ВВЕДЕНИЕ

1 Анализ предметной области

1.1 Техническое задание на разработку

1.2 Программа, методика и результаты испытаний

2 Проектирование

2.1 Выбор СУБД

2.2 Выбор среды разработки

2.3 Выбор метода разработки

3 Листинг кода

ЗАКЛЮЧЕНИЕ

**ВВЕДЕНИЕ**

В современном мире мобильные приложения стали неотъемлемой частью нашей повседневной жизни, оказывая существенное влияние на различные сферы деятельности. Они позволяют пользователям быстро получать доступ к информации, общаться, делать покупки и управлять своими задачами. С каждым годом наблюдается рост популярности приложений, что связано с их удобством и функциональностью. Именно поэтому разработка специализированных мобильных приложений, таких как приложение для автомастерских, становится особенно актуальной.

Одним из ключевых аспектов создания мобильных приложений являются базы данных. Базы данных — это организованные структуры для хранения, управления и обработки информации. Они позволяют приложениям эффективно обрабатывать большие объемы данных, обеспечивая пользователям быстрый доступ к информации и возможность ее обновления. Без хорошо спроектированной базы данных приложение может работать неэффективно или даже терять свои функциональные возможности. Поэтому понимание работы с базами данных является основополагающим при разработке любого приложения.

Мобильное приложение для автомастерских может стать незаменимым инструментом как для владельцев бизнеса, так и для их клиентов. Оно может облегчить процессы записи клиентов, управления запасами, отслеживания статуса обслуживания и уведомления пользователей о готовности автомобиля. Кроме того, приложение может предоставить полезные ресурсы, такие как советы по уходу за автомобилем и информацию о часто задаваемых вопросах. Все это не только повышает уровень сервиса в автомастерских, но и создает более удобные условия для взаимодействия с клиентами.

В данном отчете я отображу свой процесс создания мобильного приложения для автомастерских.

**1 Анализ предметной области**

Анализ предметной области является важнейшим этапом проектирования информационной системы для управления учебным процессом в деканате. Рассмотрим ключевые аспекты данной предметной области.

* Основной сущностью системы являются Клиенты. Каждый клиент характеризуется уникальным идентификатором, ФИО, номером телефона и адресом проживания. Эти атрибуты позволяют однозначно идентифицировать посетителей автосервиса.
* Сущность автомобиль содержит информацию о транспорте клиентов. Основные атрибуты включают идентификатор автомобиля, бренд, мощность, год выпуска и цвет. Эти данные необходимы для распределения нагрузки на персонал и определение требуемых запчастей.
* Механики представляют собой важный компонент системы. По каждому автомеханику есть аналогичная клиентам информация, но с добавлением категории. Эта информация используется при формировании рабочих планов.
* Особое значение имеет сущность Работы, которая отражает процесс проведения ремонтных услуг. Атрибуты включают идентификатор мероприятия, клиента, механика, автомобиля, автомастерской и затраченных запчастей. Эти данные позволяют анализировать рабочий процесс.
* Сущность Автомастерские отображает филиалы компании. Автомастерские включают в себя такие атрибуты, как адрес, номер телефона и название.
  1. **Техническое задание на разработку**

В качестве предметной области выбрана тема «Автомастерские».

Этап разработки раздела «Общие сведения»:

* + Полное наименование БД: «RepairShopDb»;
  + Шифр темы:06;
  + Предприятие-разработчик системы: Дом;
  + Предприятие-заказчик системы: Рязанский Станкостроительный колледж РГРТУ».

Система создается на основании технического задания (ТЗ). ТЗ на АС является основным документом, определяющим требования и порядок создания автоматизированной системы, в соответствии с которым проводится разработка АС и ее приемка при вводе в действие. Кроме того, при создании системы используются ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы».

Порядок оформления и предъявления заказчику результатов работы по созданию системы определяется после получения начальной версии продукта, в которой должны быть реализованы все основные функции, определенные в ТЗ и утвержденные заказчиком.

Этап разработки раздела «Назначение и цели создания системы»:

* Вид автоматизируемой деятельности: Распределение задач;
* Перечень автоматизируемых процессов: распределение работ между сотрудниками университета;
* Наименование и значение показателей, которые будут достигнуты в результате внедрения БД: уменьшение затрат рабочего времени на ввод, редактирование и удаление данных о выполняемых задач.

Этап разработки раздела «Характеристики объекта автоматизации»

“RepairShopManager” автоматизирована с использованием sqlite и visual studio. Управление клиентскими данными обеспечивают персонализированный сервис. Автоматизация оптимизирует процессы, повышает эффективность и способствует анализу выполняемой работы между сотрудниками.

* 1. **Программа, методика и результаты испытаний**

Общее положение

Структура и оформление документа устанавливается в соответствии с ГОСТ 19.105-78.

Составление информационной части (аннотации и содержания) является необязательным.

Документ «Программа и методика испытаний» должен содержать следующие разделы:

* Объект испытаний.
* Цель испытаний.
* Состав предъявляемой документации.
* Технические требования.
* Порядок проведения испытаний.
* Методы испытаний.

В зависимости от особенностей документа допускается вводить дополнительные разделы.

СОДЕРЖАНИЕ РАЗДЕЛОВ

Объект испытаний.

Наименование: «Практика-07».

Область применения: Университет.

Обозначение испытуемой программы: П-9

Цель испытаний:

* Проверить выводятся ли данные БД.
* Проверить работу добавления, редактирования и удаления данных.

Состав предъявляемой документации:

1. Техническое задание (ТЗ):

* Важнейший документ, определяющий требования к продукту или системе, которые подлежат испытаниям. ТЗ является основой для создания тестов и оценки их эффективности.

1. План испытаний:

* Ключевой документ, определяющий общий подход и стратегию проведения испытаний. Включает в себя информацию о методах тестирования, расписании, ресурсах, и критериях успешности.

1. Протоколы испытаний:

* Записи, сделанные во время проведения тестов, являются важными документами, фиксирующими результаты, обнаруженные дефекты и другие замечания в процессе тестирования. Протоколы предоставляют детальную информацию о ходе испытаний.

1. Руководство оператора

* Документ, предоставляющий оператором подробные инструкции по настройке, безопасной эксплуатации и обслуживанию. Также содержит техническое обучение, справочную информацию, процедуры управления и контроля, необходимые для правильного функционирования оборудования.

Технические требования:

* Процессор i3 12100
* Оперативная память 1 ГБ или более.
* Жесткий диск 500 МБ свободного пространства на диске.
* Операционная система Windows 10.

Требования к программной документации включают в себя определённые стандарты и ожидания по созданию, оформлению и содержанию документации для программного продукта. Ниже приведены основные требования к программной документации:

1. Полнота:

* Документация должна полностью охватывать все аспекты программного продукта, включая функциональность, интерфейсы, архитектуру, установку, настройку и т.д.

1. Понятность:

* Документация должна быть написана ясно и понятно для целевой аудитории, включая разработчиков, тестировщиков, администраторов и конечных пользователей.

1. Актуальность:

* Документация должна быть актуальной и отражать текущую версию программного продукта. В случае изменений в продукте, документация должна обновляться соответственно.

1. Структурированность:

* Документация должна быть хорошо структурированной с ясными разделами и подразделами. Информация должна быть организована логично для удобства восприятия.

1. Стандартизация:

* Документация должна соответствовать определенным стандартам и шаблонам, принятым в организации или индустрии. Это включает в себя единый стиль, форматирование и терминологию.

1. Доступность:

* Документация должна быть легкодоступной для тех, кто ей пользуется. Это может включать в себя электронные версии, онлайн-ресурсы или интеграцию с средствами справки в продукте.

1. Иллюстрации и примеры:

* Использование графиков, диаграмм, таблиц и примеров кода может значительно улучшить понимание и применимость документации.

1. Совместимость с инструментами:

* Документация должна быть предоставлена в форматах, совместимых с используемыми инструментами, такими как средства контроля версий, средства управления проектами, средства анализа кода и т.д.

1. Обратная связь:

* Документация должна предоставлять механизмы для обратной связи пользователей, чтобы можно было быстро исправлять ошибки или дополнять информацию по необходимости.

1. Управление изменениями:

* Документация должна включать в себя механизмы для отслеживания изменений и обновлений, а также архивацию предыдущих версий документации.

Соблюдение этих требований способствует созданию высококачественной программной документации, которая облегчит понимание, использование и сопровождение программного продукта.

«Требования к техническим характеристикам». Требования к характеристикам программы применительно к условиям эксплуатации:

* Версия Visual Studio 2024 года.

Требование к информационной и программной совместимости:

* Процессор i3 12100.
* Оперативная память 100 МБ или более.
* Жесткий диск 100 ГБ свободного пространства на диске.
* Операционная система Windows 10.

Порядок проведения испытаний:

* Запуск проекта.
* Запуск программы.
* Тестирование имеющихся функций программы.

**2 Проектирование**

**2.1 Выбор СУБД**

Система управления базами данных (СУБД) — это специализированное программное обеспечение, предназначенное для создания, управления и организации данных в базе данных. СУБД позволяет пользователям эффективно хранить, извлекать, обновлять и управлять данными с помощью различных интерфейсов, включая язык SQL.

Основные функции СУБД:

Хранение данных: СУБД обеспечивает структурированное хранение больших объемов информации.

Работа с данными: Позволяет выполнять операции добавления, удаления и модификации данных.

Запросы к данным: Поддерживает язык запросов (например, SQL) для получения необходимой информации.

Безопасность: Предоставляет механизмы доступа и контроля, чтобы защитить данные.

Целостность данных: Обеспечивает соотношение и логику между различными наборами данных.

Резервное копирование и восстановление: Позволяет создавать резервные копии и восстанавливать данные в случае сбоев.

Почему важно серьезно относиться к выбору СУБД:

Производительность: Разные СУБД имеют различные уровни производительности в зависимости от объема данных и сложности операций.

Совместимость: Убедитесь, что выбранная СУБД поддерживает все необходимые технологии и инструменты, которые вы планируете использовать.

Безопасность: Разные СУБД предлагают различные уровни защиты данных. Адекватная безопасность критически важна для защиты конфиденциальной информации.

Легкость использования: Удобство интерфейса и доступность документации могут существенно упростить работу для разработчиков и администраторов.

Обслуживание и поддержка: Наличие активного сообщества или профессиональной поддержки может значительно упростить решение проблем.

Выбор подходящей СУБД может оказать большое влияние на эффективность работы приложения, скорость обработки данных и безопасность информации. Поэтому важно внимательно оценивать все аспекты перед принятием решения.

Для своего проекта я выбрал такую СУБД, как SQLite. Ее плюсы заключаются в следующем:

Простота использования: Очень легкий в настройке и использовании, не требует установки сервера.

Легковесность: Маленький размер библиотеки и базы данных, идеален для мобильных приложений и встраиваемых систем.

Надежность: Поддерживает транзакции ACID, что обеспечивает целостность данных.

Кросс-платформенность: Работает на разных операционных системах без изменений в коде.

Быстродействие: Высокая производительность для небольших и средних объемов данных.

Без установки: Нет необходимости в отдельном сервере или настройке, всё хранится в одном файле.

Поддержка SQL: Использует стандартный язык SQL, что делает его удобным для разработки.

Гибкость: Подходит для множества приложений, от тестов до мобильных и настольных программ.

Совокупность вышеперечисленных аспектов делает SQLite хорошим выбором для разработки мобильного приложения.

**2.2 Выбор среды разработки**

DE (Интегрированная Среда Разработки) — это программа, которая объединяет инструменты для разработки программного обеспечения в одном интерфейсе. Обычно IDE включает:

Редактор кода — для написания и редактирования исходного кода.

Компилятор/интерпретатор — для преобразования кода в исполняемую программу.

Отладчик — для поиска и исправления ошибок в коде.

Управление проектами — для организации файлов и ресурсов.

Интеграция с системами контроля версий — для управления изменениями в коде.

Почему важен правильный выбор IDE:

Удобство и скорость разработки:

Выбор подходящей IDE может значительно ускорить процесс написания кода благодаря функциям автозаполнения, подсветки синтаксиса и рефакторинга.

Поддержка языков программирования:

Разные IDE оптимизированы для разных языков (например, Python, Java, C#). Важно выбрать ту, которая best подходит для вашего проекта.

Инструменты для отладки:

Отладка — критически важный процесс. Хорошая IDE предлагает мощные инструменты для отладки, что помогает быстрее находить и исправлять ошибки.

Поддержка фреймворков и библиотек:

Многие современные разработки требуют использования различных фреймворков. Выбор IDE с встроенной поддержкой нужных вам фреймворков может упростить интеграцию и работу.

Сообщество и поддержка:

Большее сообщество пользователей означает больше ресурсов, таких как плагины, расширения и документация. Это может помочь вам решать возникающие проблемы.

Удобство интеграции:

Некоторые IDE лучше интегрируются с другими инструментами разработки (например, с системами CI/CD), что улучшает рабочий процесс.

Оптимизация рабочего процесса:

Функции, такие как управление задачами, автоматизация сборки и интеграция с облачными сервисами, могут сделать разработку более эффективной.

Выбор IDE — это не просто вопрос удобства, а стратегическое решение, которое может повлиять на продуктивность и качество разработки. Хорошая IDE удовлетворяет потребности конкретного проекта и команды, помогая справляться с задачами быстрее и эффективнее.

Свое мобильное приложение я разрабатывал в Microsoft Visual Studio, которое имеет следующие преимущества:

Многофункциональность: Поддерживает множество языков программирования, таких как C#, C++, VB.NET и многих других.

Интегрированная среда разработки (IDE): Удобный интерфейс с множеством инструментов для разработки, от редактирования кода до отладки.

Отладка и тестирование: Мощные инструменты для отладки и интегрированные юнит-тесты.

Поддержка Git и других систем контроля версий: Удобная работа с репозиториями прямо из IDE.

Расширяемость: Большое количество плагинов и расширений для улучшения функциональности.

Поддержка облака: Легкая интеграция с Azure и другими облачными сервисами.

Шаблоны проектов: Удобные шаблоны для быстрого создания приложений в разных категориях.

Интеграция с базами данных: Удобное взаимодействие с различными СУБД.

Сообщество и поддержка: Огромное сообщество разработчиков и множество ресурсов для обучения.

Удобство совместной работы: Поддержка командной работы и интеграция с системами управления проектами.

**2.3 Выбор метода разработки**

Поскольку фреймворк .NET MAUI не поддерживает привычные инструменты для подключения базы данных, я решил писать методом Code First.

Метод Code First — это подход в разработке приложений, использующийся в контексте работы с базами данных, особенно в Entity Framework (EF). Основная идея заключается в том, что разработчик сначала создаёт модели данных, а база данных формируется автоматически на основе этих моделей.

Преимущества метода Code First:

Удобство разработки: Разработчик может сосредоточиться на бизнес-логике и структуре классов, не отвлекаясь на SQL-запросы и создание таблиц.

Простота миграции: С помощью инструментов миграции можно легко обновлять структуру базы данных при изменениях в моделях.

Читаемость кода: Модели создаются в виде классов, что делает код более понятным и поддерживаемым.

Тестирование: Позволяет легче писать тесты, так как структуры представлены в виде классов.

Также в разработке мне помогал фреймворк Sqlite-net-pcl.

**3 Листинг кода**

Листинг сущностей:

public class Autorization

{

[PrimaryKey, AutoIncrement]

public int Id { get; set; }

public string Login { get; set; }

public string Password { get; set; }

public int AccessLevel { get; set; }

}

namespace RSManagerMobile.Models

{

public class Car

{

[PrimaryKey, AutoIncrement]

public int Id { get; set; }

public string Brand { get; set; }

public string Power { get; set; }

public int Year { get; set; }

public string Color { get; set; }

}

}

namespace RSManagerMobile.Models

{

public class Client

{

[PrimaryKey, AutoIncrement]

public int Id { get; set; }

public string Name { get; set; }

public string Adres { get; set; }

public string Phone { get; set; }

}

}

namespace RSManagerMobile.Models

{

public class Mechanic

{

[PrimaryKey, AutoIncrement]

public int Id { get; set; }

public string Name { get; set; }

public string Adres { get; set; }

public string Phone { get; set; }

public string Category { get; set; }

}

}

namespace RSManagerMobile.Models

{

public class Part

{

[PrimaryKey, AutoIncrement]

public int Id { get; set; }

public int Coast { get; set; }

public string DateOfIssue { get; set; }

public string Category { get; set; }

public string CompletionDate { get; set; }

}

}

namespace RSManagerMobile.Models

{

public class RepairShop

{

[PrimaryKey, AutoIncrement]

public int Id { get; set; }

public string Name { get; set; }

public string Adres { get; set; }

public string Phone { get; set; }

}

}

namespace RSManagerMobile.Models

{

public class Work

{

[PrimaryKey, AutoIncrement]

public int Id { get; set; }

[ForeignKey(nameof(Client))]

public int ClientId { get; set; }

[ForeignKey(nameof(Mechanic))]

public int MechanicId { get; set; }

[ForeignKey(nameof(RepairShop))]

public int RepairShopId { get; set; }

[ForeignKey(nameof(Car))]

public int CarId { get; set; }

[ForeignKey(nameof(Part))]

public int PartId { get; set; }

}

}

Листинг настройки контекста базы данных

namespace RSManagerMobile.Models

{

public class DataBase

{

private readonly SQLiteAsyncConnection \_dataBase;

public DataBase(string dbPath)

{

\_dataBase = new SQLiteAsyncConnection(dbPath);

\_dataBase.CreateTableAsync<Autorization>().Wait();

\_dataBase.CreateTableAsync<Car>().Wait();

\_dataBase.CreateTableAsync<Client>().Wait();

\_dataBase.CreateTableAsync<Mechanic>().Wait();

\_dataBase.CreateTableAsync<Part>().Wait();

\_dataBase.CreateTableAsync<RepairShop>().Wait();

\_dataBase.CreateTableAsync<Work>().Wait();

}

public Task<List<Autorization>> GetUsers()

{

return \_dataBase.Table<Autorization>().ToListAsync();

}

public Task<List<Car>> GetCars()

{

return \_dataBase.Table<Car>().ToListAsync();

}

public Task<List<Client>> GetClietns()

{

return \_dataBase.Table<Client>().ToListAsync();

}

public Task<List<Mechanic>> GetMechanics()

{

return \_dataBase.Table<Mechanic>().ToListAsync();

}

public Task<List<Part>> GetParts()

{

return \_dataBase.Table<Part>().ToListAsync();

}

public Task<List<RepairShop>> GetShops()

{

return \_dataBase.Table<RepairShop>().ToListAsync();

}

public Task<List<Work>> GetWorks()

{

return \_dataBase.Table<Work>().ToListAsync();

}

public Task<int> SaveItem(Autorization item)

{

if (item.Id != 0)

{

return \_dataBase.UpdateAsync(item);

}

else

{

return \_dataBase.InsertAsync(item);

}

}

public Task<int> SaveItem(Car item)

{

if (item.Id != 0)

{

return \_dataBase.UpdateAsync(item);

}

else

{

return \_dataBase.InsertAsync(item);

}

}

public Task<int> SaveItem(Client item)

{

if (item.Id != 0)

{

return \_dataBase.UpdateAsync(item);

}

else

{

return \_dataBase.InsertAsync(item);

}

}

public Task<int> SaveItem(Mechanic item)

{

if (item.Id != 0)

{

return \_dataBase.UpdateAsync(item);

}

else

{

return \_dataBase.InsertAsync(item);

}

}

public Task<int> SaveItem(Part item)

{

if (item.Id != 0)

{

return \_dataBase.UpdateAsync(item);

}

else

{

return \_dataBase.InsertAsync(item);

}

}

public Task<int> SaveItem(RepairShop item)

{

if (item.Id != 0)

{

return \_dataBase.UpdateAsync(item);

}

else

{

return \_dataBase.InsertAsync(item);

}

}

public Task<int> SaveItem(Work item)

{

if (item.Id != 0)

{

return \_dataBase.UpdateAsync(item);

}

else

{

return \_dataBase.InsertAsync(item);

}

}

public Task<int> RemoveItem(Autorization item)

{

return \_dataBase.DeleteAsync(item);

}

public Task<int> RemoveItem(Car item)

{

return \_dataBase.DeleteAsync(item);

}

public Task<int> RemoveItem(Client item)

{

return \_dataBase.DeleteAsync(item);

}

public Task<int> RemoveItem(Mechanic item)

{

return \_dataBase.DeleteAsync(item);

}

public Task<int> RemoveItem(Part item)

{

return \_dataBase.DeleteAsync(item);

}

public Task<int> RemoveItem(RepairShop item)

{

return \_dataBase.DeleteAsync(item);

}

public Task<int> RemoveItem(Work item)

{

return \_dataBase.DeleteAsync(item);

}

}

}

Вспомогательный класс Data для передачи информации между страницами

namespace RSManagerMobile.Models

{

public class Data

{

public static object? item;

public static bool? isAdding;

}

}

Листинг начального окна авторизации

namespace RSManagerMobile

{

public partial class MainPage : ContentPage

{

public MainPage()

{

InitializeComponent();

}

private void btnReg\_Clicked(object sender, EventArgs e)

{

RegisterPage page = new();

Navigation.PushModalAsync(page);

}

private async void btnEnter\_Clicked(object sender, EventArgs e)

{

try

{

var gen = await App.\_db.GetUsers();

var item = gen.Where(x => x.Login == entryLogin.Text).FirstOrDefault();

if (item != null)

{

if (item.Password == entryPassword.Text)

{

await Navigation.PushModalAsync(new GeneralPage());

}

}

else

{

await DisplayAlert("Ошибка!",

"Такого пользователя не существует!",

"Ок");

}

}

catch (Exception ex)

{

await DisplayAlert("Ошибка!", ex.Message, "Ох...");

}

}

}

}

Листинг окна регистрации

public partial class RegisterPage : ContentPage

{

public RegisterPage()

{

InitializeComponent();

}

private async void btnReg\_Clicked(object sender, EventArgs e)

{

try

{

if (entryPassword.Text == entryPassword2.Text)

{

Autorization user = new();

user.Password = entryPassword.Text;

user.Login = entryLogin.Text;

user.AccessLevel = 0;

App.\_db.SaveItem(user);

await Navigation.PushModalAsync(new GeneralPage());

}

else await DisplayAlert("Ошибка!", "Пароли не совпадают!", "Ok");

}

catch (Exception ex)

{

await DisplayAlert("Ошибка!", ex.Message, "Ok");

}

}

private void btnCancel\_Clicked(object sender, EventArgs e)

{

Navigation.PopAsync();

}

}

Листинг главной страницы с выбором таблицы

public partial class GeneralPage : ContentPage

{

public GeneralPage()

{

InitializeComponent();

}

private void btnClients\_Clicked(object sender, EventArgs e)

{

Data.item = new Client();

Navigation.PushModalAsync(new ListPage());

}

private void btnMechanics\_Clicked(object sender, EventArgs e)

{

Data.item = new Mechanic();

Navigation.PushModalAsync(new ListPage());

}

private void btnRepairShops\_Clicked(object sender, EventArgs e)

{

Data.item = new RepairShop();

Navigation.PushModalAsync(new ListPage());

}

private void btnParts\_Clicked(object sender, EventArgs e)

{

Data.item = new Part();

Navigation.PushModalAsync(new ListPage());

}

private void btnWorks\_Clicked(object sender, EventArgs e)

{

Data.item = new Work();

Navigation.PushModalAsync(new ListPage());

}

private void brnCars\_Clicked(object sender, EventArgs e)

{

Data.item = new Car();

Navigation.PushModalAsync(new ListPage());

}

private void btnChangeUser\_Clicked(object sender, EventArgs e)

{

Navigation.PopModalAsync();

}

}

Листинг страницы для просмотра таблиц

public partial class ListPage : ContentPage

{

public ListPage()

{

InitializeComponent();

}

private void btnAdd\_Clicked(object sender, EventArgs e)

{

Data.isAdding = true;

Navigation.PushModalAsync(new ViewPage());

}

private void lvClients\_ItemSelected(object sender, SelectedItemChangedEventArgs e)

{

Data.item = lvClients.SelectedItem;

Navigation.PushModalAsync(new ViewPage());

}

private void lvCars\_ItemSelected(object sender, SelectedItemChangedEventArgs e)

{

Data.item = lvCars.SelectedItem;

Navigation.PushModalAsync(new ViewPage());

}

private void lvMechanics\_ItemSelected(object sender, SelectedItemChangedEventArgs e)

{

Data.item = lvMechanics.SelectedItem;

Navigation.PushModalAsync(new ViewPage());

}

private void lvParts\_ItemSelected(object sender, SelectedItemChangedEventArgs e)

{

Data.item = lvParts.SelectedItem;

Navigation.PushModalAsync(new ViewPage());

}

private void lvShops\_ItemSelected(object sender, SelectedItemChangedEventArgs e)

{

Data.item = lvShops.SelectedItem;

Navigation.PushModalAsync(new ViewPage());

}

private void lvWorks\_ItemSelected(object sender, SelectedItemChangedEventArgs e)

{

Data.item = lvWorks.SelectedItem;

Navigation.PushModalAsync(new ViewPage());

}

private async void ContentPage\_Loaded(object sender, EventArgs e)

{

if (Data.item is Client)

{

lvClients.IsVisible = true;

lvClients.ItemsSource = await App.\_db.GetClietns();

}

else if (Data.item is Car)

{

lvCars.IsVisible = true;

lvCars.ItemsSource = await App.\_db.GetCars();

}

else if (Data.item is Mechanic)

{

lvMechanics.IsVisible = true;

lvMechanics.ItemsSource = await App.\_db.GetMechanics();

}

else if (Data.item is Part)

{

lvParts.IsVisible = true;

lvParts.ItemsSource = await App.\_db.GetParts();

}

else if (Data.item is RepairShop)

{

lvShops.IsVisible = true;

lvShops.ItemsSource = await App.\_db.GetShops();

}

else if (Data.item is Work)

{

lvWorks.IsVisible = true;

lvWorks.ItemsSource = await App.\_db.GetWorks();

}

}

private void btnBack\_Clicked(object sender, EventArgs e)

{

Navigation.PopModalAsync();

}

}

Листинг страницы для просмотра, добавления, редактирования и удаления записей

public partial class ViewPage : ContentPage

{

public ViewPage()

{

InitializeComponent();

}

private void btnEdit\_Clicked(object sender, EventArgs e)

{

UpdateItem();

if (Data.item is Client client)

{

App.\_db.SaveItem(client);

}

if (Data.item is Car car)

{

App.\_db.SaveItem(car);

}

if (Data.item is Mechanic mechanic)

{

App.\_db.SaveItem(mechanic);

}

if (Data.item is RepairShop repairShop)

{

App.\_db.SaveItem(repairShop);

}

if (Data.item is Part part)

{

App.\_db.SaveItem(part);

}

if (Data.item is Work work)

{

App.\_db.SaveItem(work);

}

}

private void btnRemove\_Clicked(object sender, EventArgs e)

{

if (Data.item is Client client)

{

App.\_db.RemoveItem(client);

}

if (Data.item is Car car)

{

App.\_db.RemoveItem(car);

}

if (Data.item is Mechanic mechanic)

{

App.\_db.RemoveItem(mechanic);

}

if (Data.item is Part part)

{

App.\_db.RemoveItem(part);

}

if (Data.item is RepairShop shop)

{

App.\_db.RemoveItem(shop);

}

if (Data.item is Work work)

{

App.\_db.RemoveItem(work);

}

}

private void btnBack\_Clicked(object sender, EventArgs e)

{

Navigation.PopModalAsync();

}

private void ContentPage\_Loaded(object sender, EventArgs e)

{

if (Data.isAdding == true)

{

gridAdd.IsVisible = true;

gridView.IsVisible = !(gridAdd.IsVisible);

}

else

{

gridAdd.IsVisible = false;

gridView.IsVisible = !(gridAdd.IsVisible);

}

if (Data.item is Client client)

{

stackClient.IsVisible = true;

entryAdres.Text = client.Adres;

entryName.Text = client.Name;

entryPhone.Text = client.Phone;

}

if (Data.item is Car car)

{

stackCar.IsVisible = true;

entryBrand.Text = car.Brand;

entryPower.Text = car.Power;

entryYear.Text = car.Year.ToString();

entryColor.Text = car.Color;

}

if (Data.item is Mechanic mech)

{

stackMechanic.IsVisible = true;

entryAdresM.Text = mech.Adres;

entryNameM.Text = mech.Name;

entryPhoneM.Text = mech.Phone;

entryCategory.Text = mech.Category;

}

if (Data.item is Part part)

{

stackPart.IsVisible = true;

entryCoast.Text = part.Coast.ToString();

entryDateOfI.Text = part.DateOfIssue;

entryCategoryPart.Text = part.Category;

entryDateOfC.Text = part.CompletionDate;

}

if (Data.item is RepairShop shop)

{

stackShops.IsVisible = true;

entryAdresS.Text = shop.Adres;

entryNameS.Text = shop.Name;

entryPhoneS.Text = shop.Phone;

}

if (Data.item is Work work)

{

stackWorks.IsVisible = true;

entryClient.Text = work.ClientId.ToString();

entryMechanic.Text = work.MechanicId.ToString();

entryShop.Text = work.RepairShopId.ToString();

entryCar.Text = work.CarId.ToString();

entryPart.Text = work.PartId.ToString();

}

}

private void UpdateItem()

{

if (Data.item is Client)

{

((Client)Data.item).Name = entryName.Text;

((Client)Data.item).Adres = entryAdres.Text;

((Client)Data.item).Phone = entryPhone.Text;

}

if (Data.item is Car)

{

((Car)Data.item).Brand = entryBrand.Text;

((Car)Data.item).Power = entryPower.Text;

((Car)Data.item).Year = Convert.ToInt32(entryYear.Text);

((Car)Data.item).Color = entryColor.Text;

}

if (Data.item is Mechanic)

{

((Mechanic)Data.item).Adres = entryAdresM.Text;

((Mechanic)Data.item).Name = entryNameM.Text;

((Mechanic)Data.item).Phone = entryPhoneM.Text;

((Mechanic)Data.item).Category = entryCategory.Text;

}

if (Data.item is Part)

{

((Part)Data.item).Coast = Convert.ToInt32(entryCoast.Text);

((Part)Data.item).DateOfIssue = entryDateOfI.Text;

((Part)Data.item).Category = entryCategoryPart.Text;

((Part)Data.item).CompletionDate = entryDateOfC.Text;

}

if (Data.item is RepairShop)

{

((RepairShop)Data.item).Name = entryNameS.Text;

((RepairShop)Data.item).Adres = entryAdresS.Text;

((RepairShop)Data.item).Phone = entryPhoneS.Text;

}

if (Data.item is Work)

{

((Work)Data.item).ClientId = Convert.ToInt32(entryClient.Text);

((Work)Data.item).MechanicId = Convert.ToInt32(entryMechanic.Text);

((Work)Data.item).RepairShopId = Convert.ToInt32(entryShop.Text);

((Work)Data.item).CarId = Convert.ToInt32(entryCar.Text);

((Work)Data.item).PartId = Convert.ToInt32(entryPart.Text);

}

}

}

**ЗАКЛЮЧЕНИЕ**

В заключении проекта мобильного приложения для автомастерской можно отметить, что эффективное управление данными играет ключевую роль в повышении операционной эффективности и качества обслуживания в данном бизнесе. Создание базы данных, которая содержит информацию о клиентах, автомобилях, заказах, запчастях и других важных аспектах деятельности автомастерской, помогает улучшить процессы управления, оптимизировать ресурсы и повысить уровень обслуживания.

База данных для автомастерской позволяет автоматизировать учет работ, контролировать наличие запасных частей, отслеживать историю обслуживания клиентов, а также анализировать данные для принятия обоснованных решений. Это способствует повышению производительности рабочих процессов, снижению рисков ошибок и улучшению взаимодействия с клиентами.

Результативное использование базы данных автомастерской помогает не только оптимизировать текущую деятельность, но и создает базу для дальнейшего развития и улучшения сервиса. Иметь доступ к надежной и актуальной информации о заказах, клиентах и процессах обслуживания становится ключом к долгосрочному успеху и конкурентоспособности автомастерской в современной индустрии.

В ходе работы я спроектировал и разработал базу данных для ведения автомастерской. Также к ней было разработано мобильное приложение.