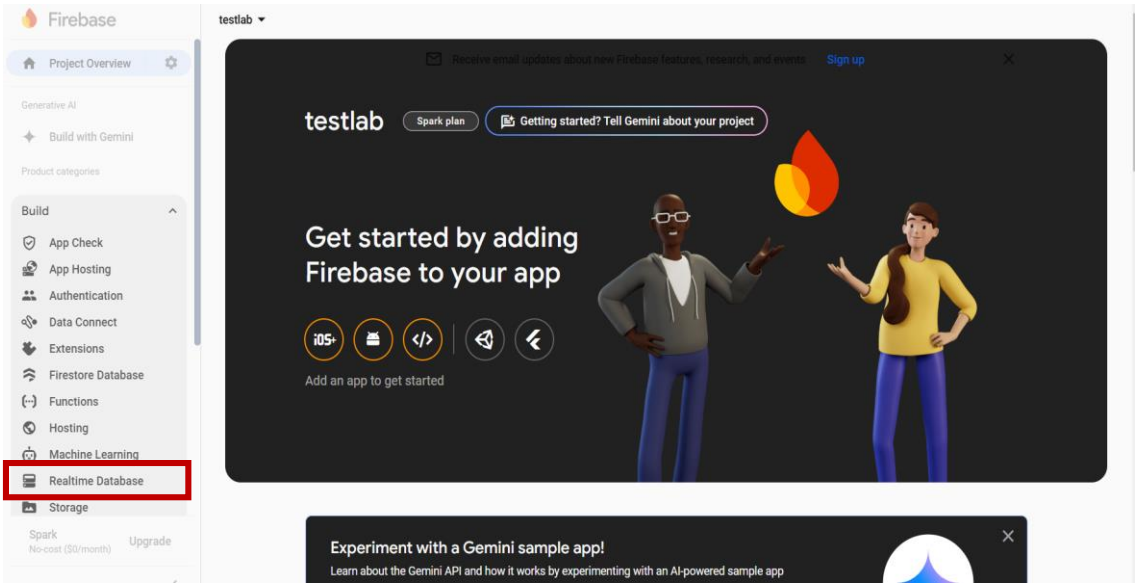
CSE 2200: Advanced Programming
Lab Tutorial: Firebase Realtime Database

In this lab tutorial, we will implement **CRUD (Create, Read, Update, Delete)** system in Android applications using **Firebase Realtime Database.**

Create and Insert Data

Step-1: go to https://console.firebase.google.com/ > Sign in through your google ID > Click "Getting Started" > "Create a Project with name and id" > You can turn off Analytics option for now. Then you will have the below interface in your console.



Then go to **"Build"> Choose "Realtime Database" > "Get Started" > Create Database.**

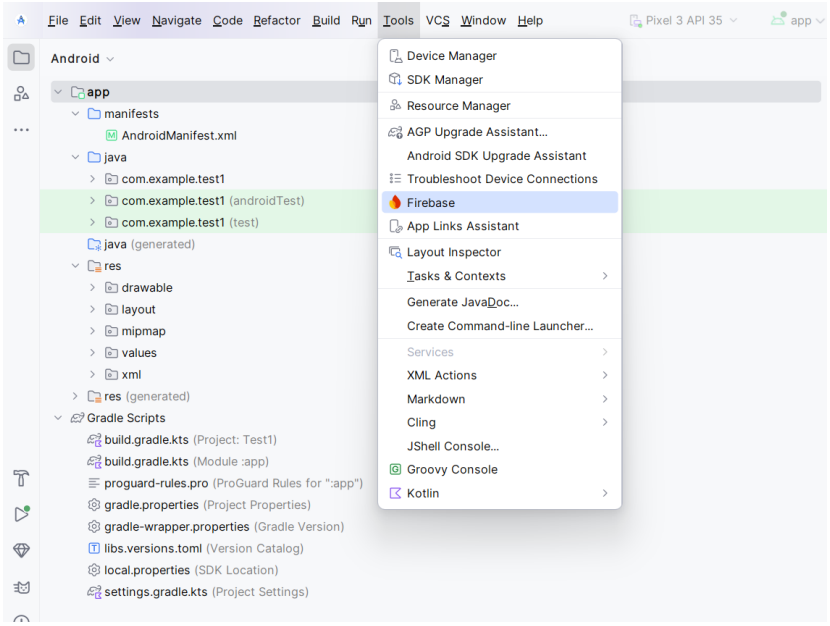
Go to 'Rules' and change as the following.

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

Once you publish this, Step-1 is complete.

CSE 2200: Advanced Programming
Lab Tutorial: Firebase Realtime Database

Step-2: Now you need to go to Android Studio, Create an empty project. Go to **"Tools".** Choose Firebase.



Now with the assistant window, go to Realtime Database and Click **"Get Started with Realtime Database".**

Step-3:

- Click "Connect your app to Firebase", a pop up window will be opened in your browser. Choose the app in your firebase control that you have created in Step-1.
- Click "Add the Firebase Realtime Database SDK to your app".
- Once both of the process are marked done, you can check in the Gradle files to see if the dependencies are added properly. If not, you have to do it manually.

build.gradle.kts (Test1)

```
1  // Top-level build file where you can add configuration options common to all sub-projects/modules.
2  plugins {
3      alias(libs.plugins.android.application) apply false
4      alias(libs.plugins.google.gms.google.services) apply false
5  }
```

Step-4: For adding data to Firebase you have to give permissions for accessing the internet. For adding these permissions navigate to the **app > AndroidManifest.xml** and inside that file **add** the below **permissions** to it.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

Step-5: Go to the **activity_main.xml** file and refer to the provided code **activity_main.xml**. Make necessary changes according to your project. You can also design as per your requirements.

Step-6: Add the action bar to your app. Go to **'AndroidManifest.xml'** and replace the theme portion to this line:

```
android:theme="@style/Theme.AppCompat.Light.DarkActionBar"
```

go to each activity classes and inside **'onCreate'** function add the following line:

```
this.setTitle("Insert Data"); //inside main activity class
```

Add similarly to other activity classes too.

Step-7: For sending multiple data to the Firebase Realtime database you have to **create an Object class** and **send** that whole **object class to Firebase**. For creating an object class navigate to the **app > java > your app's package name** > Right-click on it and Click on **New > Java Class >** Give a name to your class. In this case, it is **EmployeeInfo**. Now, define private attributes `employeeName, employeeContactNumber, employeeAddress, employeeID` according to your layout textfields. Then Right Click on your mouse > Select 'Generate > Constructor. (Create an empty constructor required when using Firebase Realtime Database.) Then Right Click again on your mouse > Select 'Generate > Getter and Setter. (Implement getter and setter for all the attributes.)

Step-8: Go to the **MainActivity.java** file. Now initialize objects for your **EditText,** and **Button** widgets that you have created in your layout and connect their IDs. In addition to these objects you also need to initialize the **FirebaseDatabase, DatabaseReference** (will be used for referencing to the database you have created in Firebase) and **EmployeeInfo** class objects.

```
private EditText employeeNameEdt, employeePhoneEdt, employeeAddressEdt;
private Button sendDatabtn;
FirebaseDatabase firebaseDatabase;
DatabaseReference databaseReference;
EmployeeInfo employeeInfo;
```

Now inside the onCreate function: Connect the IDs. Create instances for **FirebaseDatabase, DatabaseReference** and **EmployeeInfo** class.

```
firebaseDatabase = FirebaseDatabase.getInstance();
databaseReference = firebaseDatabase.getReference("EmployeeInfo");
//EmployeeInfo will be the name of the firebase database root node. Each
entries will be added to this node once inserted.
employeeInfo = new EmployeeInfo();
```

Next you have to implement the button **click** listener for **sendDatabtn**. Inside the '**onClick**' method:

```
// getting text from our edittext fields.
String name = employeeNameEdt.getText().toString();
String phone = employeePhoneEdt.getText().toString();
String address = employeeAddressEdt.getText().toString();
```

You can check **input constraints** too.

```
// below line is for checking whether the
// edittext fields are empty or not.
if (TextUtils.isEmpty(name) && TextUtils.isEmpty(phone) &&
TextUtils.isEmpty(address)) {
    // if the text fields are empty
    // then show the below message.
    Toast.makeText(MainActivity.this, "Please add some data.",
Toast.LENGTH_SHORT).show();
} else {
    // else call the method to add
    // data to our database.
    addDatatoFirebase(name, phone, address);
}
```

Finally, you need to define the body of the function: **`addDatatoFirebase`**.

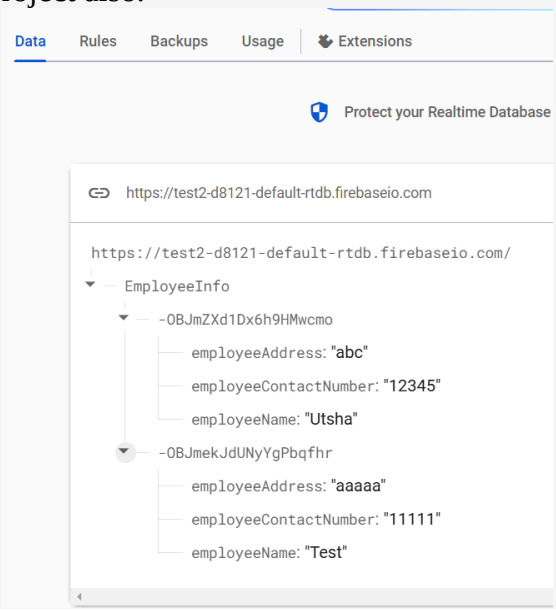
Step-9: `addDatatoFirebase` receives name, phone and address parameters. Set these values to your employeeInfo class object.

```
employeeInfo.setEmployeeName(name);
employeeInfo.setEmployeeContactNumber(phone);
employeeInfo.setEmployeeAddress(address);
```

Now push the object to your databaseReference object, clear the editText fields and show Toast message.

```
databaseReference.push().setValue(employeeInfo)
        .addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
                Toast.makeText(MainActivity.this, "Data added",
Toast.LENGTH_SHORT).show();
                employeeNameEdt.setText("");
                employeePhoneEdt.setText("");
                employeeAddressEdt.setText("");
            } else {
                Toast.makeText(MainActivity.this, "Failed to add data: " +
task.getException(), Toast.LENGTH_SHORT).show();
            }
        });
```

You can now run the app and try inserting some data. You can check if the data is properly inserted in your Firebase project also.



CSE 2200: Advanced Programming
Lab Tutorial: Firebase Realtime Database

Retrieve Data

Now you need to retrieve data from Firebase Realtime Database and displays it in a **RecyclerView.** RecyclerView is used in Android for displaying large data sets efficiently with smooth scrolling.

Step-1: Create a new empty activity class **'ReadActivity'** along with its layout. Add a new Button **'Display Details'** in your **layout_main.xml** file. Then, add the button click listener method in your **MainActivity.java** class and inside the **onClick** method: Change the intent to the new **ReadActivity** class you just created.

Step-2: Create a new XML file in res/layout called **employee_item.xml** to define how each employee item will appear. Update your **activity_main.xml** to include the **RecyclerView**. For now, you can use the xml files provided.

Step-3: Go to app > java > your project package > Create a new Java class **EmployeeAdapter.java** to **bind** the data to the **RecyclerView**. Now do the followings steps.

- Inside the **EmployeeAdapter** class create a class **EmployeeViewHolder** class which will inherit the **RecyclerView.ViewHolder** class. The **ViewHolder** class in **RecyclerView** is used to improve performance by reducing the number of findViewById calls. ViewHolder optimizes RecyclerView by caching item views and making scrolling smoother.

```
public static class EmployeeViewHolder extends RecyclerView.ViewHolder
{
    TextView employeeName, employeeContact, employeeAddress;

    public EmployeeViewHolder(View itemView) {
        super(itemView);
        employeeName = itemView.findViewById(R.id.tvEmployeeName);
        employeeContact = itemView.findViewById(R.id.tvEmployeePhone);
        employeeAddress =
itemView.findViewById(R.id.tvEmployeeAddress);
    }
}
```

  Inside this class, connect all the IDs of the textView items from **employee_item.xml** file.
- **EmployeeAdapter** will have to inherit RecyclerView.Adapter<EmployeeAdapter.EmployeeViewHolder> class so that the adapter will update the RecyclerView with each items from the database.
- Click **Alt+Enter** and select **'implement methods'**.
- Create the employee list as the attribute of the class. Generate constructor.

```
private List<EmployeeInfo> employeeList;

public EmployeeAdapter(List<EmployeeInfo> employeeList) {
    this.employeeList = employeeList;
}
```

- Now change the bodies of all the methods implemented according to the list.
  getItemCount() function

```
public int getItemCount() {
    return employeeList.size();
}
```

- **onCreateViewHolder** function is responsible for creating a new ViewHolder when needed. Here's a breakdown:
  - **Inflate Layout**: LayoutInflater.from(parent.getContext()).inflate(...) creates a View object from the employee_item XML layout file.
  - **Return ViewHolder**: The method then wraps this View in a new EmployeeViewHolder instance and returns it, ready to be used for displaying data.

  In short, onCreateViewHolder prepares and provides a ViewHolder with a layout for displaying each item in the list.

```
public EmployeeViewHolder onCreateViewHolder(@NonNull ViewGroup parent,
int viewType) {
    View view =
LayoutInflater.from(parent.getContext()).inflate(R.layout.employee_item
, parent, false);
    return new EmployeeViewHolder(view);
}
```

- **onBindViewHolder** populates the ViewHolder with data for the current item in the list.

```
public void onBindViewHolder(@NonNull EmployeeViewHolder holder, int
position) {
    EmployeeInfo employee = employeeList.get(position);
    holder.employeeName.setText(employee.getEmployeeName());

holder.employeeContact.setText(employee.getEmployeeContactNumber());
    holder.employeeAddress.setText(employee.getEmployeeAddress());
}
```

Step-4: Now you need to fetch data from Firebase Realtime Database. In your Read activity, retrieve the data from Firebase and populate it into the RecyclerView. Create objects for **RecyclerView, EmployeeAdapter, List.**

```
private RecyclerView recyclerView;
private EmployeeAdapter employeeAdapter;
private List<EmployeeInfo> employeeList;
DatabaseReference databaseReference;
```

CSE 2200: Advanced Programming
Lab Tutorial: Firebase Realtime Database

Inside **onCreate** function: initialize the objects and connect IDs.

```
recyclerView = findViewById(R.id.recyclerView);
recyclerView.setLayoutManager(new LinearLayoutManager(this));
employeeList = new ArrayList<>();
employeeAdapter = new EmployeeAdapter(employeeList);
recyclerView.setAdapter(employeeAdapter);
```

Fetch data from firebase:

```
databaseReference =
FirebaseDatabase.getInstance().getReference("EmployeeInfo");
```

Add **Event listener** to the **databaseReference** object.

```
databaseReference.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {

    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {

    }
});
```

Inside **onDataChange** function:

```
public void onDataChange(@NonNull DataSnapshot snapshot) {
    employeeList.clear();
    for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
        EmployeeInfo employee = dataSnapshot.getValue(EmployeeInfo.class);
        employeeList.add(employee);
    }
    employeeAdapter.notifyDataSetChanged();
}
```

The code **clears** the **employeeList** to remove any existing data. Then **loops** through the **DataSnapshot** (**which contains data from a database, like Firebase**). For each child in the snapshot, it converts the data into an EmployeeInfo object and adds it to the list. After updating the list, it calls **notifyDataSetChanged()** on the employeeAdapter, which **refreshes** the RecyclerView to display the updated data.

Inside **onCancelled** function: You can implement a toast.

Finally, the data stored in Firebase can be visualized once you run the app.

CSE 2200: Advanced Programming
Lab Tutorial: Firebase Realtime Database

Update and Delete Data (Home Work)

- In your employee_item.xml file for each entry add two more buttons **'Update'** and **'Delete'**.
- Make necessary changes in your **EmployeeAdapter.java** and **ReadActivity.java** class to implement the functionalities for update and delete.
- Design an **Alert Dialog Box** so that when the **Update** button is pressed, an alert dialog box will pop up and ask user for the new attributes for any rows needed to be updated.
- Bring the complete project in the next sessional class to demonstrate.

Thank you. 😊