Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«**СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**»
Институт космических и информационных технологий
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой

_____О.В.Непомнящий

«____»_____20___г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 «Информатика и вычислительная техника»

Мобильное приложение для планирования задач

Руководитель	_____	_____	Старший преподаватель	И. В. Матковский
	подпись	*дата*	*должность, ученая*	
Выпускник	_____	_____	_____	Н. М. Гусейнова
	подпись	*дата*		
Нормоконтролер	_____	_____	Старший преподаватель	И. В. Матковский
	подпись	*дата*	*должность, ученая*	

Красноярск 2022

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«**СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**»
Институт космических и информационных технологий
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой

_____О. В. Непомнящий

«____» _____ 20___г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме бакалаврской работы

Красноярск 2022

Студенту Гусейновой Нигар Мамедрагим кызы
Фамилия, имя, отчество

Группа КИ18-07Б **Направление** 09.03.01
номер *код*

Информатика и вычислительная техника
наименование

Тема выпускной квалификационной работы

Мобильное приложение для планирования задач

Утверждена приказом по университету № _____ от _____

Руководитель ВКР И. В. Матковский, старший преподаватель кафедры
инициалы, фамилия, должность, учёное звание, место работы

ВТ ИКИТ СФУ

Исходные данные для ВКР

Задание от руководителя ВКР

Перечень разделов ВКР

Разработка спецификаций требований, этапы проектирования приложения, разработка приложения, разработанное приложение.

Перечень графического материала

Презентация.

Руководитель ВКР _____ И. В. Матковский
подпись

Задание принял к исполнению _____ Н. М. Гусейнова
подпись

«____» _____ 20____ г.

РЕФЕРАТ

Выпускная квалификационная работа, выполненная по теме «Приложение для планирования задач» содержит в себе: 51 страниц текстового документа, 13 рисунков, 1 таблицу, 18 использованных источников, 7 приложений.

ПРИЛОЖЕНИЕ, ПРОГРАМННОЕ ОБЕСПЕЧЕНИЕ, РАЗРАБОТКА НА JAVA, СПИСОК ЗАДАЧ, ПЛАНИРОВАНИЕ ДЕЛ

Целью работы является разработка программного обеспечения на языке JAVA для планирования задач.

В данной работе отражены структура, реализация, а также тонкости разработки мобильных приложения. В результате создано приложение для мобильных телефонов с операционной системой андроид.

Результат работы имеет практическую ценность и может быть использован в повседневной жизни.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Каждый человек сталкивается с ситуацией, когда ему необходимо структурировать цели, которые он хочет достичь и оптимизировать пути их достижения. Именно для этого существуют приложения, которые позволяют: отмечать цели, их выполнение, дают возможность создавать заметки и подзадачи, отображают статистику, напоминают о целях. Данные приложения получили популярность благодаря тому, что визуальное представление задачи облегчает ее решение.

Для того, чтобы разработать приложение позволяющие помочь пользователю оптимизировать время, необходимо ознакомиться с принципами тайм-менеджмента [1].

Тайм-менеджмент – это методы и техники позволяющее наиболее эффективно управлять временем, помогающие сэкономить время и ресурсы. Основными принципами тайм-менеджмента являются: приоритизация, планирование и структурирование.

Приоритизация – это задания задаче важности, срочности и сложности.

Планирование – это определение сколько времени потребует задача и в какие сроки ее следует выполнить.

Структурирование – это отслеживание выполнения задачи и ее результатов.

Большая часть техник тайм-менеджмента опирается на приоритизацию и планирование, и лишь малая часть содержит все три принципа.

Цель работы – Создание мобильного приложения, совмещавшего различные варианты наиболее удачной реализации существующих решений, а также обладающего особенностями, которых нет у других аналогичных приложений, обеспечивающих преимущества перед конкурентами.

Интерфейс приложения должен быть интуитивно понятным и внешне привлекательным.

Актуальность и практическая значимость работы подтверждается популярностью аналогичных приложений и повышенным к ним интересом.

Структура работы отображает решаемые задачи:

– анализ существующих решений и разработка технического задания;

– разработка структуры приложения и решений на основе технического задания;

– реализация всех компонентов мобильного приложения;

– тестирование приложения.

1 Разработка спецификации требований

1.1 Анализ задания на выпускную квалификационную работу

Есть множество аналогов приложений, в которых, можно создать, редактировать и отмечать выполненными цели, реализованных как на компьютерные версии, так и в мобильных приложениях. Согласно заданию, реализуются мобильное приложение, в следствии чего, необходимо реализовать преимущества среди аналогичных мобильных приложений, путем введения новых возможностей или более удачных решений. Для выявления сильных и слабых сторон, был проведен анализ схожих приложений, а также исходного открытого кода. Учитывая это, была разработана спецификация требований к создаваемой системе.

1.2 Анализ методов интегрируемых в приложения

Эффективное выполнение максимального количества дел в короткие сроки означает продуктивность и без системного подхода с этим никак не справится. Необходимо последовательно составлять список дел, учитывая приоритет и объем задач. Существующие инструменты и методы позволяют четко структурировать задачи и последовательно разложить их в расписании.

У любого аналогичного приложения есть возможность создания целей, редактирования, удаления целей, а также возможность ставить отметки о выполнении цели.

Существуют основные методы, которые разработчики интегрируют в свои приложения [2]. Один из них это Getting Thing Done (GTD), что в переводе означает - доводить дело до конца. Основные постулаты данной методики:

– записывать все дела, освобождая место в голове;

– приоритизация и сортировка задач. Все задачи, которые можно делегировать – перепоручить. Все, что можно сделать за две минуты, делаем сейчас;

– дела, на которые нужно больше времени, разбиваем и сортируем на маленькие шаги и расставляем в последовательности их выполнения;

– начать делать что-то прямо сейчас. Расставленные в правильном порядке приоритетности и времени выполнения задачи - составляют четкий план действий, который позволяет без стресса решить дела.

Вторым по популярности решением является Kanban, метод Канбан-доски, в данном контексте означает “сигнальная карточка”. Сам метод, в первую очередь о визуализации и управлении потоком задач, идеально подходит для сложных проектов. Содержит довольно простые правила:

– для личного пользования достаточно доски из трех столбцов. Которые отображают текущий статус расположенных в них задач: “To do” – Дела, ожидающие своей очереди “In Progress” - то, что вы выполняете в данный момент и “Done” - выполненные задачи;

– Канбан, как и GTD, требуют ограничения находящихся в процессе выполнения задач. В совершенстве – выполнять дела по одному переходя к следующему, лишь по завершению предыдущего.

Такая система задач показывает вашу реальную загруженность, положение дел и помогает отслеживать прогресс.

И последнее, довольно востребованное решение, Метод Pomodoro — это техника повышения продуктивности. Основной принцип — это деление рабочего времени на четкие интервалы, во время которых требуется концентрироваться и выполнять планы и интервалы отдыха между ними. Название метод получил благодаря таймеру в виде помидора. Представляет собой четыре простых правила:

– определение задач на день;

– установка таймера на 25 минут;

– концентрация на одной задаче из списка задач;

– после выполнение задачи, устраивается пятиминутный перерыв и аналогично предыдущей задаче выполняется следующая;

– полноценный перерыв на 15-30 минут после цикла из четыре задач.

Данная методика помогает покончить с прокрастинацией, благодаря четко заданным временным промежуткам.

Также полезным инструментом, является отслеживание привычек, которые способствуют повышению индивидуальной производительности, помогают получить полезные привычки и избавиться от вредных привычек.

Для своего приложения предпочтем метод GTD, как наиболее удобный и простой в реализации способ планирования задач.

1.3 Анализ существующих приложений

Богатые функциями приложения подойдут тем, кто загружен разнообразными задачами, любит все структурировать, помечать и записывать, а более простые тем, кто предпочитает тратить меньше времени на внесение и обработку дел.

Независимо от типа планировщика, главным критерием является баланс наглядности, интуитивности и функциональности использования.

Одним из популярных примеров является – Trello [3], разработанная Fog Creek Software, облачная программа, имеющая компьютерную и мобильную версию. Trello – инструмент, который позволяет организовать, что угодно от крупного проекта до мелких задач.

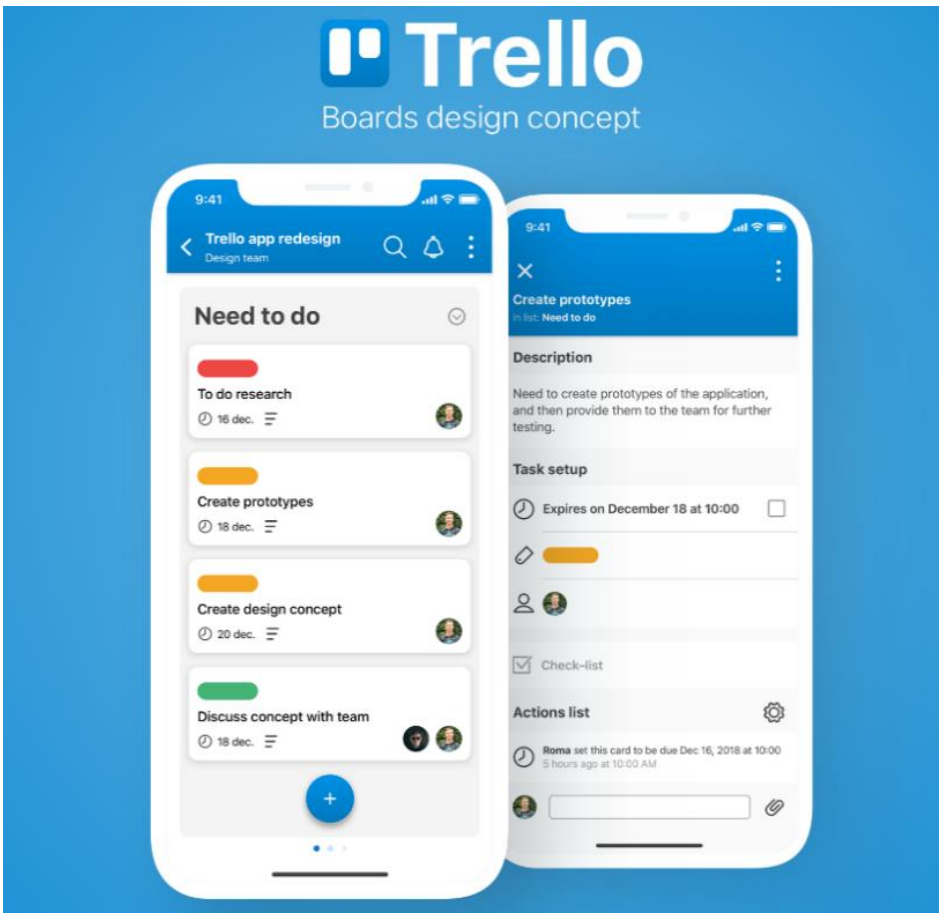

Рисунок 1 – Приложение Trello

Одним из гармоничных сочетаний между возможностями и интерфейсом является - Any.do [4]. Основной особенность данного приложения являются – моменты. При активации моментов, задаются вопросы о некоторых задачках. На основе, которых создается и содержится в порядке список дел, которые необходимо выполнить сегодня и завтра.

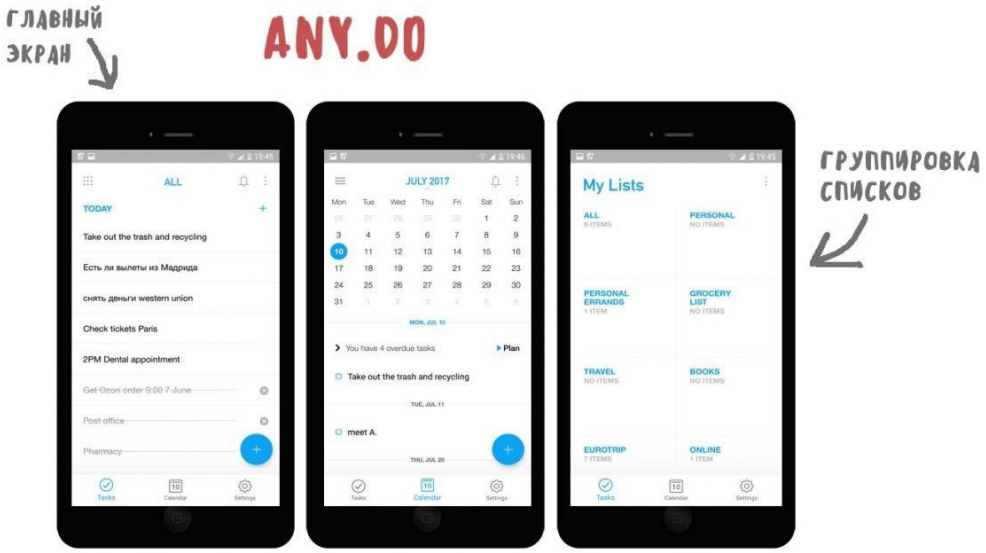

Рисунок 2 – Приложение Any.do

Представим результаты анализа в виде таблицы, которая описывает основной функционал приложений (Таблица 1).

Таблица 1 – Сравнение характеристик приложений для планирования задач

Название	Any.do	Trello
Создание задач	+	+
Командная работа	+	+
Интеграция с другими приложениями	-	+
Поиск	-	+
Уведомление	+ (платно)	+
Статистика и визуализация	-	-
Используемый метод	GTD, Pomodoro	Kanban

На основе данных приложений, можно сделать логическое заключение о том, что приложение должно обладать большинством из описанных выше характеристик. Пользователям нравится видеть статистику выполнения

плана задач, поэтому — это также станет одной из возможностей приложения.

1.4 Разработка технического задания

Назначение приложения

Планировщик задач предназначен для выполнения следующих задач:

– добавление, редактирование и удаление задач;

– наличие подзадачи;

– выбор уровня приоритета для каждой задачи

– выбор даты выполнения для задачи;

– вывод списка задач, в зависимости от текущей даты;

– вывод списка задач, в календарном представлении;

– сортировка списка задач.

Цель приложения

Приложение должно помогать пользователю структурировать и отсортировать его текущие, а также будущие дела, и выводить задачи в удобном порядке, в соответствии с приоритетом.

1.5 Вывод по разделу 1

В первом разделе выпускной квалификационной работы были определены задачи и сформулирована цель. В данном разделе рассмотрены аналоги приложений для планирования задач, также рассмотрены основные методы планирования, которые используются для реализации в аналогах.

2 Этапы проектирования приложений

2.1 Структура приложения

Разработчики предпочитают применять шаблон архитектуры программного обеспечения для разработки приложений, для операционной системы Android. Применение шаблона архитектуры придает проекту модульность и упрощает возможность тестирования. Это облегчает дальнейшее сопровождение программного обеспечения и расширения функционала приложения в будущем. Одним из популярных шаблонов среди разработчиков является шаблон – чистая архитектура [5]. Данный шаблон предлагает разделить код на три составляющих компонента. При создании класса или файла, необходимо поделить его на один из следующих уровней:

– **уровень представления**: представляет собой слой пользовательского интерфейса, содержащий компоненты, видимые пользователям, с которыми он может взаимодействовать. Представление также отвечает за визуализацию данных из базы данных. В нашем приложении эти операции разделены между уровнем представления и пользовательского интерфейса. Уровень пользовательского интерфейса содержит активность и фрагменты, фиксирующие события и отображающие данные. Уровень представления отвечает за формирования данных для того, чтобы пользовательский интерфейс отображал их необходимым способом;

– **уровень бизнес-логики**: уровень, на котором выполнятся требования, которым приложение должно соответствовать, здесь выполняются все необходимые операции, такие как сортировка списка, подсчет статистики;

– **уровень данных:** на этом уровне находятся данные и способ доступа к ним. Делится на уровень хранилища и источника данных. Уровень хранилища отвечает за получение данных, определяет способ получения доступа к данным. Уровень источника отвечает за получение данных.

Взаимодействие между слоями происходит через два интерфейса один для запроса и один для ответа Структура представлена на рисунке 3.

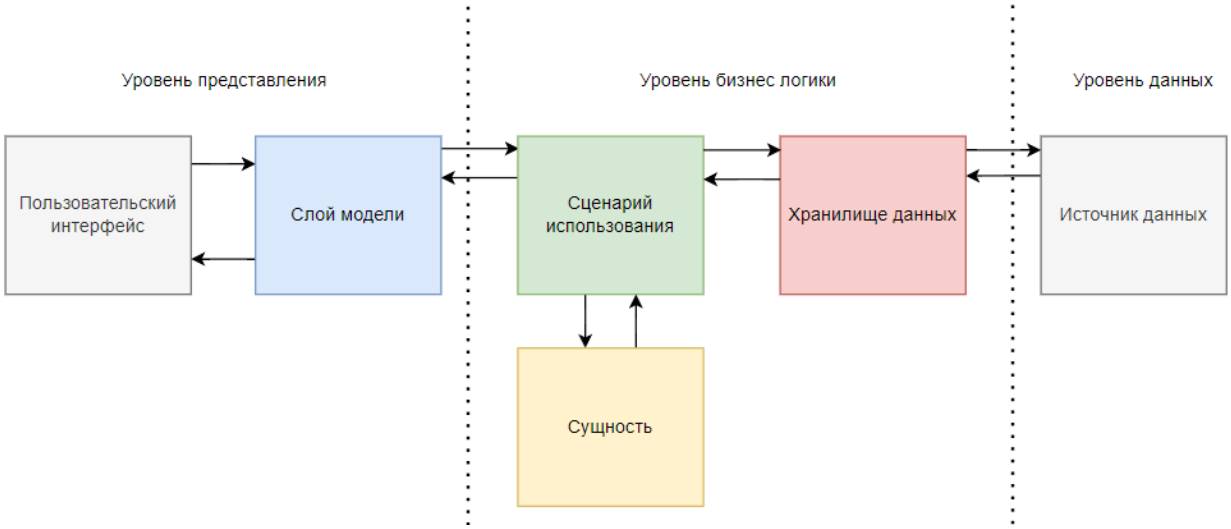

Рисунок 3 – Структура чистой архитектуры на Android

2.2 Взаимодействие классов

Хранилище данных содержит классы: App.java и AppDatabase.java. В классе App, будет инициализироваться наша база данных, AppDatabase содержит реализацию базы данных SQLite.

На уровне бизнес-логики содержаться классы: TaskDao.java, Task.java. TaskDao представляет собой интерфейс для работы с данными, а если точнее с базой данных. Task – это модель в которой содержаться поля базы данных.

Слой представления, классы: TaskDetailsActivity, MainActivity, содержат в себе логику взаимодействия с пользователем. Класс – ModelViewActivity, содержит в себе список, а класс Adapter - работает со списком.

Схема представлена на рисунке 6

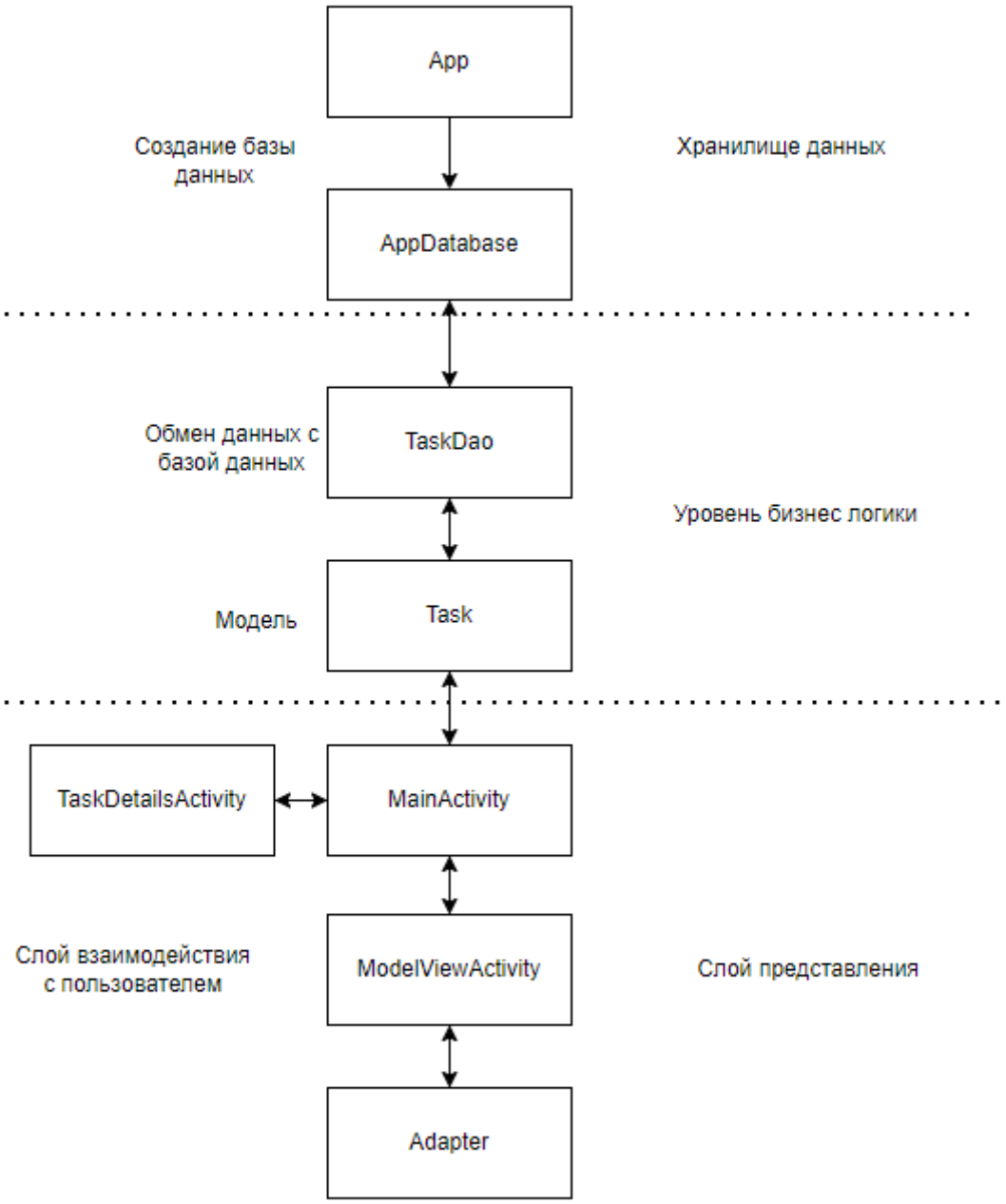

Рисунок 6 – Схема взаимодействия классов

2.3 Основной функционал

Описание основного функционала представлено с помощью диаграммы вариантов использования UML [7] на рисунке 5.

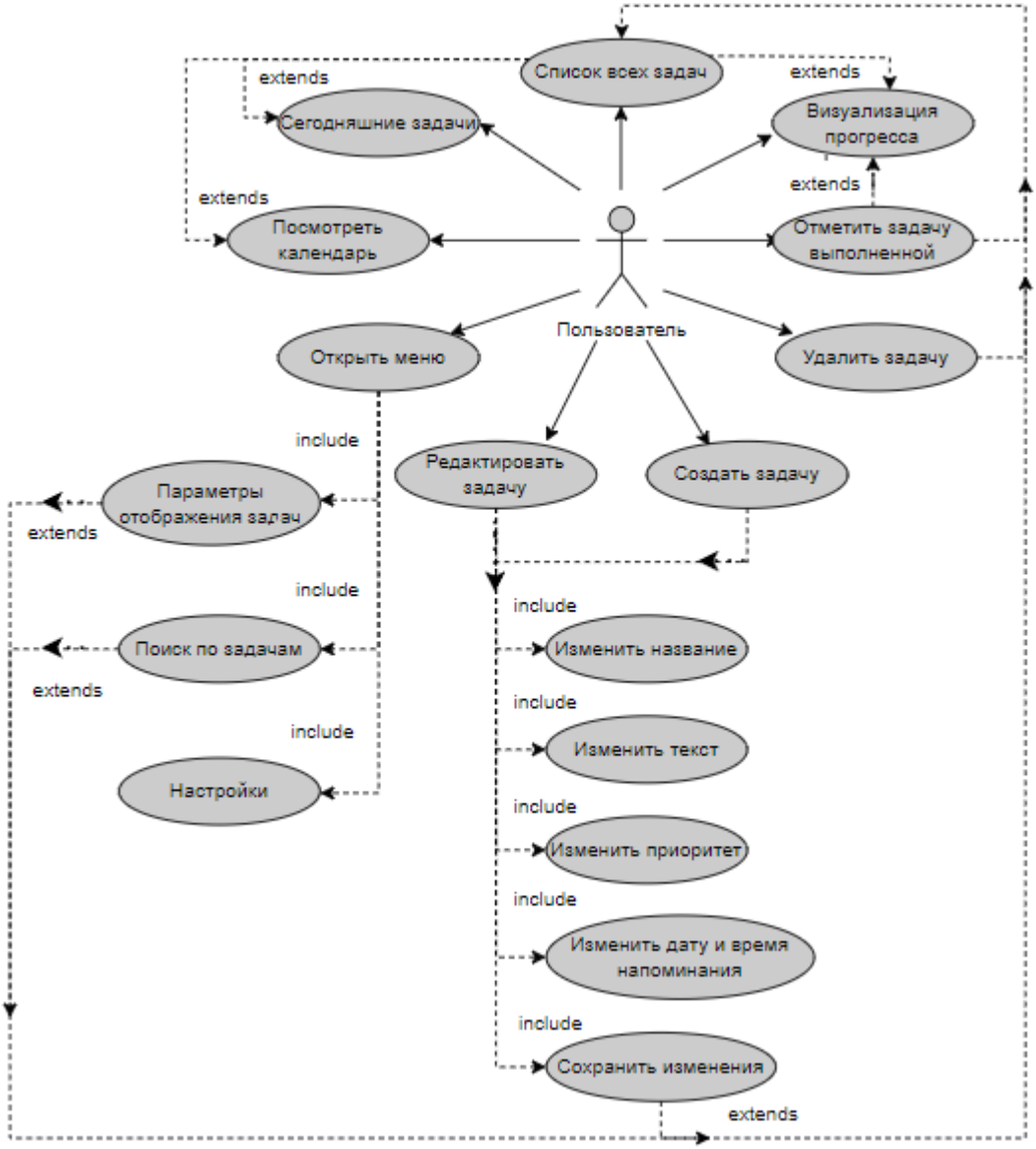

Рисунок 5 – Диаграмма вариантов использования UML

На главном экране пользователь получает доступ к:

– сегодняшнему списку задач;

– списку всех задач;

– календарю;

– меню;

– визуализации прогресса;

– созданию задач.

К любой задаче из списка могут быть применены следующие действия:

– отметить выполненной;

– редактировать;

– удалить.

В меню пользователь получает доступ к:

– поиску;

– параметрам отображения задач;

– настройкам.

В календаре пользователь видит дни в зависимости от загруженности и при выборе даты, видит список задач на выбранный день и может его изменять, добавлять и удалять задачи.

Визуализация статистики – отображает окно, на котором находится, виртуальный домашний питомец, чье состояние динамично изменяется в зависимости от текущего прогресса пользователя.

2.4 Вывод по разделу 2

В данной главе был обозначен основной функционал и были определены основные способы реализации. Приведена структура архитектуры, схема взаимодействия классов, прототип и диаграмма вариантов использования.

3 Разработка приложения

3.1 Взаимодействие экранов

Главный экран, подгружает в себя компонент со списком, в зависимости от выбора пользователя. При создании, редактировании или удалении задачи, открывается экран редактирования или создания задачи, после выхода из него пользователь возвращается в главный экран. Слои взаимодействуют между собой как показано на рисунке ниже.

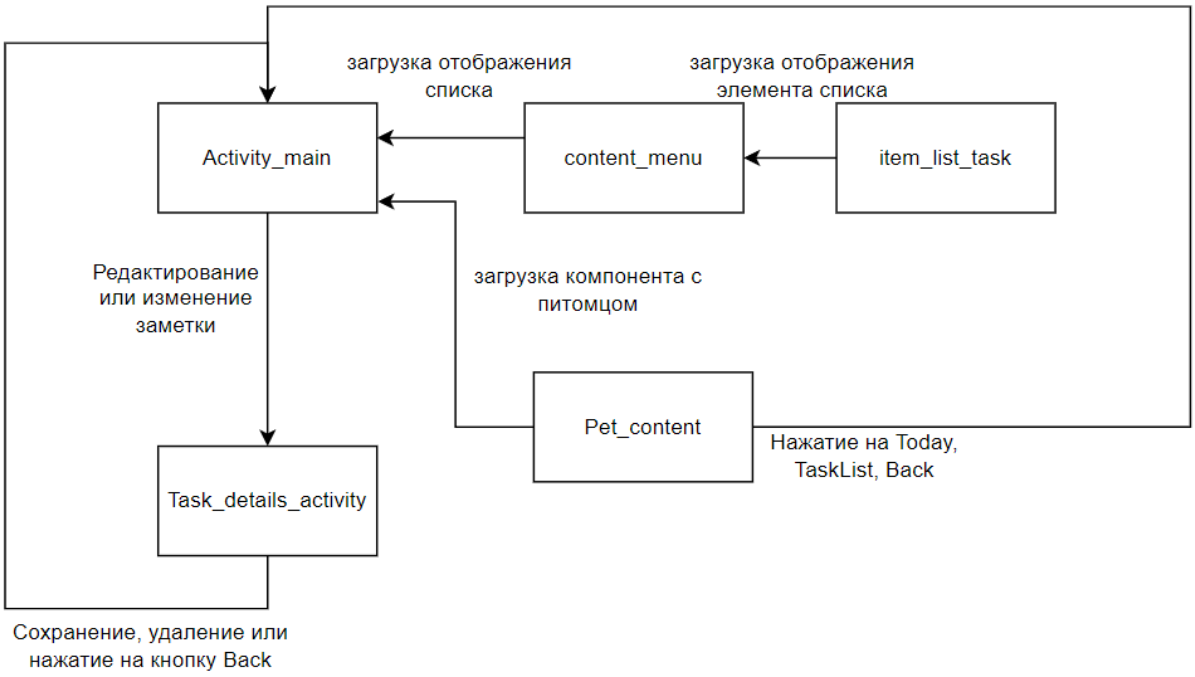

Рисунок 7 – Схема изменений интерфейса для пользователя

3.2 Используемые библиотеки

Для реализации приложения используются библиотеки ROOM [8], которая представляет собой интерфейс для работы с SQLite, упрощая и ускоряя работу с базами данных. SQLite – представляет собой локальную базу данных доступную пользователю и хранящуюся на его устройстве.

Архитектурный компонент жизненного цикла - Lifecycle [9], представляет собой удобную реализацию обновления данных. Вместо того, чтоб пересоздавать окно при изменениях, можно использовать данный компонент, который автоматически сообщает о своих изменениях.

3.3 Слой модели

Слой модели — это внутренний слой, независящий от всех слоев, будет содержать класс “Task”, в модели мы обозначаем основные поля нашей базы данных SQLite:

– идентификатор задачи;

– текст задачи;

– текст подзадачи;

– время создания;

– время напоминания;

– уровень приоритета;

– степень выполнения.

Реализовываем интерфейс Parcelable [10] для того, чтобы передавать задачу между активностями.

Используем аннотацию, это специальные библиотеки, которые упрощают написания кода. В данному случае помечаем класс как сущность с помощью, аннотации Entity [11] для того, чтоб его можно было использовать для сопоставления с SQLite в базе данных. Все поля данного класса будут сохраняться, чтобы при выходе из приложения наши данные сохранялись. Entity нужен обязательный первичный ключ - Primary Key, который и будет идентификатором для нашей заметки. Остальные типы будет соответствовать – ColumInfo.

Добавим метод контракт – Equlas [12] для сравнения объектов. И будем использовать хэш-код для их первоначального сравнения. Хэш-код [12] объекта создается на основе всех его полей, но тип int имеет ограничения, поэтому они могут быть равны у разных объектов, если хеш-коды равны, то будем сравнивать объекты через Equals. Если в Equals объекты равны, то будет их сравнивать по хеш-коду.

Код представлен в приложении (Приложение А).

3.4 Слой презентации

Переходим на следующий уровень на уровень класса - слой презентации. Date Access Object (DAO) [12]- это объект через который осуществляется доступ к базе данных SQLite: запись, чтение, выборка. Добавим интерфейс, который имеет название TaskDao. Помечаем код специальной аннотацией “@Dao” для того, чтобы библиотека room определял его, как DAO. В Объекте доступа к данным, представляются методы, которая остальная часть приложения использует для взаимодействия с данными в таблице. Для начала напишем выборку всех запросов и, аналогичную выборку, использующую LiveData [13], для обновления в режиме реального времени. Пользовательский интерфейс подписывается на данный тип данных и будет обновляться вместе с ним. Также, добавим выборку по дате и поиск по заметкам, выборку по Id, выборку по выполнению. Реализуем вставку и удаление. Используем, OnConflict == OnCondlictStrategy.Replace для вставки, чтобы при совпадении id, вовремя редактировании заметки она изменялась. Интерфейс реализуется с помощь SQL запросов. Результат представлен в приложении (Приложение Б).

Разработаем базу данных. Добавим в пакет date новый класс – AppDatabase. Указываем аннотацию для баз данных @Database, вносим список сущностей, у нас она одна — это Task.class, и добавим текущую версию, для будущих обновлений, чтобы избежать пересоздания базы

данных, а чтоб ее экспортировать. Далее расписываем создание абстрактного класса и в нем указываем абстрактный метод интерфейса TaskDao. Так, при создании класса базы данных, автоматически генерируется реализация интерфейса библиотекой Room.

База данных используется во время работы всего приложения, поэтому для того, чтобы создать базу данных, необходимо использовать объект, который живет все время. Объект типа Apllication [14]– это базовый класс, который поддерживает глобальное состояние приложения. Создаем свою собственную реализацию данного класса, создав подкласс и подключив его в файле AndroidManifest.xml в качестве класса “android=name” указав имя класса. Метод OnCreate() в классе Apllication, который гарантированно будет вызван до старта работы приложения.

Добавим файл типа Application - App. В котором укажем базу данных и интерфейс. Для создания базы данных используется следующая структура: AppDatabase db = Room.databaseBuilder(getApplicationContext(), AppDatabase.class, "database-name").build();.

Создадим методы для доступа к объектам. Реализуем через паттерн singleton [15]. Потому что данный паттерн очень удобно использовать для объекта, у которого один экземпляр, а также он прост в реализации. Готовый файл date содержится в приложении В.

Главный экран состоит из нескольких компонентов, активность, отображающая список, активность для создания и редактирования заметок и календаря, создадим слой презентации - модель для отображения данных в списке. Представляем доступ к данным о списке, для его отображения. Используем Singleton и реализованные в базе данных методы (Приложение Г).

Далее переходим к слой View, это слой непосредственной работы с пользователем, в файле activity_main.xml, располагаем нужные нам элементы, кнопки. В файле content_main.xml задаем список с помощью

И подгружаем его в главную активность.

На главном экране располагается кнопка сегодняшних дел, кнопка всего списка, календарь, меню, кнопка добавления заметки и список. Главный экран изображен на рисунке 8.

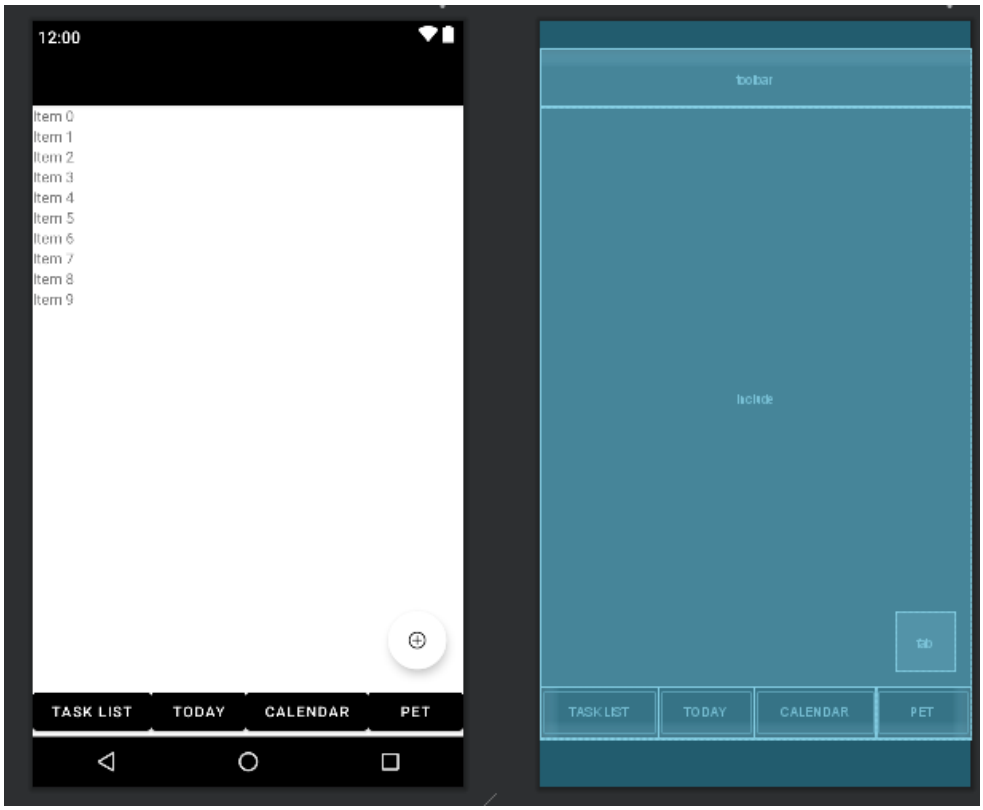

Рисунок 8 – главный экран

Инициализируем Toolbar [16] – это панель инструментов, она уже поддерживает набор функций, таких как кнопки навигации, отображения заголовков и логотипов, добавление меню.

Для отображения списка мы используем RecyclerView [17]– он может отображать любые данные и как угодно, главное дать соответствующий

менеджер размещения, для отображения списка сверху вниз необходимо использовать LinearLayputManadger – они отвечают за размещения компонентов в списке, а для отображения разделителей между элементами списка DividerItemDecoraration.

Для определения вида каждого элемента используем адаптер. Но адаптеру нечего отображать поэтому создадим экран для создания новой задачи.

Экран заметки будет универсальным, использовать будем его, как для создания, так и для редактирования задачи.

Добавим в пакет screen пакет details, создадим в нем TaskDetailsActivity. Добавил файл в Manifest.

Передавать задачу будем целиком, используя Bundle, он предназначен для передачи данных между компонентами приложения, в данном случае, между активностями. Добавим вызова активности для того, чтобы каждый раз не писать код вызова, используем функцию для вызова и будем обращаться к ней по необходимости. Мы можем вызывать данную активность из разных мест в программе с помощью данной универсальной функции.

Перед передачей задачи для ее дальнейшего редактирования или создания, проверим задачу на существования если она есть прикрепляем его к Intent.

Далее создадим Layout activity_task_deatils. Добавим ToolBar, Текст задачи, текст подзадачи, календарь, и выбор приоритета, кнопку удаления и сохранения задачи (Рисунок 9).

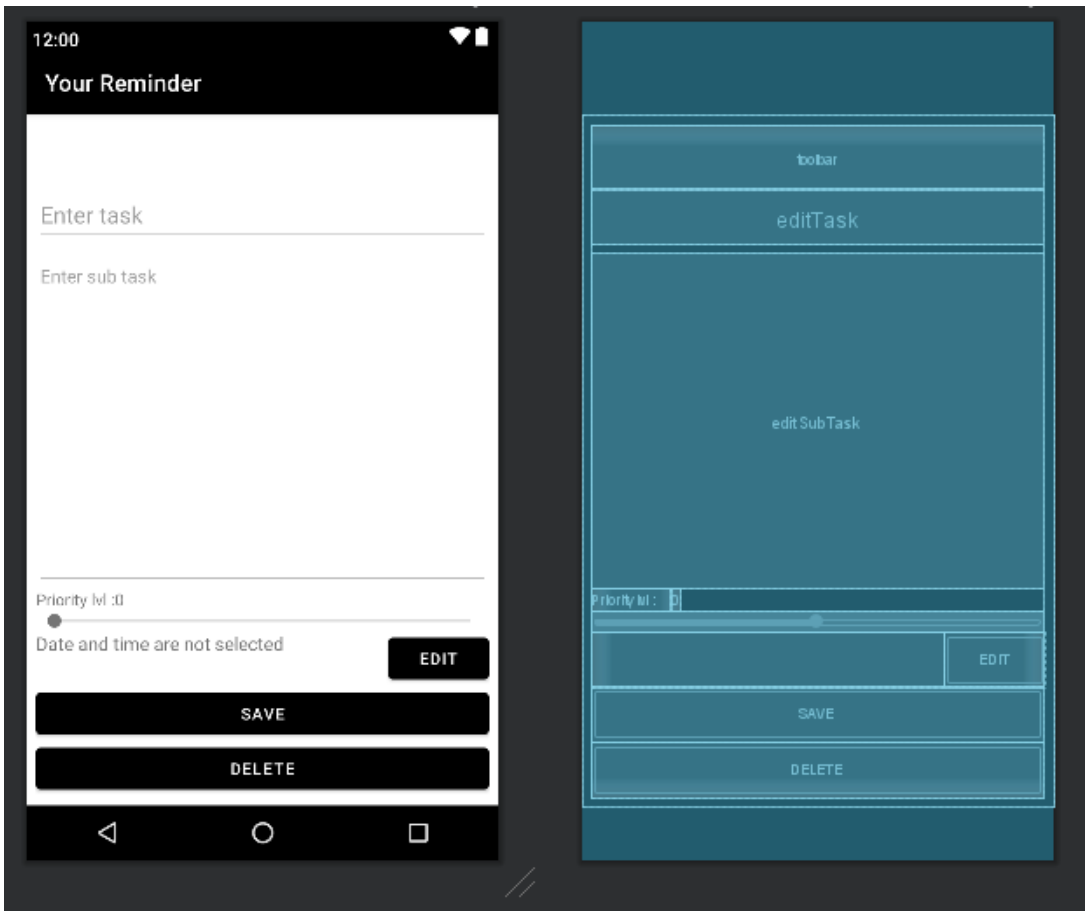

Рисунок 9- внешний вид окна редактирования и создания задачи

Функционал в создании заметки будет реализовать постепенно для начала добавим простую функцию добавления заметки, а далее реализуем остальные поля.

Добавим инициализацию компонентов с активности. Зададим Toolbar в качестве ActionBar, включаем в него кнопку назад и введем текст заголовка.

Используем функцию getIntent(), для получения данных, которые использовались при создании активности. Если заметка передана, значит мы ее редактируем, считываем параметры с полей. Если заметки нет, значит мы ее создаем.

Для обработчика кнопок используем атрибут в xml файле и далее в java реализуем его

Мы считываем данные из поля редактирования или создания заметки в поля модели, которые передаются в базу данных. Время сохраняем

нынешнее. По Intent проверяем, если по нему передавалась задача, то это редактирование заметки, если нет, то создание. Используем функции, созданные нами в TaskDao для работы с базой данных. Обновление передавать в главную активность не нужно, потому что мы используем LiveDate и при изменении в базе данных, данные передадутся автоматически.

При удалении заметки проверяем была ли такая заметка созданной, если нет, то выходим из активности, если да, удаляем ее и также закрываем активность редактирования и создания заметки.

Приступим к созданию адаптера списка. Создадим layout для отображения одного элемента списка item_task_list (Рисунок 10).

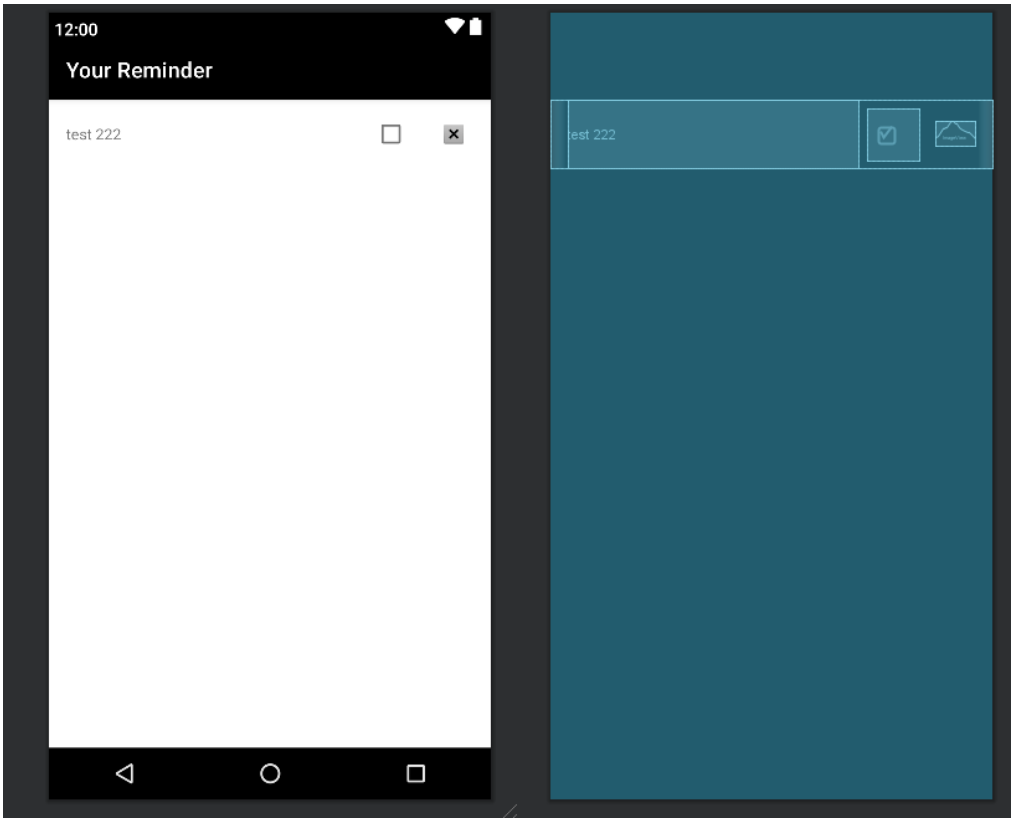

Рисунок 10 – слой для отображения одной задачи

3.5 Слой контроля

Создадим адаптер на основе этого слоя в папке main, создадим класс Adapter, добавим ему ViewHolder[18] – это класс, который описывает представление объекта. RecycleView.Adapter должен содержать ViewHolder. Создание элемента в списке через ViewHolder происходит по необходимости, он хранит в себе ссылки на элементы. Инициализируем: текст, кнопку выполнения и кнопку удаления заметки

В функции bind, передадим задачу, запомним ее. Загрузим ее задачу. И установим обновления вычеркивания. Используем переменную в качестве костыля, чтоб при создании задачи, обновление вычеркивания не срабатывало.

Реализуем оставшиеся функции. OnCheckedChanged сохраняет значение отметки заметки из списка и обновляет ее значение, также обновляет значение из списка

Отображение зачеркивание в списке реализуем отдельной функцией, она работает путем побитового умножения на нужный байт флага и побитового сложения.

Если задача отмечена выполненной, то при сложении с STRING_THRU_TEXT_FLAG, которая содержит 1 на 16 позиции бита, в любом случае заметка отметиться положительной, ну и обратная операция, в случае, если заметка отмеченная не выполненный это умножение на эту ж переменную, но инверсивную, тогда мы получим в любом случае 0, в необходимой позиции, для дальнейшей отрисовки, зачеркнутой или нет. Кнопка удаления в списке использует созданные функции из TaskDao.

Нажатие на элемент в списке открывает окно редактирования. Вызывая активность, через созданную нами функцию start().

Для автоматической работы анимации используем класс SortedList, он автоматически определяет изменения и выполняет соответствующие команды, на основе CallBack, который имеет метод – compare, сравнения двух элементов, записываем позиции, которые мы хотим сравнить, выполненные дела будут ниже всего, далее по дате создания, потом по дате напоминания, и по приоритету.

OnChanged обновляет измененные позиции.

AreContentTheSame реализует тоже, что и Equals, она возвращает true, если одинаковые id и одинаковое содержание, то есть элементы равны.

AreItemsTheSame указывает на элемент после изменения, у них одинаковые id, но разный текст.

OnInserted – функция, которая сообщает о добавление в список, с помощью NotifyItemRangeInserted уведомляем адаптер об изменениях.

Аналогично поступаем с функциями onRemoved и onMoved. OnRemoved – сигнализирует об удалении, а onMoved об перемещение элемента в списке.

Создаем ViewHolder. Привязываем отдельную заметку к ViewHolder, используем созданный нами item_task_list. Число элементов в адаптере равно числу элементов в списке. Добавим функцию, которая позволит обновить или заменить существующий список.В MainActivity добавим адаптер. Также в MainActivity добавим создание задачи, открытие активности, для создания и редактирования задач. Добавим MainViewModel со списком в главную активность.

3.6 Вывод по разделу 3

В разделе 3, было реализовано готовое приложение и описаны средства и тонкости реализации. Также была приведена схема взаимодействия слоев отображения.

4 Разработанное приложение

4.1 Главное меню

Приложение имеет светлую и темную версию, которые отображаются в соответствии с настройками устройства. Главное меню выглядит как представлено на рисунке 11. В нем можно: получить доступ ко всему списку задач, добавить задачу, открыть окно календаря и меню. При нажатии на кнопку добавление заметки открывается редактор заметки (Рисунок 12).

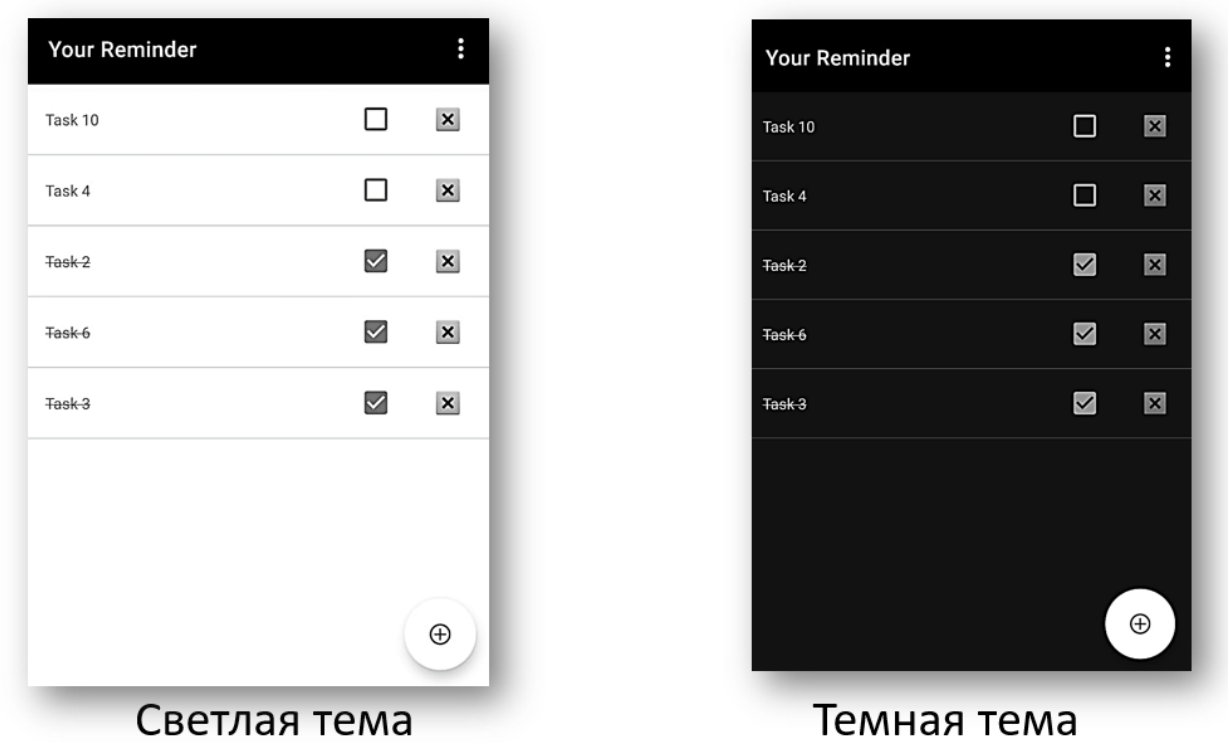

Рисунок 11 – Главное меню

При нажатии на кнопку – Task List, выводится общий список дел.

При нажатии на кнопку – Today, выводится список дел на сегодня.

При нажатии на кнопку – Calendar, открывается календарь, в котором можно изменять, создавать и удалять дела, запланированные на определенную дату.

При нажатии на кнопку – Pet, открывается окно с виртуальным питомцем, который отображает прогресс дел на сегодня.

При нажатии на задача открывается окно редактирования заметки (Рисунок 12).

При нажатии на кнопку выполнения в виде квадрата у конкретной задачи, задача отмечается выполненной и вычеркивается как представлено на рисунке 13.

При нажатии на крестик возле заметки, заметка удаляется.

При нажатии на кнопку меню в верхнем левом углу открывается меню, в котором можно получить доступ, к настройкам, поиску и фильтру отображений заметок

4.2 Создание и редактирование заметки

В редакторе заметке можно добавлять, изменять такие параметры заметки как: текст задачи, текст подзадачи, уровень приоритета и менять дату для выполнения.

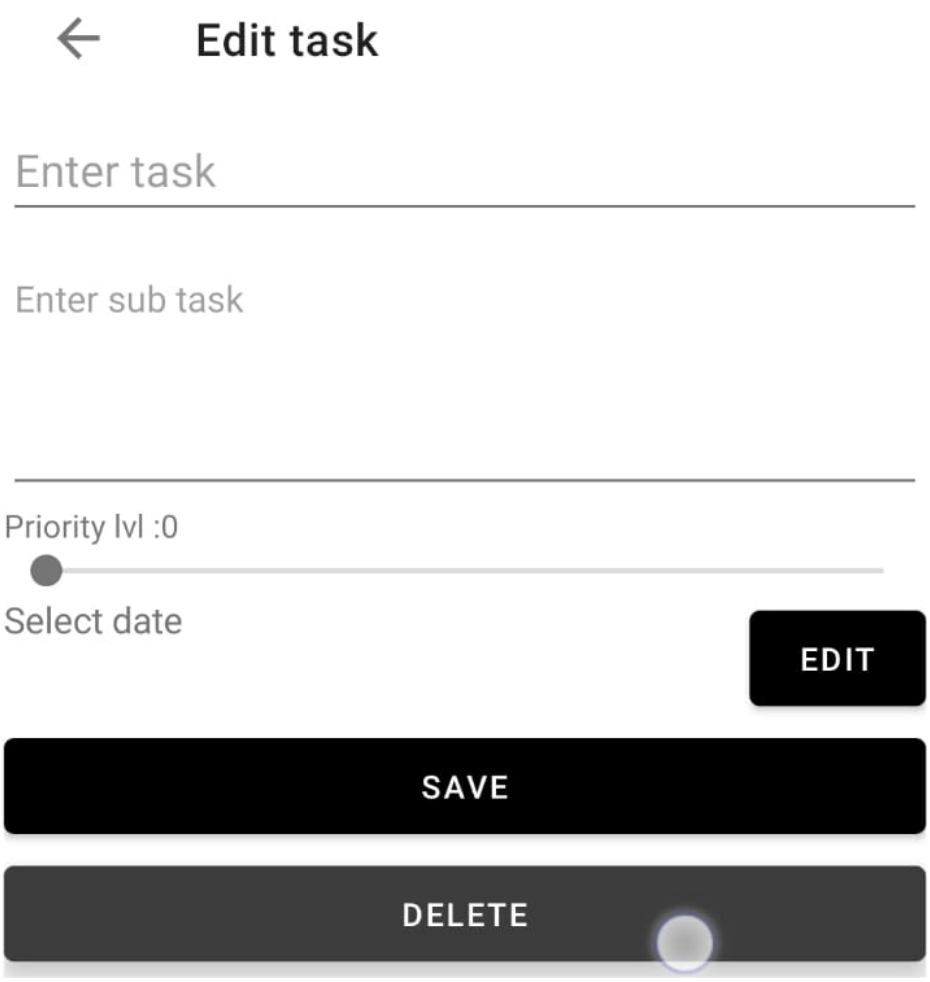

Рисунок 12 – Редактирование заметки

В поле заголовок вводится название задачи.

В поле подзадача вводятся подзадачи, при желании пользователя.

При перемещении ползунка на слайдере, меняется уровень приоритета задачи, в диапазоне от 0 до 10.

При нажатии на кнопку – Edit, открывается окно выбора даты и времени.

При нажатии на кнопку – Save, изменения в заметки сохраняются, если заметка не существовала ранее, она создается.

При нажатии на кнопку – Delete, окно закрывается, пользователь возвращается в главное меню, изменения не сохраняются, и, если задача существовала ранее, она удаляется.

При нажатии на кнопку – Back, окно закрывается, и пользователь возвращается в главное меню, изменения не сохраняются.

4.3 Изменение положения в списке

Список настроен динамично и при изменении состояния задачи, обновляет список и все выполненные задачи опускаются в конец списка (Рисунок 13). При любых изменениях, список задач сортируется заново, сначала по отметке о выполенности, далее по приоритетности и по дате напоминания.

Рисунок 13 – Отметка о выполнение заметки

При изменении поля приоритет на более высший уровень, задача поднимается выше по списку.

При отмечании задачи выполненной, она опускается в низший уровень списка.

При изменении даты и времени на более раннюю, задача поднимается в верх, но лишь на тот уровень приоритета, в котором состоит.

4.4 Тестирование

Данное приложение было протестировано вручную, есть недостатки в виде долгой загрузки базы данных при открытии приложения. Проверка сохранения задач, показала, что независимо от выхода новых версий, список будет сохранен и не вредим. Остальной реализованный функционал работает без нареканий.

4.5 Перспектива развития

Приложение имеет гибкую структуру, в следствии чего, его просто улучать и обновлять, в дальнейшем планируется добавить в приложение: интеграцию с другими приложениями, статистику в виде виртуального питомца, выбор различных виртуальных питомцев, мотивирующие цитаты на каждый день, поиск, корзину удаленных задач. Также в грядущих реализациях ожидается глобальное расширение, в котором приложение будет иметь связь с сервером, в котором, также можно будет посмотреть свои задачи и благодаря которому появится: авторизация пользователей, синхронизация списка, создания совместных задач с другими пользователями.

ЗАКЛЮЧЕНИЕ

В данной дипломной работе был разработан планировщик для задачи на языке Java. Были реализованы функции добавления, редактирования и удаления задачи. Также реализованы возможность сортировки списка, просмотра выполненных задач, поиска по задачам, добавлены напоминания.

Данный планировщик доступен в свободном доступе [18].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Наталья Шацкова, Тайм-менеджмент: 15 методов эффективного управления временем. – 2022. - URL: https://trends.rbc.ru/trends/education/606335659a7947a191c4b092 (дата обращения: 13.11.2021).
2. 10 сервисов для организации списка дел и повышения продуктивности в 2021 году. - 2021. - URL: https://habr.com/ru/post/541088/ (дата обращения: 13.11.2021).
3. Trello: официальный сайт. - URL: https://trello.com/ (дата обращения: 13.11.2021).
4. Any.do: официальный сайт. - URL https://www.any.do/ (дата обращения 13.11.2021).
5. Как реализовать чистую архитектуру на Android? – 2019. – URL: https://habr.com/ru/post/459402/ (дата обращения 01.02.2022).
6. Макет проекта, Figma: официальный сайт. - URL: https://www.figma.com/file/pU5AqYIrhZXNATpJV6cWLN/A_for_U?node-id=0%3A1 (дата обращения 23.12.2021)
7. Диаграмма вариантов использования UML. – 2019. - URL: https://coderlessons.com/tutorials/kompiuternoe-programmirovanie/uchebnik-uml/7-diagramma-variantov-ispolzovaniia-uml (дата обращения -02.06.2022)
8. Дмитрий Виноградов, Урок 5 Room Основы. -2018. - URL: https://startandroid.ru/ru/courses/architecture-components/27-course/architecture-components/529-urok-5-room-osnovy.html (дата обращения -03.03.2022)
9. Developer Android, AndroidX, Release Notes, Lifecycle. -2022. – https://developer.android.com/jetpack/androidx/releases/lifecycle(дата обращения -03.03.2022)
10. Александр Климов, Parcelable. Передаем объекты. - 2022. - URL: http://developer.alexanderklimov.ru/android/theory/parcelable.php (дата обращения -03.03.2022)

11. Developer Android, AndroidX, Annotation, Entity -2022. – URL: https://developer.android.com/reference/androidx/room/Entity (дата обращения - 03.03.2022)

12. Developer Android, AndroidX, reference, java.lang, Object - 2021. – URL https://developer.android.com/reference/java/lang/Object (дата обращения -06.03.2022)

13. Developer Android, AndroidX, libraries, livedata -2022. – URL https://developer.android.com/topic/libraries/architecture/livedata (дата обращения -06.03.2022)

14. Developer Android, AndroidX, app, Aplication -2022. – URL https://developer.android.com/reference/android/app/Application (дата обращения -12.04.2022)

15. Singleton https://javarush.ru/groups/posts/2365-patternih-proektirovanija-singleton

16. Developer Android, AndroidX, widget, Toolbar -2021. – URL https://developer.android.com/reference/android/widget/Toolbar (дата обращения -12.04.2022)

17. Developer Android, AndroidX, widget, RecyclerView -2021. – URL: https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView (дата обращения -12.04.2022)

18. Github –репозиторий - URL: https://github.com/Skyrllex/YourReminder_TodoList (дата обращения - 08.06.2022)

ПРИЛОЖЕНИЕ А

Task.java

```
@Entity
public class Task implements Parcelable {
@PrimaryKey(autoGenerate = true)
public int uid; //id
@ColumnInfo(name = "texttask" )
public String texttask; //TaskText
@ColumnInfo(name = "textsubtask" )
public String textsubtask; //SubTaskText
@ColumnInfo(name = "timectreate" )
public long timectreate; //date on create
@ColumnInfo(name = "calendar" )
public long calendar; //date and time
@ColumnInfo(name = "priority" )
public int priority; //priority
@ColumnInfo(name = "done" )
public boolean done; //chek do or not
public Task()
{
}
@Override
```

```
public boolean equals (Object otask){
if (this == otask) return true;
if (otask == null || getClass()!= otask.getClass()) return false;
Task task = (Task) o;
if (uid != task.uid) return false;
if (timectreate != task.timectreate) return false;
if (calendar != task.calendar) return false;
if (priority != task.priority) return false;
if (!textsubtask.equals(task.textsubtask)) return false;
return Objects.equals(texttask, task.texttask);
}
@Override
public int hashCode() {
int resultat = uid;
resultat = (31 * resultat) + ((texttask != null) ? texttask.hashCode() : 0);
resultat = (31 * resultat) + ((textsubtask != null) ? textsubtask.hashCode() :
0);
resultat = (31 * resultat) + (int) (timectreate ^ (timectreate>>>32));
resultat = (31 * resultat) + (int) (calendar ^ (calendar>>>32));
resultat = (31 * resultat) + (done ? 1 : 0);
return result;
}
protected Task(Parcel in){
```

```
uid = in.readInt();
texttask = in.readString();
textsubtask = in.readString();
timectreate = in.readLong();
calendar = in.readLong();
priority = in.readInt();
done = in.readByte() != 0;
}
@Override
public void writeToParcel(Parcel dest, int flag) {
dest.writeInt(uid);
dest.writeString(texttask);
dest.writeString(textsubtask);
dest.writeLong(timectreate);
dest.writeLong(calendar);
dest.writeInt(priority);
dest.writeByte((byte) (done ? 1 : 0));
}
@Override
public int describeContents() {
return 0;
}
```

```
public static final Creator <Task> CREATOR = new Creator<Task>() {
@Override
public Task createFromParcel(Parcel in) {
return new Task(in);
}
@Override
public Task[] newArray(int size) {
return new Task[size];
}
};
}
```

TaskDao.java

```
@Dao
public interface TaskDao {
@Query("SELECT * FROM Task")
List<Task> getAll();
@Query("SELECT * FROM Task")
LiveData<List<Task>> getAllLiveData();
@Query("SELECT * FROM Task WHERE calendar IN (:setDate)")
List<Task> loadAllByCalendar(int[] setDate);
/* @Query("SELECT * FROM Task WHERE texttask LIKE :searchtext Or
" +" textsubtask LIKE :searchtext")
```

```
Task findByText(String searchtext);*/
@Query("SELECT * FROM Task WHERE done=:done")
Task findByDone(boolean done);
@Query("SELECT * FROM Task WHERE uid=:uid LIMIT 1")
Task findById(int uid);
@Insert(onConflict =  OnConflictStrategy.REPLACE)
void insertTask(Task task);
@Update
void update(Task task);
@Delete
void delete(Task task);
}
```

AppDatabase.java

```
@Database(entities = {Task.class}, version = 1, exportSchema = false)
public abstract class AppDatabase extends RoomDatabase {
public abstract TaskDao taskDao();
}
```

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
public RecyclerView recyclerView;
```

```
public Adapter adapter;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
MobileAds.initialize(this);
Toolbar toolbar = findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
FloatingActionButton fab = findViewById(R.id.fab);
OnConnected();
fab.setOnClickListener(view -> TaskDetailsActivity.start(MainActivity.this, null));
LinearLayoutManager linearLayoutManager = new LinearLayoutManager(this, RecyclerView.VERTICAL, false);
recyclerView.setLayoutManager(linearLayoutManager);
recyclerView.addItemDecoration(new DividerItemDecoration(this, DividerItemDecoration.VERTICAL));
View contentView = findViewById(R.id.include);
recyclerView = (RecyclerView) contentView.findViewById(R.id.list);
adapter = new Adapter();
recyclerView.setAdapter(adapter);
MainViewModel mainViewModel = new ViewModelProvider(this).get(MainViewModel.class);
```

```
mainViewModel.getTaskLiveData().observe(this, adapter::setItems);
}
@Override
public boolean onCreateOptionsMenu(@NonNull Menu menu) {
getMenuInflater().inflate(R.menu.menu_main, menu);
return true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
int id = item.getItemId();
if (id == R.id.action_settings) {
Toast toast = Toast.makeText(getApplicationContext(),
"Need to Update!", Toast.LENGTH_SHORT);
toast.show();
return true;
    }
return super.onOptionsItemSelected(item);
}
}
```

MainViewModel.java

```
public class MainViewModel extends ViewModel {
```

```
private final LiveData<List<Task>> taskLiveData =
App.getInstance().getTaskDao().getAllLiveData();
public LiveData<List<Task>> getTaskLiveData(){
return taskLiveData;
}
```

App.java

```
public class App extends Application {
private AppDatabase database;
private TaskDao taskDao;
private static App instance;
public static  App getInstance(){
return instance;
}
@Override
public void onCreate() {
super.onCreate();
instance = this;
database = Room.databaseBuilder(getApplicationContext(),
AppDatabase.class, "app-db-task").build();
taskDao = database.taskDao();
}
public AppDatabase getDatabase(){
```

```
return  database;
}
public void setDatabase(AppDatabase database){
this.database = database;
}
public TaskDao getTaskDao(){
return  taskDao;
}
public void setTaskDao(TaskDao taskDao){
this.taskDao = taskDao;
}
}
```

Adapter.java

```
public class Adapter extends RecyclerView.Adapter<Adapter.TaskViewHolder> {
public SortedList<Task> sortedList;
public Adapter(){
sortedList = new SortedList<>(Task.class, new SortedList.Callback<Task>() {
@Override
public int compare(Task o1, Task o2) {
if (!o2.done && o1.done){
```

```
return 1;
}
if (o2.done && !o1.done){
return -1;
}
if (o2.timectreate > o1.timectreate)
{
return 1;
}
if (o2.priority > o1.priority){
return 1;
}
return (int) (o1.calendar - o2.calendar);
}
@Override
public void onChanged(int position, int count) {
notifyItemRangeChanged(position, count);
}
@Override
public boolean areContentsTheSame(Task oldItem, Task newItem) {
return oldItem.equals(newItem);
}
```

```
@Override
public boolean areItemsTheSame(Task item1, Task item2) {
return item1.uid==item2.uid;
}
@Override
public void onInserted(int position, int count) {
notifyItemRangeInserted(position, count);
}
@Override
public void onRemoved(int position, int count) {
notifyItemRangeRemoved(position, count);
}
@Override
public void onMoved(int fromPosition, int toPosition) {
notifyItemMoved(fromPosition, toPosition);
}
});
}
@NonNull
@Override
public TaskViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
```

```
return new TaskViewHolder(LayoutInflater.from(parent.getContext()).inflate(R.layout.item_task_list,parent,false));
}
@Override
public void onBindViewHolder(@NonNull TaskViewHolder holder, int position) {
holder.bind(sortedList.get(position));
}
@Override
public int getItemCount() {
return sortedList.size();
}
public void setItems(List<Task> tasks){
sortedList.replaceAll(tasks);
}
static class TaskViewHolder extends RecyclerView.ViewHolder {
TextView itemTask;
CheckBox itemDone;
View itemDelete;
Task task;
boolean silentUpdate;
public TaskViewHolder(@NonNull View itemView)
```

```
{
super(itemView);
itemTask=itemView.findViewById(R.id.taskItem);
itemDone=itemView.findViewById(R.id.doneItem);
itemDelete=itemView.findViewById(R.id.deleteItem);
itemView.setOnClickListener(view -> TaskDetailsActivity.start((Activity)
Objects.requireNonNull(itemView).getContext(), task));
itemDone.setOnCheckedChangeListener((compoundButton, checked) -> {
if (!silentUpdate){
task.done = checked;
App.getInstance().getTaskDao().update(task);
}
updateStrokeOut();
});
itemDelete.setOnClickListener(view ->
App.getInstance().getTaskDao().delete(task));
}
public void bind( Task task){
this.task = task;
itemTask.setText(task.texttask);
updateStroke();
silentUpdate = true;
itemDone.setChecked(task.done);
```

```
        silentUpdate = false;
        }
        private void updateStroke (){
        if (!task.done){
        itemTask.setPaintFlags(itemTask.getPaintFlags() &
~Paint.STRIKE_THRU_TEXT_FLAG);}
        else{ itemTask.setPaintFlags(itemTask.getPaintFlags()|
Paint.STRIKE_THRU_TEXT_FLAG);}
        }
        }
        }
```

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт космических и информационных технологий
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В.Непомнящий
«20» 06 2022г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 «Информатика и вычислительная техника»

Мобильное приложение для планирования задач

Руководитель	подпись	24.06.22 дата	Старший преподаватель должность, ученая	И. В. Матковский
Выпускник	подпись	24.06.22 дата		Н. М. Гусейнова
Нормоконтролер	подпись	24.06.22 дата	Старший преподаватель должность, ученая	И. В. Матковский

Красноярск 2022