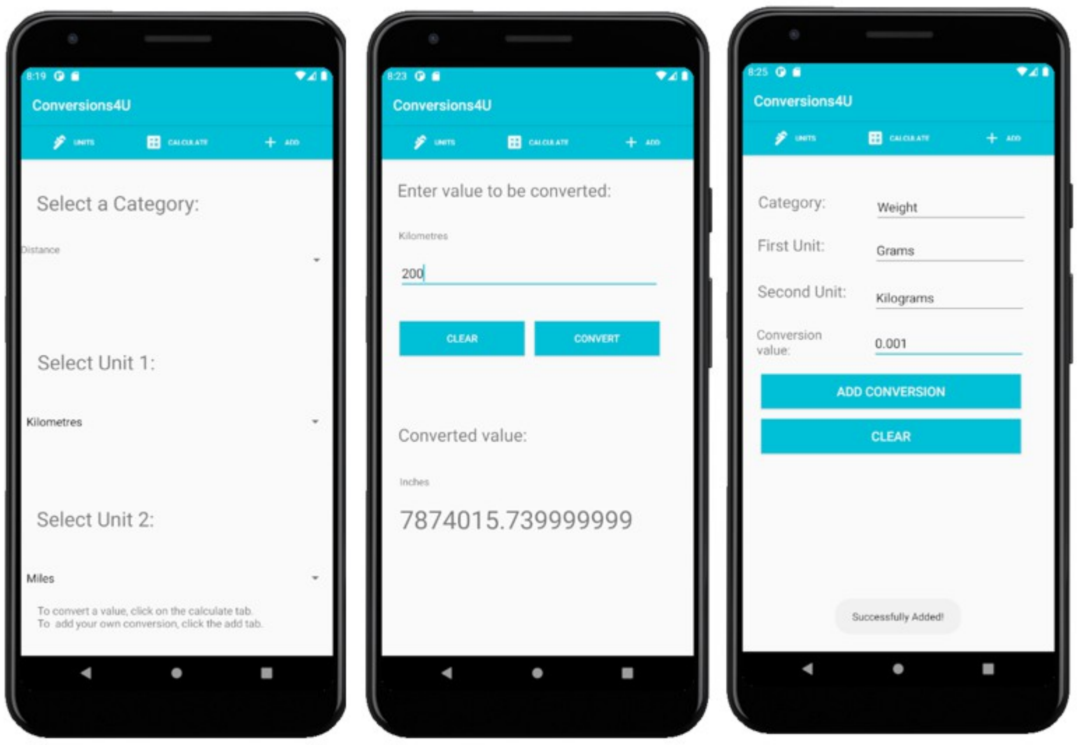
CSCU9YH

Mobile App Development

"UNIT CONVERSION APP"

2535156



Select a Category:
Distance
Select Unit 1:
Kilometres
Select Unit 2:
Miles
To convert a value, click on the calculate tab.
To add your own conversion, click the add tab.

Enter value to be converted:
Kilometres
200
CLEAR CONVERT
Converted value:
Inches
7874015.739999999

Category: Weight
First Unit: Grams
Second Unit: Kilograms
Conversion value: 0.001
ADD CONVERSION
CLEAR
Successfully Added!

1 – Implementation Detail

1.1 Layout & Design

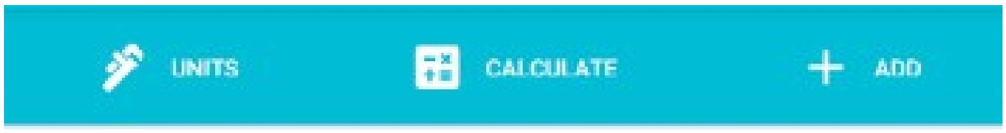
The design of the Conversions4U app has been kept as simple as possible

to provide ease of use to the user. It features an easy to navigate tablayout with fragments operating from the main activity, achieved using a pagerAdapter that implements the FragmentManager interface to keep track of and handle the switching from fragment to fragment.

When the Conversions4U app is opened the user is greeted with the option to

select their desired Category, Initial Unit and target Unit, they can navigate to either the calculate tab or add tab to convert a value and add a new conversion respectively.

On each tab, vector graphics have been used to add some visual aids for the user, featuring a calculate numerical vector graphic for the calculate tab, and a standard plus graphic for the add tab. These give the app more substance and makes it appear less 'bare bones'.



The colour scheme was picked for the contrasting nature as to make it easier for the user to differentiate between text and layout/functional elements and makes viewing easier. The app attempts to stay away from colour combinations like Red and Green as this could cause problems for people who suffer from colour blindness.

Additional features:

- Multiple tabs
- Clean, non-standard user interface
- Categories of conversions units
- Add fragment to add conversions to the local database
- A local "Room" database allowing local storage of data.
- Toast notification messages

1.1 Fragments

The Conversions4U app makes use of three fragments, each one with its own specific functionality and layout. The use of fragments is beneficial because they are reusable, more scalable and eliminate the tight coupling as seen in the more traditional activity and view architecture. This allows us to pass data from fragment to fragment, reuse them where necessary instead of having to create a new view, and are easier to cycle. Below is a breakdown of what each fragment does and how it achieves its task:

- Units Fragment – The units fragment allows the user to select their desired category, and unit pair using spinners. Upon selection of a category, the units spinners are updated and populated with units that are of the chosen category. There are three prompt labels (TextViews) to clarify which spinner is which, as well as a small information label at the bottom of the page. From here the user can navigate to any other fragment using the tabLayout at the top of the app.

- Calculate Fragment – The calculate fragment presents the user with an input box and two buttons; clear and convert. The user can enter the value they want converted and click the convert button to get the desired conversion. They can also click the clear button if they wish to enter a different value. If the user does not enter anything in the input box, a toast message is displayed reminding them to enter an input. Furthermore, if the user has selected a pair of units in the previous fragment (units) that are both the same – a toast message is displayed asking them to change one of the units.

- Add Fragment – The Add fragment presents the user with four text fields that can be submitted to the local database, thus creating their own specific conversion. In descending order the user can specify category, firstUnit, secondUnit and the conversion value and add it to the database. They can then navigate back to the units fragment by clicking on the units tab, and will be able to see their newly updated selection of conversions in the unit spinners once they have selected the new conversions category in the first spinner. If the user attempts to enter a pair of units that are the same, a toast message is displayed reminding them to change one of the units. The user can also make use of the clear button to clear the fields after an entry has been added. Upon clicking the add button, the user will be notified with a toast message to let them know the conversion has been successfully added to the database.

1.1 Communicating Between Fragments

The app uses a sharedViewModel as an interface between fragments, the view model is comprised of value setters. Once the user has selected their desired units, they are cast to a string and the setter methods are called from the sharedViewmodel, and the strings are passed as parameters. These are then assigned to argument values that can be accessed by another fragment. In the calculate fragment these string values are passed to a search query function by calling the conversionsViewModel, which in turn interfaces with the repository to query the database and returns a conversion value on observable using LiveData type String. This is then cast to a double or an int type with the input and multiplied together to produce the conversion result. This is then cast to a string and set as the text for the results plaintext element. The different conversion formats (decimal points/ whole numbers) are handled by identifying if the input or conversion value contains a decimal point, if so it is cast to a double, and if not, it is cast to an int type.

1.2 ROOM Database

For the purpose of storing and accessing data locally an android room database has been implemented. Room is a persistence library that offers an abstraction layer on top of sqlite and makes it easier to work with, it also reduces boilerplate code and allows us to easily interface with the database and run queries using a Data Access Object (DAO). We can then interface with this DAO using a repository class, with the equivalent query functions, and this can be called from the conversionsViewModel. Below is the list of classes/interfaces used to build and perform operations to the database – as well as their functions:

- Conversions.kt – This is class creates an entity (database table) were columns and their types can be specified – allowing us to specify what data will be stored and used.

- ConversionsDatabase.kt – This class creates an instance of the database locally.

- ConversionsDAO.kt – The Data Access Object class is where the queries used are specified and the corresponding functions and their return types implemented. It is the main interface used to access and perform actions on the data.

- ConversionsRepository.kt – The repository class is a repository for the functions that call and run the queries, it acts as an intermediate interface.

- ConversionsViewModel.kt – The view model allows us to interface with the repository and calls the repository functions. It can be accessed and used by the fragments to retrieve and add/change data.

1.1 Cases of error

Because the app database is not pre-populated, if conversions are added they are discrete. As in one conversion miles -> metres could be added, and kilometres -> feet could be added too as initial conversions. This will display both miles and kilometres in the first unit spinner, and metres and feet in the second unit spinner. Therefore, one could select kilometres -> metres as a conversion pair, even though the corresponding conversion value does not exist in the table. This causes the app to crash occasionally.

This could be remedied using a prepopulated database with all the conversion units and their category equivalents. However, this has not been implemented due to time constraints.

1.2 Potential Future Features

- Rotatable orientation – When the device is rotated, the application adapts to a new orientation with a new layout. This was not implemented due to time constraints.
- Delete/Edit conversion entries – Functionality could be implemented that allows the user to delete added conversions and edit conversions if they wish.
- List of database data with recyclerView – A recycler view could be implemented to allow the user to view a list of all the conversion data in the database.
- Prepopulate database with a full set of conversion units – On deployment the app would have access to a full list of conversions from the start.

1.1 Code Listings

Below are the code listings including xml files:

MainActivity :

```
/**
 * All fragments operate under the same activity - we use the pager adapter class
```

```kotlin
to add fragments to
 * the corresponding layout.
 */
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        setUpTabs()
    }

    private fun setUpTabs() {
        val adapter = ViewPagerAdapter(supportFragmentManager)

        adapter.addFragment(UnitsFragment(), "Units")
        adapter.addFragment(CalculateFragment(), "Calculate")
        adapter.addFragment(AddFragment(), "Add")
        viewPager.adapter = adapter
        tabs.setupWithViewPager(viewPager)

        tabs.getTabAt(0)!!.setIcon(R.drawable.ic_baseline_plumbing_24)
        tabs.getTabAt(1)!!.setIcon(R.drawable.ic_baseline_calculate_24)
        tabs.getTabAt(2)!!.setIcon(R.drawable.ic_baseline_add_24)


    }
}
```

ViewPagerAdapter :

```kotlin
class ViewPagerAdapter(supportFragmentManager: FragmentManager) :
    FragmentPagerAdapter(supportFragmentManager,
BEHAVIOR_RESUME_ONLY_CURRENT_FRAGMENT) {

    private val mFragmentList = ArrayList<Fragment>()
    private val mFragmentTitleList = ArrayList<String>()

    override fun getCount(): Int {
        return mFragmentList.size
    }

    override fun getItem(position: Int): Fragment {
        return mFragmentList[position]
    }

    override fun getPageTitle(position: Int): CharSequence? {
        return mFragmentTitleList[position]
    }

    fun addFragment(fragment: Fragment, title: String) {
        mFragmentList.add(fragment)
        mFragmentTitleList.add(title)
    }
}
```

UnitsFragment :

```kotlin
/**
 * The Units Fragment is the first page the user interacts with, it handles the
populating of spinners with data,
 * user inputs and selected units. It uses a conversions view model to interface
with DAO queries and a shared view model
 * to communicate with other fragments.
 */
class UnitsFragment : Fragment() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        mConversionsViewModel =
ViewModelProvider(this).get(ConversionsViewModel::class.java)

    }

    /**
     * Initialise both the conversions view model for handling queries and shared
view model for handling
     * communication between fragments.
     */
    private lateinit var mConversionsViewModel: ConversionsViewModel
    private val sharedViewModel: SharedViewModel by activityViewModels()
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.fragment_units, container, false)
        return view
    }

    /**
     * onViewCreated method calls all functions that populate spinner data once
the view inside the fragment has been
     * created and is being used - otherwise would return null.
     */

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // Create the landing page information.

        val info = instructionText.findViewById<TextView>(R.id.instructionText)
        info.setText("To convert a value, click on the calculate tab.\nTo  add
your own conversion, click the add tab.")

        // Defining a set list of categories.
        val setCategories = mutableListOf<String>("Distance", "Weight", "Speed")

        // Create category spinner and assign list to spinner

        val categorySpinner = view?.findViewById<Spinner>(R.id.catSpinner)
        categorySpinner.adapter = ArrayAdapter(
            requireActivity().baseContext,
            R.layout.support_simple_spinner_dropdown_item,
            setCategories
        )
        categorySpinner.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
            override fun onItemSelected(
                parent: AdapterView<*>?,
                view: View?,
                position: Int,
                id: Long
            ) {
                /**
                 * Once user has selected category from spinner - the selected
item is cast to string to
                 * be used as a parameter to set up and populate the following
units spinners using the database.
                 */
                initFirstUnitSpinnerData(categorySpinner.selectedItem.toString())
                initSecondUnitSpinnerData(categorySpinner.selectedItem.toString())
            }

            override fun onNothingSelected(parent: AdapterView<*>?) {
            }
        }

    }


    fun onUnitOneSelection(firstSelectedUnit: String) =
        sharedViewModel.setFirstSelectedUnit(firstSelectedUnit)

    private fun initFirstUnitSpinnerData(categorySelected: String) {

        // create spinner
        val spinnerFirstUnit = view?.findViewById<Spinner>(R.id.firstUnitSpinner)

        if (spinnerFirstUnit != null) {

            // Create arraylist adapter to store values from livedata list.
            val allConversions = context?.let {
                ArrayAdapter<Any>(it,
R.layout.support_simple_spinner_dropdown_item)
            }
            /**
             * Call view model to access repositoy and DAO to query the database
with parameter 'category' -
            passed as a parameter from onViewCreated method.
             */

            mConversionsViewModel.getByCategory(categorySelected)
                .observe(viewLifecycleOwner, { conversions ->
                    conversions?.forEach {
                        allConversions?.add(it)
                    }
                })
            spinnerFirstUnit.adapter = allConversions

            spinnerFirstUnit.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
                override fun onItemSelected(
                    parent: AdapterView<*>?,
                    view: View?,
                    position: Int,
                    id: Long
                ) {
                    Toast.makeText(requireContext(), "$allConversions",
Toast.LENGTH_LONG).show()
                    // pass unit to SharedModelView so it can be used by other
fragments
                    onUnitOneSelection(spinnerFirstUnit.selectedItem.toString())
                }

                override fun onNothingSelected(parent: AdapterView<*>?) {
                }
            }
        }

    }

    /**
     * REPEAT
     */

    fun onUnitTwoSelection(secondSelectedUnit: String) =
        sharedViewModel.setSecondSelectedUnit(secondSelectedUnit)

    private fun initSecondUnitSpinnerData(categorySelected: String) {
        val spinnerSecondUnit =
view?.findViewById<Spinner>(R.id.secondUnitSpinner)

        if (spinnerSecondUnit != null) {
            val allConversions = context?.let {
                ArrayAdapter<Any>(it,
R.layout.support_simple_spinner_dropdown_item)
            }
            mConversionsViewModel.getByCategoryTwo(categorySelected)
                .observe(viewLifecycleOwner, { conversions ->
                    conversions?.forEach {
                        allConversions?.add(it)
                    }
                })
            spinnerSecondUnit.adapter = allConversions

            spinnerSecondUnit.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
                override fun onItemSelected(
                    parent: AdapterView<*>?,
                    view: View?,
                    position: Int,
                    id: Long
                ) {
                    Toast.makeText(requireContext(), "$allConversions",
Toast.LENGTH_LONG).show()
                    onUnitTwoSelection(spinnerSecondUnit.selectedItem.toString())
                }

                override fun onNothingSelected(parent: AdapterView<*>?) {
                }
            }
        }
    }


}
```

CalculateFragment :

```
class CalculateFragment : Fragment() {

    private lateinit var mConversionsViewModel: ConversionsViewModel

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_calculate, container, false)

    }

    private val sharedViewModel: SharedViewModel by activityViewModels()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        mConversionsViewModel =
ViewModelProvider(this).get(ConversionsViewModel::class.java)
        // Initialise UI Elements

        val unitsFrom = view.findViewById<TextView>(R.id.unitsText)
        val unitsTo = view.findViewById<TextView>(R.id.secondUnitsText)

        // Create two "holder" variables
        var searchItemOne = ""
        var searchItemTwo = ""
        /**
         * Get selected units from spinners in unitsFragment using the
sharedViewModel
         */
        sharedViewModel.firstSelectedUnit.observe(
            viewLifecycleOwner,
            Observer { firstSelectedunit ->
                unitsFrom.text = firstSelectedunit
            })

        sharedViewModel.secondSelectedUnit.observe(
            viewLifecycleOwner,
            Observer { secondSelectedunit ->
                unitsTo.text = secondSelectedunit
            })
        /**
         * Assign chosen unit strings to holder variables
         */

        convertButton.setOnClickListener {
            var searchItemOne = unitsFrom.text.toString()
            var searchItemTwo = unitsTo.text.toString()
            val input = inputField.text.toString()

            /**
             * Check user input - if empty display message.
             */
            if (input == "") {
                Toast.makeText(
                    requireContext(), "Please enter a value to Convert!",
Toast.LENGTH_LONG
                ).show()
            } else {
                /**
                 * Holder variables are then passed as string parameters to the
getResult function.
                 */
                getResult(input, searchItemOne, searchItemTwo)
            }
        }

        convertClearButton.setOnClickListener {
            clearFields()
        }


    }

    /**
     * getResult function calls the conversionsViewModel to query the database
with the two selected units
     * as string parameters.
     * The returned observable value and user input is then  checked for a
present decimal points to avoid dataloss
     * if just casted to an int etc. This allows the conversion function to
handle differently delimited double values
     * and integers .
     * The function then assigns the outputted conversion value after
multiplication to the text attribute of the
     * resultField TextView.
     */
    private fun getResult(input: String, firstUnit: String, secondUnit: String) {

        if (firstUnit == secondUnit) {
            Toast.makeText(
                requireContext(), "Please select different units!",
Toast.LENGTH_LONG
            ).show()
        } else {
            mConversionsViewModel.getConversionUnit(firstUnit, secondUnit)
                .observe(viewLifecycleOwner, { convertValue ->
                    if (convertValue.contains(".") && input.contains(".")) {
                        resultField.text = ((convertValue.toDouble() *
input.toDouble()).toString())
                    } else if (!convertValue.contains(".") && !
input.contains(".")) {
                        resultField.text = ((convertValue.toInt() *
input.toInt())).toString()
                    } else if (convertValue.contains(".") && !input.contains("."))
{
                        resultField.text = (convertValue.toDouble() *
input.toInt()).toString()
                    } else {
                        resultField.text = (convertValue.toInt() *
input.toDouble()).toString()
                    }
                }
                )
        }
    }

    /**
     * This function clears the user input field and the result ouput field.
     */
    private fun clearFields() {
        try {
            inputField.setText("")
            resultField.setText("")

        } catch (exception: Exception) {
            Toast.makeText(
                requireContext(),
                "Could not clear data - please try again.",
                Toast.LENGTH_SHORT
            ).show()
        }

    }
}
```

AddFragment :

```
/**
 * This fragment contains functionality to add a new conversion to the room
database.
 */
class AddFragment : Fragment() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

    }

    private lateinit var mConversionsViewModel: ConversionsViewModel
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.fragment_add, container, false)

        mConversionsViewModel =
ViewModelProvider(this).get(ConversionsViewModel::class.java)

        view.addButton.setOnClickListener {
```

```kotlin
            insertDataToDatabase()

        }
        view.clearButton.setOnClickListener {
            clearFields()
        }
        return view
    }

    /**
     * fun insertDataToDatabase creates a new entity object to the database using
the
     * conversionsViewModel.
     */
    private fun insertDataToDatabase() {
        val category = categoryInput.text.toString()
        val firstUnit = firstUnitInput.text.toString()
        val secondUnit = secondUnitInput.text.toString()
        val convertValue = convertValueInput.text.toString()

        if (inputCheck(category, firstUnit, secondUnit, convertValue)) {
            val conversion = Conversions(0, category, firstUnit, secondUnit,
convertValue)

            if(firstUnit == secondUnit){
                Toast.makeText(requireContext(), "Please select different units!",
Toast.LENGTH_LONG
                ).show()
            }
            else {
                mConversionsViewModel.addConversion(conversion)
                Toast.makeText(requireContext(), "Successfully Added!",
Toast.LENGTH_LONG).show()
            }
        } else {
            Toast.makeText(requireContext(), "Please fill out all fields.",
Toast.LENGTH_LONG)
                .show()
        }
    }

    /**
     * Checks user input.
     */
    private fun inputCheck(
        category: String,
        firstUnit: String,
        secondUnit: String,
        convertValue: String
    ): Boolean {
        return !(TextUtils.isEmpty(category) && TextUtils.isEmpty(firstUnit) &&
TextUtils.isEmpty(
            secondUnit
        ) && TextUtils.isEmpty(convertValue))
    }

    /**
     * Clears all fields.
     */
    private fun clearFields() {
        try {
            categoryInput.setText("")
            firstUnitInput.setText("")
            secondUnitInput.setText("")
            convertValueInput.setText("")
        } catch (exception: Exception) {
            Toast.makeText(requireContext(), "Invalid data -try again",
Toast.LENGTH_SHORT).show()
        }

    }

}
```

SharedViewModel :

```
/**
 * This is a view model class shared by all fragments so that they may
communicate with each other and pass data.
 * Using a view model has proved easier than using the factory instance method
specified in the boilerplate fragment template,
 * as that is designed to be called from the shared activity the fragments are a
part of. It also prevents data loss when a fragment
 * is deestroyed ( e.g. when app is rotated ).
 */
class SharedViewModel : ViewModel() {
    /**
     * To pass data from onr fragment to another it must be in the form of
livedata
     * that can be changed/updated everytime there is an action performed in the
source fragment.
     */

    // First selected unit - unit to convert from.
    val firstSelectedUnit = MutableLiveData<String>()

    // Second selected unit - unit to convert to.
    val secondSelectedUnit = MutableLiveData<String>()

    /**
     * Set up setters to set the values of source fragment data - so destination
fragment listener returns updated values.
     */

    fun setFirstSelectedUnit(firstSelection: String) {
        firstSelectedUnit.value = firstSelection
    }

    fun setSecondSelectedUnit(secondSelection: String) {
        secondSelectedUnit.value = secondSelection
    }
}
```

Conversions :

```kotlin
/**
 * This is the conversions entity class, in this class we define a table within
the database and declare each field in
 * the table ( each field corresponds to a column. )
 */
@Entity(tableName = "conversion_table")
data class Conversions(
    @PrimaryKey(autoGenerate = true)
    val id: Int,
    val category: String,
    val firstUnit: String,
    val secondUnit: String,
    val convertValue: String
)
```

ConversionsDatabase :

```kotlin
/**
 * This class creates and returns an instance of the database
 */
@Database(entities = [Conversions::class], version = 1, exportSchema = false)
abstract class ConversionsDatabase : RoomDatabase() {

    abstract fun conversionsDAO(): ConversionsDAO

    companion object {
        @Volatile
        private var INSTANCE: ConversionsDatabase? = null

        fun getDatabase(context: Context): ConversionsDatabase {
            val tempInstance = INSTANCE
            if (tempInstance != null) {
                return tempInstance
            }
            synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    ConversionsDatabase::class.java,
                    "conversions_database"
                ).build()
                INSTANCE = instance
                return instance
            }
        }
    }
}
```

ConversionsDAO :

```
/**
 * This class is a DATA ACCESS object and is used to  access the database and run
queries.
 */
@Dao
interface ConversionsDAO {

    // This query adds a row to the table ( i.e. one whole conversion)
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun addConversion(conversions: Conversions)

    // This query lists all conversions held in the database.
    @Query("SELECT * FROM conversion_table ORDER BY id ASC")
    fun readAllData(): LiveData<List<Conversions>>

    // The two queries below select all rows with category input.

    @Query("SELECT DISTINCT firstUnit FROM conversion_table WHERE category LIKE
:search")
    fun getByCategory(search: String): LiveData<List<String>>

    @Query("SELECT DISTINCT secondUnit FROM conversion_table WHERE category
LIKE :search")
    fun getByCategoryTwo(search: String): LiveData<List<String>>

    // The query below fetches the conversion value that corresponds to the
selected spinner units.

    @Query("SELECT convertValue FROM conversion_table WHERE firstUnit LIKE
:searchOne AND secondUnit LIKE :searchTwo")
    fun getConversionUnit(searchOne: String, searchTwo: String): LiveData<String>
}
```

ConversionsRepository :

```
/**
 * The database repository acts like an API to keep the data and the app UI
seperate - it is used by
 * the conversionsViewModel to access the DAO's set of queries so that it can
retrieve, change and
 * perform operations on the database.
 */
class ConversionsRepository(private val conversionsDAO: ConversionsDAO) {

    val readAllData: LiveData<List<Conversions>> = conversionsDAO.readAllData()

    suspend fun addConversion(conversions: Conversions) {
        conversionsDAO.addConversion(conversions)
    }

    fun getByCategory(search: String): LiveData<List<String>> {
        return conversionsDAO.getByCategory(search)
    }

    fun getByCategoryTwo(search: String): LiveData<List<String>> {
        return conversionsDAO.getByCategoryTwo(search)
    }

    fun getConversionUnit(searchOne: String, searchTwo: String): LiveData<String>
{
        return conversionsDAO.getConversionUnit(searchOne, searchTwo)
    }

}
```

ConversionsViewModel :

```
/**
 * Conversions View model acts as an interface between the repository
 * and the application activities and fragments.
 */

class ConversionsViewModel(application: Application) :
AndroidViewModel(application) {
    private val readAllData: LiveData<List<Conversions>>
    private val repository: ConversionsRepository

    init {
        val conversionsDAO =
ConversionsDatabase.getDatabase(application).conversionsDAO()
        repository = ConversionsRepository(conversionsDAO)
        readAllData = repository.readAllData
    }

    fun addConversion(conversions: Conversions) {
        viewModelScope.launch(Dispatchers.IO) {
            repository.addConversion(conversions)
        }
    }

    fun getByCategory(search: String): LiveData<List<String>> {
        return repository.getByCategory(search)

    }

    fun getByCategoryTwo(search: String): LiveData<List<String>> {
        return repository.getByCategoryTwo(search)
    }

    fun getConversionUnit(searchOne: String, searchTwo: String)
            : LiveData<String> {
        return repository.getConversionUnit(searchOne, searchTwo)
    }


}
```

activity_main.xml :

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/appBarLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <com.google.android.material.tabs.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="#00BCD4"
            app:tabBackground="@color/colorPrimaryDark"
            app:tabGravity="fill"
            app:tabInlineLabel="true"
            app:tabMode="fixed"
            app:tabTextAppearance="@style/CustomTabStyle"
            app:tabTextColor="@android:color/white"/>

    </com.google.android.material.appbar.AppBarLayout>

    <androidx.viewpager.widget.ViewPager
        android:id="@+id/viewPager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@id/appBarLayout"/>

</RelativeLayout>
```

fragment_add.xml :

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".fragments.UnitsFragment">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <Button
        android:id="@+id/clearButton"
        android:layout_width="360dp"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="26dp"
        android:layout_marginTop="365dp"
        android:background="#00BCD4"
        android:text="Clear"
        android:textColor="#FAF9F9"
        android:textSize="18sp" />

    <EditText
        android:id="@+id/categoryInput"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="183dp"
        android:layout_marginTop="49dp"
        android:ems="10"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/secondUnitInput"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="181dp"
        android:layout_marginTop="175dp"
        android:ems="10"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/convertValueInput"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="180dp"
        android:layout_marginTop="238dp"
        android:ems="10"
        android:inputType="textPersonName" />

    <TextView
        android:id="@+id/secondULabel"
        android:layout_width="127dp"
        android:layout_height="43dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="21dp"
        android:layout_marginTop="173dp"
        android:text="Second Unit:"
        android:textSize="22sp" />

    <TextView
        android:id="@+id/valueLabel"
        android:layout_width="127dp"
```

```xml
            android:layout_height="43dp"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="20dp"
            android:layout_marginTop="236dp"
            android:text="Conversion value:"
            android:textSize="18sp" />

        <EditText
            android:id="@+id/firstUnitInput"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="182dp"
            android:layout_marginTop="109dp"
            android:ems="10"
            android:inputType="textPersonName" />

        <TextView
            android:id="@+id/CatLabel"
            android:layout_width="127dp"
            android:layout_height="43dp"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="22dp"
            android:layout_marginTop="49dp"
            android:text="Category:"
            android:textSize="22sp" />

        <TextView
            android:id="@+id/firstULabel"
            android:layout_width="127dp"
            android:layout_height="43dp"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="21dp"
            android:layout_marginTop="109dp"
            android:text="First Unit:"
            android:textSize="22sp" />

        <Button
            android:id="@+id/addButton"
            android:layout_width="360dp"
            android:layout_height="wrap_content"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="26dp"
            android:layout_marginTop="303dp"
            android:background="#00BCD4"
            android:text="Add Conversion"
            android:textColor="#FAF9F9"
            android:textSize="18sp" />
    </RelativeLayout>
</FrameLayout>
```

fragment_calculate.xml :

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".fragments.UnitsFragment">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">


        <TextView
            android:id="@+id/resultField"
            android:layout_width="362dp"
            android:layout_height="67dp"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="25dp"
            android:layout_marginTop="475dp"
            android:ems="10"
            android:inputType="textPersonName"
            android:textSize="35sp" />

        <TextView
            android:id="@+id/promptOne"
            android:layout_width="364dp"
            android:layout_height="55dp"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="22dp"
            android:layout_marginTop="26dp"
            android:text="Enter value to be converted:"
            android:textSize="24sp" />

        <TextView
            android:id="@+id/promptTwo"
            android:layout_width="364dp"
            android:layout_height="55dp"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="23dp"
            android:layout_marginTop="365dp"
            android:text="Converted value:"
            android:textSize="24sp" />

        <EditText
            android:id="@+id/inputField"
            android:layout_width="361dp"
            android:layout_height="wrap_content"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="24dp"
            android:layout_marginTop="135dp"
            android:ems="10"
            android:inputType="textPersonName" />
        <Button
            android:id="@+id/convertClearButton"
            android:layout_width="173dp"
            android:layout_height="wrap_content"
            android:layout_alignParentStart="true"
            android:layout_alignParentBottom="true"
            android:layout_marginStart="25dp"
            android:layout_marginBottom="416dp"
            android:background="#00BCD4"
            android:text="Clear"
            android:textColor="#FAF8F8" />

        <Button
            android:id="@+id/convertButton"
            android:layout_width="173dp"
            android:layout_height="wrap_content"
            android:layout_alignParentStart="true"
            android:layout_alignParentBottom="true"
            android:layout_marginStart="212dp"
            android:layout_marginBottom="416dp"
            android:background="#00BCD4"
            android:text="Convert"
            android:textColor="#FAF8F8" />

        <TextView
            android:id="@+id/unitsText"
            android:layout_width="98dp"
            android:layout_height="30dp"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="24dp"
            android:layout_marginTop="96dp" />

        <TextView
            android:id="@+id/secondUnitsText"
            android:layout_width="98dp"
            android:layout_height="30dp"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="25dp"
            android:layout_marginTop="437dp" />

    </RelativeLayout>
</FrameLayout>
```

fragment_units.xml :

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".fragments.UnitsFragment">

<RelativeLayout
```

```xml
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Spinner
        android:id="@+id/secondUnitSpinner"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true"
        android:layout_marginTop="556dp"
        android:layout_marginEnd="0dp"
        android:scrollbarSize="@dimen/scrollbarSizeTwo" />

    <Spinner
        android:id="@+id/firstUnitSpinner"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true"
        android:layout_marginTop="339dp"
        android:layout_marginEnd="0dp"
        android:scrollbarSize="@dimen/scrollbarSizeTwo" />

    <TextView
        android:id="@+id/secondUnitLabel"
        android:layout_width="368dp"
        android:layout_height="38dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="22dp"
        android:layout_marginTop="479dp"
        android:text="Select Unit 2: "
        android:textSize="28sp" />

    <TextView
        android:id="@+id/firstUnitLabel"
        android:layout_width="368dp"
        android:layout_height="38dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="23dp"
        android:layout_marginTop="262dp"
        android:text="Select Unit 1: "
        android:textSize="28sp" />

    <TextView
        android:id="@+id/catLabel"
        android:layout_width="368dp"
        android:layout_height="38dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="22dp"
        android:layout_marginTop="41dp"
        android:text="Select a Category:"
        android:textSize="28sp" />

    <Spinner
        android:id="@+id/catSpinner"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true"
        android:layout_marginTop="115dp"
        android:layout_marginEnd="-2dp"
        android:scrollbarSize="@dimen/scrollbarSizeTwo" />

    <TextView
        android:id="@+id/instructionText"
        android:layout_width="364dp"
        android:layout_height="64dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentBottom="true"
        android:layout_marginStart="23dp"
        android:layout_marginBottom="9dp"
        android:textSize="15sp" />

</RelativeLayout>

</FrameLayout>
```