть приложения, описывающая действие программы. Например, нажатие кнопок вызывает обработчик событий. Такие как:

- переход на другую страницу;
- сохранение изменений в базу данных;
- удаление элемента из базы данных;
- подключение математических модулей;
- И т.д.

Написание качественной программы требует глубоких познаний frontend и backend. Но так как объем информации содержащийся в каждом из них очень

велик, и в компаниях занимающимися разработкой приложений ими занимаются разные сотрудники, тем самым делая сложные проекты.

Код приложения составляется из двух языков XML и C#. XML (eXtensible Markup Language) – расширяемый язык разметки страниц, представляет собой набор тегов и атрибутов. Атрибуты могут наследоваться другими тегами, тем самым упрощая разработку и масштабируемость кода. XML позаимствовал свой функционал у html и их структура и семантика языка очень похожи, то есть его можно применять на любом устройстве с разными по ширине и длине экранов из адаптивной верстки.

В наше время XML стал набирать популярность и очень много современных платформ используют его из-за простоты использования.

На C# пишется весь функционал приложения, а метод InitializeComponent() инициализирует страницу XML. Тем самым связывает код C# и XML.

3.2 Взаимодействие с базой данных

Чтобы приложение начало взаимодействовать с базой данных SQLite, ее нужно подключить к проекту через “NuGet Package Manager” скачав библиотеку sqlite-net-pcl на официальном сайте SQLite рисунок 3.3.

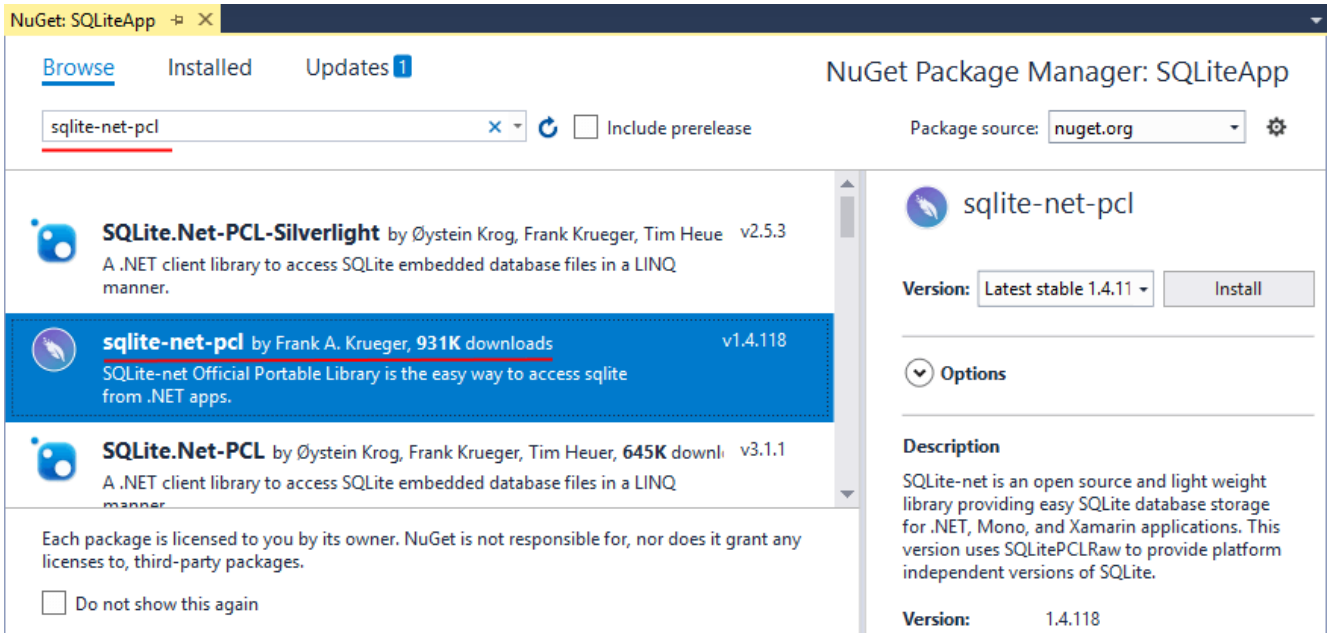

Рисунок 3.3 – Скачивание библиотеке через NuGet Package Manager

Библиотека sqlite-net-pcl представляет собой хранилищем объектов и обращение к базе данных построенных на языке C# как к обычным объектам этого

языка. Таблицы пишутся как обычный класс языка C#, а столбцы как поля этого класса [12].

При создании таблиц прежде всего необходимо определить, как будет исполнена связь между таблицами в базе данных. Это очень важный этап проектирования в разработке программы и для дальнейшей ее модификации. Существует несколько типов связей таблиц:

- многие ко многим;
- один ко многим;
- один к одному.

Чтобы определить связь таблиц, с начало нужно определить сущности, которые будут в проекте.

Сущность «Пользователь» определяет пользователей данного приложения. В нее входят:

- Id пользователя;
- Email пользователя;
- Пароль пользователя.

Сущность «Начальник подразделения» определяет начальника подразделения и группу его подчиненных. В нее входят:

- Id начальника;
- Id подразделения.

Сущность «Подразделение» определяет подразделение и работников подразделения [12]:

- Id подразделения;
- Id работника.

Сущность «Задания» определяет кто и кому выдает задание, и состояние задания. В нее входят:

- Id задания;
- Id начальника;
- Id работника;
- Сложность задания;
- Дата выдачи задания.
- Состояние задание (выполнено или не выполнено)

Сущность «Архив заданий» определяет, работника и выполненные им задания. В нее входят:

- Id работника;
- Id задания;
- дата выполнения;

- сложность задания.

Сущность «Приборы» определяет номера прибора и дату следующей поверки. В нее входит:

- Id прибора;
- номер прибора;
- дата следующей поверки.

При проектировании базы данных было принято решение. Из-за сложной связи данных, сделать связь многие ко многим за исключением таблицы «Приборы», так как она не зависит ни от одной таблицы и связей у нее пока нет, но в дальнейшем при разработке приложение может и появиться [12].

Для того чтобы при работе с приложение не возникли проблемы, связанные с базой данных, таблицы должны быть в одной из нормальных форм. Всего существуют 6 нормальных форм.

Нормальные формы – это условия по которым структура таблицы будет считаться нормальной. Нормальные формы служат для того чтобы избежать избыточности зависимостей атрибутов и дальнейших аномалий которые усложняют обработку информации или потерю данных. Основные обозначения, применяющиеся при проектировании баз данных рисунок 3.4

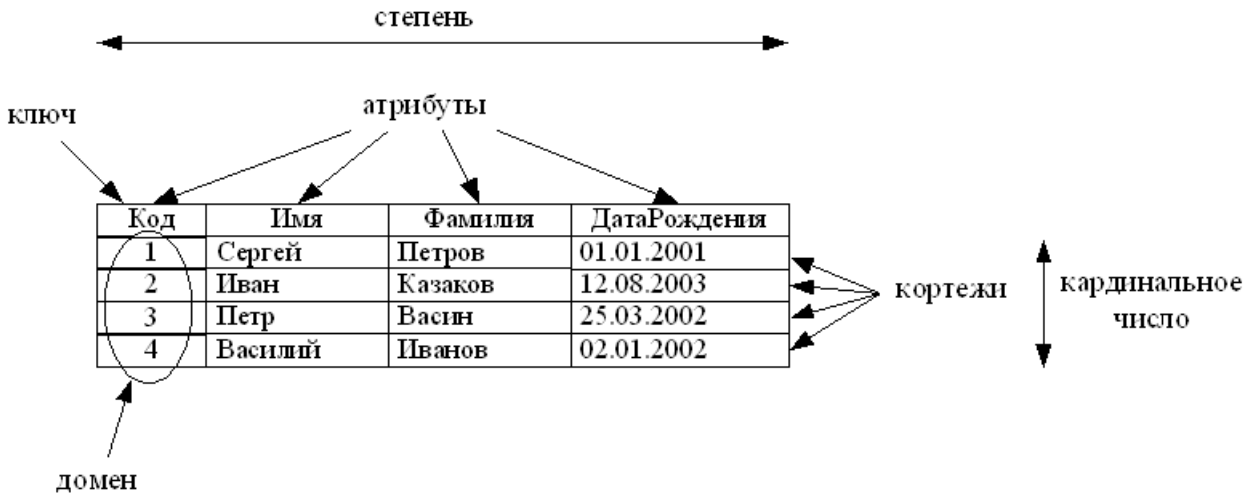

Рисунок 3.4 – Основные обозначения

Существует шесть нормальных форм:

- таблица считается в первой нормальной форме, когда поля таблицы являются простыми. В кортеже не должно быть повторяющихся атрибутов;

- таблица считается во второй форме, если она не противоречит первой нормальной форме и все не ключевые атрибуты неприводимо зависят от ключевого атрибута;
- таблица находится в третьей нормальной форме, когда она не противоречит второй нормальной форме и все не ключевые атрибуты не транзитивно зависят от ключевого атрибута;
- таблица находиться в четвертой нормальной форме, не нарушает условий третьей нормальной формы и все нетипичные универсальные зависимости являются функциональными от её потенциальных ключей.
- таблица находиться в пятой нормальной форме, если не нарушает условий четвертой нормальной форме и не имеет сложные зависимостей между атрибутами;
- таблица находиться в шестой нормальной форме только тогда, она удовлетворяет все простым зависимостям соединения атрибутов.

В данном приложение все таблицы находятся в третьей нормальной форме так как выполняются все ее условия.

База данных должна храниться на сервере чтобы приложение могло взаимодействовать с ней удаленно, но поскольку все хостинги платные то для данной выпускной квалификационной работы такой вариант рассматриваться не будет и взаимодействие с базой данных будет локальным в рамках одного приложения. Также в этой работе будет реализован один модуль с одной таблицей «Приборы».

3.3 Алгоритмы работы приложения

В данном приложение били разработаны алгоритмы, для правильного его функционирования. Рассмотрим первый алгоритм по фильтрации запрещенных знаков. Если не делать фильтрацию в полях ввода пользователем, то какой-нибудь злоумышленник может этим воспользоваться. В поля ввода должны вводится только строчные переменные. Это делается для того чтобы нельзя было написать какой-либо код для обращения к базе данных. Последствие от таких запросов могут быть серьезными такие как:

- потеря данных;
- хищение данных;
- вывод из строя сервера.

Алгоритм по обработки строк представлен на рисунке 3.5 [14].

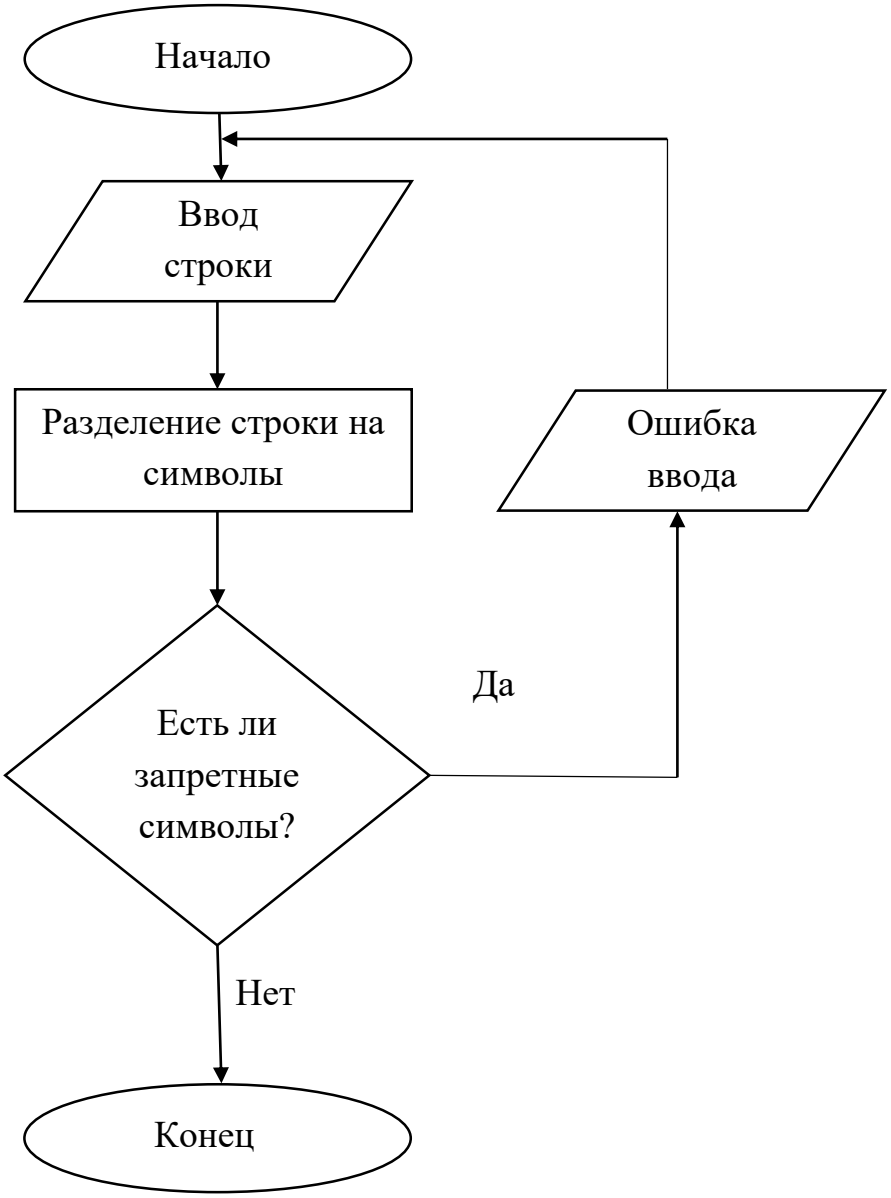

Рисунок 3.5 – Алгоритм обработки строки

Благодаря этому алгоритму нельзя будет написать, запрещенные знаки тем самым увеличивается безопасность приложения.

Алгоритм по обработки входа пользователя в систему рисунок 3.6 [14].

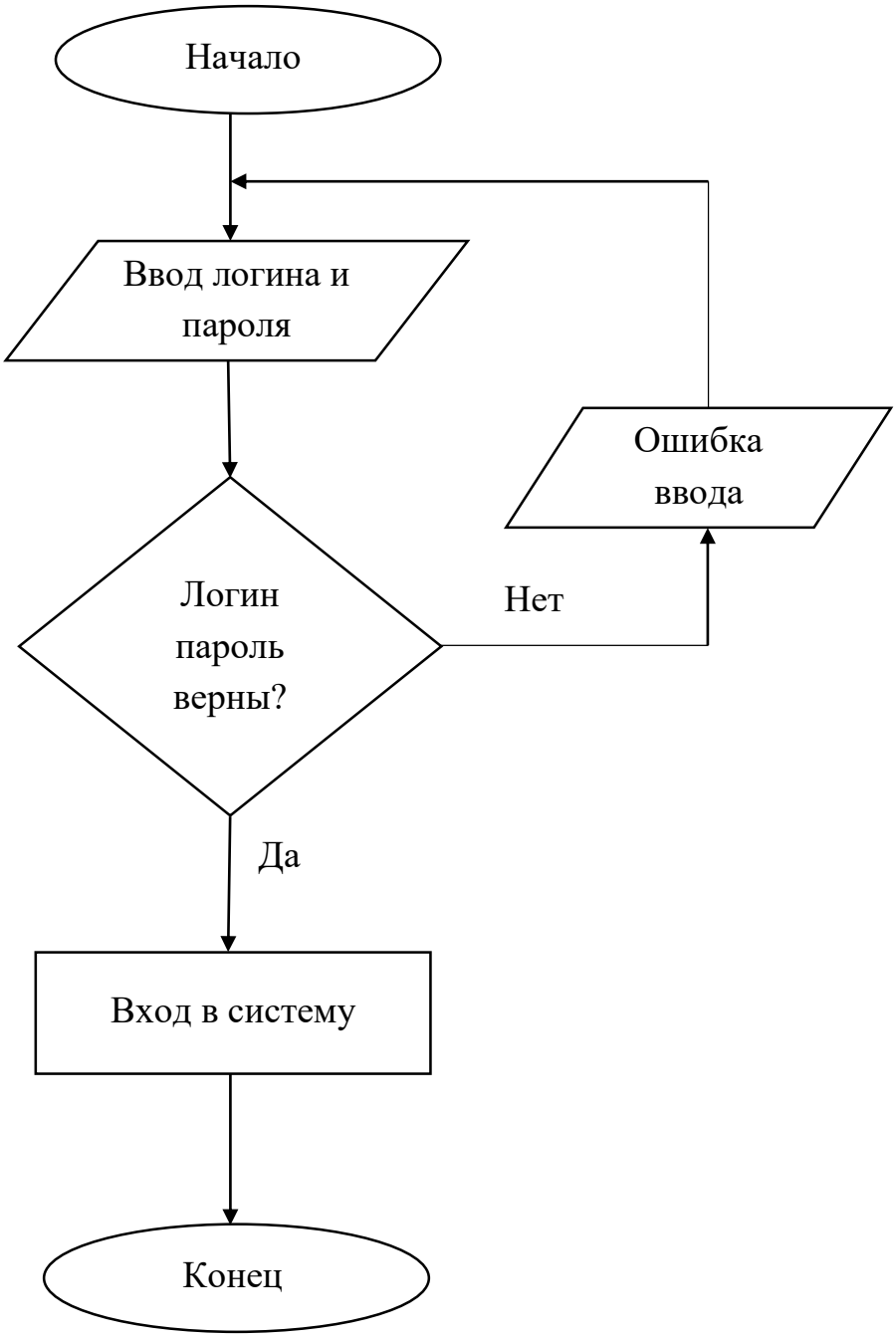

Рисунок 3.6 – Инициализация пользователя

Алгоритм по доступности функционала приложения рисунок 3.7. Этот алгоритм разделяет интерфейс на подчиненного и начальника [14].

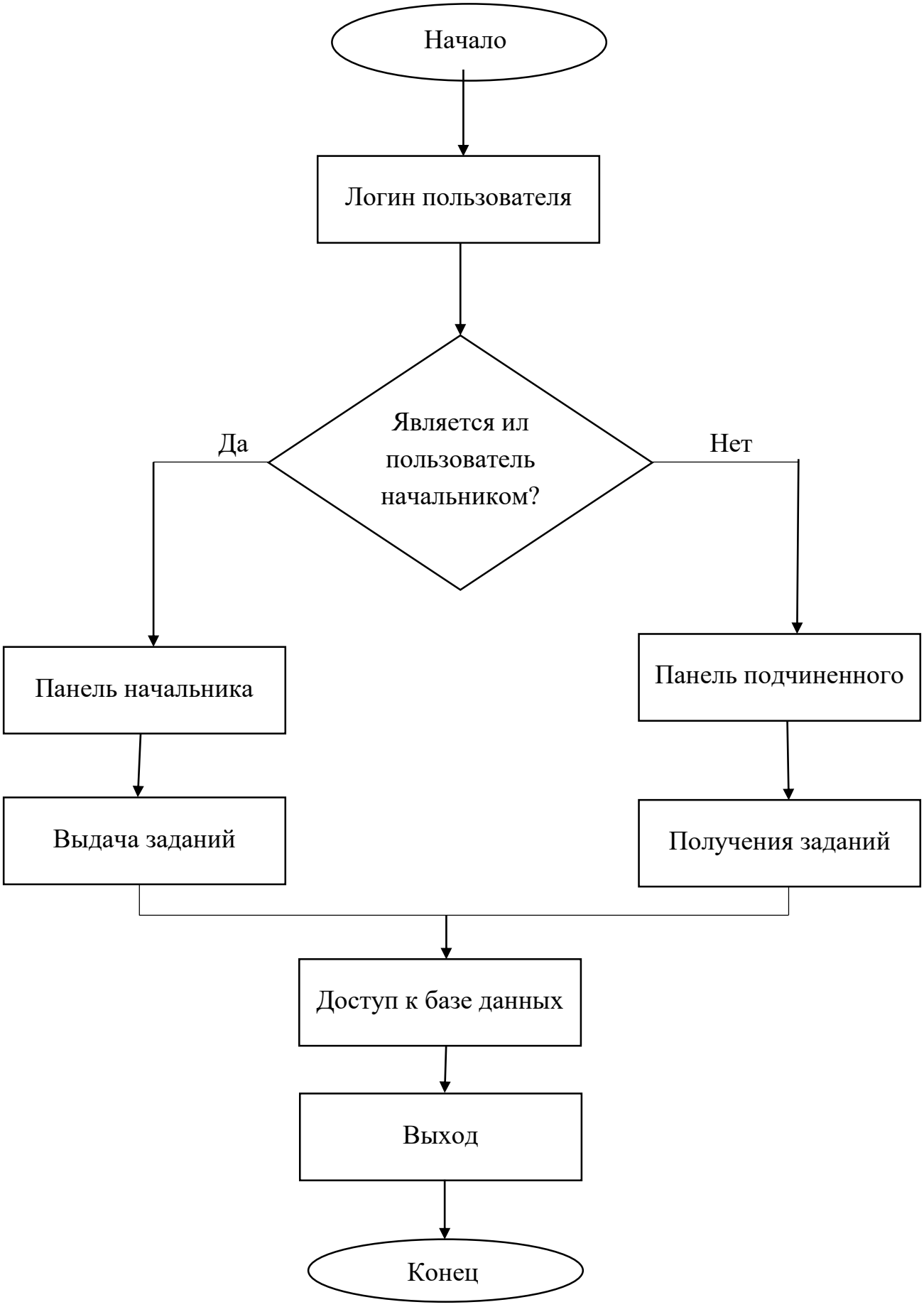

Рисунок 3.7 – Алгоритм интерфейса

Алгоритм для работы с базой данных на удаление, рисунок 3.8

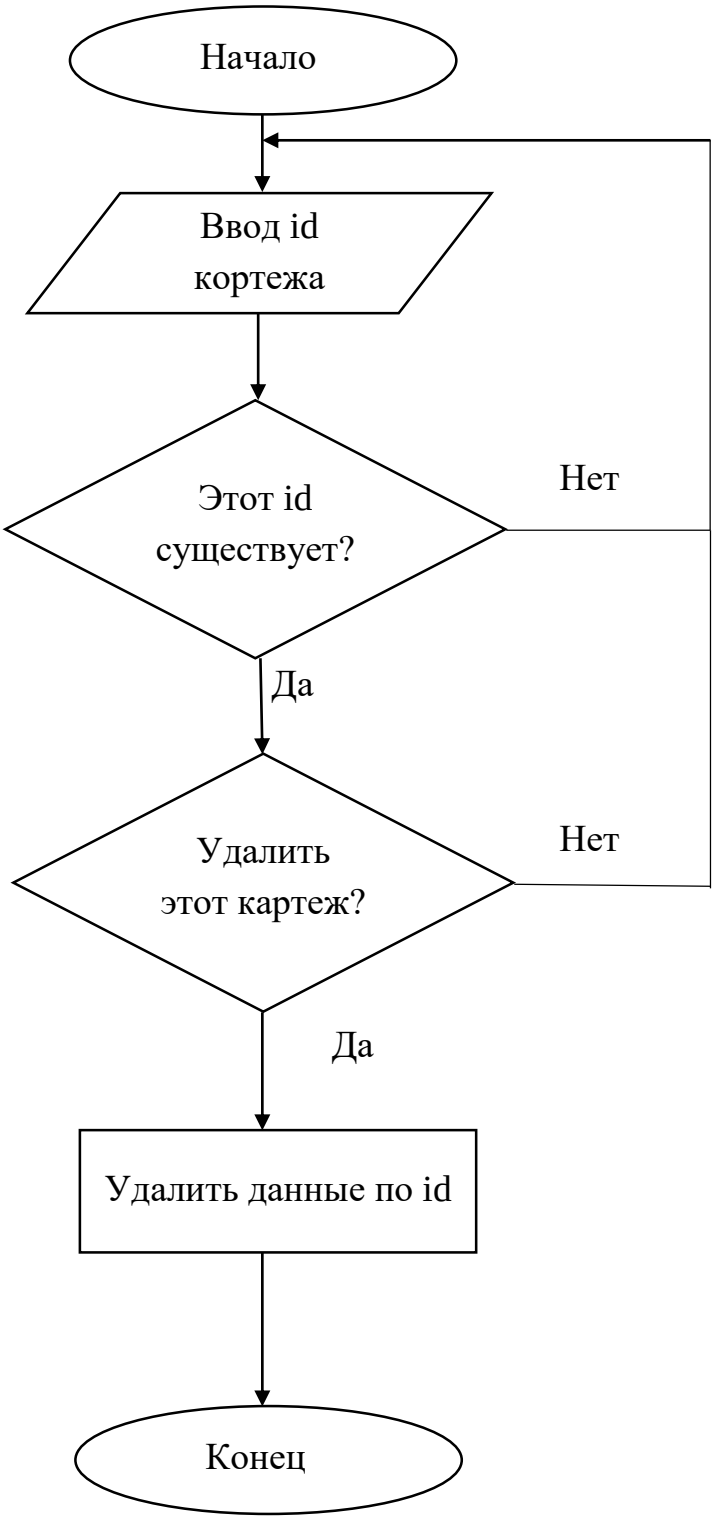

Рисунок 3.8 – Алгоритм удаление из базы данных

Алгоритм для работы с базой данных на запись рисунок 3.9

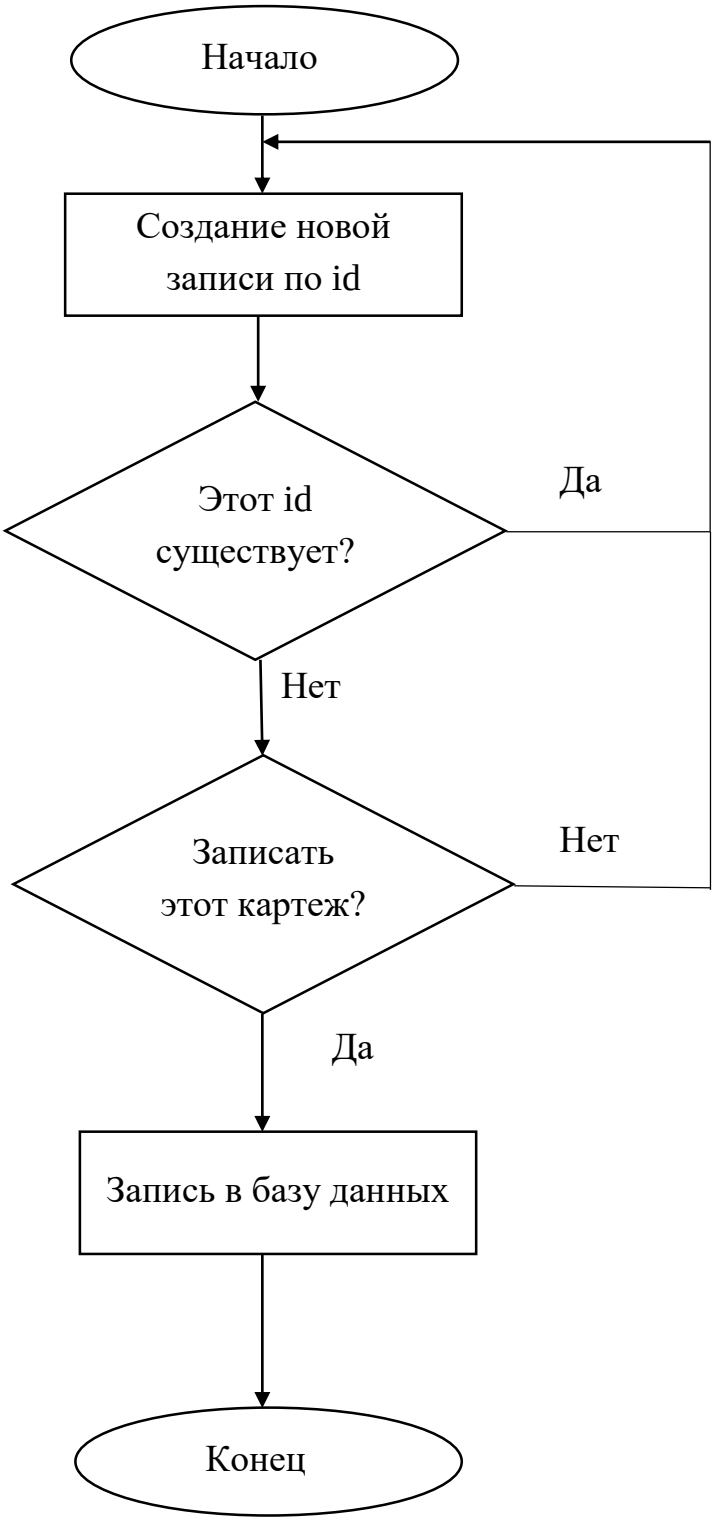

Рисунок 3.9 – Алгоритм сохранение в базу данных

Алгоритмы позволяют корректно обрабатывать ошибки без экстренного завершения программы.

3.4 Загрузка приложения на телефон

Чтобы установить приложение на телефон, необходимо выполнить ряд действий. Зайти в настойки телефона и установить режим разработчика, для этого надо, найти номер сборки и нажать на нее 7 раз рисунок 3.10

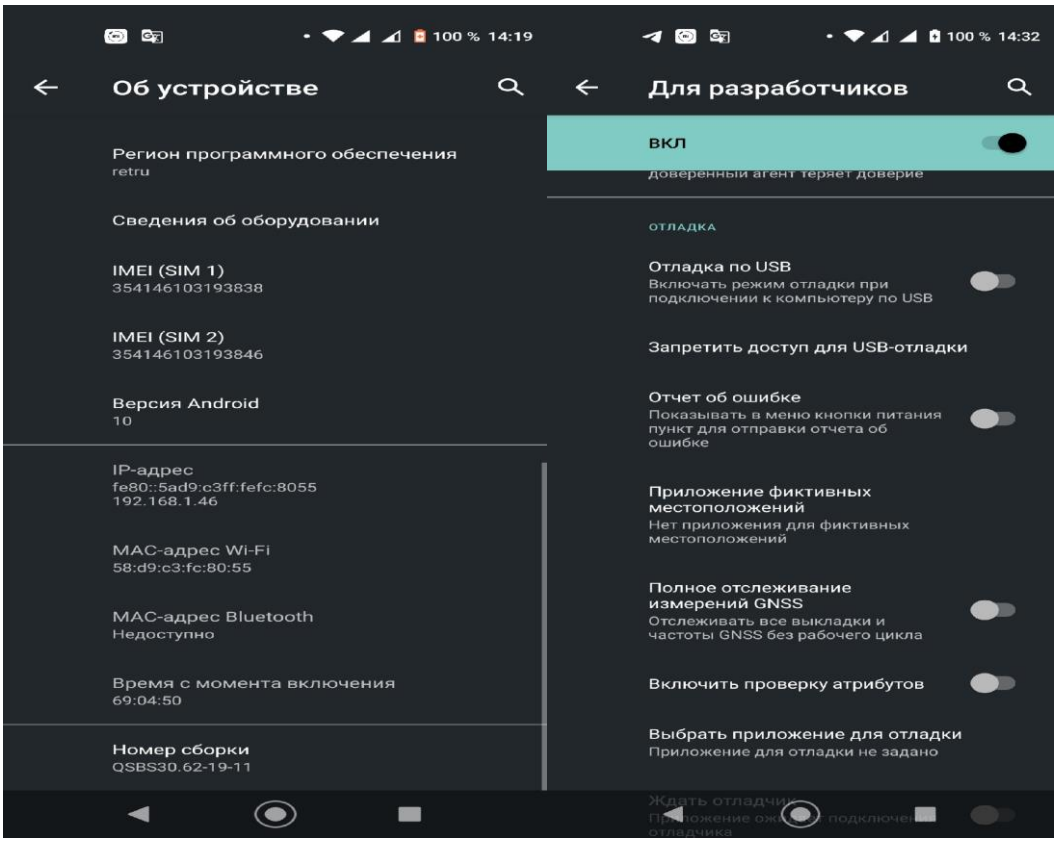

Рисунок 3.10 – Настройки телефона

Следующий шаг подключить телефон по USB кабелю. При подключении вылезет окно Отладка по USB. Выбираем «разрешить» рисунок 3.11.

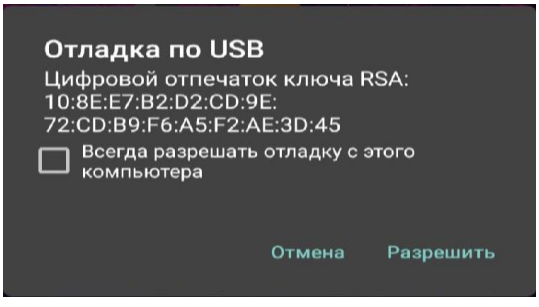

Рисунок 3.11 – Разрешение на отладку

Чтобы загрузить готовое приложение на телефон у Visual Studio есть инструменты для загрузки приложения. По умолчанию в настройках стоит функция «Debug» она служит для того чтобы отлаживать приложение на телефоне без загрузки на устройство. Готовую продукцию загружают с помощью функции «Release» рисунок 3.12.

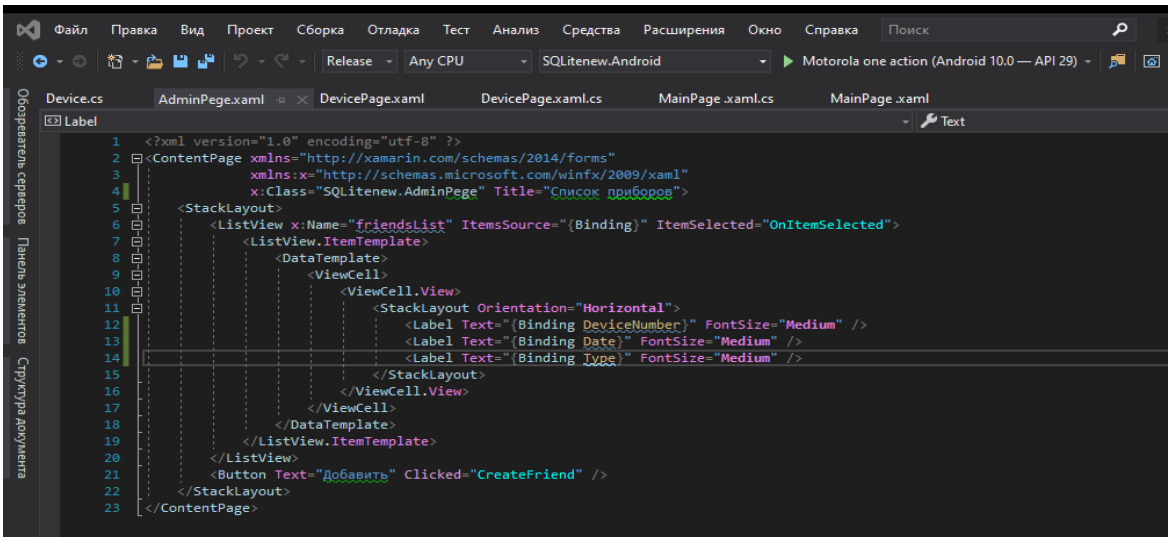

Рисунок 3.12 – Загрузка приложения на телефон

После всех этих действий, приложение готово к использованию.

3.4 Работа приложения

При первом запуске, приложение предлагает ввести логин и пароль рисунок 3.13

Рисунок 3.13 – Форма логина

После входа появляется список существующих приборов

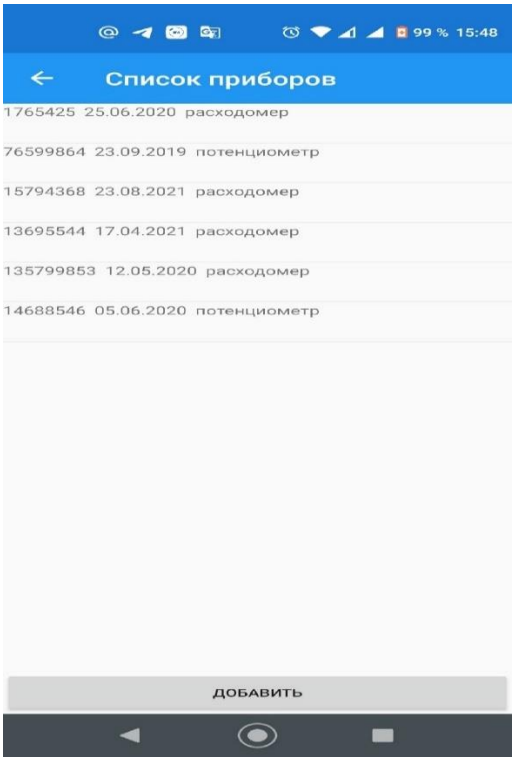

Рисунок 3.14 – Меню списка приборов

С помощью кнопки «добавить» можно добавить новый прибор в список приборов.

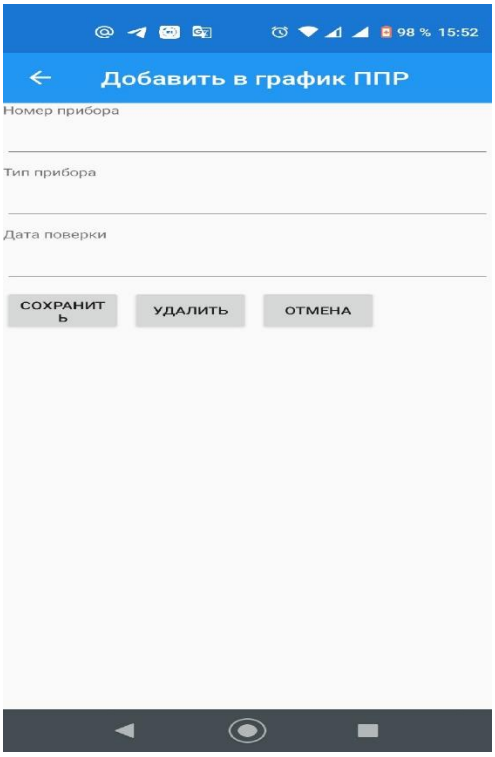

Рисунок 3.15 – Меню добавления прибора

После добавление прибора в список, его можно редактировать.

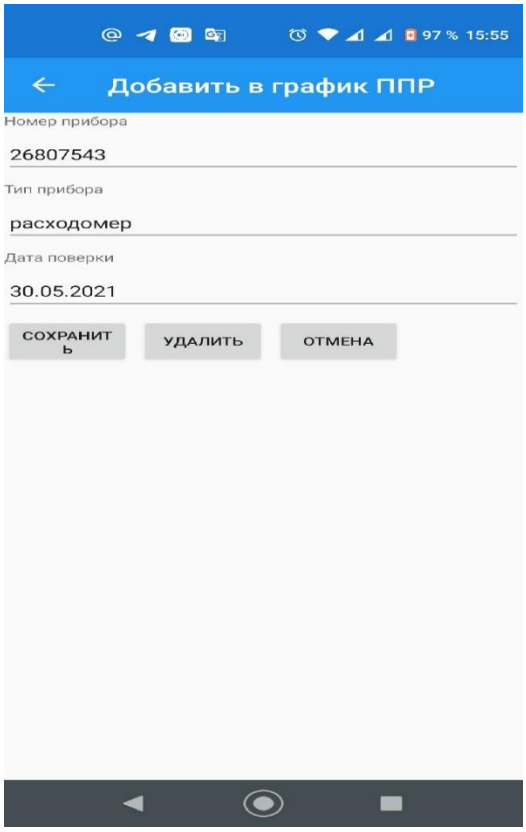

Рисунок 3.16 – Редактирование параметров прибора

Также те приборы, которые попали в брак можно удалить из базы данных.

Рисунок 3.17 – Удаление прибора из базы данных

Выводы по третьей главе

В третьей главе была проделана работа по разработке базы данных. Была рассмотрена библиотека с помощью которой было написано приложение, и был разработан один модуль этого приложения.

В рамках данной выпускной квалификационной работы был создан один модуль приложения. Так как объемы работы очень большие и такие приложение разрабатывают группа разработчиков в составе:

- один Team Leader;
- два backend разработчика;
- два frontend разработчика;
- один дизайнер.

ЗАКЛЮЧЕНИЕ

Был проведен анализ для формирования функций приложения, которые требуются участку КИПиА, что позволило перейти к разработке и внедрению приложения в рабочий процесс.

В ходе проектирование была разработана база данных для полного функционирования приложения. База данных содержит в себе шесть таблиц, пять из которых имеют сложные связи между собой, и одна простая таблица «Приборы», которая и была задействована в данной ВКР. Создано программное обеспечение в объеме одного главного модуля учета приборов.

Была проведена исследовательская работа для создания приложения, разработано устройство базы данных и алгоритмов программного обеспечения.

Созданное приложение соответствует требованием предприятия и успешно внедрено в рабочий процесс. Что позволяет качественнее планировать предшествующий рабочий процесс и сокращать время подачи распоряжений, проверки документов и данных о работе сотрудников.

В результате, это экономит примерно 1,5 – 2 часа рабочего времени начальника участка и позволяет без понижения эффективности труда обходиться без вакансии бригадира. Для остальных работников использование приложения, также, помимо экономии времени, создает дополнительные преимущества в виде переноса актуальных сведений непосредственно в базу данных.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Ёранссон, А. Эффективное использование потоков в операционной системе Android / А. Ёранссон. – М.: ДМК-Пресс, 2018. – 312 с

2 Ахмедханлы, Д.М. Основы алгоритмизации и программирования. Учебное пособие / Д.М. Ахмедханлы, Н.В. Ушмаева. – Тольятти: Изд-во ТГУ, 2016. – 120 с.

3 Тарасов, С.В, СУБД для программиста. Базы данных изнутри / С.В. Тарасов. – М.: СОЛОН-Пресс, 2015. – 320 с.

4 Моргунов, Е.П. Основы языка SQL. Учебное пособие / Е.П. Моргунов, Е.В. Рогова, П.В. Лузанова. – СПб.: «БХВ-Петербург», 2018. – 336 с.

5 Маркин, А.В. Программирование на SQL. Часть 1 / А.В. Маркин. – М.: Юрайт, 2017. – 363 с.

6 Тейт, Б. Семь языков за семь недель / Б. Тейт. – М.: ДМК-Пресс, 2017. – 384 с.

7 Васильев, А.Н. Python на примерах. Практический курс по программированию / А.Н. Васильев. – СПб.: Наука и Техника, 2016. – 432 с.

8 Шарп, Д. Microsoft Visual C#. Подробное руководство / Д. Шарп. – 8-е изд. – СПб.: Питер, 2017. – 848 с.

9 Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Джонсон, Д. Влиссидес, Р. Хелм. – СПб.: Питер, 2015. – 368 с.

10 Щербаков, В.П. Моделирование и автоматизированное проектирование систем управления. Учебное пособие / В.П. Щербаков, О.О. Павловская. – Челябинск: Издательский центр ЮУрГУ, 2015. – 32 с.

11 Пселтис, Э. Потоковая обработка данных. Конвейер реального времени / Э. Пселтис. – М.: ДМК-Пресс, 2018. – 218 с.

12 Коннолли, Т. Базы данных. Проектирование, реализация и сопровождение / Т. Коннолли, К. Бегг – 3-е изд. – М.: Издательский дом «Вильямс», 2017. – 1440 с.

13 Фило, В.Ф. Теоретический минимум по Computer Science / В.Ф. Фило. – СПб.: Питер, 2018. – 224 с.

14 Стивенс, Р. Алгоритмы. Теория и практическое применение / Р. Стивенс. – М.: Эксмо, 2016. – 544 с.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ.А Листинг программы управления

App.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:d="http://xamarin.com/schemas/2014/forms/design"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d"
        x:Class="SQLitenew.App">
   <Application.Resources>

   </Application.Resources>
</Application>
using System;
using System.IO;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
```

App.cs

```
namespace SQLitenew
{
   public partial class App : Application
   {
      public const string DATABASE_NAME = "friends.db";
      public static DeviceRepository database;
      public static DeviceRepository Database
      {
         get
         {
            if (database == null)
            {
               database = new DeviceRepository(
                  Path.Combine(
```

```
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
DATABASE_NAME));
            }
            return database;
        }
    }
    public App()
    {
        InitializeComponent();

        MainPage = new NavigationPage(new MainPage());
    }

    protected override void OnStart()
    {
    }

    protected override void OnSleep()
    {
    }

    protected override void OnResume()
    {
    }
  }
}
```

AdminPege.xaml

```
?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="SQLitenew.AdminPege" Title="Список приборов">
    <StackLayout>
        <ListView x:Name="friendsList" ItemsSource="{Binding}"
ItemSelected="OnItemSelected">
            <ListView.ItemTemplate>
                <DataTemplate>
```

```
                <ViewCell>
    <ViewCell.View>
                    <StackLayout Orientation="Horizontal">
                      <Label  Text="{Binding  DeviceNumber}"  FontSize="Medium"
/>
                      <Label Text="{Binding Date}" FontSize="Medium" />
                      <Label Text="{Binding Type}" FontSize="Medium" />
                    </StackLayout>
                  </ViewCell.View>
                </ViewCell>
              </DataTemplate>
            </ListView.ItemTemplate>
          </ListView>
          <Button Text="Добавить" Clicked="CreateFriend" />
        </StackLayout>
    </ContentPage>

    AdminPege.cs
    using System;
    using System.ComponentModel;
    using Xamarin.Forms;

    namespace SQLitenew
    {

      public partial class AdminPege : ContentPage
      {
        public AdminPege()
        {
          InitializeComponent();
        }
        protected override void OnAppearing()
        {
          friendsList.ItemsSource = App.Database.GetItems();
          base.OnAppearing();
        }
```

```
        // обработка нажатия элемента в списке


    private async void OnItemSelected(object sender, SelectedItemChangedEventArgs
e)
        {
            Device selectedFriend = (Device)e.SelectedItem;
            DevicePage devicePage = new DevicePage();
            devicePage.BindingContext = selectedFriend;
            await Navigation.PushAsync(devicePage);
        }
        // обработка нажатия кнопки добавления
        private async void CreateFriend(object sender, EventArgs e)
        {
            Device friend = new Device();
            DevicePage devicePage = new DevicePage();
            devicePage.BindingContext = friend;
            await Navigation.PushAsync(devicePage);
        }
    }
}

DevicePage.xaml
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="SQLitenew.DevicePage" Title="Добавить в график ППР">
    <StackLayout>
        <Label Text="Номер прибора" />
        <Entry Text="{Binding DeviceNumber}" />
        <Label Text="Тип прибора" />
        <Entry Text="{Binding Type}" />
        <Label Text="Дата поверки" />
        <Entry Text="{Binding Date}" />
        <StackLayout Orientation="Horizontal">
            <Button Text="Сохранить" Clicked="SaveDevice" />
            <Button Text="Удалить" Clicked="DeleteDevice" />
```

```
            <Button Text="Отмена" Clicked="Cancel" />
        </StackLayout>

    </StackLayout>
</ContentPage>
```

DevicePage.cs

```
using System;
using Xamarin.Forms;

namespace SQLitenew
{

    public partial class DevicePage : ContentPage
    {
        public DevicePage()
        {
            InitializeComponent();
        }
        private void SaveDevice(object sender, EventArgs e)
        {
            var device = (Device)BindingContext;
            if (!String.IsNullOrEmpty(device.DeviceNumber))
            {
                App.Database.SaveItem(device);
            }
            this.Navigation.PopAsync();
        }
        private void DeleteDevice(object sender, EventArgs e)
        {
            var device = (Device)BindingContext;
            App.Database.DeleteItem(device.Id);
            this.Navigation.PopAsync();
        }
        private void Cancel(object sender, EventArgs e)
```

```
        {
            this.Navigation.PopAsync();
        }
    }
}
```

DeviceRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using SQLite;

namespace SQLitenew
{
    public class DeviceRepository
    {
        SQLiteConnection database;
        public DeviceRepository(string databasePath)
        {
            database = new SQLiteConnection(databasePath);
            database.CreateTable<Device>();
        }
        public IEnumerable<Device> GetItems()
        {
            return database.Table<Device>().ToList();
        }
        public Device GetItem(int id)
        {
            return database.Get<Device>(id);
        }
        public int DeleteItem(int id)
        {
            return database.Delete<Device>(id);
        }
        public int SaveItem(Device item)
```

```
    {
            if (item.Id != 0)
            {
               database.Update(item);
               return item.Id;
            }
            else
            {
               return database.Insert(item);
            }
         }
      }
   }
```

MainPage .xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="SQLitenew.MainPage" Title="Вход в систему">
   <StackLayout>
      <Entry x:Name="loginEntry" Placeholder = "    Login    "
             HorizontalOptions="Center" VerticalOptions ="StartAndExpand"/>

      <Entry  x:Name="passwordEntry"   Placeholder  =  "  Password  "  IsPassword
="True"
             HorizontalOptions="Center" VerticalOptions ="StartAndExpand"/>


<Button Text="Войти" Clicked="ButtonAdmin" />
   </StackLayout>
</ContentPage>
```

MainPage .cs

```
using System;
using System.ComponentModel;
using Xamarin.Forms;
```

```
namespace SQLitenew
{
    // Learn more about making custom code visible in the Xamarin.Forms previewer
    // by visiting https://aka.ms/xamarinforms-previewer
    [DesignTimeVisible(false)]
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        private async void ButtonAdmin(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new AdminPege());
        }
    }
}
```

Device.cs

```
using SQLite;

namespace SQLitenew
{
    [Table("Device")]
    public class Device
    {
        [PrimaryKey, AutoIncrement, Column("_id")]
        public int Id { get; set; }

        public string DeviceNumber { get; set; }
        public string Type { get; set; }
        public string Date { get; set; }
    }
}
```