**Практическая работа №3**

**«Знакомство с БД Android Studio, работа с библиотекой Room»**

**I ТЕОРЕТИЧЕСКИЙ МИНИМУМ**

Базы данных играют важную роль в разработке любого ПО, поскольку они обеспечивают структурированное и удобное хранение, а также быстрый доступ к данным. В мобильных приложениях БД обычно используются для решения следующих задач:

* **Хранение пользовательских данных**: сохранение настроек, личных данных, историй покупок и прочей информации, что делает приложение более персонализированным и удобным.
* **Офлайн доступ**: базы данных позволяют приложениям функционировать даже без постоянного доступа к интернету, сохраняя критически важную информацию локально.
* **Повышение производительности**: доступ к локальным данным значительно быстрее в сравнении с запросами к серверу, что критично для пользовательского опыта.
* **Долгосрочное хранение**: данные, сохраненные в мобильных устройствах, не удаляются при закрытии приложения, что позволяет использовать данные в долгосрочной перспективе.



1. **С какими локальными базами можно работать?**

При разработке приложений под Android существует три наиболее популярных СУБД, каждая из которых имеет свои особенности:

1. **SQLite:**

* **Встроенная поддержка в Android**: SQLite является реляционной БД, которая встроена в Android. Это делает её идеальным выбором для разработчиков приложений, которым нужна простота и эффективность.
* **Прямое взаимодействие с SQL**: разработчики могут напрямую использовать SQL-запросы для управления данными.

1. **Room:**

* **Часть Android Jetpack**: Room предоставляет абстрактный слой поверх SQLite для более удобной работы с базой данных.
* **Компиляционная проверка запросов**: Room позволяет избегать распространенных ошибок во время компиляции через проверку SQL-запросов.

1. **Realm:**

* **Нереляционная база данных**: Realm – это база данных, ориентированная на объекты, что делает её отличным выбором для приложений, которые работают с частично структурированными или неструктурированными вовсе данными, сохраняя при этом высокую скорость работы с большим объемом данных.
* **Реактивное программирование**:Realm поддерживает реактивные обновления данных, что упрощает создание интерактивных приложений.

Подробнее про реактивное программирование можно почитать тут: https://habr.com/ru/companies/oleg-bunin/articles/545702/

1. **Различия, особенности подхода.**

Давайте более подробно поговорим об указанных выше СУБД, чтобы иметь возможность сделать более осознанный выбор при разработке ПО.

**SQLite:**

SQLite – это встраиваемая реляционная база данных, которая не требует отдельного серверного процесса для работы. Она является частью стандартного SDK Android и подходит для большинства простых приложений. Главные особенности данной СУБД заключаются в простоте её использования:

1. **Прямое взаимодействие с SQL**:разработчики напрямую пишут SQL-запросы для взаимодействия с базой данных, что обеспечивает полный контроль над БД и её структурой.
2. **Легковесность и производительность**: отсутствие необходимости в дополнительных библиотеках делает SQLite очень легковесным выбором.
3. **Низкоуровневый доступ:** непосредственный доступ к базе данных позволяет детально настроить работу с данными, однако требует хорошего понимания SQL и специфики работы с базами данных.

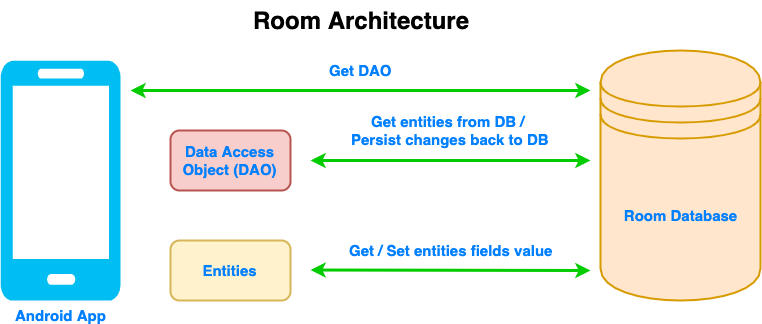


Таким образом, SQLite подходит в том случае, если вам нужен полный контроль и максимальная гибкость без необходимости использования дополнительных библиотек. Но для его использования нужен хороший уровень владения SQL.

**Room:**

Room предоставляет абстракцию поверх SQLite и входит в состав Android Architecture Components. Являясь встроенной библиотекой языка Kotlin, Room позволяет отказаться от «чистого» SQL, заменив его DAO интерфейсом (о нём поговорим позже). Можно выделить следующие преимущества работы с Room:

1. **Уменьшение boilerplate (шаблонного) кода:** Room значительно упрощает работу с базой данных, предоставляя удобные аннотации и проверяемые на этапе компиляции SQL-запросы. Таким образом, SQL-запросы превращаются в подобие методов/функций, которые удобно вызывать из разных частей кода.
2. **Поддержка LiveData и RxJava:** Room легко интегрируется с библиотеками для асинхронной работы и реактивного программирования, что позволяет легко реализовывать реактивные пользовательские интерфейсы.
3. **Type-safe запросы:** запросы в Room строятся так, что ошибки в SQL могут быть обнаружены еще на этапе компиляции, что предотвращает многие распространенные ошибки при работе с базами данных.



Исходя из этого, Room даёт большее удобство работы, а также интеграции с современными Android-подходами. При этом одним из главных преимуществ является возможность сохранить «чистый» код, избавившись от дублирования SQL-запросов в разных местах программы.

Подробней про room можно почитать здесь: <https://habr.com/ru/articles/713518/>

**Realm:**

Realm – это объектно-ориентированная база данных, предназначенная для мобильных приложений с акцентом на высокую производительность и простоту использования. Особенности Realm:

1. **Объектно-ориентированное хранилище:** в Realm данные представлены в виде объектов, что делает модель данных более наглядной и удобной для разработчиков, привыкших к ООП.
2. **Автоматическое обновление представлений:** Realm может автоматически обновлять UI при изменении данных в базе, что упрощает создание динамичных интерфейсов.
3. **Мультиплатформенность:** Realm поддерживает не только Android, но и другие платформы, что делает его хорошим выбором для кросс-платформенных приложений.
4. **Отсутствие языка запросов SQL:** Realm использует свои методы запросов, что может быть как плюсом, так и минусом в зависимости от вашего опыта и предпочтений.



Таким образом, Realm следует выбирать при создании мультиплатформенных приложений с акцентом на производительность и простоту работы с данными.

В нашем курсе мы будем использовать **Room**, так как она является идеальным выбором для создания простых приложений в учебных целях. Она позволяет работать с базами данных, используя уже знакомый нам язык SQL, но при этом не вынуждает использовать его в большом количестве (всё же у нас не курс БД).

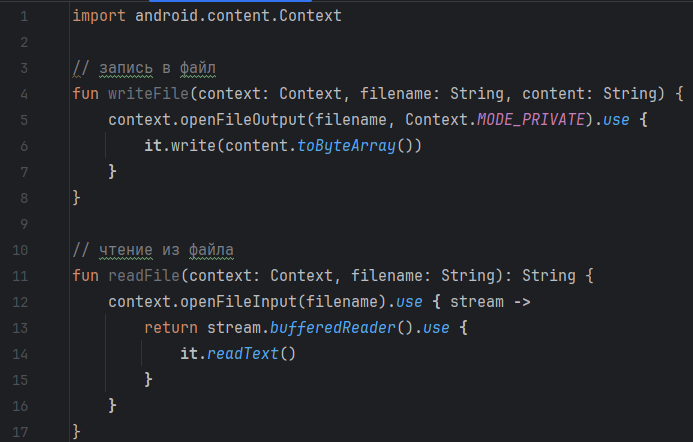
1. **Можно ли хранить не в базе, а в файлах? Работа с файлами вообще в AS есть?**

В Kotlin, как и в большинстве других языков программирования имеется возможность работы не только с БД, но и непосредственно с файловой системой устройства. Данный подход также имеет свои преимущества:

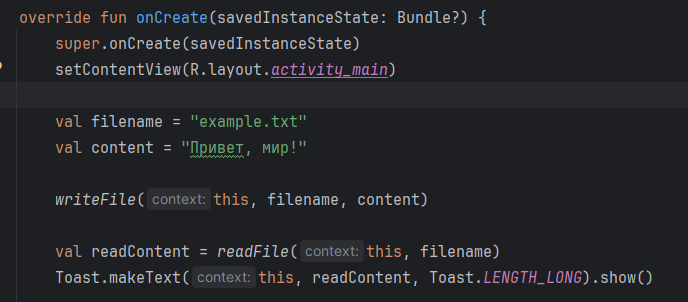
* **Простота:** для некоторых типов данных, таких как конфигурационные файлы или большие неструктурированные текстовые документы, использование файлов может быть более простым решением по сравнению с организацией базы данных.
* **Портабельность:** файлы легче переносить между устройствами или системами. Это может быть полезно для функций экспорта/импорта данных приложения.
* **Нет зависимости от СУБД:** работа с файлами не требует подключения и настройки СУБД, что может ускорить разработку и упростить поддержку приложения.

Android Studio позволяет работать с такими типами файлов как текстовые файлы, бинарные файлы, XML и JSON.

Вот пример работы с файлами на языке программирования Kotlin в Android Studio:



Эти функции можно вызвать из любого места приложения (например, из Activity или Fragment):

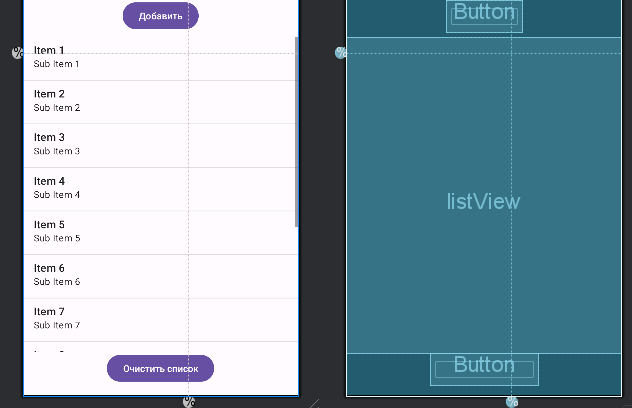


1. **Работа с новыми элементами**

В данной практической работе будут использованы два новых элемента: listView для отображения списка и checkbox для отмечания выполненных дел.

**ListView:**

**ListView —** это компонент интерфейса в Android, который отображает элементы в виде вертикального списка. Это один из базовых видов для представления списковых данных, который подходит для простых сценариев, где не требуется сложный дизайн или высокая производительность при обработке больших объемов данных.



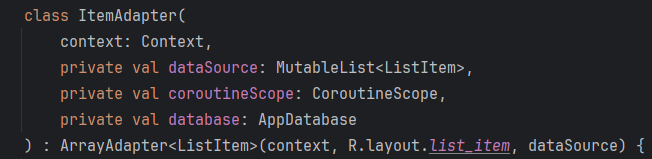
Более подробно прочитать про ListView можно здесь:

<https://developer.android.com/reference/kotlin/android/widget/ListView>

<https://java-blog.ru/kotlin/kak-sozdat-listview-v-kotlin-android>

Основные особенности ListView:

* **Адаптер:** ListView работает в паре с адаптером, который отвечает за превращение массива элементов данных в элементы интерфейса, которые затем отображаются в ListView. Адаптер создает представление для каждого элемента данных при его показе на экране.



* **Взаимодействие:** пользователи могут взаимодействовать с элементами списка, например, выбирать их с помощью тапа. Обработка кликов по элементам обычно осуществляется через прослушиватели событий.



**CheckBox:**

**CheckBox в Android —** это пользовательский элемент управления, который позволяет пользователю сделать выбор, включающий один или несколько пунктов. Очень часто используется для задач, где необходимо указать выполнение или выбор чего-либо, например, в списке задач или настроек приложения.



Более подробно прочитать про CheckBox можно здесь:

<https://developer.android.com/reference/kotlin/android/widget/CheckBox>

<https://developer.alexanderklimov.ru/android/views/checkbox.php>

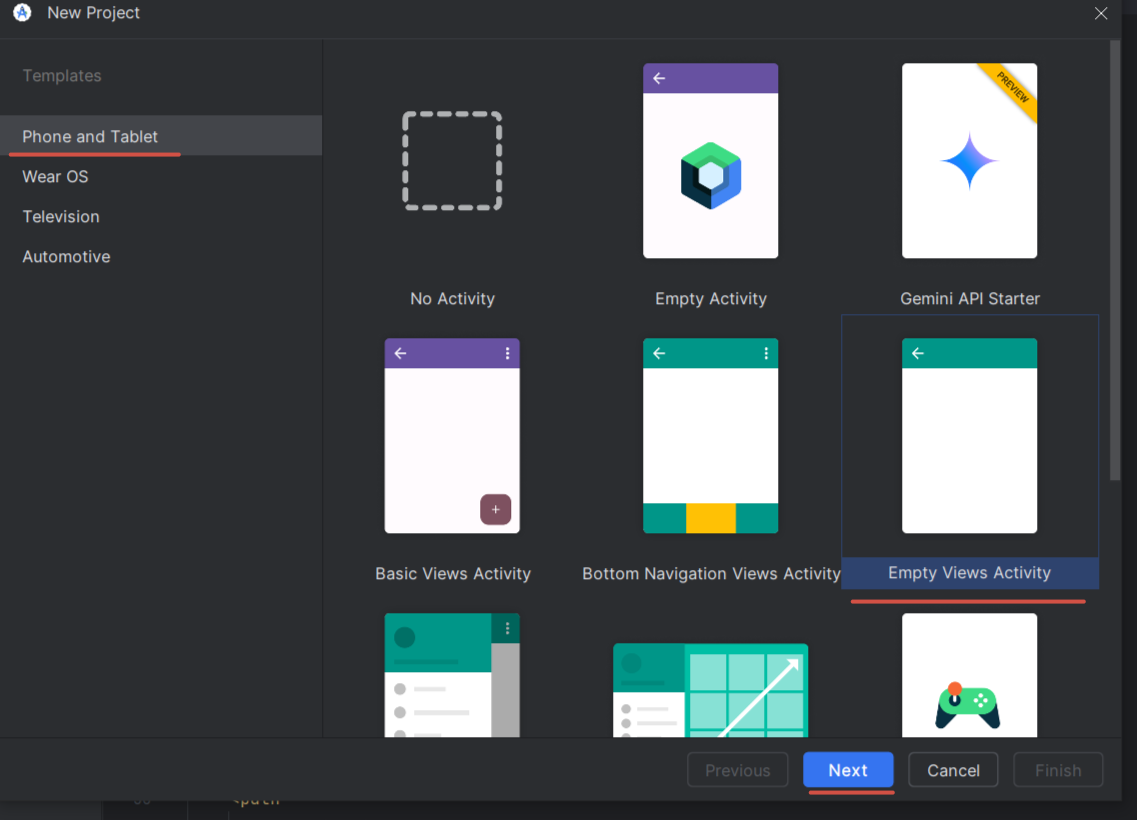
Основные особенности ListView:

* **Двухсоставное состояние:** как правило, CheckBox может находиться в одном из двух состояний: отмеченном (checked) или неотмеченном (unchecked). Некоторые чекбоксы могут включать третье состояние indeterminate (неопределенное), когда значение не может быть четко определено, но это редко используется.
* **Пользовательский интерфейс:** CheckBox обычно представлен в виде квадратной кнопки, к которой слева или справа прикреплен текстовый меток, описывающий выбор.
* **Программное управление:** CheckBox поддерживает программное управление его состоянием, что позволяет приложению автоматически устанавливать или снимать отметку, а также реагировать на изменения состояния через слушатели событий.

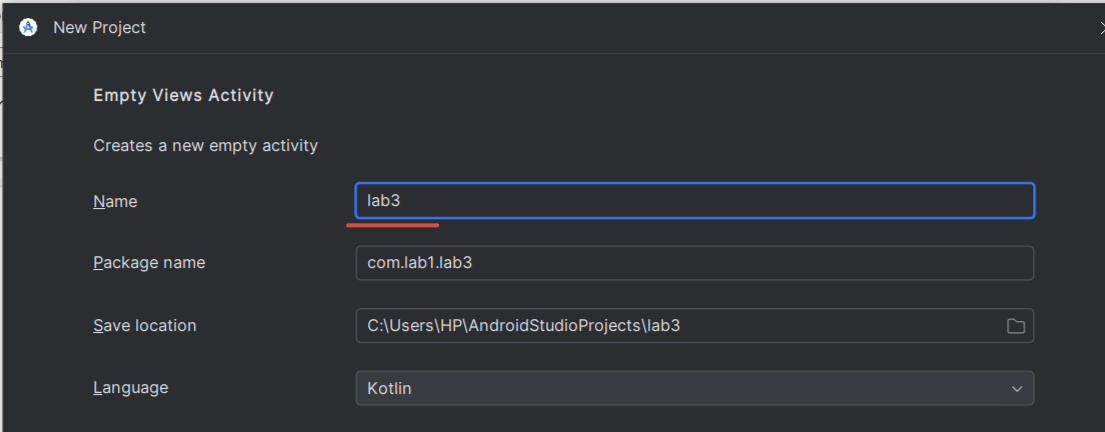
**II ХОД РАБОТЫ**

Android Studio мы установили в рамках предыдущей работы. Также были изучены кнопки и поле ввода/вывода, spinner. В этой работе мы разработаем более сложное приложение с использованием всех предыдущих навыков плюс с новыми элементами и локальной БД.

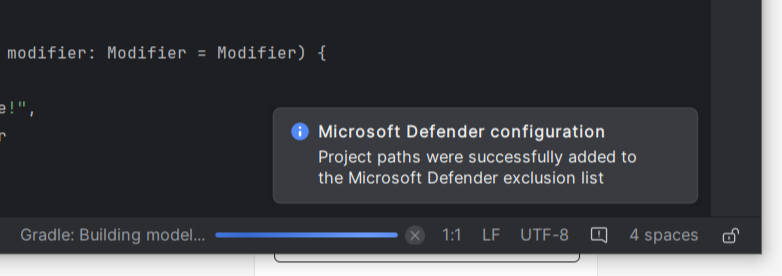
1. **Создание нового проекта**
   1. Создаём проект с Empty Views Activity.



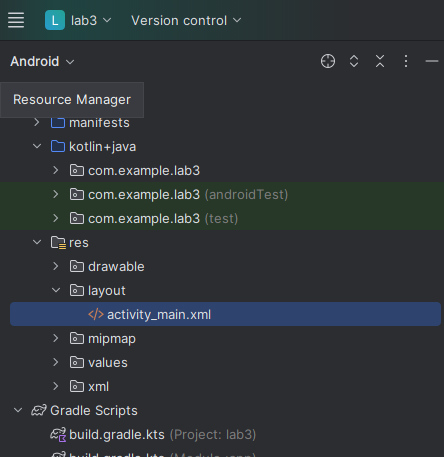
Назовём его lab3.



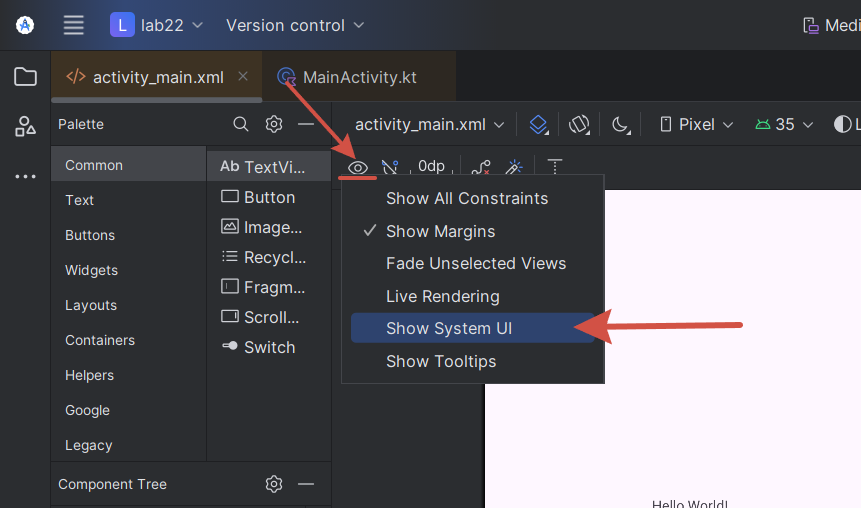
* 1. Также как и в прошлый раз, после создания нового проекта Android Studio нужно дать пару минут. Она подгружает различные библиотеки, добавляет их внутрь проекта, синхронизирует/докачивает зависимости. Также может потребоваться дать разрешение на эти действия встроенному файрволлу, например, Windows Defender’у. Нужно это сделать и после дождаться загрузки проекта. Пока программа это делает, вы не можете писать никакого кода и использовать функции приложения.



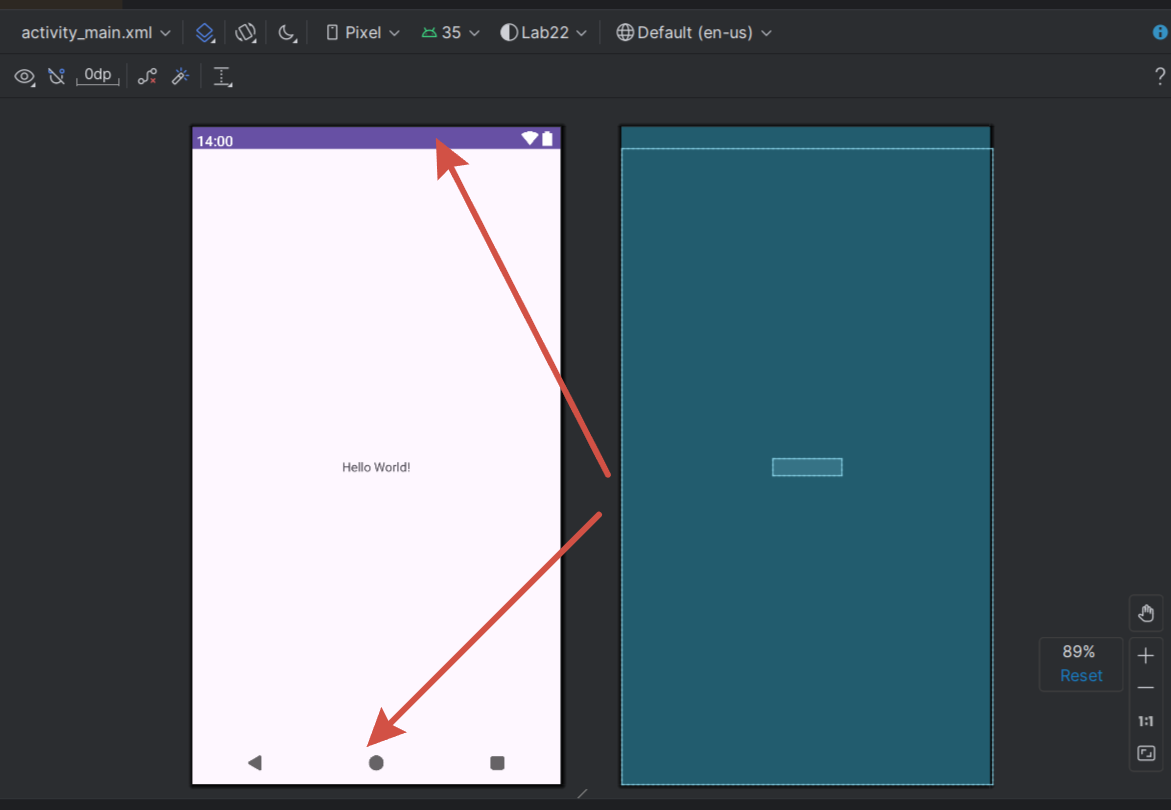
1. **Создание дизайна приложения или компоновка Activity**
   1. Не будем отходить от традиции. Если в прошлый раз был «Калькулятор», то в этой работе мы создадим приложение – «список дел». После того, как проект запустится находим и кликаем по главной и пока единственной Activity в приложении.



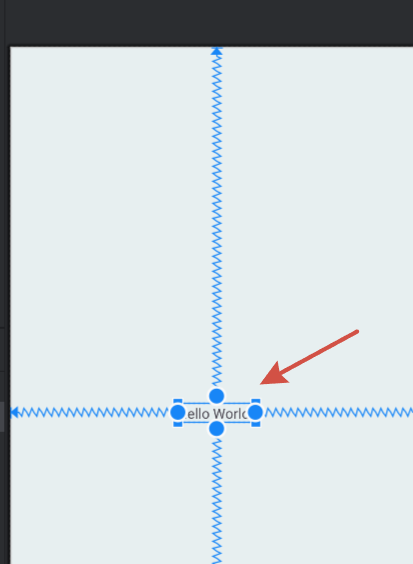
* 1. Включим отображение системных элементов в пользовательском интерфейсе, чтобы наша страница выглядела на экране дизайна также как на реальном устройстве.



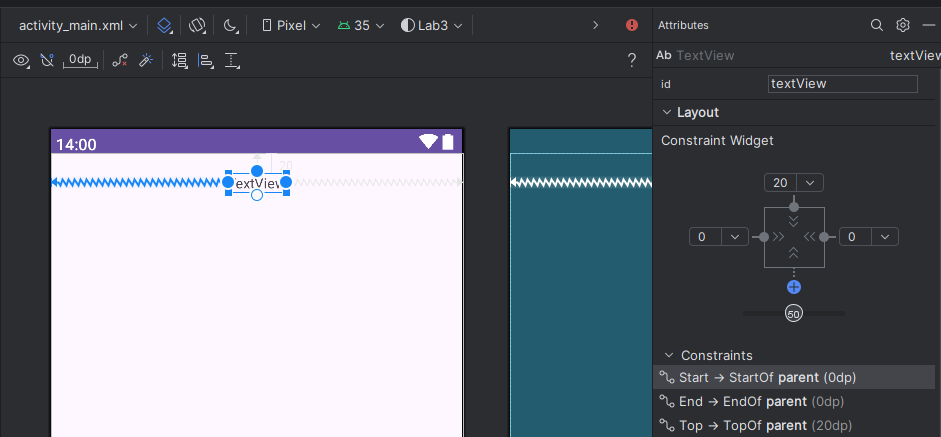
Появятся 2 панели, сверху и снизу:



* 1. Давайте удалим пока единственную имеющуюся у нас надпись в проекте. Выбираем её и нажимаем Del

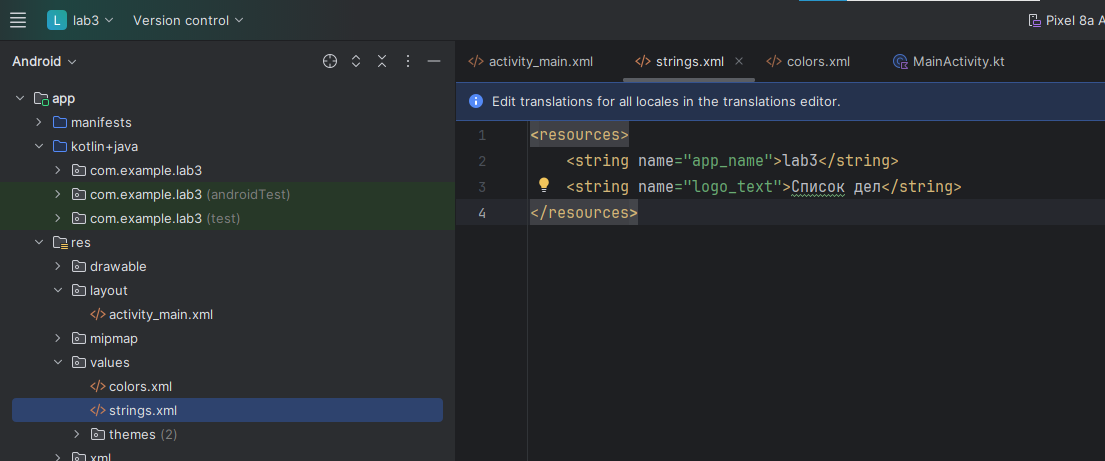


* 1. Добавляем на слой текстовую надпись (TextView). Связываем наш элемент с границами экрана, используя указанные значения:

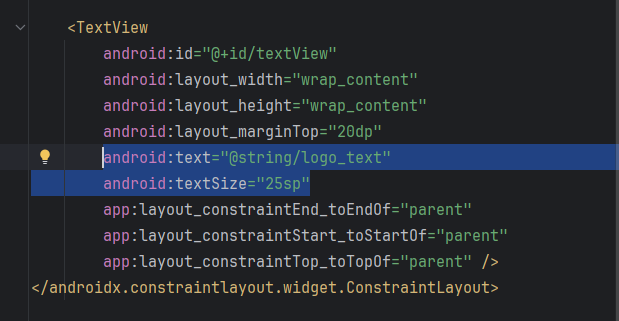


Сверху смещение должно быть 20dp, + по горизонтали выравнивание должно быть по центру.

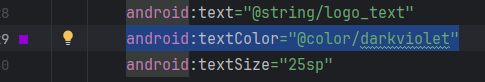
* 1. Далее поменяем текст этой надписи. Напоминаю, что это нужно сделать с помощью strings.xml. Добавляем туда надпись logo\_text.



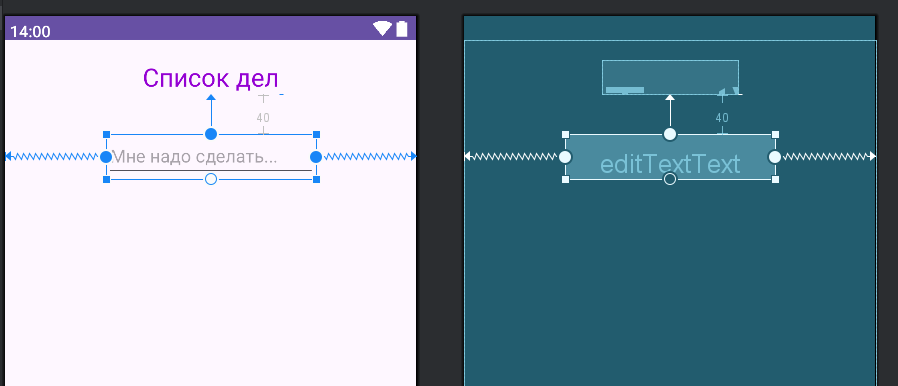
* 1. Далее вернемся к Activity. Изменим текст элемента и размер этого текста, как делали это в предыдущей работе.



* 1. Давайте теперь также поменяем цвет шрифта. По аналогии используем свойство textColor. Пусть он будет темно-фиолетовым. HEX #9400D3 (не забудьте добавить цвет в colors.xml)



* 1. Давайте добавим поле PlainText. По аналогии с предыдущей работой свяжем его с элементами интерфейса, а также добавим подсказку (hint). Должно получиться нечто подобное:

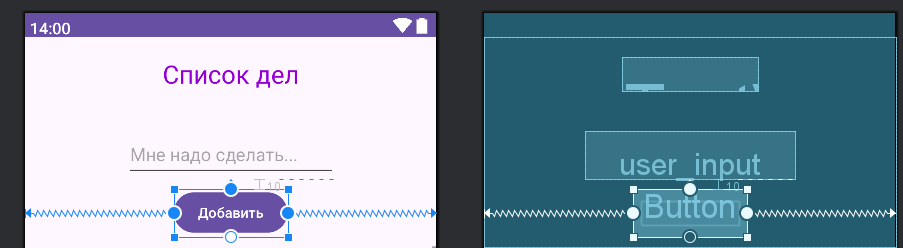


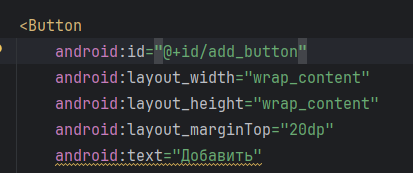
Далее необходимо убрать ошибку, связанную с размером поля ввода. Исправляем её при помощи встроенных средств AS, как делали это в прошлой работе (либо вручную увеличиваем высоту до 48dp).

Также сразу изменим id данного элемента на user\_input



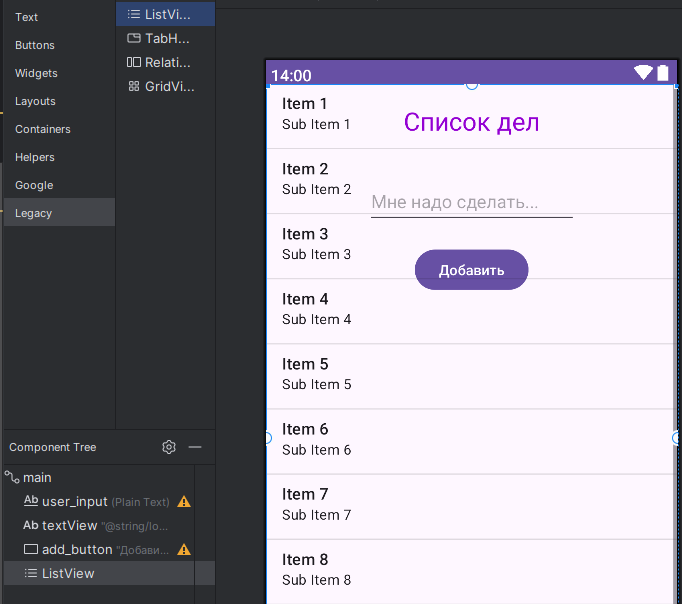
* 1. Далее добавим кнопку, связав её с полем ввода и границами экрана, а также указав id add\_button и текст «Добавить».



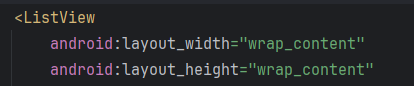


Цвет самой кнопки и текста в ней можно изменить по желанию.

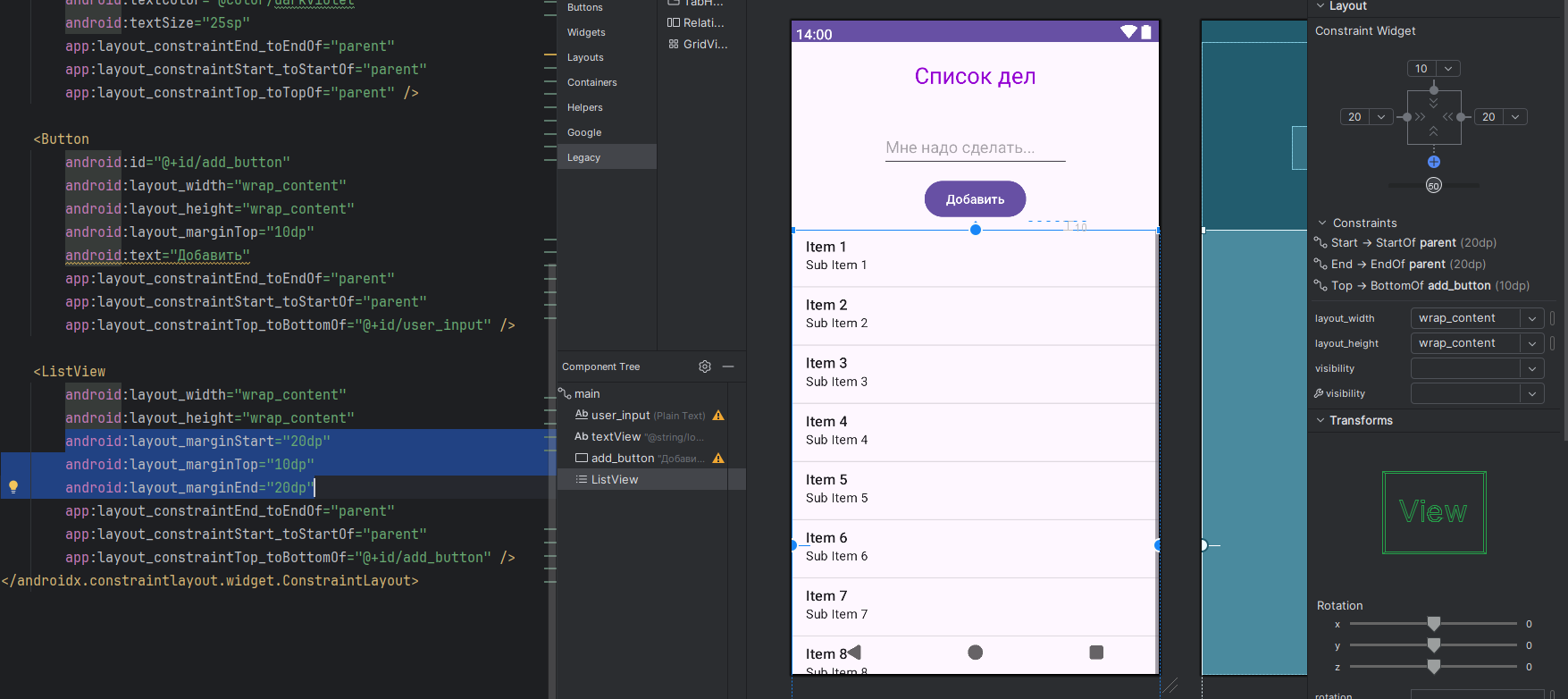
* 1. Теперь перейдем к добавлению нового элемента, который не использовался ранее. Добавим ListView, перенеся его в Component Tree в раздел main. Сделаем ему id «list\_view».



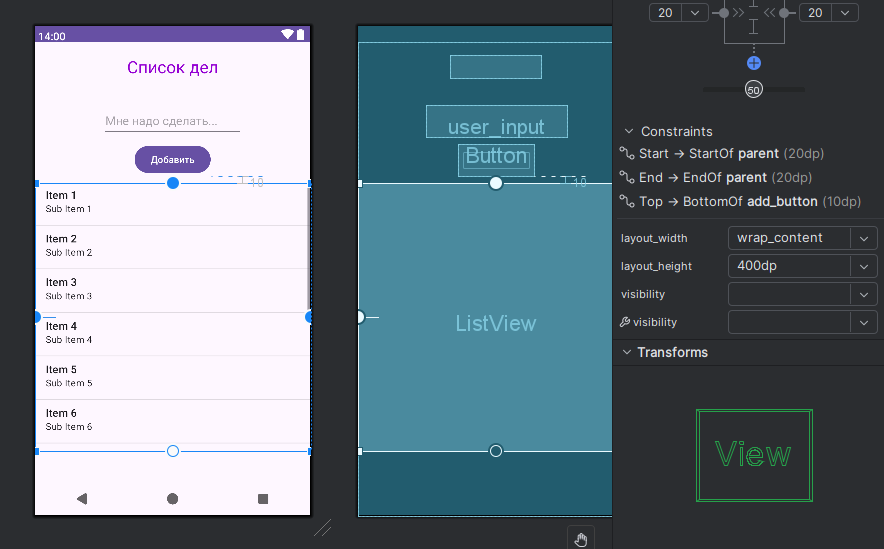
Теперь необходимо связать элемент с границами экрана, а также с кнопкой, расположенной выше. Мы можем увидеть, что наш ListView продолжает закрывать остальные элементы. Для того, чтобы это исправить, необходимо изменить поля layout\_width и layout\_height на wrap\_content



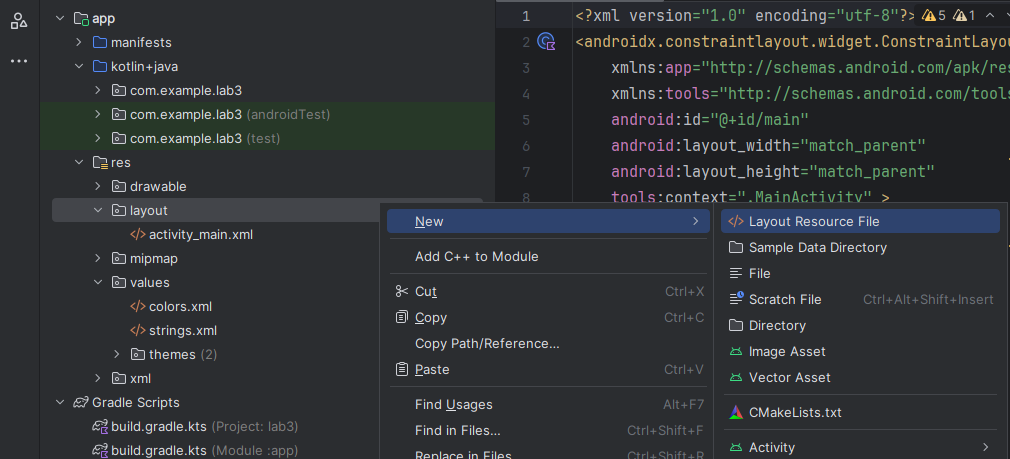
Далее мы можем добавить боковые отступы по 20 пикселей (либо другие, это на ваш выбор) и верхний отступ на 10. Получается следующая картина:



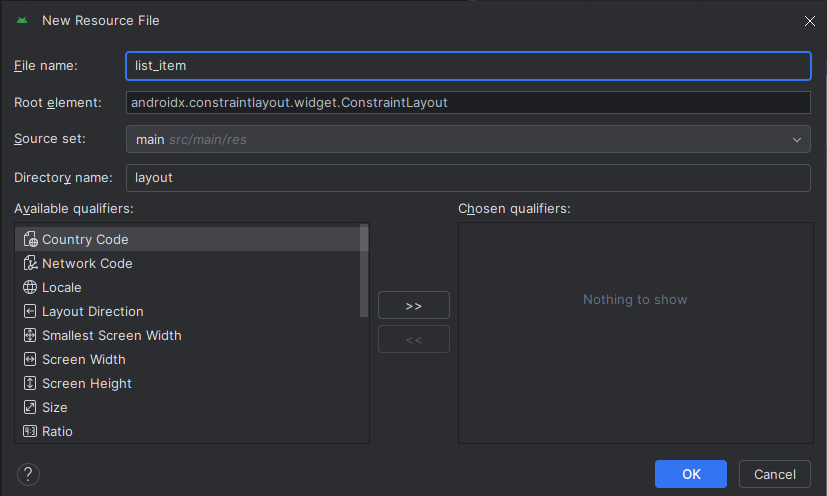
Из-за отсутствия нижней привязки элемента наш список «провалился» вниз за пределы экрана. Давайте исправим это, указав фиксированное значение высоты данного элемента (например, 400dp).



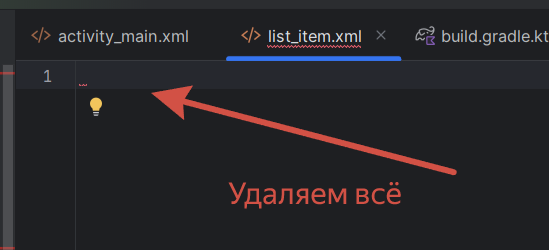
* 1. Мы создали список, но он какой-то слишком скучный. А ведь нам нужен более широкий функционал. Давайте сделаем таким элементы списка, которые будут идеально подходить под наше приложение. Для начала создадим ещё один xml файл. Для этого нажимаем ПКМ на папку layout и выбираем new > Layout Resource File.



Называем данный файл list\_item, больше ничего не меняем и жмём ОК.

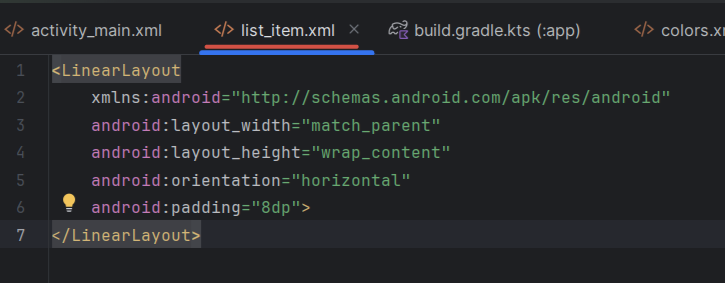


* 1. Удалим весь имеющийся xml код, так как вместо ConstraintLayout мы будем использовать LinearLayout.



* 1. Ключевое отличие этих двух элементов в том, что первый тип позволяет нам работать с другими компонентами, привязывая их к границам экрана, а второй располагать их в виде одной строки/столбца.

Создадим LinearLayout с горизонтальной ориентацией. Код нужно будет вводит вручную:

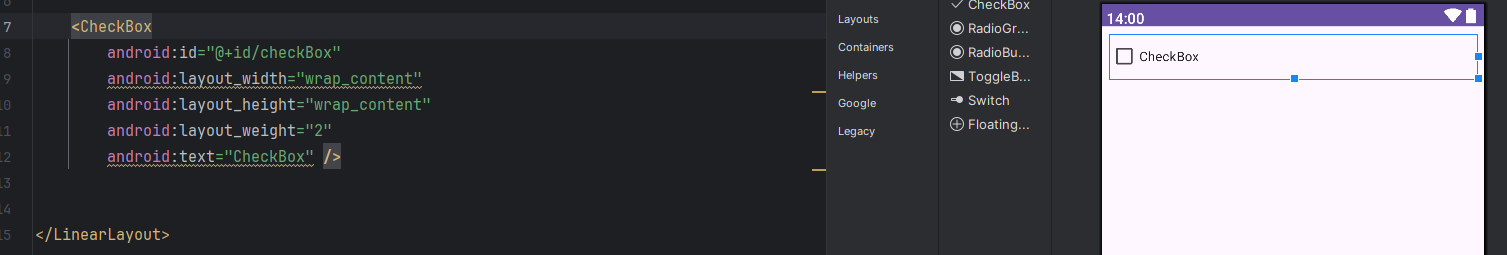


Задаем параметр ширины «match\_parent». Он означает, что элемент должен полностью заполнить родительский по вертикали/горизонтали. В нашем случае он должен занять всю длину строки в элементе списка.

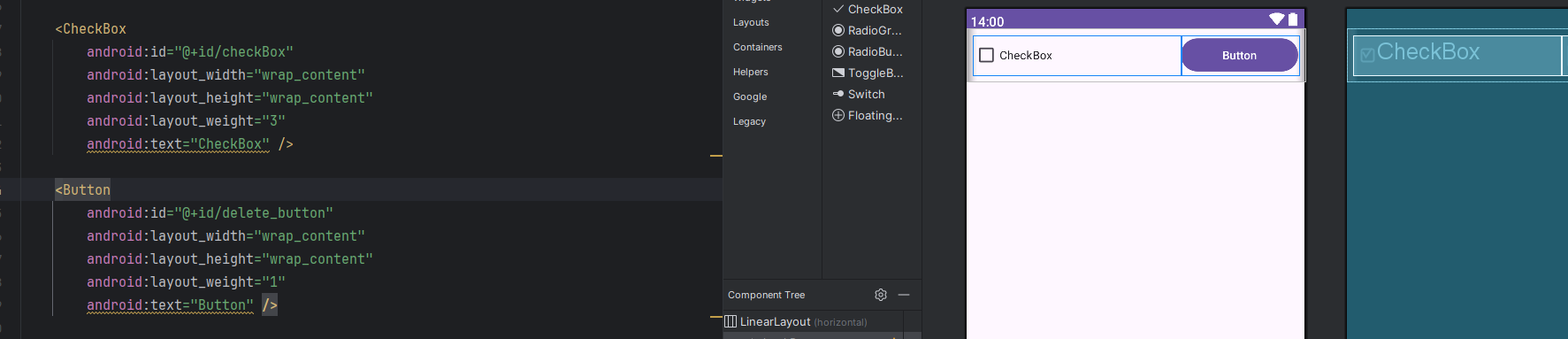
Высота же задаётся как wrap\_content, что делает элемент с минимальным размером, достаточным для того, чтобы всё поместилось.

Параметр padding задаёт «подкладку» для элемента. Не будет задавать слишком большое значение, чтобы строки списка не занимали много места.

* 1. Далее нам надо добавить элемент CheckBox, в котором мы будем отмечать выполненные дела. Ширину и высоту оставим “Wrap content”, чтобы элемент занимал максимальное необходимое место.



* 1. Теперь, когда у нас есть элемент для проставления отметок о выполнении дел, надо задуматься о возможности удаления элементов из списка. Давайте создадим кнопку, которая будет отвечать за данное действие. Высоту и ширину задаем аналогично чекбоксу. Также поменяем id элемента на delete\_button, чтобы не путай её с другими кнопками.



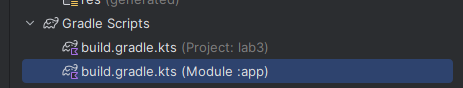
Как вы могли заметить, у последних двух добавленных элементов автоматически появляется поле android:layout\_weight. Оно отвечает за пространство, занимаемое элементов относительно нашего LinearLayout. Если установить значение «1» для обоих элементов, то каждый из них будет занимать ровно половину пространства. Нам это не подходит, так как мы будет писать свой текст в чекбокс, поэтому установим для него значение «3», что позволит занимать 3/4 ширины строки.

На этом работа с созданием макета завершена, и мы можем приступить непосредственно к программированию.

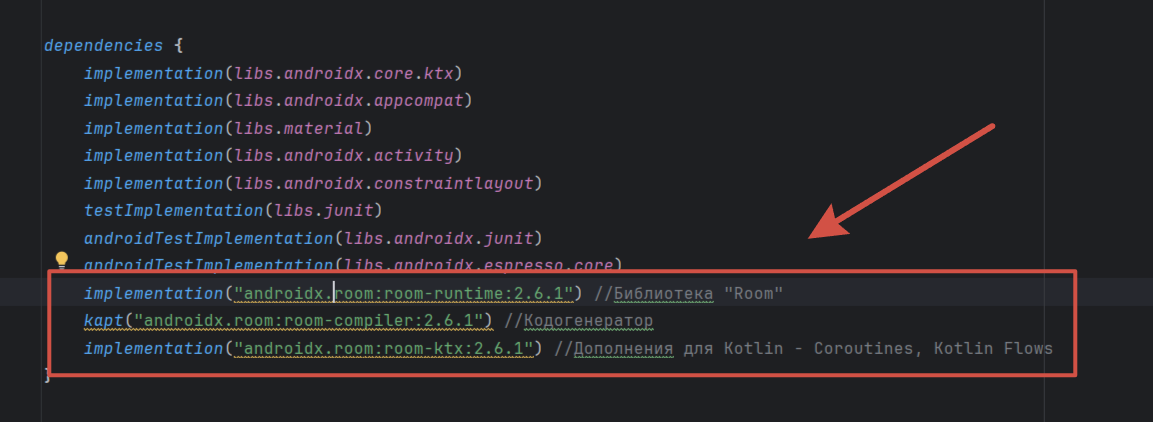
1. **Создание локальной БД71**

На данный момент у нас не написано никакого кода в нашем проекте. Давайте начнем с того, что создадим локальную базу данных при помощи библиотеки Room.

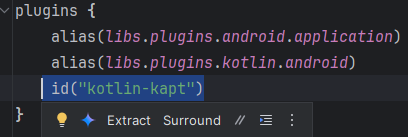
* 1. Первым делом нужно указать зависимости, без которых мы не сможем подключить библиотеку. Открываем файл build.gradle.kts, расположенный в Gradle Scripts. Обратите внимание, что нам нужен именно файл Module :app.

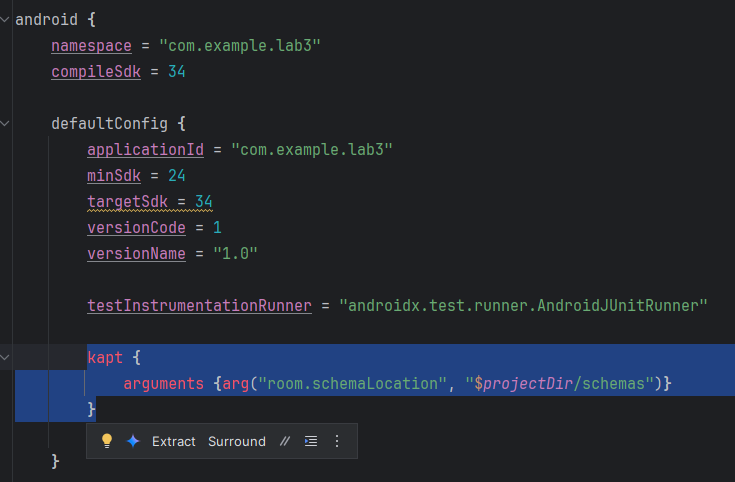


Здесь необходимо вставить следующий код:



Kapt выдает ошибку, так как такого элемента не существует в нашей программе. Давайте добавим его. Необходимо дописать следующие фрагменты кода:

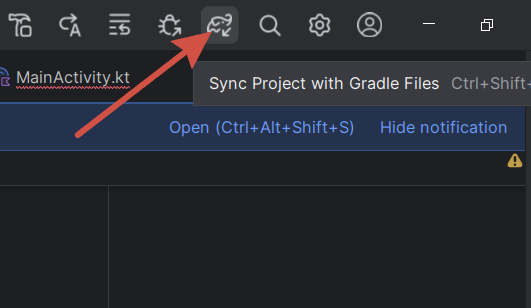




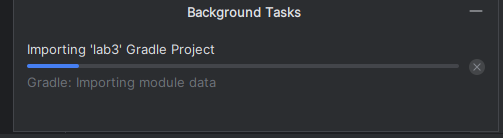
Ошибка всё ещё остаётся. В верхней части экрана можно заметить уведомление о том, что Gradle был изменен.



Нам необходимо нажать кнопку синхронизировать, чтобы AS смогла подгрузить необходимые данные



После загрузки всех нужных файлов ошибка пропадает.

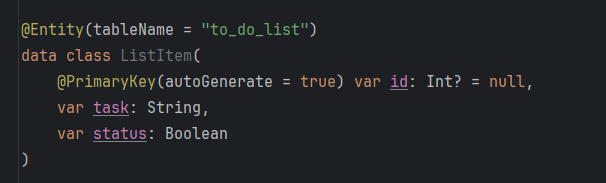


* 1. Теперь мы можем приступать к созданию базы данных. Её можно создавать как в отдельном файле, так и в том, где находится наша MainActivity. Обычно создание происходит отдельно, но наш проект учебный и содержит мало кода, так что создание рядом с MainActivity не будет слишком критичным.

Первым делом импортируем библиотеку для работы с Room в наш файл.

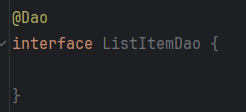


* 1. Давайте подумаем, какие типы данных будут содержаться в нашей базе. Первое это текст типа String с данными о наших делах. Второе – статус дел типа Bool, чтобы помечать выполнение. Для приложения, которое мы создаем в рамках текущей работы, данных полей будет достаточно.



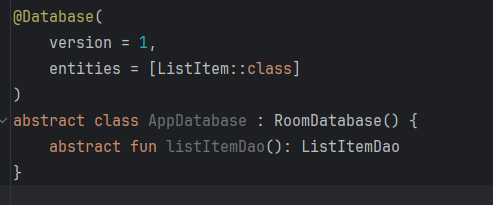
Создание таблицы происходит при помощи аннотации @Entity. В нашем лабораторной работе указывать название таблицы не обязательно, так как у нас она всего одна. Но в рамках крупных проектов это необходимо, чтобы БД работала корректно. Внутри Entity мы создаем data class, который называется «ListItem». В нем указываем автоматическую генерацию ключей (можно сделать переменную типа Long, но вряд ли кто-то из нас сможет создать достаточно дел, чтобы переполнить Int).

* 1. Следующим шагом является создание Dao интерфейса. Он позволит взаимодействовать с базой при помощи специальных методов.



Сейчас оставим его пустым, а все необходимые методы добавим позже.

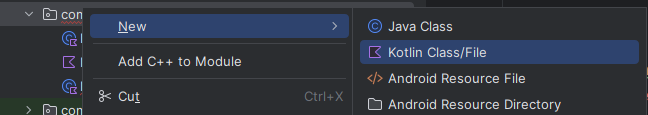
* 1. После создания сущностей и Dao-интерфейса необходимо создать абстрактный класс AppDatabase, который будет описывать базу данных.

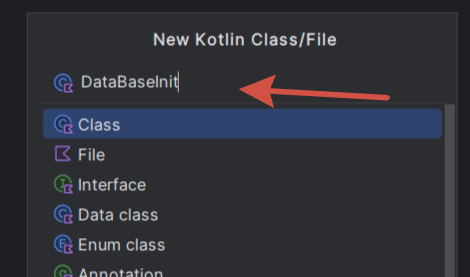


Данный класс помечен аннотацией @Database, где необходимо описать два свойства: *entities* и *version*. Первое свойство принимает все сущности, которые были описаны выше, а второе свойство задаёт версию базы данных. Версия базы данных используется для контроля версионности структуры базы данных и используется при миграции базы данных с одной версии на другую, но это уже совсем другая история...

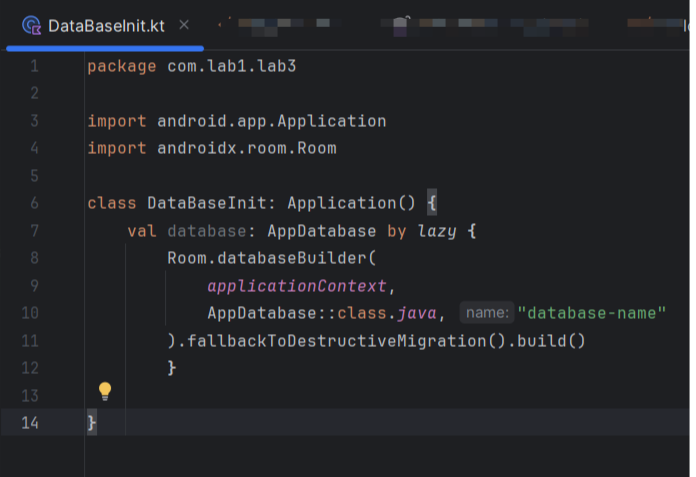
В самом классе создан абстрактный метод, который возвращает dao-интерфейс.

* 1. Далее создадим отдельный класс DataBaseInit.kt. Для этого на папку, где находится наша MainActivity, необходимо нажать правой кнопкой мыши и создать новый класс



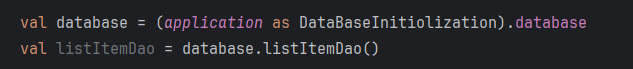


* 1. Теперь в созданном классе пропишем код для «ленивой» инициализации БД при помощи специального билдера: Room.databaseBuilder, который принимает контекст, класс, содержащий описание нашей базы данных, и название самой базы.

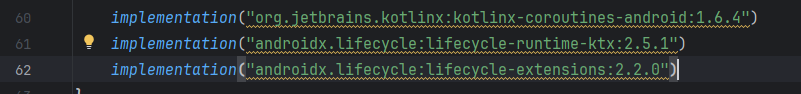


Если появляются ошибки, необходимо импортировать нужные библиотеки при помощи быстрых подсказок.

* 1. Последним шагом является объявление БД и Dao­-интерфейса в классе MainActivity внутри функции OnCreate().



* 1. Раз уж мы делаем проект в целях обучения, давайте сразу изучим ещё один полезный навык: работу с асинхронными потоками. В данной работе можно обойтись и без неё, но в больших приложениях данных подход позволяет намного эффективнее использовать ресурсы. Также будет затронут жизненный цикл. Но всё это будет сделано на очень простом уровне. Давайте вернемся к файлу build.gradle.kts, куда мы уже добавляли зависимости и подключим ещё несколько.



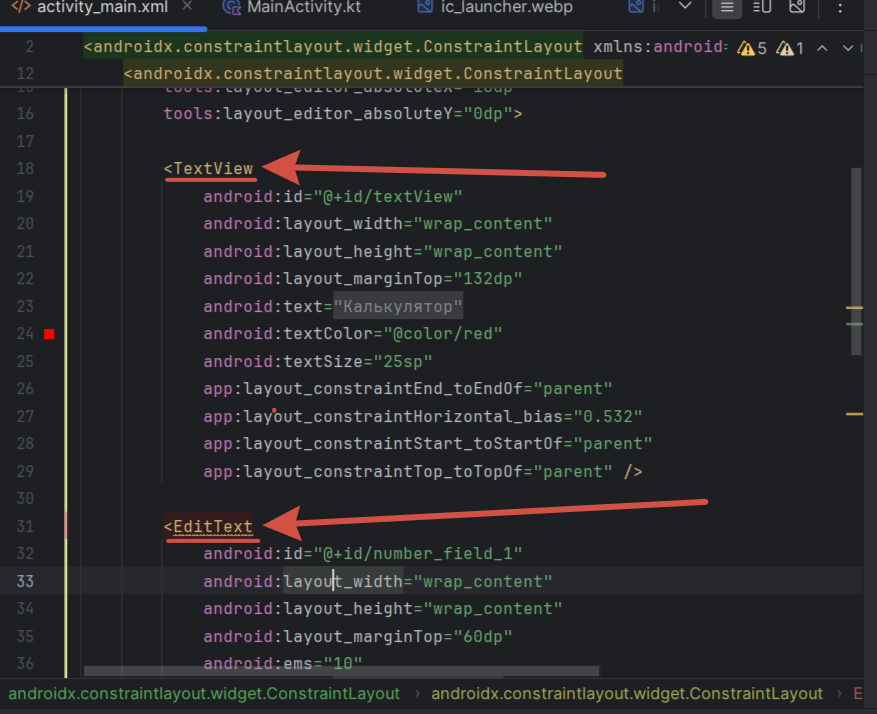
Как и в прошлый раз после добавления зависимостей необходимо синхронизировать Gradle.

1. **Непосредственно программирование**

Чтобы приложение заработало нам необходимо для всех визуальных объектов создать переменные.

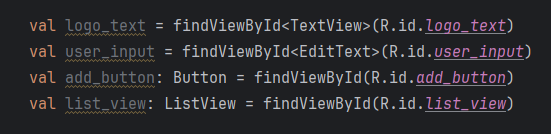
Добавим для каждого нашего элемента программный код. Это можно сделать, объявив переменные полями класса, а можно просто добавить их вызов по месту. Будем использовать второй вариант.

* 1. Для начала нужно посмотреть в Activity названия узлов для каждого из необходимых нам объектов:



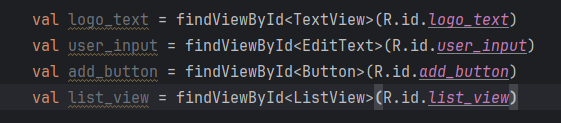
В конечном итоге это будут типы создаваемых нами объектов.

* 1. Открываем файл MainActivity.kt и там мы можем добавить для каждого элемента однотипную конструкцию (пока не вводите код):

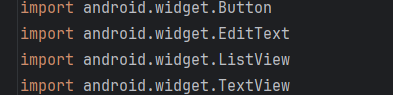


На данном рисунке показаны 2 различных способа инициализации элементов. В первом мы указываем тип объекта напрямую сразу после создания переменной, а во втором создаем переменную неопределенного типа, а присваиваем его уже после знака «=». В рамках одного проекта лучше использовать одинаковый синтаксис, так что сделаем все элементы в одинаковом стиле (будем использовать тот, что был в предыдущей работе).

На всякий случай, проверьте, что у вас activity\_main.xml заданы такие же id-ники, как и в коде ниже. Без этого код конечно же работать не станет.



Иногда при создании элемента может появляться ошибка. Этого происходит из-за того, что библиотеки для некоторых элементов необходимо подключать отдельно. Это делается либо вручную через код, либо автоматически, используя подсказки для быстрого исправления ошибок.



Проблемы как правило возникают, на MacOS, но это пока предварительная информация. Если у вас такое же на Windows, акцентируйте, пожалуйста, внимание преподавателя. Поправлю в методичке.

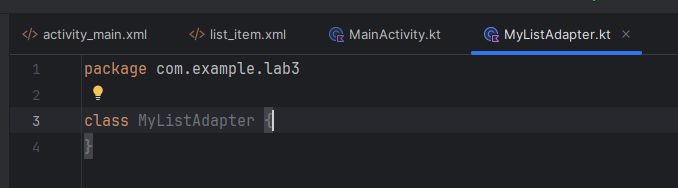
* 1. Теперь давайте сделаем так, чтобы при нажатии на кнопку «Добавить» введенный текст попадал в список. Для начала создадим сам список. В языке Kotlin это mutableList. Он будет содержать элементы типа ListItem (этот класс был создан выше).



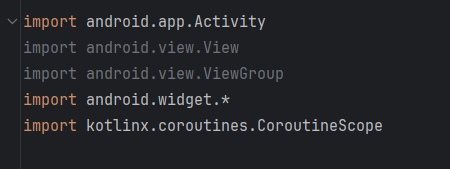
* 1. Для работы с данным списком нам нужен будет адаптер, который превратит наш пользовательский элемент в элемент списка ListView.

Для адаптера необходимо создать отдельный класс. Находим папку, где располагается наша MainActivity и добавляем туда новый котлин класс. Назовем этот класс «MyListAdapter».

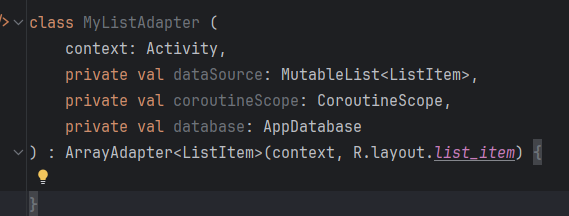
Появился новый пустой класс.



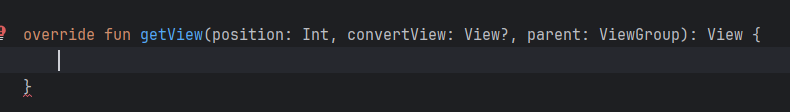
* 1. Теперь займемся созданием самого адаптера. Первым делом импортируем библиотеки, которые понадобятся нам позже



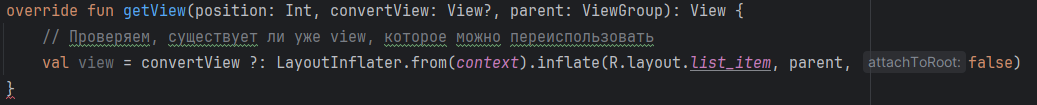
* 1. Далее наследуем наш класс от базового ArrayAdapter для элементов ListItem. В параметрах указываем контекст Activity, источник данных MutableList с тем типом данных, которые будут храниться в нашей программе, а также базу данных, созданную ранее, и корутины.



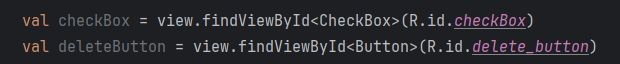
* 1. Первым делом в нашем классе надо определить метод getView(), который определяет, какие данные должны быть в каждой строке ListView.



Далее надо проверить, нет ли у нас уже готового View, которое можно использовать



После того, как мы проверили существование view в нашем проекте, мы можем приступать к основному функционалу данного метода. Для начала надо инициализировать элементы, которые расположены в нашем list\_item.xml. Это чекбокс и кнопка.



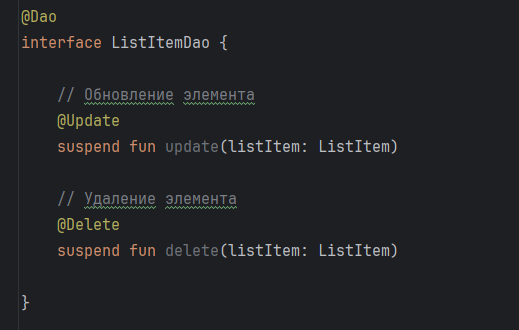
Теперь создадим переменную Item, которая будет содержать элемент из списка с выбранной позицией.



Далее в этот элемент добавим параметры, аналогичные тем, которые находятся в нашей БД.

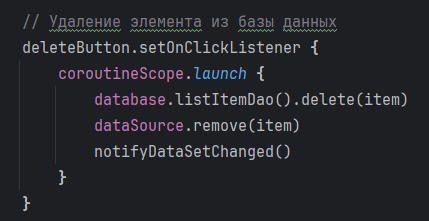


* 1. Для дальнейшей реализации функционала getView() нам понадобятся некоторые методы из Dao-интерфейса. Давайте их создадим. В каждом элементе списка есть два основных действия, которые может выполнять пользователь: проставление отметки о выполнении задания и удаление элемента списка. Если переводить данный функционал на язык SQL, то это методы Update и Delete.



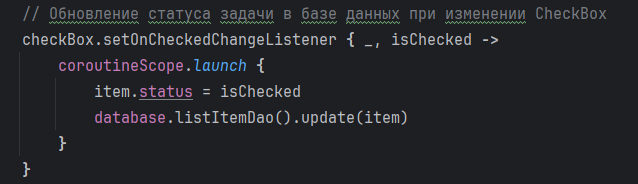
Оба созданных метода используют соответствующие аннотации, после чего прописывается синтаксис метода, который будет использоваться в основном проекте. В нашем случае это название самого метода и элемент списка в качестве параметра.

* 1. После создания необходимых методов в Dao-интерфейсе мы можем вернуться к нашему адаптеру, продолжив писать метод getView (модуль MyListAdapter.kt). Реализуем удаление элемента из БД при нажатии на соответствующую кнопку.



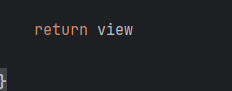
В этом куске кода мы привязываем к кнопке deleteButton действие, которое произойдет при нажатии. В данном случае у нас создается асинхронный поток, в котором выбранный элемент удаляется из базы данных и из списка, который видит пользователь. После удаления вызывается метод notifyDataSetChanged(), который оповещает адаптер о необходимости обновления списка.

* 1. Аналогично создается действие, которое будет выполняться при проставлении/удалении «галочки» в чекбоксе.



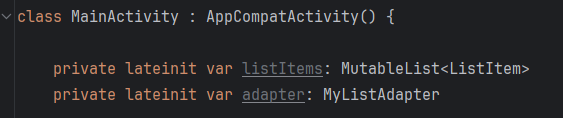
Статус элемента меняется на тот, что находится в чекбоксе, а база данных обновляется.

* 1. Последним действием в данном методе мы возвращаем полученное view.



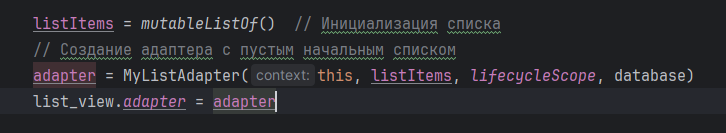
На этом наш создание адаптера завершено и можно приступать к работе с нашей MainActivity

* 1. Ранее мы уже создали список элементов listItems. Давайте подключим его к нашему адаптеру. Для начале в классе MainActivity вне метода OnCreate() необходимо инициализировать список и адаптер.

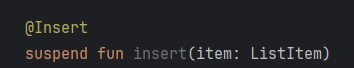


Инициализация вне конкретного метода производится для того, чтобы данные переменные были видны из всего класса, а не только из конкретного метода. Lateinit же нужен поздней инициализации переменных. Такой синтаксис позволит нам работать с элементами из любого места класс MainActivity, но инициализированы они будут только в методе OnCreate(), что позволит избежать некоторых возможных ошибок и сэкономить ресурсы.

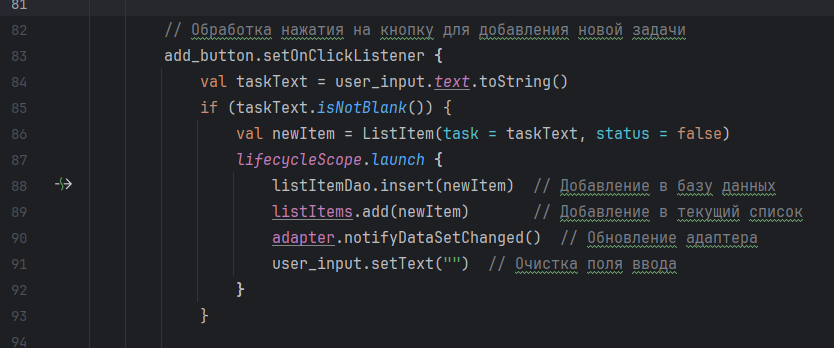
* 1. Теперь можно перейти к непосредственной инициализации списка и созданию адаптера. Мы наконец-то сможем их соединить. Инициализацию списка, которая была раньше, нужно убрать. Она изначально создалась только для того, чтобы ошибки не нервировали)



* 1. Для реализации добавления элемента в БД и список при нажатии на кнопки «Добавить» необходимо создать метод «Insert» в Dao-интерфейсе.

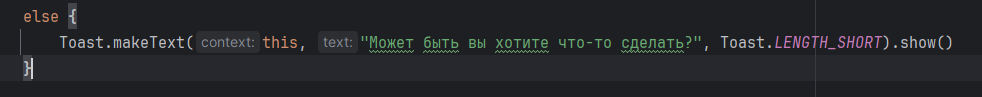


Далее можно вернуться в класс Main Activity и внутри OnCreate() создать активность для кнопки «Добавить»

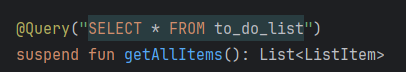


Сначала мы создаем переменную taskText, куда будет добавляться ввод пользователя и проверяем, что она не пустая. В случае, если пользователь написал хоть какой-то текст, мы добавляем его в БД и в список, обновляем адаптер и очищаем поле ввода.

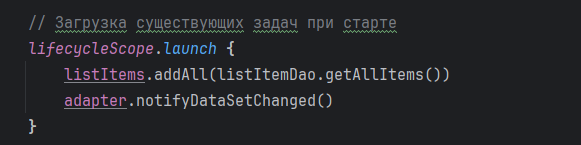
Также можно добавить всплывающую. Надпись в случае, когда пользователь ничего не ввел.



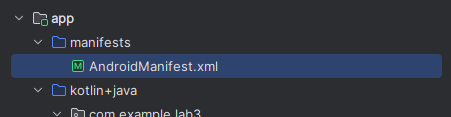
* 1. Осталось реализовать сохранение списка задач при выходе из приложения. Сначала создадим ещё один метод в Dao, который будет получать все элементы из таблицы.



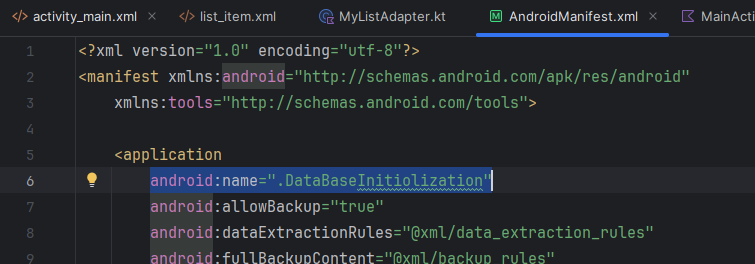
* 1. Теперь реализуем добавление всех задач при запуске приложения.



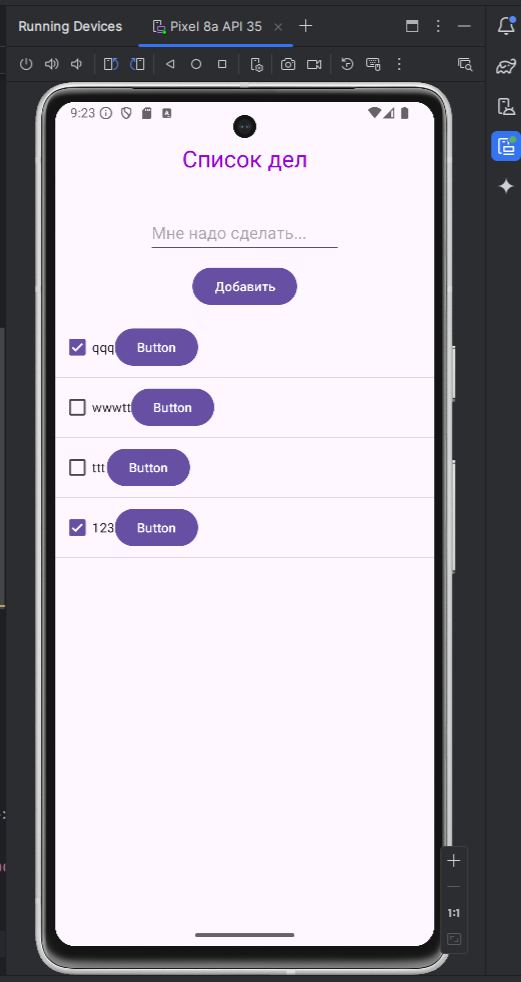
Давайте запустим наше приложение. Если оно запустилось сразу, всё замечательно. Если нет, необходимо проверить файл AndroidManifest.xml



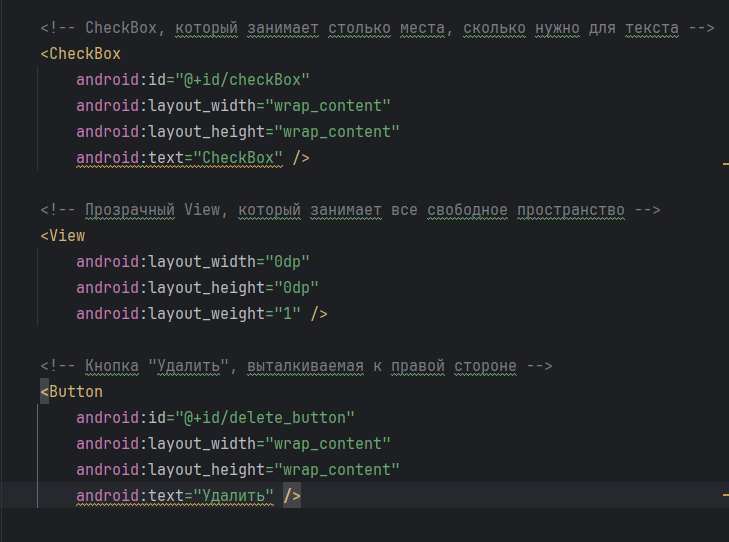
` Возможно, при создании класса с БД, он не добавился в манифест. В таком случае необходимо добавить следующий код:

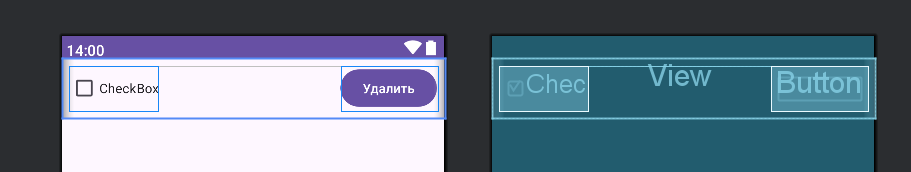


* 1. Давайте запустим наш проект и добавим несколько элементов в список. Получается не очень красивая картина, так как кнопка располагается сразу после текста.

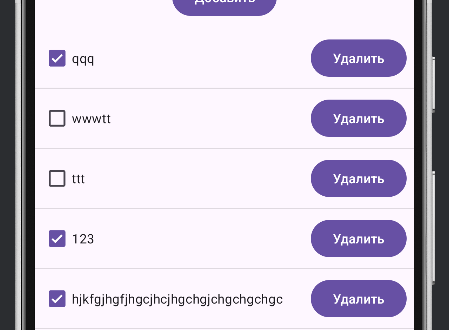


Для того, чтобы исправить данную проблему, перейдем в list\_item.xml. Мы устанавливали размеры элементов так, чтобы они занимали минимально возможное пространство. Из-за этого, кнопка сама смещается к тексту, если он короткий. Давайте не будем изобретать что-то новое и будем дальше работать с заполнением свободного пространства элементами. Но в этот раз мы поступим умнее. Добавим прозрачный View, который займет все доступное пространство, а кнопку и текст оставим такими, что они буду занимать только нужное им место.





Теперь наши элементы отображаются корректно.



На этом основной этап работы завершен. Далее будут задания для самостоятельного выполнения.

**III САМОСТОЯТЕЛЬНОЕ ЗАДАНИЕ**

1. Добавить кнопку, которая будет очищать весь список (а также полностью очищать БД).
2. Сделать так, чтобы выполненные задачи отмечались другим цветом (на ваше усмотрение).

**Вопросы, на которые хотелось бы мне получить ответ…**

1. Почему действия с БД (удаление элемента и отметка о выполнении) происходят корректно только при перезапуске приложения? То есть, если создать элемент и сразу его удалить, то он останется в БД (хотя и удалится из списка), из-за чего при перезапуске приложения он появится в списке. При этом, если его удалить уже после перезапуска приложения, то все работает корректно. Возможно, проблема в том, что элементы полноценно добавляются в базу только при закрытии приложения, а во время его работы они просто числятся в списке.
2. Почему асинхронные потоки получили такое большое распространение именно при работе с БД?