Advanced Mobile Application Development
Week 15: Android and Firebase

Firebase

Firebase approaches all platforms – iOS, Android, and Web with similar patterns. The main difference is in the setup.

We're going to use the same Recipes Firebase project that we used for iOS. Without authentication we need to make it public so our app can read and write.

In the Firebase console chose your database and then go into Authentication and in sign-in method disable all the sign-in providers.

Also back in Database go to the Rules tab and make sure they're public with read and write both set to true so our app can access the database.

```
  "rules": {
    ".read": true,
    ".write": true
  }
```

(had been "auth != null")

Android Studio

New project called Recipes

Basic Activity template

RecipeMainActivity

Setup

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

1. Create a Firebase project

Chose existing Recipes database or create a new one

2. Register your app with Firebase

Click the Android icon or Add app

Enter your app's application id which is the same as the package name which you can get from your Gradle files or Android manifest.

3. Add the Firebase configuration file

Click Download google-services.json to obtain your Firebase Android config file (google-services.json).

Go into the Project view.

Move your config file into the app-level directory of your app.

4.Add the Realtime Database to your app

Add the follow dependencies and Google services plugin to your gradle files:

build.gradle (project-level)

```
    Add Firebase Gradle buildscript dependency
        classpath 'com.google.gms:google-services:4.2.0'
```

Make sure you have the Google Maven repository

```
allprojects {
  repositories {
    jcenter()
```

```
    google()
  }
}
```

app/build.gradle

```
    Add Firebase plugin for Gradle
        apply plugin: 'com.google.gms.google-services'
```

The plugin must be at the BOTTOM of the file.

4.Add Firebase SDKs to your app

app/build.gradle

```
implementation 'com.google.firebase:firebase-database:16.1.0'
implementation 'com.firebaseui:firebase-ui-database:4.3.2'
```

The second dependency is for FirebaseUI which is a library that helps bind data into our app's UI.
You will need to add other dependencies for other Firebase functionality.
(note that the docs list 'com.google.firebase:firebase-core:16.0.8' but that adds analytics, not the realtime database).

For now go into the Firebase console and under Authentication disable Google sign-in.
In Database rules make sure anyone can read or write to your database.

```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

Model class

Go into your project's Java folder and click on the top folder and add a new Java class called Recipe for our model.

```
public class Recipe {
  private String id;
  private String name;
  private String url;

  public Recipe(){
    // Default constructor required for calls to DataSnapshot.getValue(Recipe.class)
  }

public Recipe(String newid, String newName, String newURL){
  id = newid;
  name = newName;
  url = newURL;
}
```

```
public String getId() {
   return id;
}

   public String getName(){
      return name;
   }

   public String geturl(){
      return url;
   }
}
```

Layout

Look at the activity_recipe_main.xml and look what the template has provided. It has an app bar layout with a toolbar, a FAB which we'll use, and it includes content_recipe_main.
For the FAB change the icon from email to add
app:srcCompat="@android:drawable/ic_input_add"

In the content_main.xml layout file replace the TextView with a RecyclerView to display the list(can be found in common or containers).
Add Project Dependency to add the support library with the RecyclerView. This will add it to your build.gradle(module: app) file.
Add start, end, top, and bottom constraints.
Make sure the RecyclerView has layout_width and layout_height set to “match_constraint” (0dp).
Make sure the RecyclerView has an id.

We also need a layout file that will specify the format of each row of the ListView.
File | New | Layout resource file
File name: list_item
Root element: android.support.constraint.ConstraintLayout
Source set: main
Directory name: layout

We want to have just one TextView for the receipe name with the id recipeTextView,
Make sure the TextView has layout_width and layout_height set to “wrap_content”.
Add missing constraints.
I made the text in the TextView larger.
android:textAppearance="@android:style/TextAppearance.Material.Medium"
I also added some bottom padding.
android:paddingBottom="10dp"
Once you don’t need the default text to help with layout, remove it.

Also change the constraint layout’s height to “wrap_content”. If the height is match_constraint each text view will have the height of a whole screen.
android:layout_height="wrap_content"

ViewHolder

Before we create our FirebaseRecyclerAdapter we need to define our ViewHolder class. Because FirebaseUI handles the ViewHolder construction for us, we won't have as many customization options.

In the java folder select the booklist folder (not androidTest or test)
File | New | Java class
Name: RecipeViewHolder
Kind: Class

```
@Override
public class RecipeViewHolder extends RecyclerView.ViewHolder {
    private TextView recipename;

    public RecipeViewHolder(@NonNull View itemView) {
        super(itemView);
        recipename = itemView.findViewById(R.id.recipeTextView);
    }

    public void setRecipeName(String name){
        recipename.setText(name);
    }
}
```

Adapter

In MainActivity.java we'll set up an ArrayList for our recipes, our adapter, RecyclerView and Firebase listener.
The FirebaseRecyclerAdapter binds a Query to a RecyclerView. When data is added, removed, or changed these updates are automatically applied to the UI in real time.
In onCreate() we'll define a query for our Firebase "recipes" child node.
Then we create an instance of FirebaseRecyclerOptions using the query we created.
Then we're ready to create the FirebaseRecyclerAdapter object using these options and the RecipeViewHolder class we created.
Once that's all set up we'll set up our RecyclerView and attach the adapter to it.

```
//array list of recipes
List<Recipe> recipes = new ArrayList<>();

//Firebase database recipe node reference
DatabaseReference reciperef = FirebaseDatabase.getInstance().getReference("recipes");

//define an adapter
FirebaseRecyclerAdapter adapter;

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_recipe_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
```

```
    //define a query
    Query query = FirebaseDatabase.getInstance().getReference().child("recipes");

//define a parser
SnapshotParser<Recipe> parser = new SnapshotParser<Recipe>() {
    @NonNull
    @Override
    public Recipe parseSnapshot(@NonNull DataSnapshot snapshot) {
        Recipe newRecipe = new Recipe(snapshot.getKey(),
                snapshot.child("name").getValue().toString(),
                snapshot.child("url").getValue().toString());
        recipes.add(newRecipe);
        return newRecipe;
    }
};

    //define adapter options
    FirebaseRecyclerOptions<Recipe> options = new
FirebaseRecyclerOptions.Builder<Recipe>().setQuery(query, parser).build();

    //create an adapter
    adapter = new FirebaseRecyclerAdapter<Recipe, RecipeViewHolder>(options) {

        @NonNull
        @Override
        public RecipeViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, int i) {
            View view = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.list_item,
viewGroup, false);
            return new RecipeViewHolder(view);
        }

        @Override
        protected void onBindViewHolder(@NonNull RecipeViewHolder holder, int position, @NonNull
Recipe model) {
            holder.setRecipeName(model.getName());
        }
    };

    // get the recyclerview
    RecyclerView recyclerView = findViewById(R.id.recyclerView);

    //divider line between rows
    recyclerView.addItemDecoration(new DividerItemDecoration(this,
LinearLayoutManager.VERTICAL));

    //assign the adapter to the recycle view
    recyclerView.setAdapter(adapter);
```

```
    //set a layout manager on the recycler view
    recyclerView.setLayoutManager(new LinearLayoutManager(this));
}
```

The FirebaseRecyclerAdapter uses an event listener to monitor changes to the Firebase query. To begin listening for data, call the startListening() method. To stop listening call the stopListening() method. We'll use the onStart() method to start listening and the onStop() method to stop listening.

```
@Override
protected void onStart() {
    super.onStart();
    adapter.startListening();
}

@Override
protected void onStop() {
    super.onStop();
    adapter.stopListening();
}
```

You should now be able to run your app and see all your Firebase data. If you make any changes through the console your app should automatically update.

Add data

In RecipeMainActivity we already have a listener all set up for this button so we'll use that. In onCreate() we'll replace the Snackbar with an AlertDialog and the logic to add a recipe.

```
        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                //create a vertical linear layout to hold edit texts
                LinearLayout layout = new LinearLayout(RecipeMainActivity.this);
                layout.setOrientation(LinearLayout.VERTICAL);

                //create edit texts and add to layout
                final EditText nameEditText = new EditText(RecipeMainActivity.this);
                nameEditText.setHint("Recipe name");
                layout.addView(nameEditText);
                final EditText urlEditText = new EditText(RecipeMainActivity.this);
                urlEditText.setHint("URL");
                layout.addView(urlEditText);

                //create alert dialog
                AlertDialog.Builder dialog = new AlertDialog.Builder(RecipeMainActivity.this);
                dialog.setTitle("Add Recipe");
                dialog.setView(layout);
                dialog.setPositiveButton("Save", new DialogInterface.OnClickListener() {
```

```
            @Override
            public void onClick(DialogInterface dialog, int which) {
                //get entered data
                String recipeName = nameEditText.getText().toString();
                String recipeURL = urlEditText.getText().toString();
                if (recipeName.trim().length() > 0) {
                    //get new id from firebase
                    String key = reciperef.push().getKey();
                    //create new recipe item
                    Recipe newRecipe = new Recipe(key, recipeName, recipeURL);
                    //add to Firebase
                    reciperef.child(key).child("name").setValue(newRecipe.getName());
                    reciperef.child(key).child("url").setValue(newRecipe.geturl());
                    }
                }
            });
            dialog.setNegativeButton("Cancel", null);
            dialog.show();
        }
    });
```

When we create the AlertDialog it's in an anonymous function so when implementing the listener, outer class RecipeMainActivity has to be specified to refer to the Activity instance and the keyword this in Java applies to the most immediate class being declared.

You should now be able to add recipes and see them in your recyclerview as well as in Firebase through the console.

Delete data

To delete items through your app we'll present a context menu on a long press which we'll add to the onBindViewHolder() method in our adapter.

```
//context menu
holder.itemView.setOnCreateContextMenuListener(new View.OnCreateContextMenuListener() {
    @Override
    public void onCreateContextMenu(ContextMenu menu, final View v,
ContextMenu.ContextMenuInfo menuInfo) {
        //set the menu title
        menu.setHeaderTitle("Delete " + model.getName());
        //add the choices to the menu
        menu.add(1, 1, 1, "Yes").setOnMenuItemClickListener(new
MenuItem.OnMenuItemClickListener() {
            @Override
            public boolean onMenuItemClick(MenuItem item) {
                //get recipe that was pressed
                int position = holder.getAdapterPosition();
                //get recipe id
                String recipeid = recipes.get(position).getId();
                //delete from Firebase
```

```
            reciperef.child(recipeid).removeValue();
            Snackbar.make(v, "Item removed", Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
            return false;
        }
    });
    menu.add(2, 2, 2, "No");
  }
});
```

Now you should be able to delete items on a long click and see them removed from your Firebase database and your RecyclerView.

Load web page

When the user taps on a recipe we want to load the recipe url in an app that can load web pages like a browser. We're going to use an implicit intent so the user will be prompted to chose an app to load the web page if the device has more than one capable of doing so.
Add an onClickListener to the onBindViewHolder() method in our adapter.

```
//onclick listener
holder.itemView.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    //get recipe that was pressed
    int position = holder.getAdapterPosition();
    //get recipe id
    String recipeURL = recipes.get(position).geturl();
    //create new intent
    Intent intent = new Intent(Intent.ACTION_VIEW);
    //add url to intent
    intent.setData(Uri.parse(recipeURL));
    //start intent
    startActivity(intent);
  }
});
```

Now when you tap on a recipe its web page should open in a browser.