Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Финансовый менеджер задолженностей

Студент гр.953501

И. Э. Макеенко

Руководитель ассистент кафедры информатики И.А. Удовин

Минск 2020

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой

Информатики

(подпись)

Волорова Н.А. 2020 г.

ЗАДАНИЕ

по курсовому проекту

Студенту *Макеенко Игорю Эдуардовичу*

1. Тема работы *Финансовый менеджер задолженностей.*
2. Срок сдачи студентом законченной работы *31.05.2020.*
3. Исходные данные к работе *Операционная система Android. Язык программирования C#.*
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение. 1. Анализ предметной области. 2. Разработка приложения. 3. Тестирование и проверка на ошибки. Заключение. Список использованных источников. Приложения.*

1. Консультант по курсовой работе *Удовин И. А.*
2. Дата выдачи задания *10.02.2020*
3. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

*раздел 1, Введение к 03.03.2020 – 10% готовности работы;*

*раздел 2 к 16.03.2020г. – 30% готовности работы;*

*раздел 3 к 19.04.2020г. – 60% готовности работы;*

*раздел 4 к 20.05.2020г. – 80% готовности работы;*

*Заключение, Приложение к 20.05.2020г. – 90% готовности работы; оформление пояснительной записки и графического материала к 25.05.2020г. – 100% готовности работы.*

*Защита курсового проекта с 31.05.2020 г. по 01.06.2020 г.*

РУКОВОДИТЕЛЬ *Удовин И.А.*

Задание принял к исполнению Макеенко И. Э. 10.02.2020 г.

(дата и подпись студента)

Оглавление

[**1.** **Анализ предметной области** 5](#_Toc41722920)

[**1.1.** **Введение** 5](#_Toc41722921)

[**1.2.** **Приложения на Android** 5](#_Toc41722922)

[**1.3.** **Выбор инструментария** 5](#_Toc41722923)

[**1.4.** **Плюсы и минусы Xamarin Forms** 6](#_Toc41722924)

[**1.5.** **Постановка задачи** 7](#_Toc41722925)

[**2.** **Разработка приложения** 7](#_Toc41722926)

[**2.1.** **Сущность интерфейса** 7](#_Toc41722927)

[**2.2.** **Паттерн Model-View-ViewModel(MVVM)** 8](#_Toc41722928)

[**2.3.** **Использование SQLite.NET** 8](#_Toc41722929)

[**2.4.** **Реализация данных (Model)** 9](#_Toc41722930)

[**2.4.1.** **Запись задолженности** 9](#_Toc41722931)

[**2.4.2.** **Краткая информация о человеке связанного с задолженностями** 9](#_Toc41722932)

[**2.4.3.** **Запись задолженности для правильной отработки интерфейса** 9](#_Toc41722933)

[**2.5.** **Рализация логики страниц (View-Model)** 10](#_Toc41722934)

[**2.5.1.** **Логика Comings и Outs** 10](#_Toc41722935)

[**2.5.2.** **Логика CreateDebtPage** 11](#_Toc41722936)

[**2.5.3.** **Логика PersonalityDebtPage** 12](#_Toc41722937)

[**2.5.4.** **Логика DebtViews** 14](#_Toc41722938)

[**2.6.** **Реализация страниц (Views)** 14](#_Toc41722939)

[**2.7.** **Реализация взаимодействия с SQLite** 14](#_Toc41722940)

[**2.7.1.** **База данных задолженностей** 14](#_Toc41722941)

[**3.** **Тестирование приложения** 16](#_Toc41722942)

[**3.1.** **Тест 1 – Добавление записи** 16](#_Toc41722943)

[**3.2.** **Тест 2 – Добавление категории** 16](#_Toc41722944)

[**3.3.** **Тест 3 – Добавление кошелька** 17](#_Toc41722945)

[**3.4.** **Тест 4 – Удаление записи** 17](#_Toc41722946)

[Заключение 18](#_Toc41722947)

[Список использованных источников 18](#_Toc41722948)

[Приложение А. Исходный код программы 20](#_Toc41722949)

[Приложение В. Интерфейс программы 29](#_Toc41722950)

1. **Анализ предметной области**
   1. **Введение**

Жизнь в большом городе подразумевает собой грамотное распределение и трату личных денежных средств и для того, чтобы, к примеру, контроллировать задолженности нужно удобное приложение, чтобы делать это быстро и с комфортом.

Финансовый менеджер задолженностей есть такое мобильное приложение, которое позволяет пользователю сохранять информацию о задолженностях, изменять её, структуризировать, а также использовать для полного учёта своих финансов.

* 1. **Приложения на Android**

Следует отметить, что огромное количество портативных девайсов имеют операционную систему Android, эта система успешно развивается и очень удобна для разработки приложений под неё для разработчиков.

Можно отметить, что существует огромное количество стандартных средств разработки под Android, а также большое сообщество Android разработчиков, что тускоряет процесс поиска информации для разработки собственного мобильного приложения под Android.

А также хочется заметить, что смартфоны очень удобны для повседневного использования. И из-за активного потока финансов в мире, количество задолженностей может быть большим и поэтому смартфоны на базе Android становятся подходящей платформой для реализации таких приложений.

* 1. **Выбор инструментария**

Для реализации курсового проекта была выбрана интегрированная среда разработки Microsoft Visual Studio 2019, для разработки на языке C#, используя Xamarin Forms.

Microsoft Visual Studio 2019 позволяет быстро и эффективно разрабатывать приложения с графической оболочкой. Также среда поддерживает удобный Just-In-Time отладчик кода для легкого поиска ошибок.

Xamarin Forms – это фреймворк для создания приложений для всех мобильных платформ, использующий для кроссплатформенной разработки единый язык C#.

Язык C# - объектно-ориентрованный язык программирования и является языком разработки для платформы Microsoft .NET Framework и Microsoft .NET Core. Язык был выбран потому что он является основным языком платформы .NET, на котором и была разработана платформа Xamarin.

Git – распределенная система контроля версий, для сохранения прогресса на разных этапах разработки и возможности отката.

GitHub – веб-сервис для хостинга IT-проектов и их совместной разработки. Для храния курсового проекта не только локально на компьютере.

* 1. **Плюсы и минусы Xamarin Forms**

Из плюсов можно выделить:

* Единый стек технологий для разработки на всех платформах;
* Производительность близка к нативной;
* Возможность разрабатывать платформо-независимый интерфейс;
* Открытый исходный код с корпоративной поддержкой;
* Простота в поддержке и обновления разрабатываемого ПО;
* Полный пакет инструментов разработки;
* Большое сообщество Xamarin-разработчиков;
* Удобная документация;

Тем не менее есть и минусы:

* Задержки с обновлением платформ;
* Ограниченный доступ к open-source библиотекам;
* Ограниченность экосистемы;
* Xamarin не подходит для приложений с высокопроизводительной графикой;
* Большой размер приложений;
* Сложности с интеграцией.
  1. **Постановка задачи**

Задачей курсовой работы является создание программы на языке C#, которая:

* Имеет базовый функционал в виде:
  + добавления записей о задолженностях/долгах,
  + удаления записей о задолженностях/долгах,
  + хранение пользовательских данных,
* Имеет дополнительный функционал в виде:
  + изменение пользовательских данных,
  + сортировка и структуризация задолженностей/долгов,
* Разработать простой, понятный и удобный пользователю интерфейс приложения.

Иначе говоря, разработать прикладное программное обеспечение для менеджмента задолженностей.

1. **Разработка приложения**
   1. **Сущность интерфейса**

При созданиии пользовательского интерфейса (см. приложение В) были использованы следующие критерии:

1. Интерфейс должен быть функционально оправданным, то есть не иметь лишних элементов и чтобы последние не загромождали все пространство. Максимально упрощать интерфейс, но не во вред функциональности.
2. Пользователь должен получить эстетическое удовольствие от пользования. Иными словами, управление данными и их представление должно быть приятным. Это обеспечит долгую продолжительность желания пользователя использовать данное приложение.
3. Все элементы интерфейса должны быть логически объеденены и доступны. Его структура должна быть интуитивно понятна, чтобы пользователь первый раз зайдя в приложение сразу в нем разобрался.

Пользовательский интерфейс данного (см. приложение) приложения представляет собой 2 главных страницы и 3 вспомогательных:

1. **Comings** – главная страница, обеспечивающая вывод информации о задолженностях других людей, которые должны закрыть их нашему пользователю;
2. **Outs** – главная страница, обеспечивающая вывод информации о задолженностях нашего пользователя, которому нужно закрыть их перед другими людьми;
3. **CreateDebtPage** – вспомогательая страница, осуществляющая ввод и редактирование пользовательской информации по поводу задолженности;
4. **PersonalityDebtPage** – вспомогательная страница, осуществляющая вывод собирательной информации о задолженностях конкретного человека;
5. **DebtViews** – вспомогательная страница, обеспечивающая возможность переключения между двумя главными страницами;
   1. **Паттерн Model-View-ViewModel(MVVM)**

Паттерн MVVM основывается на разделении функциональной части приложения на три ключевых компонента:

* **View** – представление или пользовательский интерфейс,
* **Model** – модель или данные, которые используются в приложении,
* **ViewModel** – промежуточный слой между интерфейсом и данными, который обеспечивает их взаимодействие.

Благодарая механизму привязки Xamarin стало возможно привязывать элементы интерфейса к свойствам тех или иных объектов, что дало возможность легко реализовывать выше описанный паттерн.

Следует отметить, что ViewModel ничего не знает о представлении. Он лишь определяет логику страницы.

* 1. **Использование SQLite.NET**

Для локального хранения и доступа данных была выбрана библиотека SQLite.NET. Данная библиотека является упрощенной версией классического SQL реализованная на С. Является крайне популярным решением при создании баз данных(БД) для мобильных приложений. Основной плюс в том, что используя SQLite можно легко, не используя синтаксиса SQL, организовать хранение данных и достуа к ним в коде C#.

Таким образом, помимо реализации описанного выше паттерна в приложении будут дополнительно реализованы классы для взаимодействия с каждой базой данных по отдельности.

* 1. **Реализация данных (Model)**
     1. **Запись задолженности**

Запись задолженности приложения реализована в классе DebtModel (см. приложение А), в котором находятся:

|  |  |
| --- | --- |
| **Имя поля** | **Описание** |
| Name | Поле имени человека, с которым связана задолженность |
| DebtChange | Поле значения задолженности |
| Description | Поле описания задолженности |
| Date | Поле даты задолженности |

* + 1. **Краткая информация о человеке связанного с задолженностями**

Класс PreviewDebtModel (см. приложение А) описывает первичный просмотр о суммарных задолженностях человека. Он содержит:

|  |  |
| --- | --- |
| **Имя поля** | **Описание** |
| Name | Поле имени человека, с которым связана задолженность |
| DebtSum | Сумма по всем задолженностям человека |

* + 1. **Запись задолженности для правильной отработки интерфейса**

Класс ViewDebtModel (см. приложение А) описывает категорую транзакции. Он содержит:

|  |  |
| --- | --- |
| **Имя поля** | **Описание** |
| Name | Поле имени человека, с которым связана задолженность |
| DebtChange | Поле значения задолженности |
| Description | Поле описания задолженности |
| Date | Поле даты задолженности |
| IsMoneyDebt | Поле описывающие принадлежность к денежным или к неденежным задолженностям |

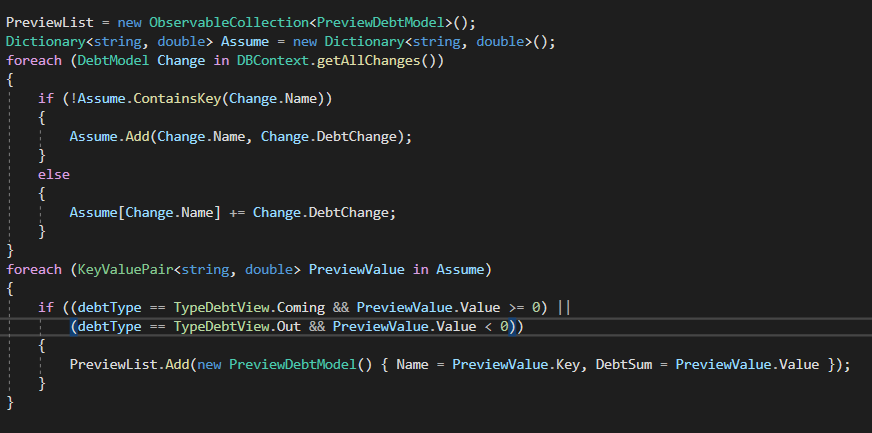
Следует отметить, что описанные выше классы данных, используются SQLite для организации записей в соответствующей каждому классу БД.

* 1. **Рализация логики страниц (View-Model)**

Каждая страница в приложении сопровождается обслуживающим ее классом, что предоставляет свойства для привязки, чтобы в последствии отобразить данные на экране.

* + 1. **Логика Comings и Outs**

Главным элементом на этой странице является ListView, данные к которой предоставляет класс CommonDebtView (см. приложение А). Он содержит PreviewList, который является объектом класса ObservableCollection. PreviewList, определением для Outs этот список или для Comings отвечает енумератор TypeDebtView (см. приложение А), собирает общую информацию о задолженностях и суммарную задолженность для каждого человека (см. Рис. 1)

Рис. 1. Получение информации для PreviewList в классе CommonDebtView

После получения необходимой информации, он иницилизируется как ItemSource для ItemView, после этого BindingContext заменяется на данные из PreviewList, таким образом реализовываются динамические элементы в Xamarin Forms.

Также присутствует кнопка создания задолженности и обработчик AddButton\_Clicked, который открывывает страницу CreateDebtPage для создания новой задолженности, тип задолженности зависит от текущей выбранно страницы в DebtViews.

ListView\_ItemTapped – метод, который вызывается, когда происходит нажатие на элемент ListView, после нажатия происходит создание страницы PersonalityDebtPage с информацией человека, имя которого было написано на элементе ListView.

Refresh – метод, который пересоздаёт CommonDebtView для изменения BindingContext.

* + 1. **Логика CreateDebtPage**

|  |  |
| --- | --- |
| **Поля ввода** | |
| NameField | Поле для ввода имени человека |
| ValueField | Поле для ввода значения долга |
| DescriptionField | Поле для ввода описания долга |
| DateDebtStart | Поле для ввода даты, реализовано с помощью DatePicker |
| DebtButton | Кнопка для создания долга |
| **Методы** | |
| DebtButton\_Clicked | Обработчик события нажатия на кнопку DebtButton, после нажатия происходит валидация введённых и переданных данных |

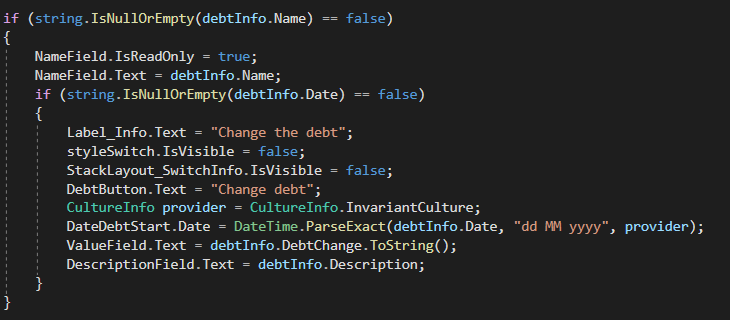
CreateDebtPage выступает не только для создания долга, но и для редактирования, логика отличается в зависимости от переданных данных (см. Рис. 2).

Рис. 2. Изменение логики в зависимости от переданных данных в конструктор класса CreateDebtPage

* + 1. **Логика PersonalityDebtPage**

Основным элементом на данной странице является PersonalDebts (ListView), данные к которому предоставляет класс PersonalityDebtView (см. приложение А), который по аналогии с CommonDebtView предоставляет объект ObservableCollection - ChangeList, с которым взаимодействует PersonalDebts.

ChangeList генерируется из \_context, который является результатом работы метода класса DBContext (см. приложение А) – getChangesByName (см. Рис. 3).

Рис. 3. Вызов статического метода из DBContext

На навигационной панеле присутсвует панель инструментов, которые позволяют сортировать задолженности текущего человека по разным полям, тип сортировки определяет енумератор TypeSort (см. приложение А).

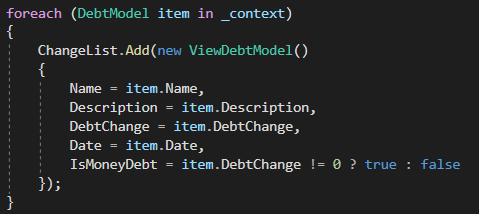
ChangeList использует шаблон ViewDebtModel, в котором отличительное от DebtModel поле IsMoneyDebt, генерируется исходя из DebtModel который находится в \_context (см. Рис. 4). Отличительное поле IsMoneyDebt нужно для измененного отображение неденежных долгов.

Рис. 4. Генерация элементов ChangeList

На панели инструментов есть элемент, отвечающий за закрывание всех задолженностей этого человека, что является возможностью вызова функции из класса DBContext(см. приложение А)- eraseByName.

На странице присутсвует кнопка создания долга и обработчик AddButton\_Clicked, по аналогии со Comings или Outs, только при нажатии этой кнопки на странице PersonalityDebtPage открывается CreateDebtPage уже сразу с именем человека, которого вы просматриваете.

PersonalDebts\_ItemTapped – метод, который вызывается при нажатии на элемент PersonalDebts, после этого создаётся контекстное меню в котором, можно либо изменить (вызвать метод CreateDebtPage c полностью заполненным DebtInfo) или закрыть эту задолженность (вызвать метод eraseByFields из DBContext, куда передать информацию, которую содержит элемент ListView по которому было совершено нажатие).

Refresh – метод, обновляет информацию о суммарном долге человека, также происзводит валидацию на ненулевое количество задолженностей и если количество задолженностей равно нулю, то закрывает текущую страницу и обновляет ресурсы предыдущих страниц, а также пересоздаёт объект класса PersonalityDebtView, чтобы обновить ресурсы PersonalDebts.

* + 1. **Логика DebtViews**

Предоставляет возможность изменения обзора текущей страницы Outs или Comings

Refresh – метод, вызывающий Refresh у страниц потомков.

* 1. **Реализация страниц (Views)**

Каждая страница представлена в виде XAML-файла, где хранится разметка и привязка к свойствам класса страницы, и прикрепленного к нему CS-файла, описывающего класс страницы. При реализации паттерна MVVM вся логика выносится в отдельные файлы (все были описаны выше). Поэтому класс каждой страницы будет иметь поле с объектом обслуживающего его класса и конструктор без аргументов, где происходит инциализация полей и настройка контекста привязки страницы.

В зависимости от логики в классах страниц могут присутствовать дополнительные методы, которые обрабатывают события в результате взаимодействия интерфейса с пользователем. Все они так или иначе вызывают методы DBContext или же ViewLists и меняются в зависимости от аргументов пользователя.

* 1. **Реализация взаимодействия с SQLite**

Для взаимодействия с БД записей был реализован класс DBContext (см. приложение А). Он содержит:

|  |  |
| --- | --- |
| **Статические поля** | |
| db | Объект класса SQLiteConnection, который обеспечивает взаимодействие с файлом БД |
| tableCreationResult | Объект класса CreateTableResult, который предоставляет результат создания таблицы в базе данных |
| **Статические методы** | |
| getAllChanges | Получение списка всех задолженностей |
| getChangesByName | Получение списка задолженностей по имени человека |
| getSummaryDebtByName | Получение суммарного долга по имени человека |
| getNumberChanges | Получение количества задолженностей по имени |
| createChange | Добавление задолженности в базу данных |
| eraseByName | Удаление всех задолженностей по имени |
| eraseByFields | Удаление конкретной задолженности по всем полям |

1. **Тестирование приложения**

В ходе тестирования приложения были проверены базовые функции, на которых и строится вся концепция приложения.

* 1. **Тест 1 – Добавление задолженности**

|  |  |
| --- | --- |
| Тестовая ситуация | Работа кнопки добавления задолженности на главной странице. |
| Ожидаемый результат | Добавление созданной задолженности, отображение корректной суммы общего долга, отображение задолженности на персональной странице всех задолженностей для конкретного человека, отображение со всеми введёнными полями |
| Фактический результат | Все работает корректно. Демонтрация находится на рис. 2, 3, 4, 5 приложения В. |

* 1. **Тест 2 – Добавление задолженности из персональной страницы**

|  |  |
| --- | --- |
| Тестовая ситуация | Работа кнопки добавления из персональной страницы. |
| Ожидаемый результат | При нажатии на кнопку, должна открытся страница создания вместе с именем человека, которого мы просматривали на персональной странице. |
| Фактический результат | Все работает корректно. Демонтрация находится на рис. 6, 7, 8 приложения В. |

* 1. **Тест 3 – Редактирование задолженности**

|  |  |
| --- | --- |
| Тестовая ситуация | Работа обработчика событий нажатия в персональной странице |
| Ожидаемый результат | Появление контекстного меню, в которой должен быть пункт изменить, после изменение будет корректное отображение данных задолженности на персональной странице |
| Фактический результат | Все работает корректно. Демонтрация находится на рис. 9, 10, 11 приложения В. |

* 1. **Тест 4 – Сортировка задолженностей внутри персональной страницы**

|  |  |
| --- | --- |
| Тестовая ситуация | Работа сортировки задолженностей внутри персональной страницы |
| Ожидаемый результат | Правильное отображение последовательности элементов задолженности персональной страницы |
| Фактический результат | Все работает корректно. Демонтрация находится на рис. 12, 13, 14, 15, 16 приложения В. |

# Заключение

В результате выполнения курсовой работы было разработано простое приложение для смартфонов на базе Android для менеджмента задолженностей, которое имеет весь базовый функционал. В ходе разработки данного программного средства были получены знания по программированию пользовательских интерфейсов в Microsoft Visual Studio 2019 на Xamarin с помощью языка С#, также были получены теоретические знания в сфере программирования на платформе .NET.

Анализируя приложение можно прийти к следующим выводам.

Xamarin forms – хороший инструмент для разработки мобильных приложений под Android. На нём можно быстро и эффективно создавать интерфейсы приложений и получать при этом почти нативную производительность. Но из-за того, что Xamarin не успевает за новыми тенденциями Android, так как глобальные обновления выходят не часто, то можно получить много проблем связанных с новыми элементами Android. Также готовые установочные пакеты Xamarin.Android занимают слишком много места на устройстве, при том, что приложение, которое установится через этот пакет, не будет является обоснованно нагруженным. Если сравнивать по памяти нативные приложения Android и Xamarin.Android, то на данный момент Xamarin проигрывает, но это не значит, что Xamarin плохой инструмент, это лишь значит, что не вышло должного количества обновлений, чтобы Xamarin стал лидером в производстве мобильных приложений.

# Список использованных источников

1. Wikipedia [Электронный ресурс]: <https://ru.wikipedia.org>
2. Microsoft .NET Documentation [Электронный ресурс]: <https://docs.microsoft.com/en-us/dotnet/>
3. Документация по Xamarin [Электронный ресурс]: <https://docs.microsoft.com/ru-ru/xamarin/>
4. Волосевич, А. А. Язык С# и основы платформы .NET: Минск: БГУИР, 2013.
5. Волосевич, А. А. Базовые технологии платформы .NET: Минск: БГУИР, 2013.

# Приложение А. Исходный код программы

Comings:

public partial class Comings : ContentPage

{

public Comings()

{

InitializeComponent();

Refresh();

}

public void Refresh()

{

BindingContext = new ViewModels.CommonDebtView(TypeDebtView.Coming);

}

private void ListView\_ItemTapped(object sender, ItemTappedEventArgs e)

{

PreviewDebtModel temp = (PreviewDebtModel)e.Item;

Navigation.PushAsync(new PersonalityDebtPage(temp));

}

private void AddButton\_Clicked(object sender, System.EventArgs e)

{

Navigation.PushAsync(new CreateDebtPage("Comings", new DebtModel() { Date = "", Name = "", Description = "", DebtChange = 0 }));

}

}

Outs:

public partial class Outs : ContentPage

{

public Outs()

{

InitializeComponent();

Refresh();

}

private void ListView\_ItemTapped(object sender, ItemTappedEventArgs e)

{

PreviewDebtModel temp = (PreviewDebtModel)e.Item;

Navigation.PushAsync(new PersonalityDebtPage(temp));

}

public void Refresh()

{

BindingContext = new CommonDebtView(TypeDebtView.Out);

}

private void AddButton\_Clicked(object sender, System.EventArgs e)

{

Navigation.PushAsync(new CreateDebtPage("Outs", new DebtModel() { Date = "", Name = "", Description = "", DebtChange = 0 }));

}

}

CreateDebtPage:

public partial class CreateDebtPage : ContentPage

{

private readonly DebtModel debtInfo;

public CreateDebtPage(string currentPage, DebtModel debtInfo)

{

InitializeComponent();

styleSwitch.IsToggled = (currentPage == "Comings") ? false : true;

DateDebtStart.Date = DateTime.Now;

this.debtInfo = debtInfo;

if (string.IsNullOrEmpty(debtInfo.Name) == false)

{

NameField.IsReadOnly = true;

NameField.Text = debtInfo.Name;

if (string.IsNullOrEmpty(debtInfo.Date) == false)

{

Label\_Info.Text = "Change the debt";

styleSwitch.IsVisible = false;

StackLayout\_SwitchInfo.IsVisible = false;

DebtButton.Text = "Change debt";

CultureInfo provider = CultureInfo.InvariantCulture;

DateDebtStart.Date = DateTime.ParseExact(debtInfo.Date, "dd MM yyyy", provider);

ValueField.Text = debtInfo.DebtChange.ToString();

DescriptionField.Text = debtInfo.Description;

}

}

}

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

{

if (string.IsNullOrEmpty(NameField.Text))

{

DisplayAlert("Error", "Name is empty, please verify all information.", "OK");

return;

}

if (string.IsNullOrEmpty(ValueField.Text))

{

DisplayAlert("Error", "Sum is empty, please verify all information.", "OK");

return;

}

if (double.TryParse(ValueField.Text, out double value) == false)

{

DisplayAlert("Error", "Please enter the correct number", "OK");

return;

}

if (styleSwitch.IsToggled)

{

value = -value;

}

DebtModel Item = new DebtModel()

{

Name = char.ToUpper(NameField.Text[0]) + NameField.Text.Substring(1),

DebtChange = value,

Description = !string.IsNullOrEmpty(DescriptionField.Text) ? char.ToUpper(DescriptionField.Text[0]) + DescriptionField.Text.Substring(1) : "",

Date = DateDebtStart.Date.ToString("dd MM yyyy")

};

if (string.IsNullOrEmpty(debtInfo.Date) == false)

{

DBContext.eraseByFields(debtInfo);

}

DBContext.createChange(Item);

if (string.IsNullOrEmpty(debtInfo.Name) == false)

{

((PersonalityDebtPage)Navigation.NavigationStack.ToList<Page>()[1]).Refresh();

}

if (string.IsNullOrEmpty(debtInfo.Date) == true)

{

DisplayAlert("Success", "Your debt has been successfully created.", "OK");

}

else

{

DisplayAlert("Success", "Your debt has been successfully changed.", "OK");

}

((DebtsViews)Navigation.NavigationStack.ToList<Page>()[0]).Refresh();

Navigation.PopAsync();

}

}

PersonalityDebtPage:

public partial class PersonalityDebtPage : ContentPage

{

private readonly string personalityName;

public PersonalityDebtPage(PreviewDebtModel DebtInfo)

{

InitializeComponent();

personalityName = DebtInfo.Name;

Refresh();

/\* Close all debts \*/

ToolbarItem closeDebtsItem = new ToolbarItem

{

Text = "Close all debts",

Order = ToolbarItemOrder.Secondary,

Priority = 0

};

closeDebtsItem.Clicked += async (s, e) =>

{

bool result = await DisplayAlert("Warning", "Are you sure you want to close all debts?", "Yes", "No");

if (result == true)

{

DBContext.eraseByName(personalityName);

((DebtsViews)Navigation.NavigationStack.ToList<Page>()[0]).Refresh();

await Navigation.PopAsync();

}

};

/\* Sorting by date (old first)\*/

ToolbarItem sortByDateOldFirstItem = new ToolbarItem

{

Text = "Sort by date (old first)",

Order = ToolbarItemOrder.Secondary,

Priority = 1

};

sortByDateOldFirstItem.Clicked += (s, e) =>

{

BindingContext = new PersonalityDebtView(personalityName, TypeSort.ByDateOldFirst);

};

/\* Sorting by date (new first)\*/

ToolbarItem sortByDateNewFirstItem = new ToolbarItem

{

Text = "Sort by date (new first)",

Order = ToolbarItemOrder.Secondary,

Priority = 2

};

sortByDateNewFirstItem.Clicked += (s, e) =>

{

BindingContext = new PersonalityDebtView(personalityName, TypeSort.ByDateNewFirst);

};

/\* Sorting by value (large first) \*/

ToolbarItem sortByValueLargeFirstItem = new ToolbarItem

{

Text = "Sort by value (large first)",

Order = ToolbarItemOrder.Secondary,

Priority = 3

};

sortByValueLargeFirstItem.Clicked += (s, e) =>

{

BindingContext = new PersonalityDebtView(personalityName, TypeSort.ByValueLargeFirst);

};

/\* Sorting by value (small first) \*/

ToolbarItem sortByValueSmallFirstItem = new ToolbarItem

{

Text = "Sort by value (small first)",

Order = ToolbarItemOrder.Secondary,

Priority = 4

};

sortByValueSmallFirstItem.Clicked += (s, e) =>

{

BindingContext = new PersonalityDebtView(personalityName, TypeSort.ByValueSmallFirst);

};

/\* Add buttons on nav bar \*/

ToolbarItems.Add(closeDebtsItem);

ToolbarItems.Add(sortByDateOldFirstItem);

ToolbarItems.Add(sortByDateNewFirstItem);

ToolbarItems.Add(sortByValueLargeFirstItem);

ToolbarItems.Add(sortByValueSmallFirstItem);

}

public void Refresh()

{

PersonalityDebtView \_context = new PersonalityDebtView(personalityName, TypeSort.Default);

if (\_context.ChangeList.Count == 0)

{

((DebtsViews)Navigation.NavigationStack.ToList<Page>()[0]).Refresh();

Navigation.PopAsync();

return;

}

BindingContext = \_context;

Title = personalityName + ": " + DBContext.getSummaryDebtByName(personalityName).ToString();

}

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

{

Navigation.PushAsync(new CreateDebtPage("Comings", new DebtModel() { Name = personalityName, DebtChange = 0, Description = "", Date = "" }));

}

private async void PersonalDebts\_ItemTapped(object sender, ItemTappedEventArgs e)

{

string result = await DisplayActionSheet("What to do...", "Cancel", null, "Close this debt", "Change this debt");

DebtModel debtInfo = (DebtModel)e.Item;

if (result == "Close this debt")

{

bool decision = await DisplayAlert("Warning", string.Format("Are you sure you want close this debt({0}, {1}, {2})", debtInfo.DebtChange.ToString(), debtInfo.Description, debtInfo.Date), "YES", "NO");

if (decision == true)

{

DBContext.eraseByFields(debtInfo);

Refresh();

}

}

if (result == "Change this debt")

{

await Navigation.PushAsync(new CreateDebtPage("Comings", debtInfo));

}

}

}

DebtViews:

public partial class DebtsViews : TabbedPage

{

private string currentPageName;

[Obsolete]

public DebtsViews()

{

InitializeComponent();

BarBackgroundColor = Color.FromHex("#4A78D6");

BarTextColor = Color.White;

/\* Childred pages to tab page \*/

Children.Add(new Comings());

Children.Add(new Outs());

}

protected override void OnCurrentPageChanged()

{

currentPageName = CurrentPage.Title;

}

public void Refresh()

{

((Comings)Children.ElementAt(0)).Refresh();

((Outs)Children.ElementAt(1)).Refresh();

}

}

PersonalityDebtView:

public class PersonalityDebtView

{

public ObservableCollection<ViewDebtModel> ChangeList { get; set; }

public PersonalityDebtView(string Name, TypeSort typeSort)

{

ChangeList = new ObservableCollection<ViewDebtModel>();

System.Collections.Generic.List<DebtModel> \_context = DBContext.getChangesByName(Name);

switch (typeSort)

{

case TypeSort.ByDateOldFirst:

\_context.Sort(delegate (DebtModel x, DebtModel y)

{

CultureInfo provider = CultureInfo.InvariantCulture;

DateTime a = DateTime.ParseExact(x.Date, "dd MM yyyy", provider);

DateTime b = DateTime.ParseExact(y.Date, "dd MM yyyy", provider);

if (a == b)

{

return 0;

}

return a > b ? 1 : -1;

});

break;

case TypeSort.ByDateNewFirst:

\_context.Sort(delegate (DebtModel x, DebtModel y)

{

CultureInfo provider = CultureInfo.InvariantCulture;

DateTime a = DateTime.ParseExact(x.Date, "dd MM yyyy", provider);

DateTime b = DateTime.ParseExact(y.Date, "dd MM yyyy", provider);

if (a == b)

{

return 0;

}

return a < b ? 1 : -1;

});

break;

case TypeSort.ByValueLargeFirst:

\_context.Sort(delegate (DebtModel x, DebtModel y)

{

if (x.DebtChange == y.DebtChange)

{

return 0;

}

return x.DebtChange < y.DebtChange ? 1 : -1;

});

break;

case TypeSort.ByValueSmallFirst:

\_context.Sort(delegate (DebtModel x, DebtModel y)

{

if (x.DebtChange == y.DebtChange)

{

return 0;

}

return x.DebtChange > y.DebtChange ? 1 : -1;

});

break;

default:

break;

}

foreach (DebtModel item in \_context)

{

ChangeList.Add(new ViewDebtModel()

{

Name = item.Name,

Description = item.Description,

DebtChange = item.DebtChange,

Date = item.Date,

IsMoneyDebt = item.DebtChange != 0 ? true : false

});

}

}

}

DBContext:

public static class DBContext

{

private static readonly SQLiteConnection db = new SQLiteConnection(Settings.dbpath);

private static readonly CreateTableResult tableCreationResult = db.CreateTable<DebtModel>();

public static List<DebtModel> getAllChanges()

{

List<DebtModel> Changes = db.Table<DebtModel>().ToList();

Changes.Sort(delegate (DebtModel x, DebtModel y)

{

if (x.DebtChange == y.DebtChange)

{

return 0;

}

return x.DebtChange > y.DebtChange ? 1 : -1;

});

return Changes;

}

public static List<DebtModel> getChangesByName(string Name)

{

return db.Table<DebtModel>().ToList().FindAll(Change => Change.Name == Name);

}

public static double getSummaryDebtByName(string Name)

{

List<DebtModel> \_context = db.Table<DebtModel>().ToList();

\_context = \_context.FindAll(Change => Change.Name == Name);

double result = 0;

foreach (DebtModel item in \_context)

{

result += item.DebtChange;

}

return result;

}

public static int getNumberChanges(string UserName)

{

return db.Table<DebtModel>().Count(Change => Change.Name == UserName);

}

public static void createChange(DebtModel Debt)

{

db.Insert(Debt);

}

public static void eraseByName(string Name)

{

db.Table<DebtModel>().Delete(Change => Change.Name == Name);

}

public static void eraseByFields(DebtModel Debt)

{

int valueSameRows = db.Table<DebtModel>().Count(change =>

change.Name == Debt.Name &&

change.Date == Debt.Date &&

change.Description == Debt.Description &&

change.DebtChange == Debt.DebtChange

) - 1;

db.Table<DebtModel>().Delete(change =>

change.Name == Debt.Name &&

change.Date == Debt.Date &&

change.Description == Debt.Description &&

change.DebtChange == Debt.DebtChange

);

while (valueSameRows > 0)

{

db.Insert(Debt);

valueSameRows--;

}

}

}

DebtModel:

public class DebtModel

{

public string Name { get; set; }

public double DebtChange { get; set; }

public string Description { get; set; }

public string Date { get; set; }

}

PreviewDebtModel:

public class PreviewDebtModel

{

public string Name { set; get; }

public double DebtSum { set; get; }

}

ViewDebtModel:

public class ViewDebtModel : DebtModel

{

public bool IsMoneyDebt { get; set; }

}

TypeDebtView:

public enum TypeDebtView

{

Coming,

Out

}

TypeSort:

public enum TypeSort

{

ByDateOldFirst,

ByDateNewFirst,

ByValueLargeFirst,

ByValueSmallFirst,

Default

}

Settings:

internal static class Settings

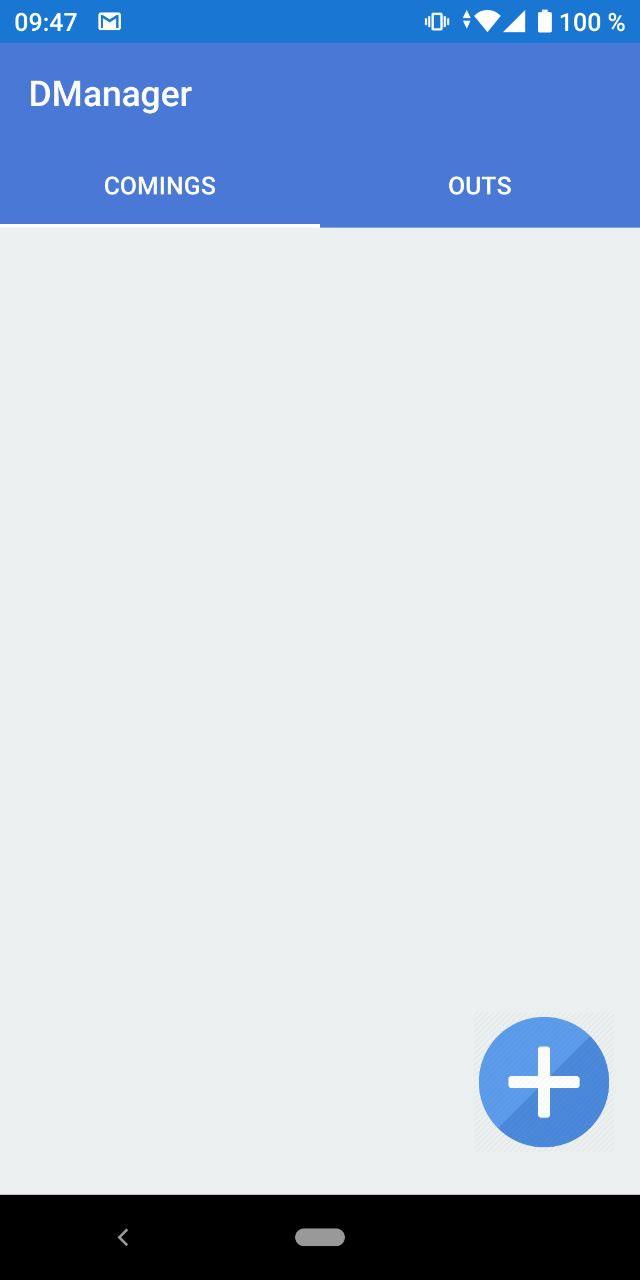
{

public static string dbpath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments), "DBContext.db");

}

# Приложение В. Интерфейс программы

Рис. 1. Структура исходных файлов

Рис. 2. Отображение превого запуска

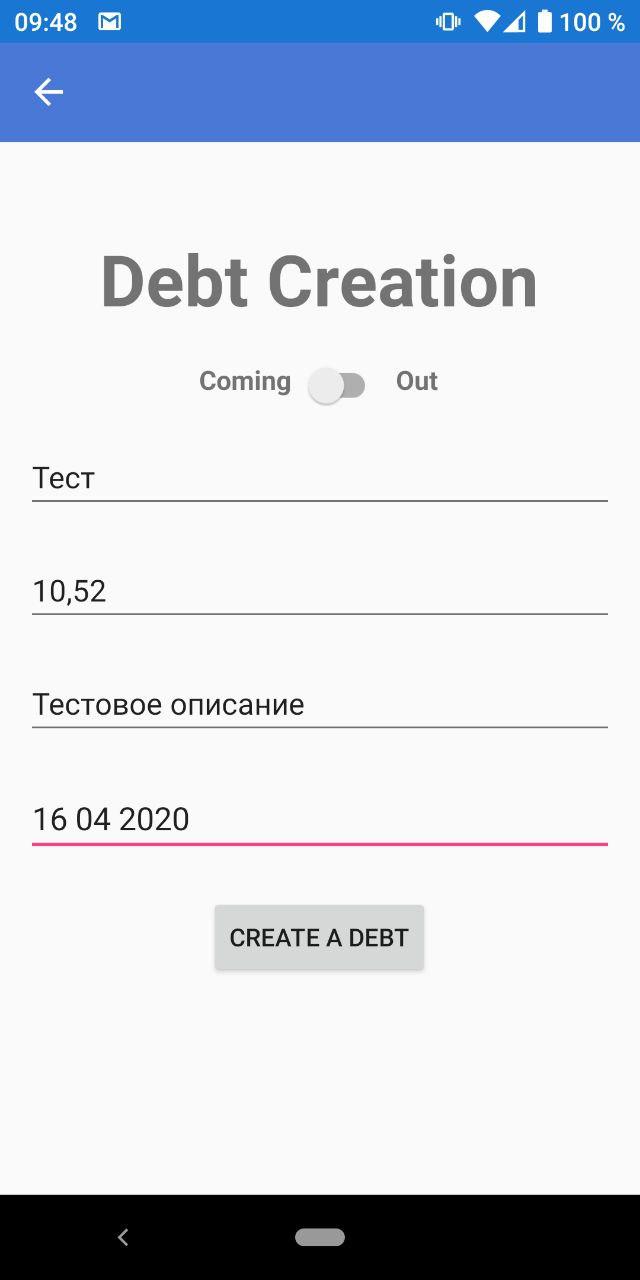


Рис. 3. Отображение при нажатии на кнопку плюс на главной странице

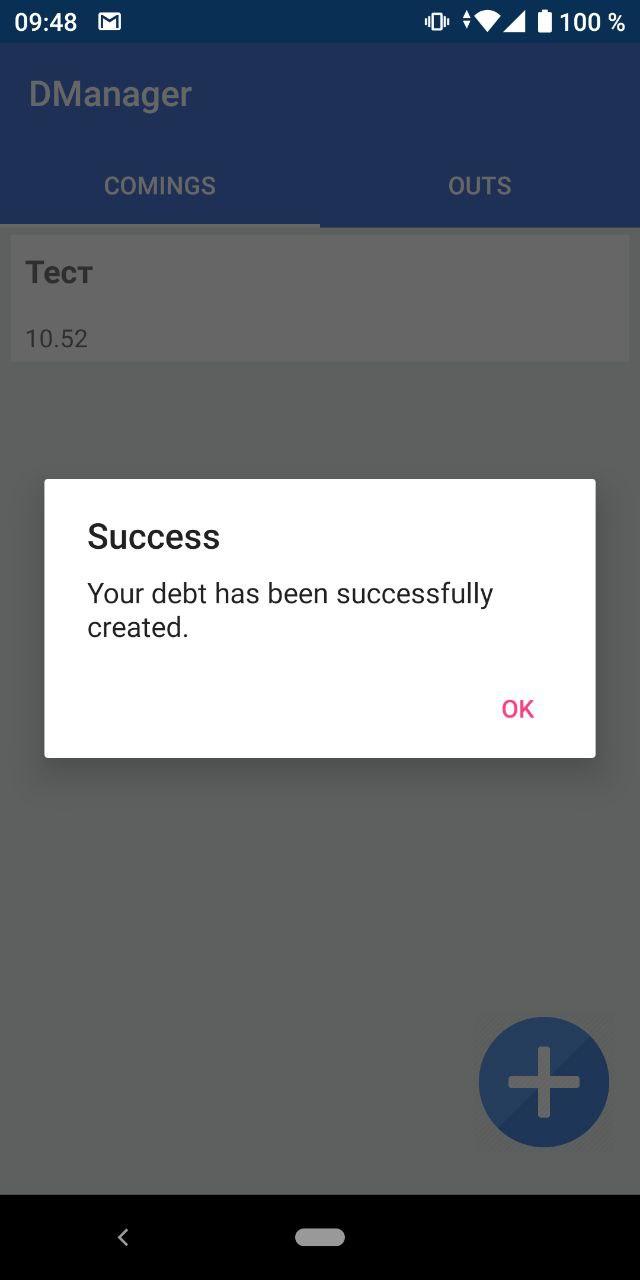


Рис. 4. Отображение при правильном вводе создания долга

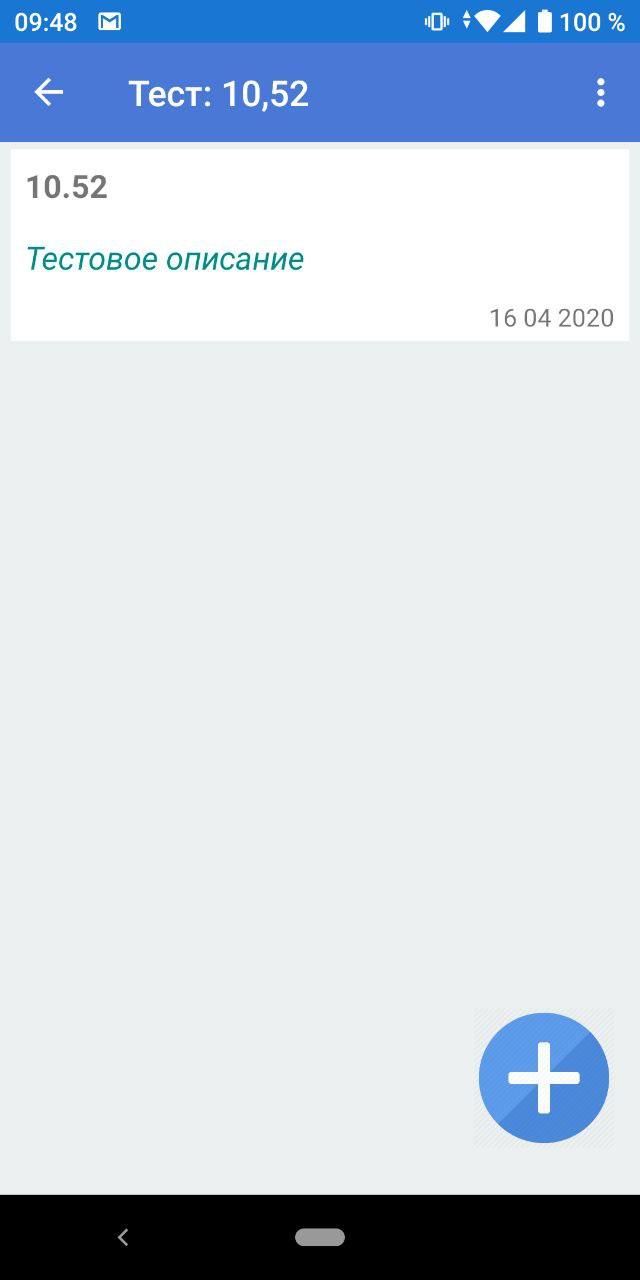


Рис 5. Отображение при клике на Тест

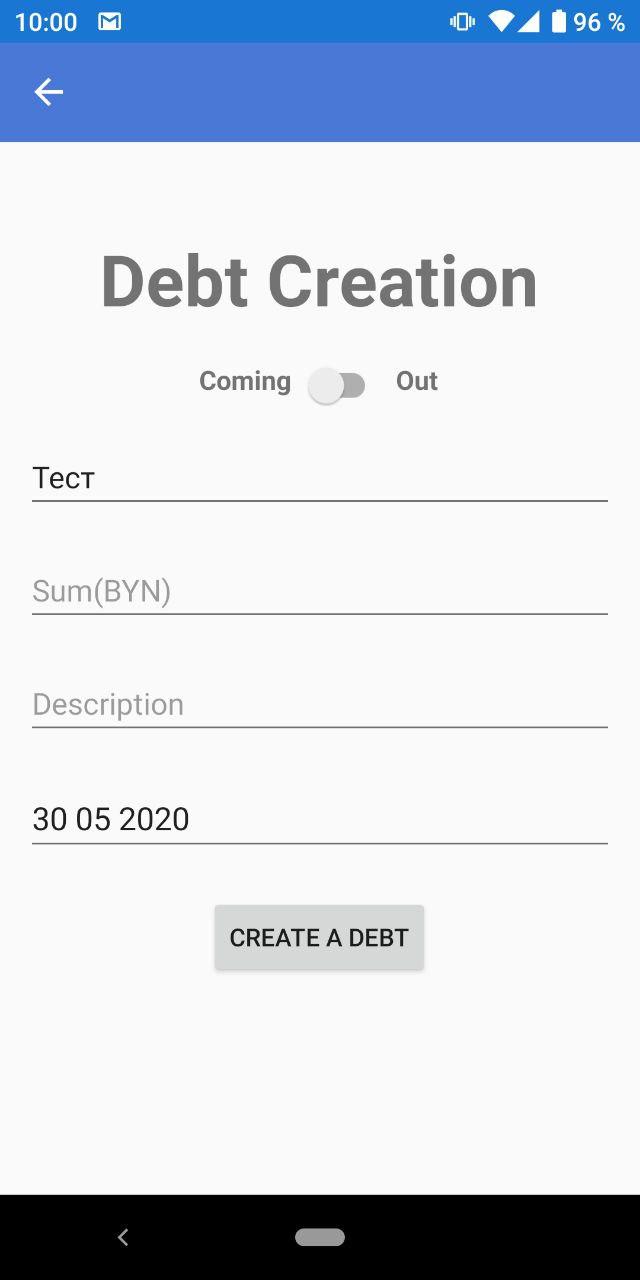


Рис 6. Нажатие кнопки из персональной страницы

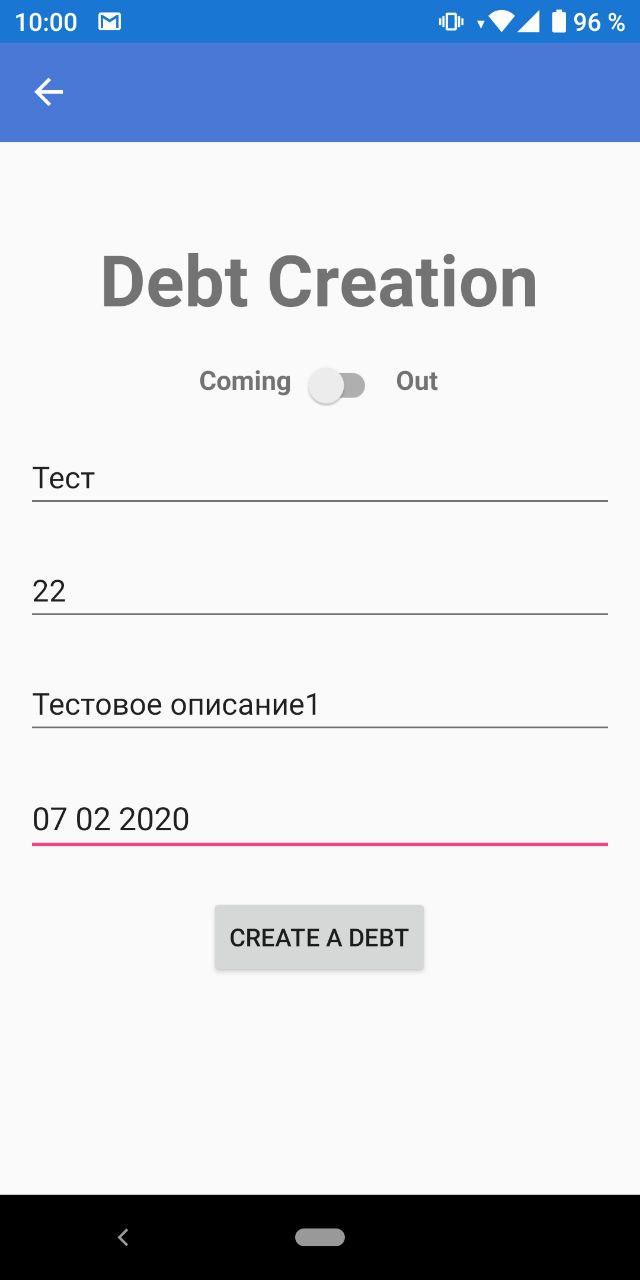


Рис. 7. Заполнение информации о долге, вызванного из персональной страницы

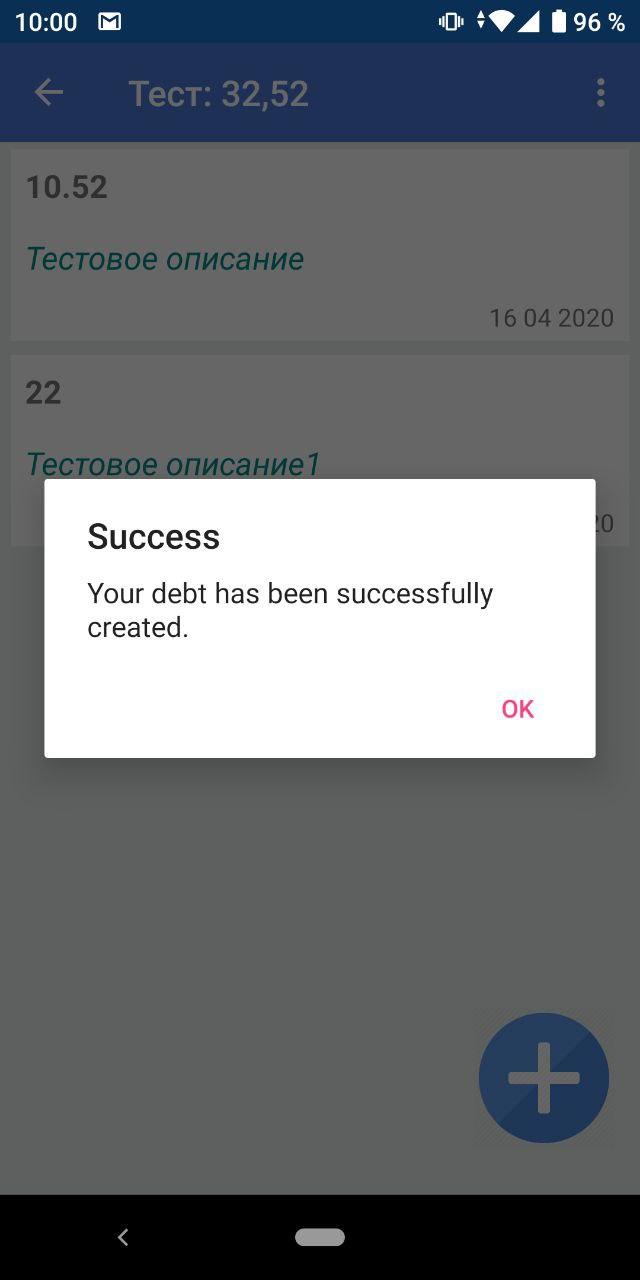


Рис. 8. Успешное создание долга из персональной страницы

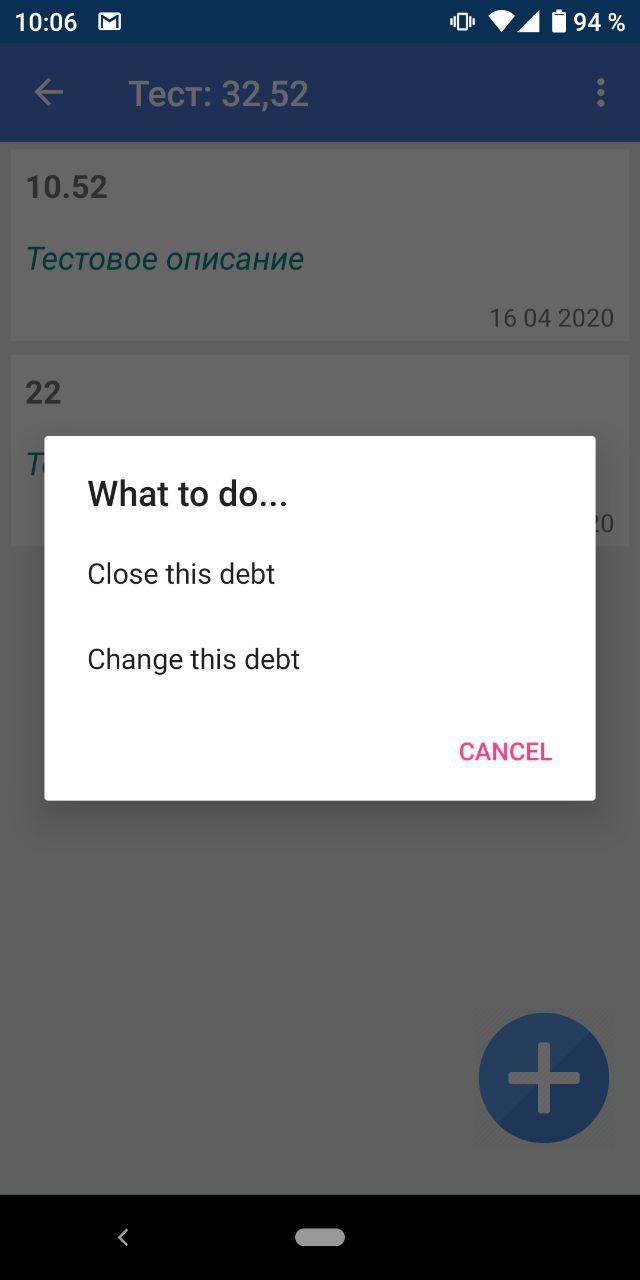


Рис. 9. Работа обработчика нажатия на элемент из описания элемента персональной страницы

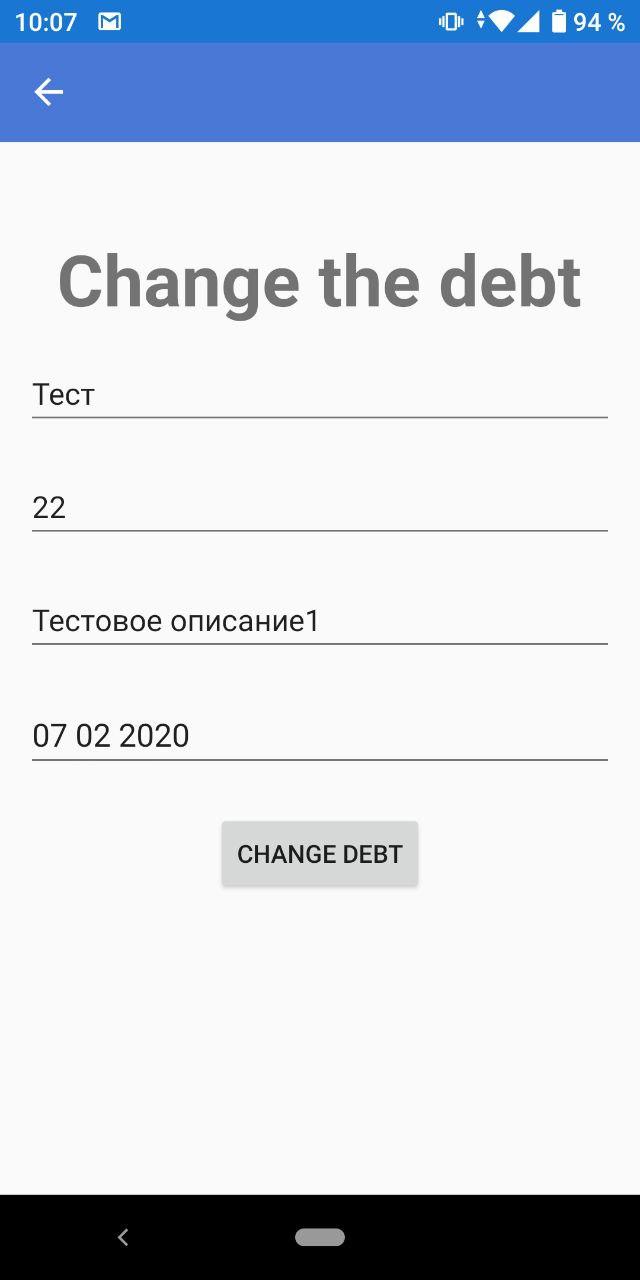


Рис 10. Появление меню изменения элемента из персональной страницы

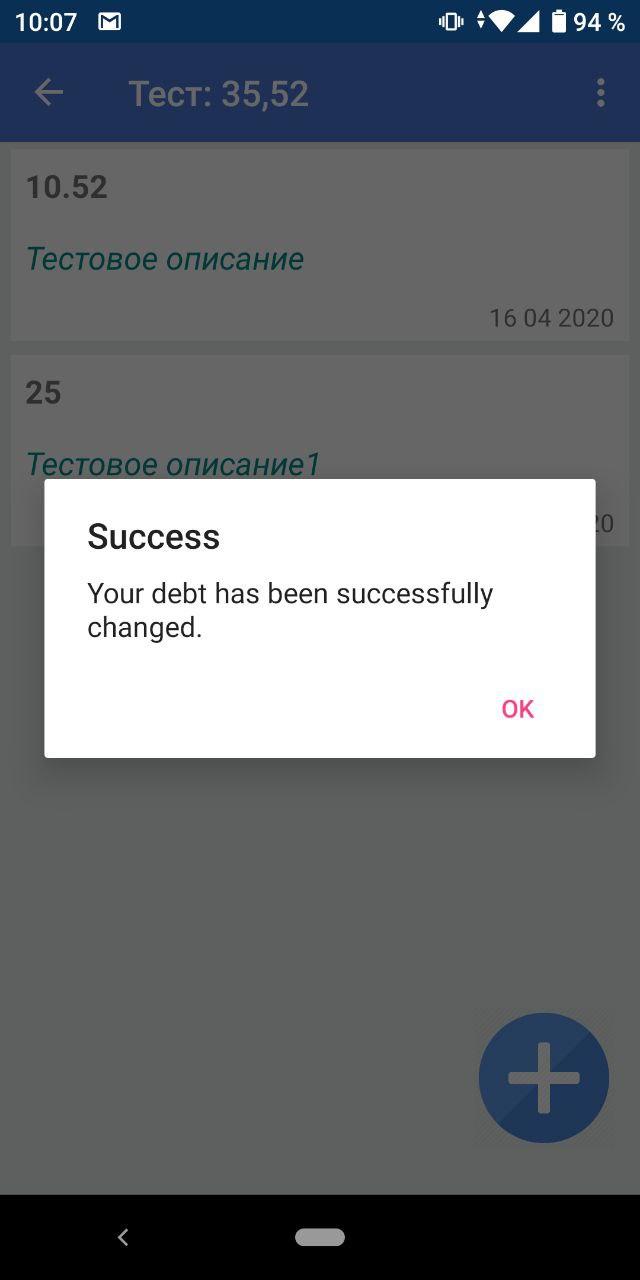


Рис 11. Успешно изменение введённых данный

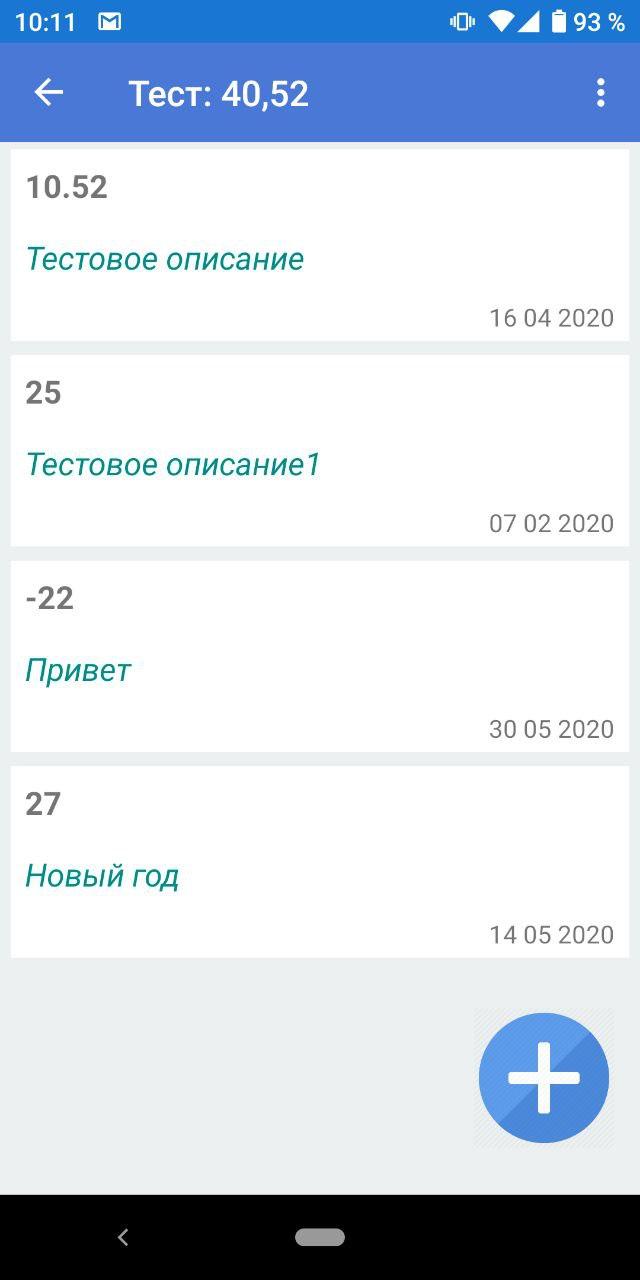
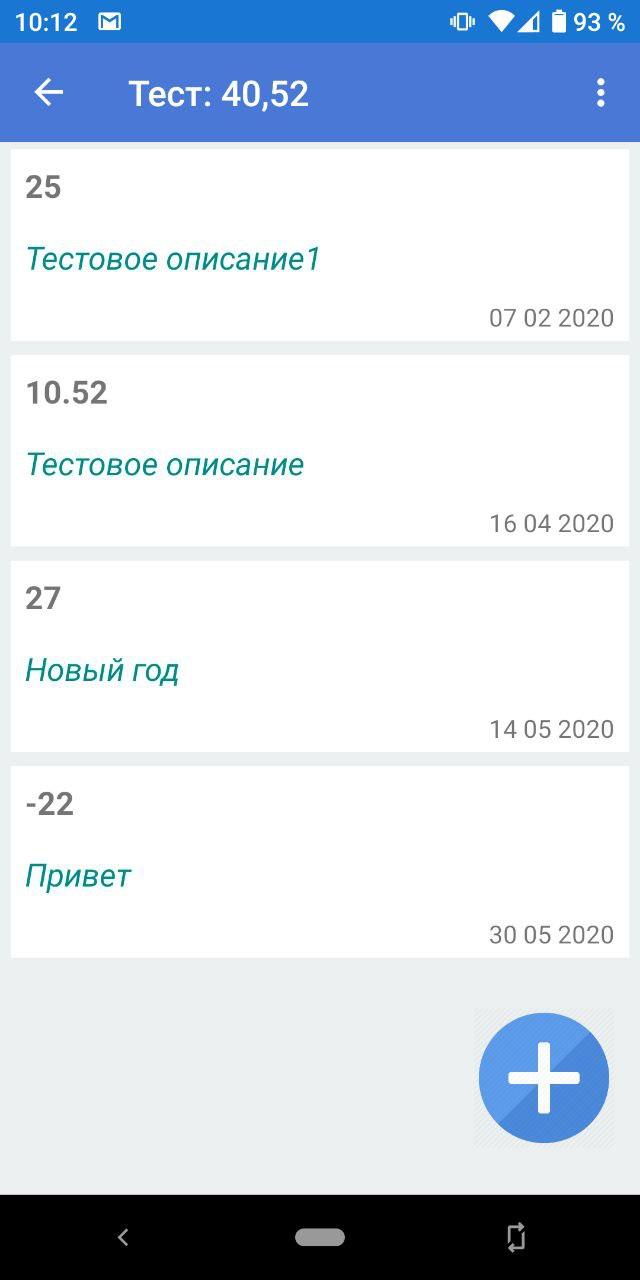


Рис. 12. Обычное отображение задолженностей



13. Сортировка по дате (сначала старые)

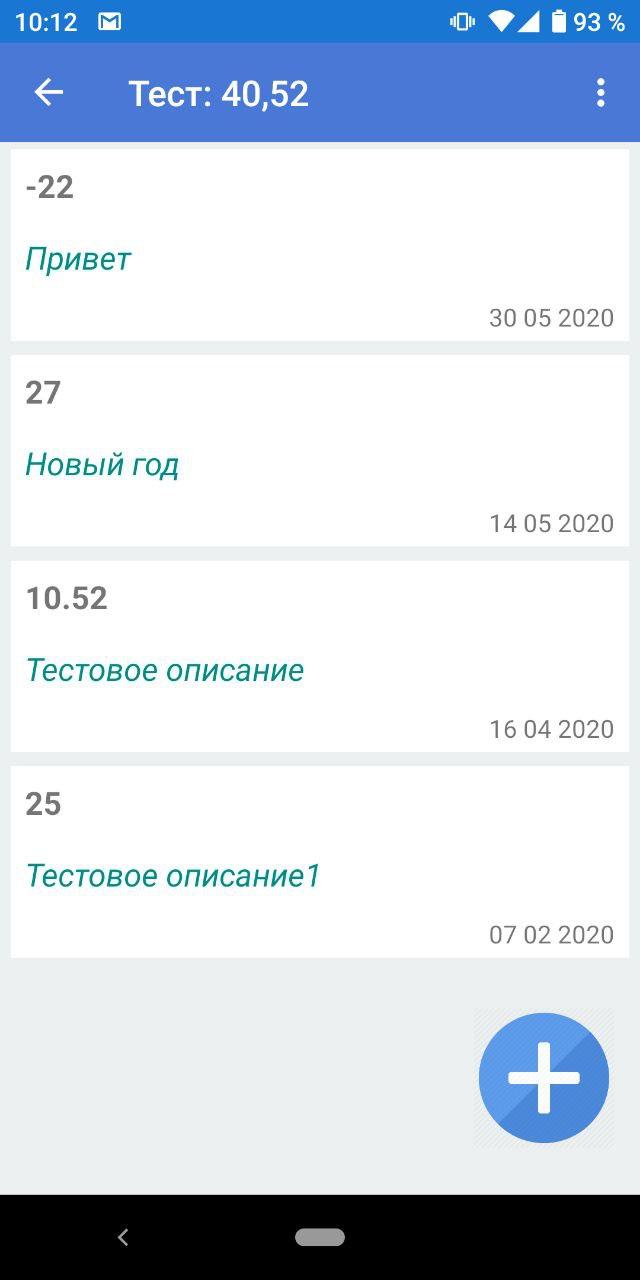


Рис. 14. Сортировка по дате (сначала новые)

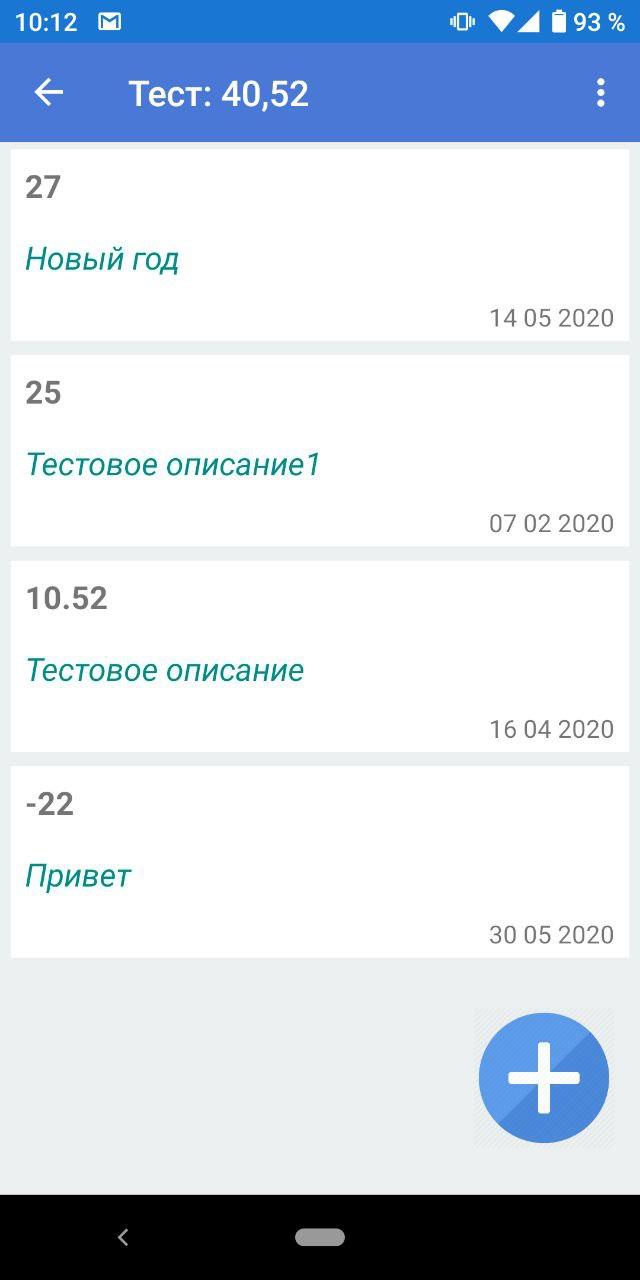


Рис 15. Сортировка по значению (сначала большие)

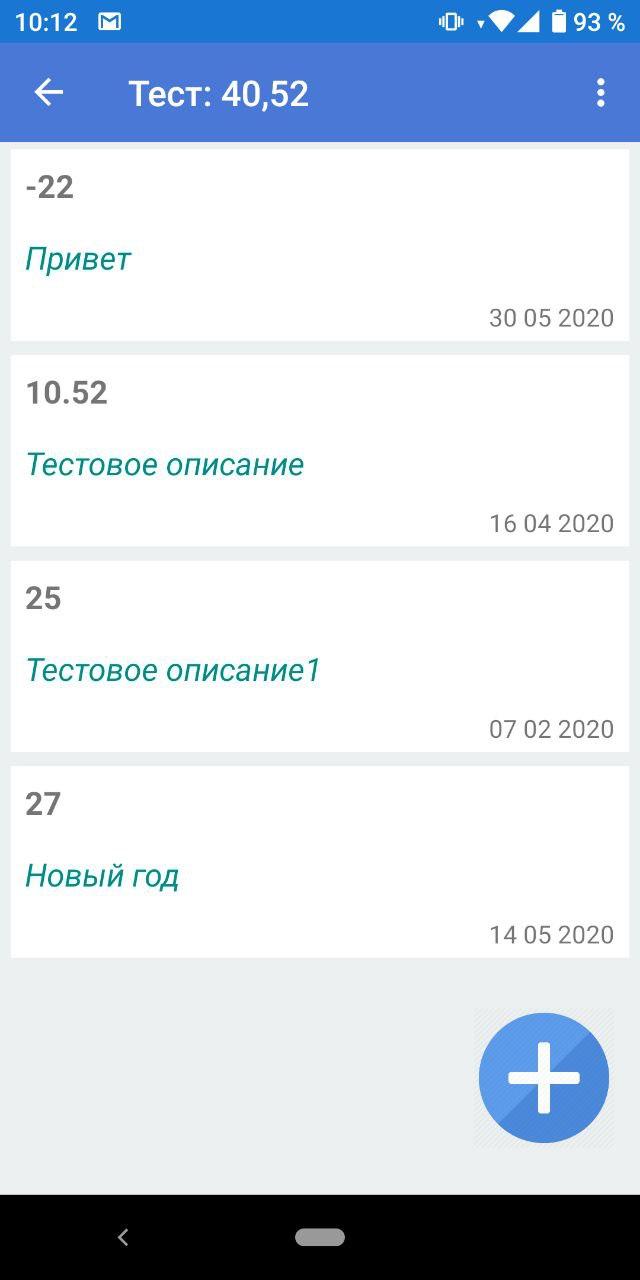


Рис 16. . Сортировка по значению (сначала маленькие)