# Data Storage on an Android Device Tutorial using JSON

By Alex Reiffer

## Overview:

The purpose of this tutorial is to allow the reader to be able to store data from their program in a JSON file to retain data across instances of their program locally. By the end of this tutorial a reader should be able to: write to a JSON file, read to a JSON file, and save a custom data class to a JSON file for larger data. For the purposes of this, readers will be instructed to create an app that saves an entered name that will be displayed as well as other information. After implementing the JSON save, this information will still be displayed after resetting the program.

## Getting Started:

Using the base project for Android Studio available at https://github.com/HomeTownUS/JSONTutorialBase provides the foundation for the tutorial. If you do not have git installed install git and then run
git clone https://github.com/HomeTownUS/JSONTutorialBase.git
to get the base needed for the tutorial installed. Similarly, by going to the repository in the link first provided, you can go to the green code button and download the repository as a zip folder. If using this method, extract all files from the zip folder.

Android Studio is needed to proceed to the next step. If you do not have Android Studio installed then install it now before proceeding. Android Studio can be installed at https://developer.android.com/studio. In Android Studio select File > Open and select either the cloned git repository or the unzipped folder and the base of the project should open.

It is recommended that readers have a basic understanding of Kotlin and Jetpack Compose as this tutorial relies upon both yet neither is taught in this tutorial. If you lack the skills to use or understand either, please learn before starting the tutorial.

## Coding Instructions:

Before we begin, it is important to note that this tutorial does not follow standard file separating conventions for the sake of simplicity. In practice, functions like save()

would go in a viewmodel or some other business logic file. Similarly, the data classes would also have their own file. This would be done to prevent the slowing down of the user interface by having these in these along with the code that runs our user interface.

Begin by adding the following line of code to the plugins portion of libs.versions.toml:

```
    kotlin-serialization = {id =
"org.jetbrains.kotlin.plugin.serialization", version.ref = "kotlin"}
```

Next add the alias to your top(project) level gradle and sync the gradle after adding it:

```
    alias(libs.plugins.kotlin.serialization) apply false
```

Following these two steps, go to your app level gradle and add the following two lines:

In the plugins section add

```
    alias(libs.plugins.kotlin.serialization)
```

and in the dependencies section add

```
    implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:x.x.x")
```

where x.x.x is the version number.

These are the dependencies needed to save any data to a JSON file as all data must be serialized before it can be written to a file.

After adding the dependencies, go back to the MainActivity and add the following import:

```kotlin
import kotlinx.serialization.Serializable
```

This allows us to add the data class that will be written to the JSON file. In the sample code, a name is entered as a string and displayed. Since we are only looking at the name right now, that will be the only data we will save; however, more data can be added to the data class to save as well. Below the composable add the following data class your code:

```kotlin
@Serializable

data class MyData(val name: String)
```

Following this we can begin to create our save function. The function will take in a string as input because the save function will not have access to the text variable in the composable function. From there we can put the string in our data class and save the data class to a JSON file as below:

```kotlin
fun save(text: String){

    val fileName = "save.json"

    val data = MyData(text)

    val saveString = Json.encodeToString(data)

    try {

        File(fileName).writeText(saveString)

    } catch (e: Exception) {println(e)/*Do nothing*/ }

}
```

Currently the save function does not get called so it does nothing so we will add a button in the composable that calls the save function as seen below.

```
      Button(onClick = {save(text)}) {

         Text("Click to save name")

      }
```

Now, if you attempt to run the current program as is, the save function will not work. This is because Android does not allow for files to be written to arbitrary locations. To fix this we will use Android Context to access a valid file directory to save our file. We will start by adding to the imports the following:

```
      import android.content.Context

      import androidx.compose.ui.platform.LocalContext
```

Now we can fix our save function by adding context as a parameter and adding the file path to the file we will attempt to write:

```
fun save(context: Context, text: String){//changed parameters

   val fileName = "save.json"

   val data = MyData(text)

   val saveString = Json.encodeToString(data)

   try {

       File(context.filesDir, fileName).writeText(saveString)//and filepath

   } catch (e: Exception) {println(e)/*Do nothing*/ }
}
```

Now that the save function is fixed, we need to fix the composable to properly call the save function. To begin with we will add a variable that holds the current context:

```
@Composable

fun Greeting(modifier: Modifier = Modifier) {

    val context = LocalContext.current // add this line here

    Column(modifier = Modifier.padding(16.dp)){
```

followed by changing the save function call in the button:

```
Button(onClick = {save(context,text)/*right here*/}) {

    Text("Click to save name")

}
```

Now the name will be saved properly but the issue is we never get the name from the JSON file we made. So next we will add a read function that will also need context passed to it to know the location of our JSON file. The read function should return the name as a string either from the file if it exists or an empty string if not as seen below:

```
fun read(context: Context):String{

    val file = File(context.filesDir,"save.json")

    if(file.exists()){

        val jsonString = file.readText()
        val data = Json.decodeFromString<MyData>(jsonString)

        return data.name

    }

    else {

        return ""

    }

}
```

Now that we have our read function we need to set our text to the results of the read function. Since read takes time, we will import

```
import kotlinx.coroutines.runBlocking
```

and use runBlocking to ensure read returns before we set text. Then we will add the read function to our composable function:

```
fun Greeting(modifier: Modifier = Modifier) {

    val context = LocalContext.current

    Column(modifier = Modifier.padding(16.dp)){

        val initialText = runBlocking { read(context) }

        var text by remember { mutableStateOf(initialText) }
```

Now your app will remember the saved name even after resetting the app. It is important to note that if the file that stores your name is corrupted or deleted, the app will no longer remember the name. Similarly, each save rewrites the save file so past data is lost. Since we saved our data in a data class, more variables can be added to save including arrays for example:

```
@Serializable

data class MyData(

   val name: String,

   val grades: List<Int>

   )
```

## Further Discussion:

By this point you should have a general understanding of using JSON to save information to a JSON file. Though it was not shown, a single primitive data, an array, or a string can be stored in JSON without using a data class. However, the use cases of these are limited and a more general approach using a data class was opted for.

Alternate options to save data locally include Android Datastore and Android Room Database. Android Room Database allows for more complex storage of data in the form of a database. Android Datastore is intended to save small amounts of data persistently and safely. Using a Proto Datastore allows for typesafe saving when using Android Datastore. If you wish to store data non-locally, I would recommend Android Firebase.

A completed version of this tutorial at https://github.com/HomeTownUS/JSONTutorial. Those unable to complete the tutorial should use this to follow along with the tutorial and study the differences between the base initially provided and the solution to gain an understanding.

## See Also:

AI can be a useful tool for learning. I would recommend exploring further with the help of AI but be careful as AI are not always accurate. For example, https://share.google/aimode/ENuIBLwvtXpPwH9Og

References:

https://medium.com/@midoripig1009/working-with-json-in-kotlin-parsing-and-serialization-a62300ec43b8

For other ways to use JSON in Kotlin:

https://stackoverflow.com/questions/41928803/how-to-parse-json-in-kotlin

Other methods for storing data:

Android Datastore:

- https://developer.android.com/topic/libraries/architecture/datastore

Android Room:

- https://developer.android.com/reference/androidx/room/RoomDatabase?hl=en
- https://developer.android.com/training/data-storage/room

Android Firebase:

- https://firebase.google.com/docs/android/setup
- https://firebase.google.com/