ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ РЕСПУБЛИКИ ХАКАСИЯ

«ХАКАССКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

ОТЧЕТ

по учебной практике
по профессиональному модулю

МДК.01.03 РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

ТЕМА: «РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ «МЕНЕДЖЕР ЗАМЕТОК»»

специальности 09.02.07 Информационные системы и программирование
Квалификация Программист

Студент гр. ИС(ПРО)-41 _____ Мальцева А.А.
подпись *Фамилия И.О.*

Руководитель практики от ГБПОУ РХ ХПК _____ _____ _____ Брюханова И.Н.
оценка *дата* *подпись* *Фамилия И.О.*

Абакан 2026 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире скорость обработки информации становится ключевым фактором продуктивности. Мобильные устройства превратились в основной инструмент для фиксации идей, составления списков задач и ведения личных записей. Несмотря на обилие существующих решений для ведения заметок, существует потребность в легковесных, автономных и интуитивно понятных приложениях, которые позволяют пользователю быстро создать, отредактировать или удалить запись без лишней визуальной нагрузки. Разработка собственного мобильного приложения «Менеджер заметок» позволяет решить эту задачу, используя современные технологии платформы Android и язык программирования Kotlin.

Цель работы: разработать мобильное приложение под операционную систему Android «Менеджер заметок», обеспечивающее полный цикл управления текстовыми записями с использованием локальной базы данных SQLite.

Для достижения поставленной цели необходимо решить следующие задачи:

- Провести анализ предметной области и существующих аналогов мобильных приложений для ведения заметок;
- Сформулировать функциональные и нефункциональные требования к разрабатываемому программному продукту;
- Спроектировать архитектуру приложения, пользовательский интерфейс и структуру базы данных;
- Реализовать клиентскую часть приложения на языке Kotlin с использованием среды разработки Android Studio;
- Организовать хранение данных с помощью СУБД SQLite и реализовать CRUD-операции;

Объект исследования: процесс разработки мобильных приложений для платформы Android.

Предмет исследования: методы и инструменты создания приложения для управления заметками с использованием языка Kotlin, фреймворка Android SDK и базы данных SQLite.

Методы исследования: анализ технической документации, объектно-ориентированное проектирование, модульное программирование, тестирование программного обеспечения.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Исследование предметной области

1.1.1 Анализ существующих аналогов

На современном рынке программного обеспечения представлен широкий спектр продуктов для реализации функции ведения заметок. Для сравнительного анализа были выбраны наиболее популярные решения:

- Obsidian — мощное приложение для работы с базой знаний, ориентированное на связь между заметками;
- Evernote — комбайн-доска для заметок с широким функционалом, включая веб-клиппер и сканирование документов;
- Google Keep — легкое приложение для быстрых записей, интегрированное в экосистему Google.

1.1.2 Обоснование необходимости разработки

Несмотря на обилие существующих решений, пользователи часто сталкиваются с проблемой избыточного функционала. Многие популярные приложения перегружены инструментами, которые не требуются для решения базовых задач, что усложняет интерфейс и снижает скорость работы. При этом, ряд аналогов требует обязательной подписки для доступа к ключевым функциям или привязки к конкретным экосистемам. В связи с этим было принято решение разработать приложение, которое фокусируется на минимализме и производительности. Основная цель проекта — предоставить пользователю инструмент, позволяющий интуитивно понятно и оперативно выполнять ключевые действия: создание, редактирование, сохранение и удаление заметок, без излишней визуальной и функциональной нагрузки.

1.2 Спецификация требований к приложению

1.2.1 Функциональные требования

Приложение должно обеспечивать следующий функционал:

- Создание заметок - возможность добавления новой заметки с заголовком и основным текстом;

- Редактирование - изменение содержания существующих заметок в реальном времени;
- Удаление - безвозвратное удаление выбранных заметок с подтверждением действия опционально или через свайп;
- Просмотр списка, отображение всех сохраненных заметок в виде адаптивного списка с сортировкой по дате создания или изменения;
- Поиск в виде фильтрации списка заметок по их заголовку или телу заметки;
- Навигация с переход между экранами списка и редактирования без перезагрузки активности.

1.2.2 Нефункциональные требования

Основные нефункциональные требования должны включать в себя:

- Производительность - время запуска приложения не должно превышать 2 секунд; интерфейс должен оставаться отзывчивым при прокрутке списков;
- Адаптивность - корректное отображение интерфейса на экранах различных размеров и ориентаций благодаря использованию ConstraintLayout и Guideline.
- Дизайн и UX:
- Единый стиль оформления с акцентом на минимализм;
- Интуитивно понятная навигация и плавные переходы между фрагментами;
- Надежность через сохранение данных при повороте экрана и сворачивании приложения.

1.3 Выбор технологий и инструментов разработки

1.3.1 Выбор языка программирования

Для реализации был выбран язык программирования kotlin - это современный статически типизированный язык программирования, разработанный компанией JetBrains. Данный язык более лаконичен и безопасен, а так же имеет полную совместимость с Java, сочетая в себе универсальность и мультиплатформенность.

1.3.2 Выбор среды разработки

Для создания приложения была выбрана популярная среда разработки от Google — Android Studio. Android Studio построена на базе IntelliJ IDEA от JetBrains и доступна на любых устройствах. Оно содержит в себе основные возможности: написание кода,

визуальный редактор, эмулятор, анализ производительности и ИИ-помощник, обеспечивающий разработчика возможностью создать мобильное приложение и довести его до момента публикации в магазине приложений.

1.3.3 Инструменты для проектирования и верстки

Для проектирования пользовательских интерфейсов и создания макетов приложения был использован графический редактор Figma. Этот инструмент позволил разработать прототип желаемого приложения, продумать логику взаимодействия элементов. Использование Figma обеспечило представлением о создаваемом приложении.

Непосредственно верстка выполнена в среде разработки Android Studio. В зависимости от выбранного подхода к разработке UI, верстка осуществлялась посредством традиционной XML-разметки с использованием визуального редактора Layout Editor, встроенного в Android Studio. Среда предоставила все необходимые средства для точного переноса макетов из Figma в работающее приложение, включая предварительный просмотр интерфейса на различных разрешениях экранов и эмуляцию состояния элементов.

2 ПРОЕКТИРОВАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

2.1 Проектирование пользовательского интерфейса

2.1.1 Разработка макетов экрана

Схематично был составлен экран редактирования и создания заметки. Он имеет заголовок, место для ввода текста заметки и кнопку «сохранить» в соответствии с рисунком 2.1.

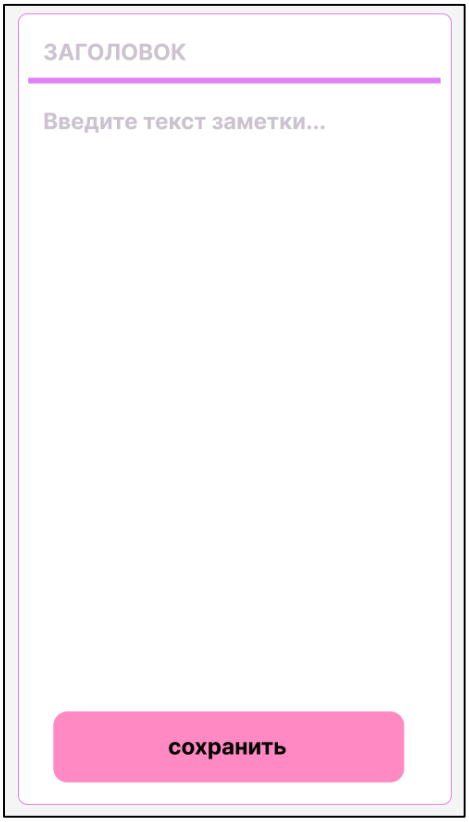

Рисунок 2.1 — Макет главного экрана в figma

Далее схематично был составлен главный экран заметок. Он имеет заголовок в виде названия «заметки», картинку с лупой, место для поиска заметок по названию или по описанию, кнопку «найти» и плавающую кнопку в нижнем правом углу отвечающую за создание новой заметки. Так же на макете был представлен схематичный список заметок в соответствии с рисунком 2.2.

Рисунок 2.2 — Макет главного экрана в figma

2.1.2 Описание навигации

Навигацию по приложению описана при помощи схемы в рисунке 2.3.

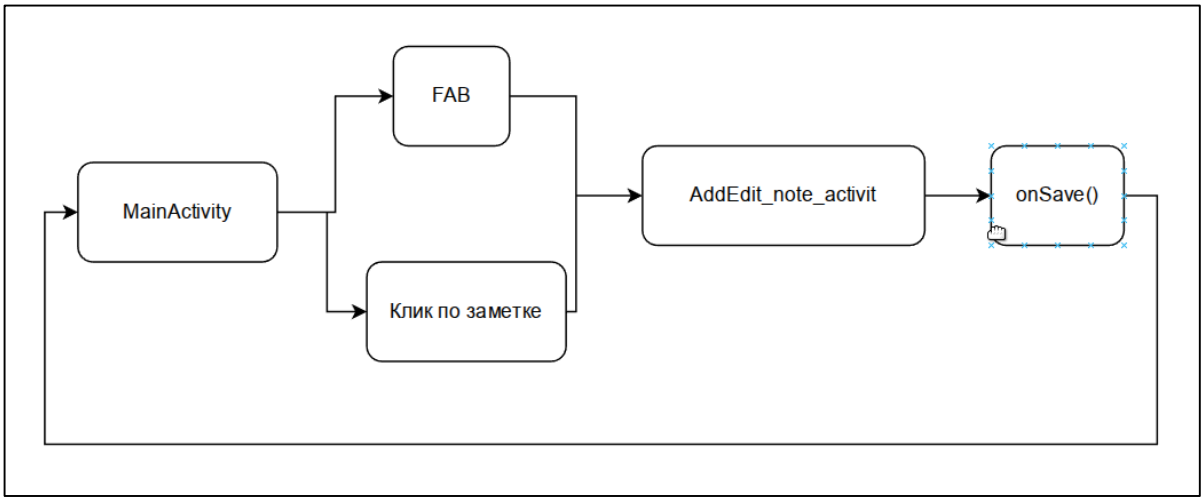

Рисунок 2.3 — Схема навигации по приложению

Посмотреть логику схемы навигации по приложению можно в соответствии с таблицей 1.

Таблица 1 — Схема навигации приложения

Действие	Метод	Описание
Открытие экрана создания	startActivityForResult(intent, REQ_ADD)	Запуск AddEdit_note_activity без передачи данных
Открытие экрана редактирования	startActivityForResult(intent, REQ_EDIT)	Запуск с передачей id, title, content, date через putExtra()
Возврат с результатом	setResult(RESULT_OK) + finish()	Уведомление главного экрана об изменении данных
Обновление списка	onActivityResult() → loadNotes()	Перезагрузка RecyclerView после возврата

2.2 Проектирование базы данных

2.2.1 ER-диаграмма

Для начала разработки была спроектирована ER-диаграмма заметок. Данная диаграмма содержит в себе одну таблицу «Notes», которая имеет в себя поля:

- Id для хранения своего уникального номера;
- Content это поле, которое хранит в себе текст заметки;
- Title является полем для хранения названия заметки;
- Date хранит в себе значение даты созданной заметки.

Посмотреть созданную ER-диаграмму можно в соответствии с рисунком 2.4.

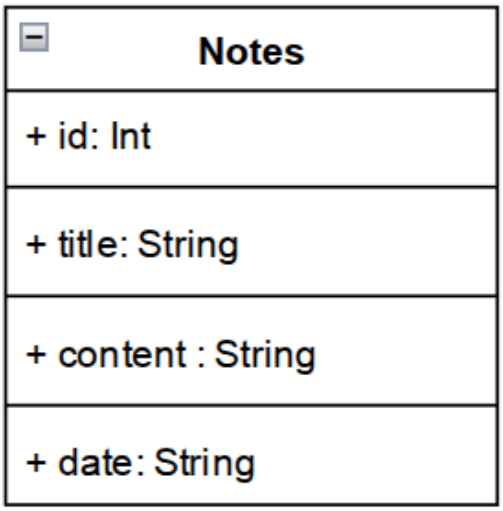

Рисунок 2.4 - ER-диаграмма

2.2.2 Физическая модель базы данных

Посмотреть физическую модель можно в соответствии с таблицей 2.

Таблица 2 — Физическая модель

Название	Описание	Тип данных	Ограничение
Id	Хранение уникального номера в виде числа	INT	От 0 до 100
Title	Хранение названия заметки в виде строки	STRING	От1 до 50 символов
Content	Хранение описания в виде строки	STRING	От 0 до 3000 символов
Date	Хранение даты создания в виде строки	STRING	Дата в формате дд.мм.гггг

2.3 Проектирование архитектуры приложения

2.3.1 Диаграмма классов

Была разработана диаграмма классов в соответствии с рисунком 2.5.

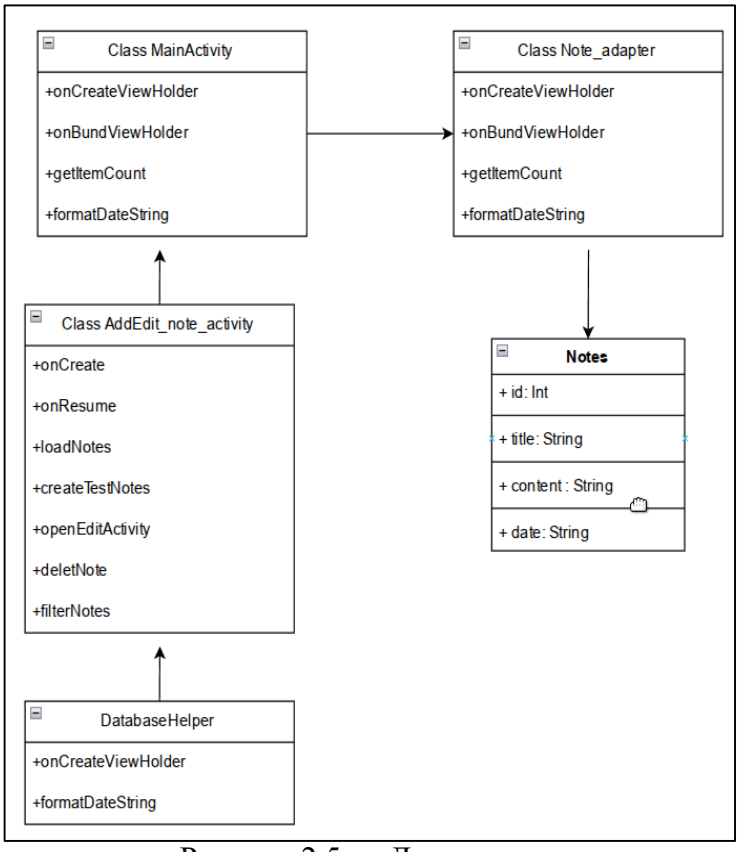

Рисунок 2.5 — Диаграмма классов

2.3.2 Описание структуры проекта

Класс MainActivity является главной активностью приложения и отвечает за отображение списка заметок на главном экране. В этом классе реализованы методы onCreate для инициализации активности и компонентов интерфейса, onResume для обновления списка заметок при возврате к экрану, loadNotes для загрузки всех заметок из базы данных и их сортировки по дате, createTestNotes для создания тестовых записей при первом запуске приложения, openEditActivity для перехода к экрану редактирования выбранной заметки с передачей необходимых данных через Intent, deleteNote для удаления заметки по идентификатору с последующим обновлением списка, а также filterNotes для фильтрации отображаемых заметок по поисковому запросу введённому пользователем в поле поиска.

Класс Note_adapter наследуется от RecyclerView.Adapter и предназначен для отображения списка заметок в виде карточек. В данном классе реализованы методы onCreateViewHolder для создания держателя представления ViewHolder для каждого элемента списка, onBindViewHolder для привязки данных конкретной заметки к элементам интерфейса карточки включая заголовок текста и дату создания, getItemCount для получения общего количества элементов в списке, updateNotes для обновления данных в адаптере при изменении содержимого базы данных, а также formatDateString для форматирования строки даты из базы данных в читаемый пользователем вид. Внутри класса определён вложенный класс NoteViewHolder который содержит ссылки на элементы интерфейса карточки заметки tvTitle tvSnippet и tvDate для оптимизации работы со списком и предотвращения частых вызовов findViewById.

Класс AddEdit_note_activity представляет собой вторую активность приложения, отвечающую за создание новых заметок и редактирование существующих. В классе реализованы методы onCreate для инициализации экрана определения режима работы создание или редактирование на основе полученных данных и заполнения полей при редактировании, а также saveNote для сохранения заметки с предварительной валидацией введённых данных заголовка и содержания после чего происходит запись в базу данных и возврат результата в MainActivity через setResult с кодом RESULT_OK. Для передачи данных между активностями используются константы EXTRA_NOTE_ID EXTRA_NOTE_TITLE EXTRA_NOTE_CONTENT и EXTRA_NOTE_DATE объявленные в companion object.

Класс Notes представляет собой data-класс модели данных который описывает структуру заметки и используется для передачи данных между компонентами приложения. Класс содержит четыре свойства: id типа Int для хранения уникального идентификатора заметки в базе данных, title типа String для хранения заголовка заметки, content типа String для хранения основного текста содержания заметки, и date типа String для хранения даты создания заметки в формате строки что обусловлено особенностями хранения временных меток в SQLite в рамках данного проекта.

Класс DatabaseHelper.OurBase наследуется от SQLiteOpenHelper и инкапсулирует всю логику работы с базой данных SQLite. В классе реализованы методы onCreate для создания таблицы Notes при первом запуске приложения с определением структуры колонок и типов данных, onUpgrade для обновления схемы базы данных при изменении версии, addNotes для добавления новой заметки в таблицу с возвратом идентификатора новой записи, getAllNotes для чтения всех записей из базы данных и преобразования курсора в список объектов Notes, updateNotes для обновления существующей заметки по идентификатору, и deleteNotes для удаления заметки из базы данных по идентификатору.

Взаимосвязи между классами построены следующим образом. MainActivity использует Note_adapter для отображения списка заметок в RecyclerView что представляет собой отношение агрегации. Note_adapter работает с моделью Notes для получения данных что является отношением ассоциации. MainActivity взаимодействует с AddEdit_note_activity через механизм Intent для навигации между экранами что представляет собой отношение зависимости. MainActivity и AddEdit_note_activity используют DatabaseHelper для выполнения операций с данными что также является отношением зависимости. Все классы работающие с данными зависят от модели Notes что обеспечивает единообразие представления информации в приложении.

3 РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ

3.1 Реализация базы данных

Посмотреть код создания таблицы базы данных можно в соответствии с листингом 1.

Листинг 1 — Notes.kt

```
//Таблица заметок
data class Notes (
    var id: Int = 0,
    var title: String,
    var content: String,
    var date: String, //База будет хранить в виде строки т.к в виде ДАТЫ нельзя вставить
)
```

3.2 Реализация пользовательского интерфейса

3.2.1 Верстка главного экрана

Посмотреть верстку главного экрана activity_main.xml можно в соответствии с листингом 2.

Листинг 2 — activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/background_activity"
    tools:context=".MainActivity">

    <!-- Горизонтальные гайдлайны -->
    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline_top"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        app:layout_constraintGuide_percent="0.01" />

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline_bottom"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        app:layout_constraintGuide_percent="0.98" />
```

```
            <!-- Вертикальные гайдлайны-->
            <androidx.constraintlayout.widget.Guideline
                android:id="@+id/guideline_start"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                app:layout_constraintGuide_percent="0.02" />

            <androidx.constraintlayout.widget.Guideline
                android:id="@+id/guideline_end"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:orientation="vertical"
                app:layout_constraintGuide_percent="0.98" />

            <!--что-бы fab держался на месте-->

    <com.google.android.material.floatingactionbutton.FloatingActionButton
                android:id="@+id/fab"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_margin="16dp"
                android:backgroundTint="@color/fab"
                android:contentDescription="@string/fab"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:srcCompat="@android:drawable/ic_input_add" />

            <!--Хранение всех элементов-->
            <LinearLayout
                android:id="@+id/head"
                android:layout_width="0dp"
                android:layout_height="0dp"
                android:orientation="vertical"
                app:layout_constraintTop_toTopOf="@id/guideline_top"
                app:layout_constraintBottom_toBottomOf="@id/guideline_bottom"
                app:layout_constraintStart_toStartOf="@id/guideline_start"
                app:layout_constraintEnd_toEndOf="@id/guideline_end">

                <TextView
                    android:id="@+id/name_head"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:gravity="center"
                    android:text="@string/name_head"
                    android:textColor="@color/pink_main"
                    android:textSize="35sp" />

                <!-- Строка с поиском -->
                <LinearLayout
                    android:id="@+id/search_liner"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:orientation="horizontal"
                    android:gravity="center_vertical">

                    <ImageView
                        android:layout_width="13dp"
                        android:layout_height="match_parent"
                        android:layout_gravity="start"
                        android:layout_weight="0.3"
```

```
                android:contentDescription="@null"
                android:src="@drawable/lp" />

            <EditText
                android:id="@+id/search_notes"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:hint="@string/search_notes"
                android:textColor="@color/pink_main"
                android:textColorHint="@color/grey_yo"
                android:inputType="text" />

            <Button
                android:id="@+id/search"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="0.6"
                android:backgroundTint="@color/pink_main"
                android:text="@string/find" />
        </LinearLayout>

        <!-- Все элементы -->
        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/recyclerView"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:clipToPadding="false"
            android:paddingBottom="0dp" />
    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Посмотреть реализованный макет в запущенном приложении в горизонтальном и вертикальном положении можно в соответствии с рисунком 3.1 и 3.2.

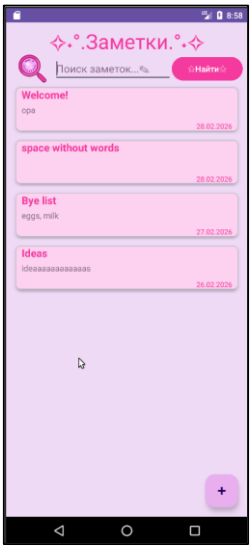

Рисунок 3.1 — activity_main.xml в вертикальном положении

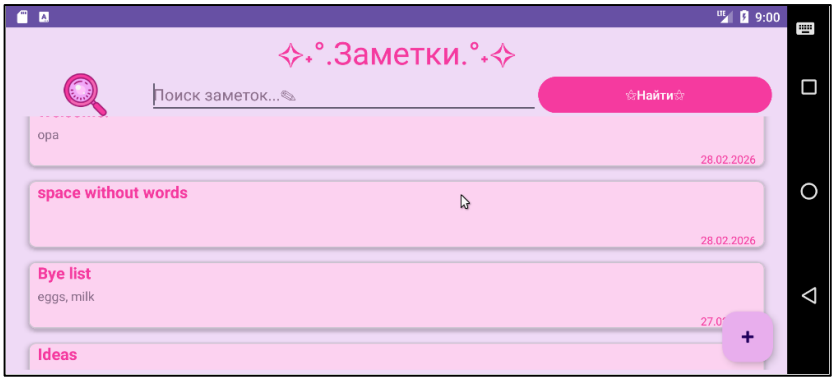

Рисунок 3.2 — activity_main.xml в горизонтальном положении

3.2.2 Верстка вспомогательных экранов

Посмотреть реализованный макет в запущенном приложении в горизонтальном положении можно в соответствии с рисунком 3.3.

Рисунок 3.3 — item_note.xml в горизонтальном положении

Посмотреть верстку экрана с элементами заметок add_edit_note.xml можно в соответствии с листингом 4.

Листинг 4 — add_edit_note.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:background="@color/background_activity"
    android:orientation="vertical"
    android:padding="10dp">

    <!-- Горизонтальные гайдлайны -->
    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline_top"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        app:layout_constraintGuide_percent="0.01" />
```

```
<androidx.constraintlayout.widget.Guideline
    android:id="@+id/guideline_bottom"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    app:layout_constraintGuide_percent="0.98" />

<!-- Вертикальные гайдлайны-->
<androidx.constraintlayout.widget.Guideline
    android:id="@+id/guideline_start"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintGuide_percent="0.02" />

<androidx.constraintlayout.widget.Guideline
    android:id="@+id/guideline_end"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintGuide_percent="0.98" />

<!--Хранение всех элементов-->
<LinearLayout
    android:id="@+id/head"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    app:layout_constraintTop_toTopOf="@id/guideline_top"
    app:layout_constraintBottom_toBottomOf="@id/guideline_bottom"
    app:layout_constraintStart_toStartOf="@id/guideline_start"
    app:layout_constraintEnd_toEndOf="@id/guideline_end">

    <EditText
        android:id="@+id/editTitle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:alpha="0.7"
        android:gravity="start"
        android:hint="@string/editTitle"
        android:textColor="@color/pink_main"
        android:textColorHint="@color/grey_yo"
        android:textSize="20sp"
        android:textStyle="bold" />

    <EditText
        android:id="@+id/etContent"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="start"
        android:hint="@string/etContent"
        android:textColorHint="@color/grey_yo"
        android:textColor="@color/pink_main"
        android:alpha="0.7"
        android:inputType="textMultiLine" />

    <Button
        android:id="@+id/btnSave"
        android:textSize="16dp"
        android:gravity="center"
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:backgroundTint="@color/pink_main"
            android:text="@string/btnSave" />
    </LinearLayout>
</LinearLayout>
```

Посмотреть реализованный макет в запущенном приложении в горизонтальном и вертикальном положении можно в соответствии с рисунком 3.5 и 3.6.

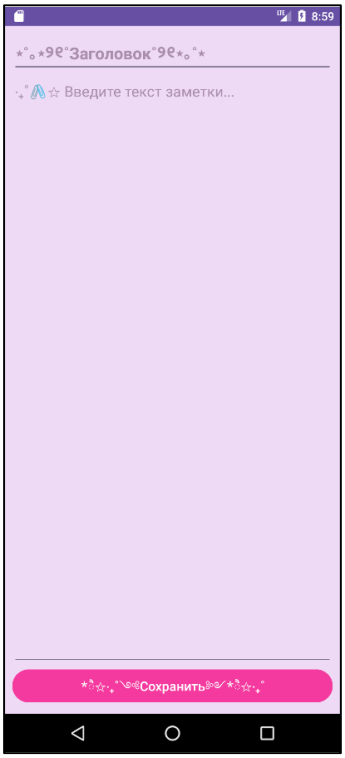

Рисунок 3.5 — add_edit_note.xml в горизонтальном положении

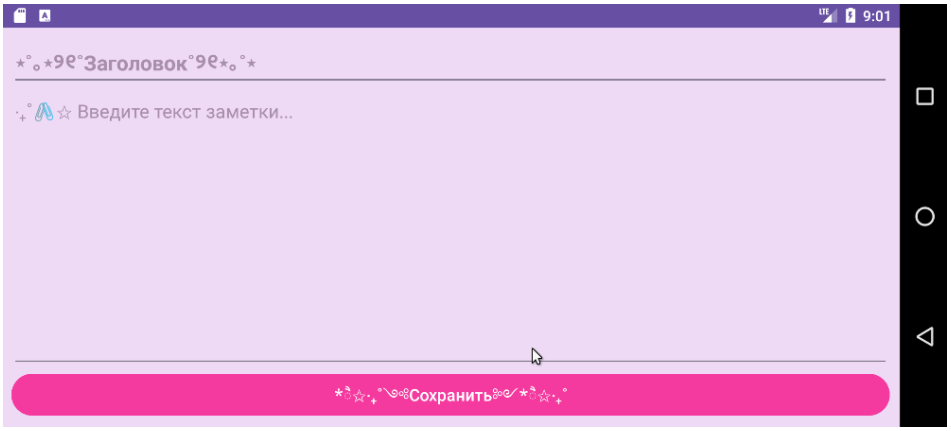

Рисунок 3.6 — add_edit_note.xml в горизонтальном положении

3.3 Реализация логики работы приложения

3.3.1 Адаптеры и работа со списками

Для отображения списка заметок в приложении был реализован кастомный адаптер Note_adapter, наследующийся от RecyclerView.Adapter. Данный подход обеспечивает эффективную работу с большими списками данных благодаря механизму пересоздания только видимых элементов интерфейса.

Адаптер принимает три параметра: список заметок notesList, обработчик короткого нажатия onItemClick для редактирования и обработчик долгого нажатия onItemLongClick для удаления. В методе onCreateViewHolder происходит создание нового элемента списка путём разметки item_note.xml в соответствии с листингом 5.

Листинг 5 — onCreateViewHolder

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): NoteViewHolder {
    val view = LayoutInflater.from(parent.context)
        .inflate(R.layout.item_note, parent, false)
    return NoteViewHolder(view)
}
```

Метод onBindViewHolder отвечает за привязку данных конкретной заметки к элементам интерфейса карточки. Здесь устанавливаются заголовок, сниппет текста и отформатированная дата создания в соответствии с листингом 6.

Листинг 6 — onBindViewHolder

```
override fun onBindViewHolder(holder: NoteViewHolder, position: Int) {
    val note = notesList[position]
    holder.tvTitle.text = note.title
    holder.tvSnippet.text = if (note.content.length > 50) {
        note.content.substring(0, 50) + "..."
    } else {
        note.content
    }
    holder.tvDate.text = formatDateString(note.date)

    holder.itemView.setOnClickListener { onItemClick(note) }
    holder.itemView.setOnLongClickListener {
        AlertDialog.Builder(holder.itemView.context)
            .setTitle("Удалить заметочку? (｡•́‿•̀｡)")
            .setMessage("\"${note.title}\" Она будет удалена безвозвратно!")
            .setPositiveButton("Да, удаляй") { _, _ -> onItemLongClick(note) }
            .setNegativeButton("Емае, пусть остается", null)
            .show()
        true
    }
}
```

Для обновления данных в адаптере реализован метод updateNotes, который принимает новый список и вызывает notifyDataSetChanged() для перерисовки RecyclerView в соответствии с листингом 7.

Листинг 7 — updateNote.

```
fun updateNotes(newNotes: List<Notes>) {
    notesList = newNotes
    notifyDataSetChanged()
}
```

3.3.2 Обработка событий

Обработка пользовательских событий распределена между тремя основными компонентами приложения. В MainActivity реализованы обработчики нажатий на плавающую кнопку FAB и кнопку поиска в соответствии с листингом 8.

Листинг 8 — Плавающая кнопка и поиск

```
// Плавающая кнопка на создание новой заметки
fab.setOnClickListener {
    val intent = Intent(this, AddEdit_note_activity::class.java)
    startActivityForResult(intent, REQUEST_ADD_NOTE)
}
// Кнопка поиска
btnSearch.setOnClickListener {
    val query = searchNotes.text.toString().trim()
    filterNotes(query)
}
```

Метод filterNotes осуществляет поиск по заголовку и содержанию заметки с учётом регистра символов в соответствии с листингом 9.

Листинг 9 — filterNotes

```
private fun filterNotes(query: String) {
    if (query.isEmpty()) {
        noteAdapter.updateNotes(allNotes)
    } else {
        val filtered = allNotes.filter {
            it.title.contains(query, ignoreCase = true) ||
            it.content.contains(query, ignoreCase = true)
        }
        noteAdapter.updateNotes(filtered)
    }
}
```

В AddEdit_note_activity реализована валидация введённых данных перед сохранением. При пустом заголовке пользователю отображается сообщение об ошибке.

Реализация ошибки внутри основного текста заметки не реализован так как предусматривается, что может быть создана пустая заметка. Посмотреть листинг можно в соответствии с листингом 10.

Листинг 10— saveNote

```
    private fun saveNote() {
        val title = editTitle.text.toString().trim()
        val content = etContent.text.toString().trim()

        if (title.isEmpty()) {
            editTitle.error = "Введите заголовочек ( ･ω･ )"
            return
        }

        val currentDate = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault()).format(Date())

        if (isEditMode) {
            dbHelper.updateNotes(noteId, title)
            Toast.makeText(this,        "Заметочка        была        обновлена        (*°▽°*)",
Toast.LENGTH_SHORT).show()
        } else {
            dbHelper.addNotes(title, content, currentDate)
    Toast.makeText(this, "Заметоча была создана (☆◡☆)", Toast.LENGTH_SHORT).show()
        }

        setResult(Activity.RESULT_OK)
        finish()
    }
```

Обновление списка заметок происходит автоматически при возврате из экрана редактирования благодаря переопределению метода onActivityResult в MainActivity в соответствии с листингом 11.

Листинг 11 — Автоматический возврат

```
    @Deprecated("Deprecated in Java")
    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (resultCode == RESULT_OK) {
            loadNotes()
        }
    }
```

3.3.3 Навигация между экранами

Навигация в приложении реализована с использованием явных Intent для перехода между активностями. Основной экран MainActivity запускает AddEdit_note_activity для создания или редактирования заметки. А так же для создания новой заметки используется

Intent без дополнительных данных. Посмотреть код реализации можно в соответствии с листингом 12.

Листинг 12 - fab.setOnClickListener

```
fab.setOnClickListener {
    val intent = Intent(this, AddEdit_note_activity::class.java)
    startActivityForResult(intent, REQUEST_ADD_NOTE)
}
```

Для редактирования существующей заметки в Intent передаются все необходимые данные через putExtra в соответствии с листингом 13.

Листинг 13 - openEditActivity

```
private fun openEditActivity(note: Notes) {
    val intent = Intent(this, AddEdit_note_activity::class.java).apply {
        putExtra(AddEdit_note_activity.EXTRA_NOTE_ID, note.id)
        putExtra(AddEdit_note_activity.EXTRA_NOTE_TITLE, note.title)
        putExtra(AddEdit_note_activity.EXTRA_NOTE_CONTENT, note.content)
        putExtra(AddEdit_note_activity.EXTRA_NOTE_DATE, note.date)
    }
    startActivityForResult(intent, REQUEST_EDIT_NOTE)
}
```

В AddEdit_note_activity режим работы определяется наличием EXTRA_NOTE_ID в полученном Intent в соответствии с листингом 14.

Листинг 14 - intent

```
isEditMode = intent.hasExtra(EXTRA_NOTE_ID)

if (isEditMode) {
    noteId = intent.getIntExtra(EXTRA_NOTE_ID, 0)
    editTitle.setText(intent.getStringExtra(EXTRA_NOTE_TITLE))
    etContent.setText(intent.getStringExtra(EXTRA_NOTE_CONTENT))
}
```

3.4 Реализация дополнительного функционала

Все операции с данными инкапсулированы в классе DatabaseHelper.OurBase, наследующемся от SQLiteOpenHelper. Класс предоставляет методы для CRUD-операций. Посмотреть реализацию в виде кода можно в соответствии с листингом 15.

Листинг 15 - CRUD-операции

```
// Добавление новой заметки
fun addNotes(title: String, content: String, date: String): Long {
    val db = this.writableDatabase
    val values = ContentValues().apply {
        put(TITLE_NAME, title)
        put(CONTENT_NAME, content)
        put(DATE_NAME, date)
    }
    return db.insert(TABLE_NAME, null, values)
}

// Получение всех заметок
fun getAllNotes(): List<Notes> {
    val notesList = mutableListOf<Notes>()
    val cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
    if (cursor.moveToFirst()) {
```

```
            do {
                notesList.add(Notes(
        id = cursor.getInt(cursor.getColumnIndexOrThrow(ID_NAME)),
        title = cursor.getString(cursor.getColumnIndexOrThrow(TITLE_NAME)),
        content = cursor.getString(cursor.getColumnIndexOrThrow(CONTENT_NAME)),
        date = cursor.getString(cursor.getColumnIndexOrThrow(DATE_NAME))
                ))
            } while (cursor.moveToNext())
        }
        cursor.close()
        return notesList
    }
```

Для отображения даты в читаемом формате реализован вспомогательный метод formatDateString в адаптере в соответствии с листингом 16.

Листинг 16 — formatDateString

```
    private fun formatDateString(dateString: String): String {
        return try {
            val input = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault())
            val output = SimpleDateFormat("dd.MM.yyyy", Locale.getDefault())
            output.format(input.parse(dateString) ?: Date())
        } catch (e: Exception) {
            dateString
        }
    }
```

Заметки отображаются в обратном хронологическом порядке где новые сверху, а старые снизу, благодаря сортировке в методе loadNotes в соответствии с листингом 17.

Листинг 17 - loadNotes

```
    private fun loadNotes() {
        allNotes = dbHelper.getAllNotes()
        if (allNotes.isEmpty()) createTestNotes()
        allNotes = allNotes.sortedByDescending { it.date }
        noteAdapter.updateNotes(allNotes)
    }
```

При первом запуске приложения, если база данных пуста, автоматически создаются три тестовые заметки через метод createTestNotes в соответствии с листингом 18.

Листинг 18 - createTestNotes

```
    private fun createTestNotes() {
        val dateFormat = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault())
        val now = Date()

        dbHelper.addNotes("Welcome!", "opa", dateFormat.format(now))
        dbHelper.addNotes("Bye list", "eggs, milk", dateFormat.format(Date(now.time -
86400000)))
        dbHelper.addNotes("Ideas", "ideaaaaaaaaaaas", dateFormat.format(Date(now.time -
172800000)))
    }
```

ЗАКЛЮЧЕНИЕ

В ходе выполнения учебной практики по профессиональному модулю МДК.01.03 разработка мобильных приложений было разработано мобильное приложение «Менеджер заметок» для платформы Android. В результате проделанной работы были достигнуты следующие результаты:

- Проведен аналитический этап;
- Выполнено проектирование;
- Реализован программный продукт;
- Реализована логика работы списка заметок через кастомный адаптер;
- Настроено локальное хранилище данных на основе SQLite;
- Реализованы все необходимые CRUD-операции;
- Внедрен функционал поиска и фильтрации заметок по заголовку и содержимому;
- Организована навигация между экранами главного списка и экрана редактирования;

В процессе разработки были закреплены навыки работы со средой Android Studio, языком программирования Kotlin, а также углублены знания в области жизненного цикла Activity, работы с базами данных и построения адаптивных интерфейсов.

Перспективы дальнейшего развития могут быть как добавление возможности выбора цветовой темы с учетом предпочтений пользователя, реализацию функции экспорта и импорта заметок в разных форматах.

Разработанное приложение полностью соответствует поставленным требованиям и готово к использованию в качестве базового инструмента для ведения личных записей.

ПРИЛОЖЕНИЯ

Программный код в системе контроля версий

Ссылка на Git - https://github.com/SnezhiK000/Android_praktica