Lab on apps development for tablets, smartphones and smartwatches

Week 6: Firebase and Lists

Giovanni Ansaloni

Dimitra Tatli, Riselda Kodra, Yuxuan Wang
Qunyou Liu, Amirhossein Shahbazinia, Christodoulos Kechris

School of Engineering (STI) – Institute of Electrical and Micro Engineering (IEM)

Mid-term

- **Nov 18th, 14.15**.
  - Be here 10 minutes early

- We will provide a map with individual seat assignments

- Do you need a desktop PC?

1. Fill this googleForm: https://forms.gle/qsR7HQcJ9RQforp76 before **Tue Nov 4th**
2. Set up your desktop on **Tue 11th** during class hours

- If you will instead use your laptop

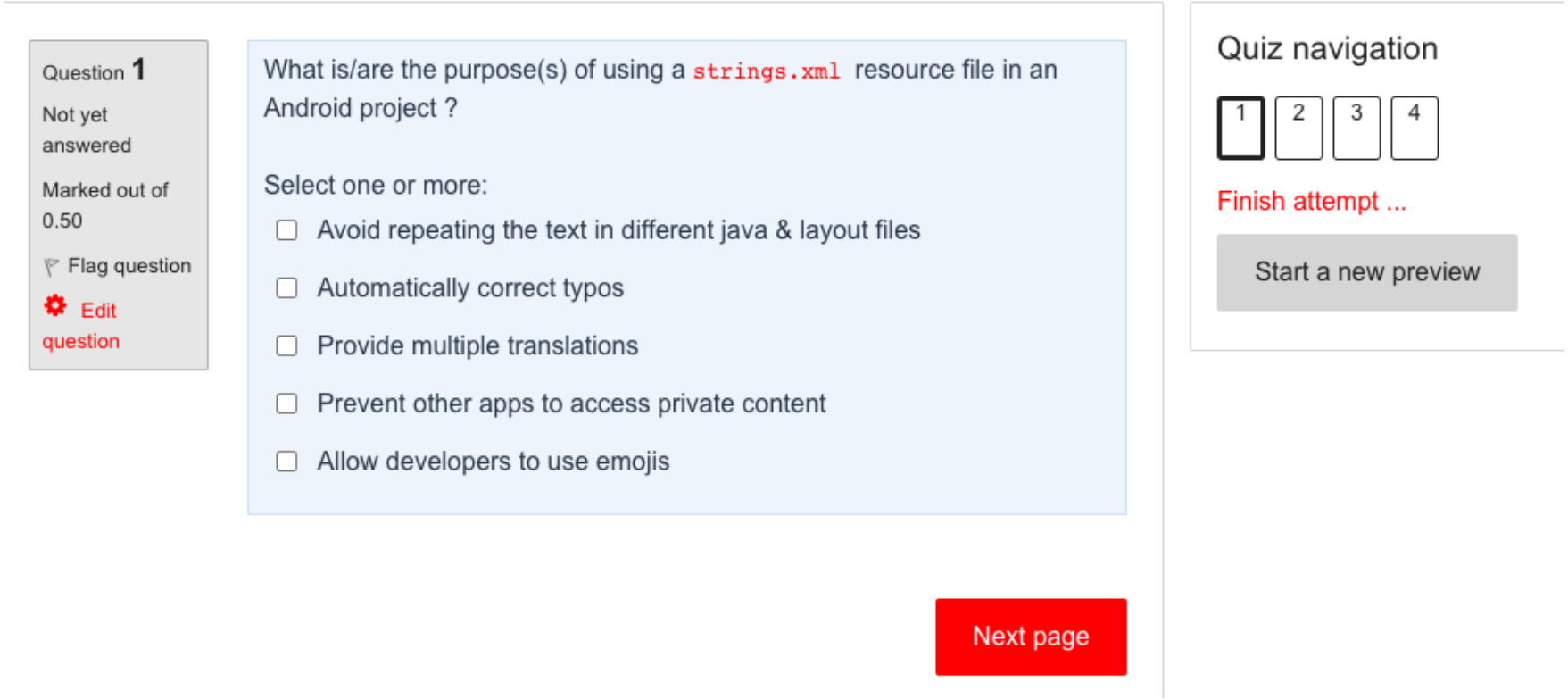
1. Do not fill in the googleForm

Mid-term

- **Nov 18st, 14.15**.
- **35%** of the grade
- 90 minutes duration
  - 5 minutes extra to upload your mini-project solution on Moodle
- 2 parts
  - mini-project to be completed
  - multiple-choices questions
- Material allowed
  - Moodle page of the course (lecture slides, labs, etc.)
  - Android developer website
  - Printed version of lecture slides and labs handouts
  - Your notes

Part1: Questions on Moodle

- ~4 questions
- one attempt



Part2: Mini-app

- You will be given a draft of an App project (as we do with labs)
- You will be tasked to implement some features

- Examples:

1. App crashes, fix the issue explaining how you did it
2. Something should be performed in response to a button press
3. …

- At the end of each task, you will call us to verify that the functionality is working
  - When you raise your hand, the emulator must be already started and the app launched

Class outline

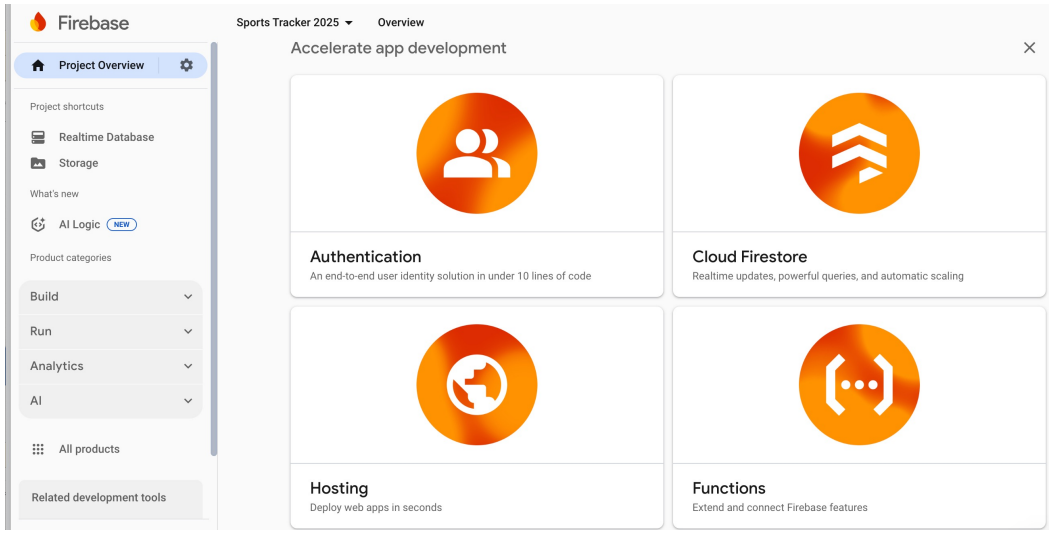
- **Firebase**
  - Realtime database
  - Cloud storage
- Lists
  - Lazy Layouts

Firebase Realtime Database

Cloud Storage
for Firebase

What is Firebase?

- Firebase is a platform that provides tools to help you
  - beta testing
  - run transactions/ads
  - store persistent data
  - …



- We will show how to use it to *sync data to the cloud*
  1. Connect your app to Firebase
  2. Enable Firebase features
  3. Add code to your app to interface Realtime and Storage databases

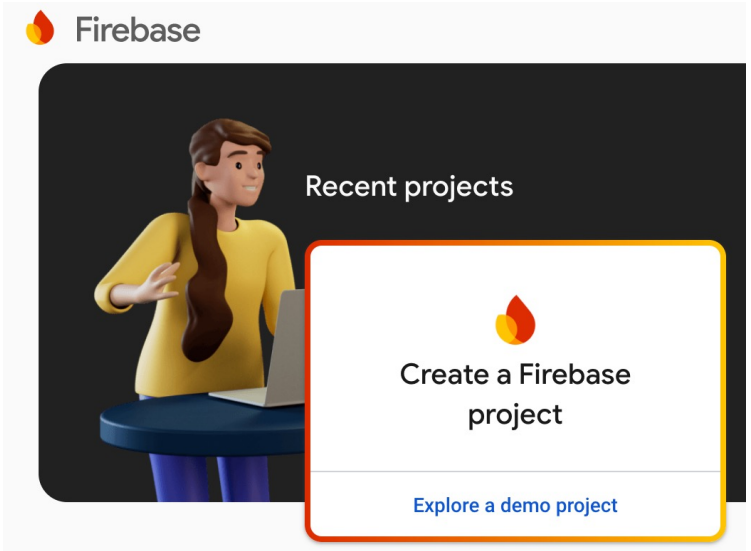
- Many other cloud features provided by Firebase
  - Authentication
  - ML deployment

  - Analytics

  - …

Firebase console

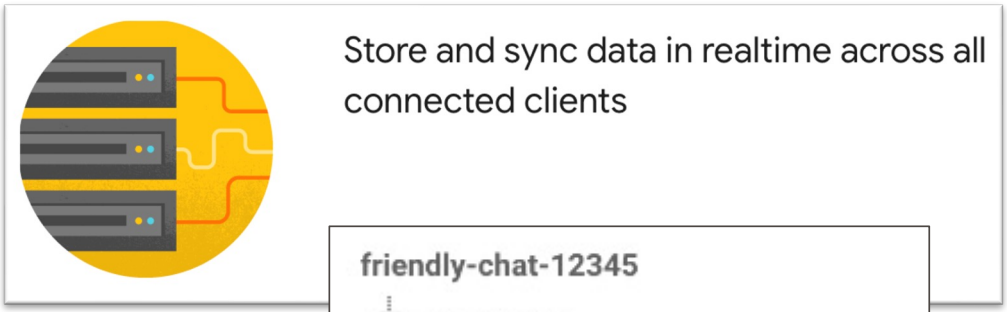
- Go to console.firebase.google.com
  - The console allows you to create new projects
  - Firebase creates a config file for your app
    - To be added to your Android Studio project
- All this happens automatically through Android Studio
  - We will teach you how to do it in the lab
- Firebase console requires to specify access rules
  - e.g. for Real-time database
    - Visit firebase.google.com/docs/database/security to learn more about security rules




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

Using Firebase Database / Storage

- Your app is connected with Firebase using the GUI-based Firebase assistant from Android Studio

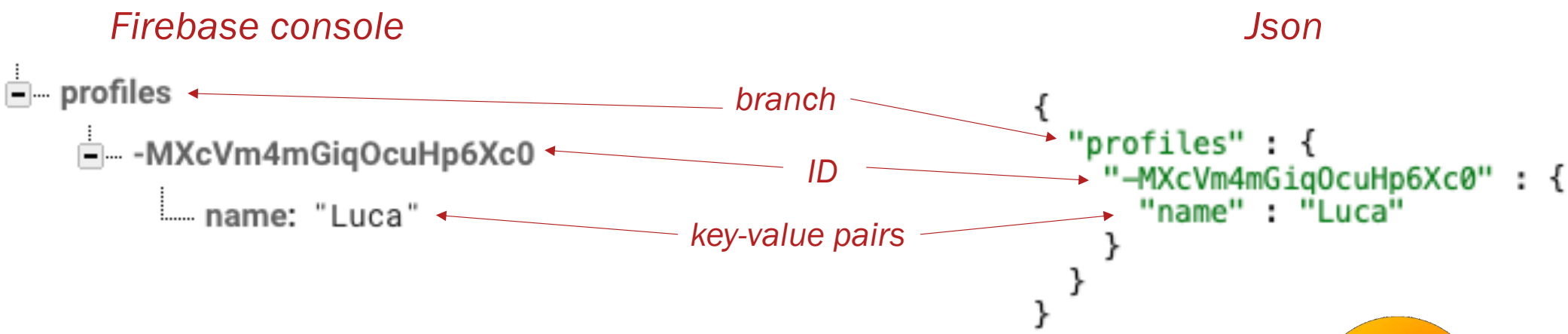


In the lab:

- Realtime Database for profiles
  - data synchronization with listeners
  - key-value database
  - data is synced across all clients, remains available when app goes offline
- Cloud Storage for pictures
  - shared cloud folder

Interfacing with Realtime database

- In Data is stored in Json trees
  - branches
  - key-value pairs



- RealTime Database is a NoSQL database
- no fixed fields
  - easily extensible

Interfacing with Realtime database

In app code:

- reference the database

```
val  database: FirebaseDatabase = FirebaseDatabase.getInstance()
```

- reference a branch of the database

```
val profileRef: DatabaseReference = database.getReference("profiles")
```

- create a new child by providing a unique key via push()

```
profileRef.push()
```

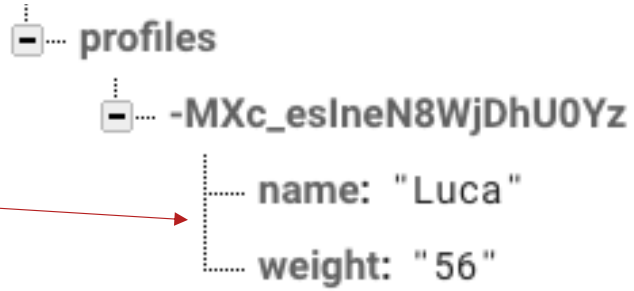
Writing to Realtime database

- The reference is used to create new entries
  - unique key provided via push()
  - navigate the json tree with child()
  - setValue()

```
val key = database.push().key
if (key != null) {
    database.child(key).child("name").setValue("Luca")
}
```

- Data classes can also be passed to setValue()

```
val key = database.push().key
if (key != null) {
    val user = User("Luca", "56")
    database.child(key).setValue(user)
}
```



Reading from Realtime database

- Attach ValueEventListener on the database Reference
  - onDataChanged() callback
    → takes a snapshot of the database when data on cloud changes
  - getValue() to read data from cloud

listener → *take snapshot* → *navigate json tree* → *get data*

```
database.addValueEventListener(object : ValueEventListener {

    override fun onDataChange(dataSnapshot: DataSnapshot) {
      for (thisUser in dataSnapshot.children) {
        usernameDatabase = thisUser.child("username")
                                   .getValue(String::class.java)
        username = usernameDatabase ?: “NULL“
      }
    }

    override fun onCancelled(error: DatabaseError) {
       /* Failed to read value */
    }
})
```

An aside: Realtime database transactions

- What about multiple concurrent accesses?
  - use transactions → abort and retry if state is modified during access

firebase.google.com/docs/database/android/read-and-write#save_data_as_transactions

```
databaseRef.runTransaction(object : Transaction.Handler {

    override fun doTransaction(mutableData: MutableData): Transaction.Result {
        val p = mutableData.getValue(String::class.java)
                ?: return Transaction.success(mutableData)

        ... // do something with “p”
        mutableData.value = p
        return Transaction.success(mutableData)
    }

    override fun onComplete(...) {
        // any error?
        Log.d(TAG, "postTransaction:onComplete:" + databaseError!!)
    }
})
```

Another aside: Firestore

- Realtime database 
  - NoSQL database
  - Data is stored as Json tree
  - No support for complex hierarchies
  - Deep queries
    - performance degrades as data set grows
  - Good for
    - synchronizing data up to few GB
    - basic querying

- Firestore 
  - NoSQL database
  - Data is stored as collection of documents
  - Supports sub-collections
  - Indexed (shallow) queries
    - performance is proportional to number of results
  - Good for
    - TBs of data
    - complex querying

Detailed documentation: https://firebase.google.com/docs/database/rtdb-vs-firestore

Interfacing with Cloud storage

Similar mechanism wrt Realtime Database:

- setup Firebase cloud storage with the Firebase assistant
  - Tools → Firebase
  - Requires Pay-as-you go plan
    - billing credential, but no cost if bucket set in **us-central1**, **us-west1** or **us-east1** zones (see firebase.google.com/pricing)
- reference the storage in your app

```
var storageRef = Firebase.storage.reference
```

Writing to Cloud storage

- Create a reference to the location where you want to store data

```
val mountainsRef = storageRef.child("mountains.jpg")
```

- now you can upload the data (e.g. image)

```
val bitmap = (binding.mountainImage.drawable as BitmapDrawable).bitmap
val baos = ByteArrayOutputStream()
bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
val data = baos.toByteArray()

var uploadTask = mountainsRef.putBytes(data)
```

- ... and listen for successful / unsuccessful uploads

```
uploadTask.addOnFailureListener {
    // Handle unsuccessful uploads
}.addOnSuccessListener { taskSnapshot ->
    // taskSnapshot.metadata contains size, content-type, etc.
}
```

Reading from Cloud storage

- Create a reference to the location where you want to read from

```
var islandRef = storageRef.child("images/island.jpg")
```

- Read byte stream from cloud storage, convert it to appropriate format

```
islandRef.getBytes(ONE_MEGABYTE)
    .addOnSuccessListener { byteArray ->
       val image: Drawable = BitmapDrawable(resources,
                             BitmapFactory.decodeByteArray(
                                          byteArray,
                                         0,byteArray.size)
                                        )
    }
    .addOnFailureListener {
    /* Handle any errors */
    }
```

Class outline

- Firebase
  - Realtime database
  - Cloud storage
- **Lists**
  - Lazy Layouts

Firebase Realtime Database

Cloud Storage
for Firebase

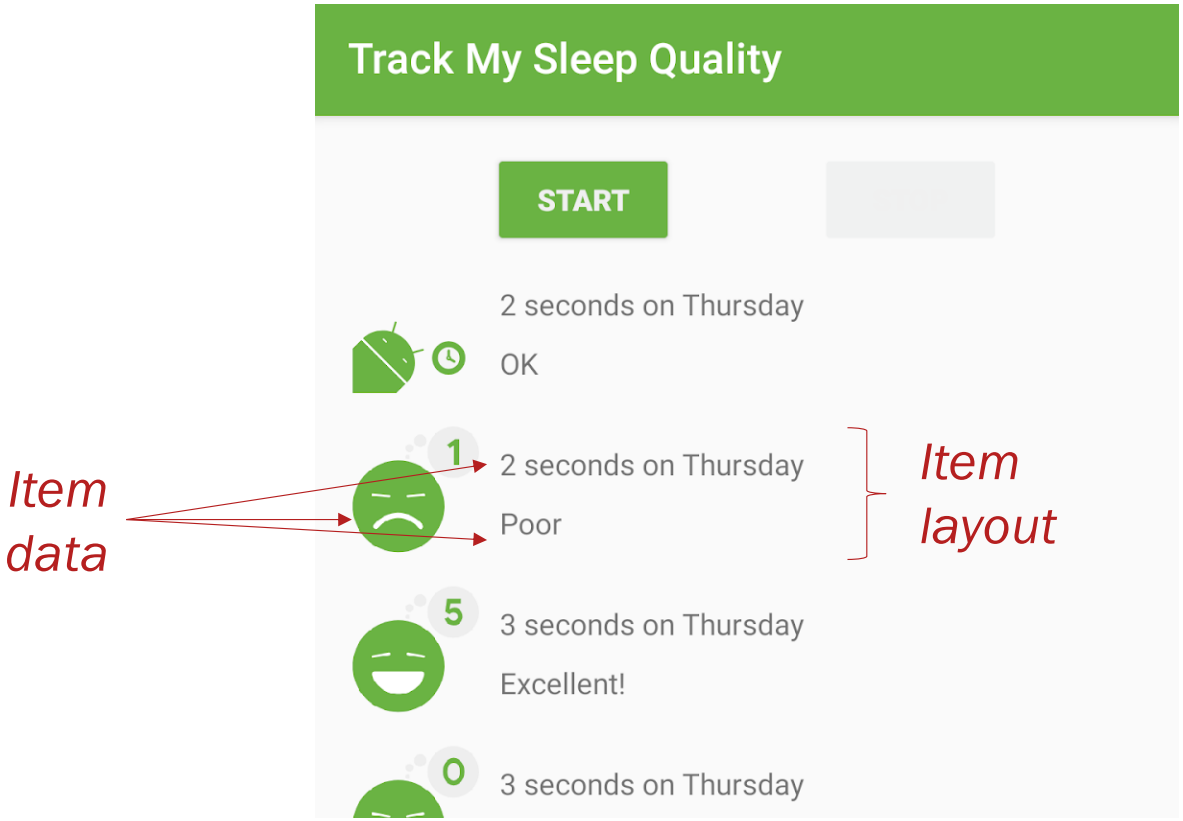
Scrollable Lists

- LazyColumn, LazyRow and Lazy grids are composable that displays scrollable lists
  - items can contain headers, icons, etc…
- only visible items are processed
  - when an item scrolls off-screen, the corresponding composable is recycled → memory efficient
  - if an item changes, Lazy Layouts updates only *that one item*

Lazy Layouts

Lazy Layout require:

- layouts of items
- data to display



Lazy Layouts

1. Use Lazy composable
2. Inside the Lazy composable block, provide the list of data items
3. Implement the item layout as a composable inside the *items* lambda

data to be displayed

```
@Composable
fun MessageList(messages: List<Message>)
{
   LazyColumn {
      items(messages) { message ->
         MessageRow(message)
      }
   }
}

@Composable
fun MessageRow(message: Message) {
   ...
}
```

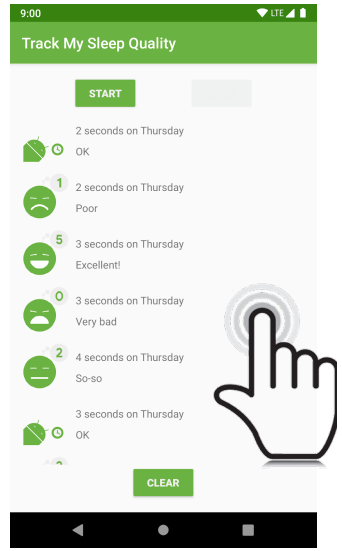
Item composable

Clickable items in Lazy Layouts

- Items can react to events

  → State Hoisting from *items* to Lazy layout

  - passing the ID of the clicked item




```
@Composable
fun MessageRow(
    message: Message,
    onItemClicked: (Message) -> Unit
) {
    Row(
        Modifier
            .fillMaxWidth()
            .clickable { onItemClicked(message) }) {
        ...
    }
}
```

```
LazyColumn {
    items(messages) { message ->
        MessageRow(
            message,
            onItemClicked = { clickedMessage ->
                Toast.makeText(
                    context,
                    "Clicked message ID: ${clickedMessage.id}",
                    Toast.LENGTH_LONG
                ).show()
            })
    }
}
```

LazyListState

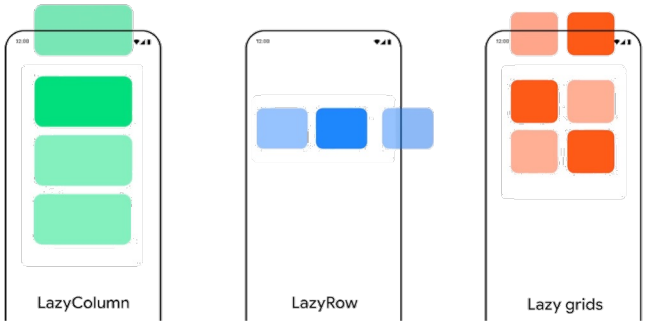
- LazyListState incapsulates the state of a LazyList
  - provided to the LazyColumn/Row/Grid as a parameter

```
val listState = rememberLazyListState()
LazyColumn(state = listState) {
    ...
}
```

- Provides information about the layout

```
if (listState.firstVisibleItemIndex == 0){
    ... //Do something if list is scrolled up
}
```

- Allows to interact with it from elsewhere in the app
  - next slide



LazyListState

- Example: scroll to the top of the list when a button is clicked
  - coroutineScope to not block the UI

1. Initiatiate a LazyListState with rememberLazyListState
2. Pass it to the LazyColumn as a parameter
3. call LazyListState methods

```
val coroutineScope =   rememberCoroutineScope()
val listState = rememberLazyListState()

LazyColumn(state = listState) {
   items(messages) { message ->
      // ...
   }
}

Button(
   onClick = {
      coroutineScope.launch {
         listState.animateScrollToItem(index = 0)
      }
   }
) {
   Text("Scroll to top")
}
```

Today's Lab

- Use Firebase to store / retrieve
  - user Profiles
    - usernames, passwords(!)[1], pictures
  - exercise session
- Show information on exercise sessions
  - session information is stored in the cloud
  - it is retrieved from Firebase ...
  - ...and displayed in a Lazy Layout

Firebase Realtime Database

Cloud Storage for Firebase

[1] Secure authentication using Firebase:
https://firebase.google.com/docs/auth/android/firebaseui#kotlin+ktx

Questions?