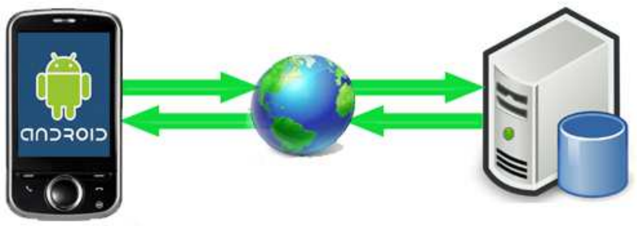
19. Remote Databases and Firebase

Where is the data?

- A database can be located in many places.
  - within your Android device (a "local database")
  - on a remote web server
  - spread throughout many remote servers ("in the cloud")
- Today we will learn to create and use **remote databases**.



Setting up remote database

- A remote database is hosted on a **web server**. Server is often called the app's "**back-end**".
- One option: Do-it-yourself
  - buy web **hosting** (e.g. DreamHost; GoDaddy)
  - use their tools to create/add a **database**
    DreamHost: MySQL (create using web panel)
  - populate the database
    import .sql file, etc.
  - set up **permissions** and authentication
    create user account(s), passwords
  - modify your **app** to connect to remote database
    use JDBC (Java Database Connectivity) connector

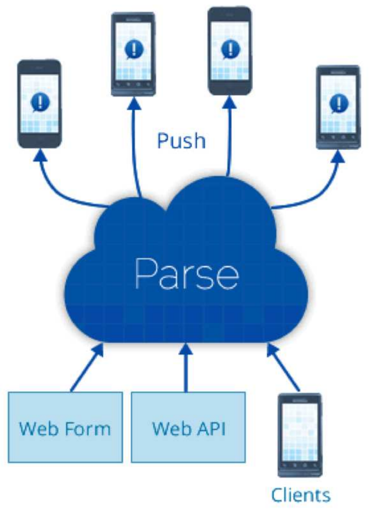
Problems with remote server

- While running your own remote database can work well, it also has potential drawbacks:
  - **cost:** have to pay to get hosting from DreamHost etc.
  - **administration:** must set up/maintain database, server yourself
  - **security:** attacker with password can connect to database
  - **privacy:** must block user from seeing other users' info
  - **robustness:** database isn't automatically backed up, protected
  - **scaling:** too many users querying server will slow it down
  - ...

BaaS web platforms

- **BaaS** (Backend as a Service): Platforms for database/service hosting, management, deployment, etc.
  - examples: Parse (RIP), **Firebase**, Google App Engine, Amazon Web Services Mobile, Azure, Kinvey, Kumulos, Backendless, ...
- Features of BaaS platforms:
  - web UI for creating accounts, databases, users, etc. as needed
  - API of classes and objects to query the data in many platforms

    web app, Android, iOS, ...
  - saves the developer from having to buy and manage servers/DBs
  - often built to scale up to very large sizes / traffic loads if needed
  - many BaaS platforms do not explicitly use SQL and instead have the user perform queries using various methods and parameters



NoSQL databases

- **NoSQL database:** Does not store data into tables and does not use SQL.
  - became popular in mid-2000s
  - *benefits:* simplicity; flexibility; "horizontal" scalability to many servers
  - *drawbacks:* less standardized; data inconsistency/loss; lack "ACID"

- Types of NoSQL databases
  - column stores (**Cassandra**, Vertica, Druid, Accumulo)
  - document stores (**MongoDB**, CouchDB, Qizx, MarkLogic, Hyperdex)
  - key/value stores (Memcached, Scalaris, Voldemort, Dynamo)
  - data structure servers (**Redis**)
  - graph stores (Allegro, Neo4J, Virtuoso, MarkLogic)

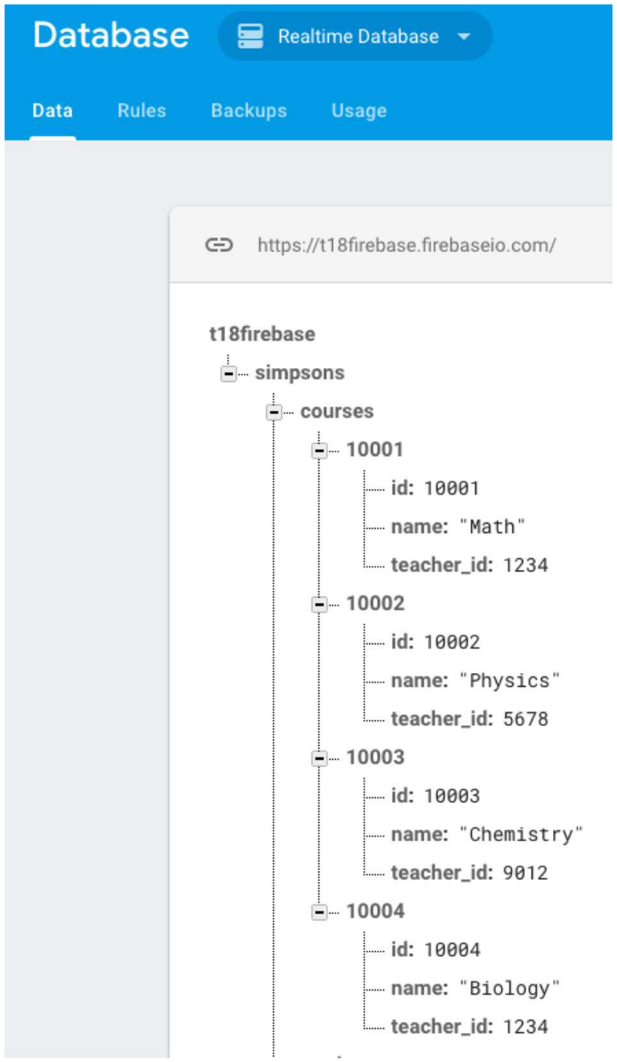
Firebase

- **Firebase:** BaaS / remote database management platform built by SF-based Google subsidiary.
  - a "real-time synchronized cloud database"
  - one of the strongest successors / replacements for now-dead **Parse**

- Key features
  - **API** to access data from Android, iOS, Java, JavaScript, Obj-C, Node.js
  - **REST API** with libraries for many common web JS frameworks
  - ability to keep data in **sync**, receive **notifications** on data changes
  - cloud **scaling**, can handle tons of requests if needed
  - other features: web hosting, login/**auth**, ...

How Firebase stores data

- The database is a giant nested **map** of string keys to values.
  - text, numbers, boolean, lists, or maps
  - object: {"key" => value} map
  - list: {index => value} map
  - overall database is tree-like map structure you can view on the web

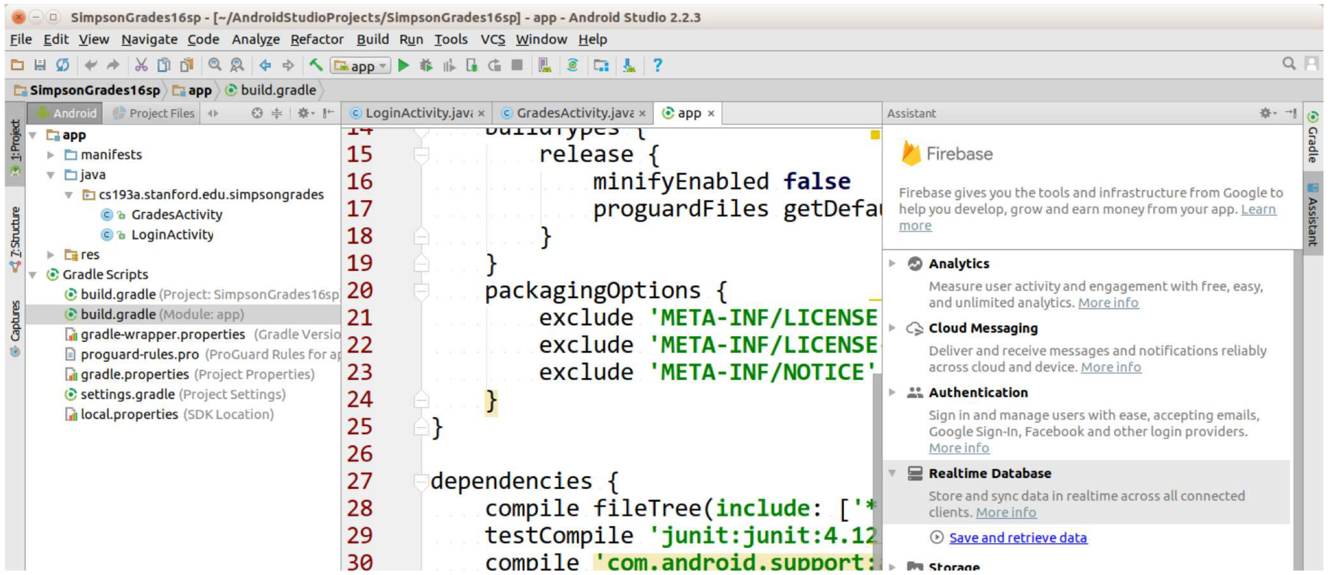


https://t18firebase.firebaseio.com/

```
t18firebase
 simpsons
  courses
   10001
    id: 10001
    name: "Math"
    teacher_id: 1234
   10002
    id: 10002
    name: "Physics"
    teacher_id: 5678
   10003
    id: 10003
    name: "Chemistry"
    teacher_id: 9012
   10004
    id: 10004
    name: "Biology"
    teacher_id: 1234
```

Set up Firebase

- **sign up** for free user account
  - https://firebase.google.com/login/
- **install** Firebase into Android Studio project
  - option 1: use UI: Tools → FireBase → Realtime Database → ...
  - option 2: do it yourself (next slide)



SimpsonGrades16sp - [~/AndroidStudioProjects/SimpsonGrades16sp] - app - Android Studio 2.2.3

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
app
SimpsonGrades16sp › app › build.gradle
Android Project Files
LoginActivity.java × GradesActivity.java × app ×
1: Project
7: Structure
Captures
Gradle
Assistant

```
app
  manifests
  java
    cs193a.stanford.edu.simpsongrades
      GradesActivity
      LoginActivity
  res
Gradle Scripts
  build.gradle (Project: SimpsonGrades16sp
  build.gradle (Module: app)
  gradle-wrapper.properties (Gradle Versio
  proguard-rules.pro (ProGuard Rules for ap
  gradle.properties (Project Properties)
  settings.gradle (Project Settings)
  local.properties (SDK Location)
```

```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefau
        }
    }
    packagingOptions {
        exclude 'META-INF/LICENSE
        exclude 'META-INF/LICENSE
        exclude 'META-INF/NOTICE'
    }
}

dependencies {
    compile fileTree(include: ['*
    testCompile 'junit:junit:4.12
    compile 'com.android.support:
```

Assistant

Firebase

Firebase gives you the tools and infrastructure from Google to help you develop, grow and earn money from your app. Learn more

Analytics
Measure user activity and engagement with free, easy, and unlimited analytics. More info

Cloud Messaging
Deliver and receive messages and notifications reliably across cloud and device. More info

Authentication
Sign in and manage users with ease, accepting emails, Google Sign-In, Facebook and other login providers. More info

Realtime Database
Store and sync data in realtime across all connected clients. More info

Save and retrieve data

Storage

DIY Firebase setup

1. modify **app build.gradle**

```
dependencies {
  api  "com.google.android.gms:play-services-base:16.1.0"

  implementation 'com.google.firebase:firebase-core:16.0.7’
  //implementation 'com.google.firebase:firebase-auth:16.1.0'
  implementation 'com.google.firebase:firebase-database:16.0.6'}

  apply plugin: 'com.google.gms.google-services'
```

→ Change SDK 26.1.0

DIY Firebase setup

2. modify **module build.gradle**

```
buildscript {
...
}

dependencies {
        ...
        classpath 'com.google.gms:google-services:4.2.0'
   }
```

3. get **google-services.json** file from Google Firebase web site

- Sign in to Firebase; open project; Click Project settings; In Your apps card, select package name of app; Click google-services.json.

Writing Firebase data

```
// create a key/value pairing

DatabaseReference fb = FirebaseDatabase.getInstance().getReference();
fb.child("name").setValue(value);
```

- Firebase stores data as key/value pairs
    - the keys are strings representing data object names
    - the values can be one of many types:
    - Boolean, Long, Double, List, Map<String, Object>
    - think of Firebase as a HashMap on steroids in the cloud

Database Reference methods

Method	Description
dbr.**child**("*name*")	return child object with given name (creates if missing)
dbr.getKey()	return key for a given data value
dbr.getParent()	return data one level up in the map
dbr.getRoot()	return data at top of map
dbr.push()	create/return an auto-created new child

dbr.**removeValue**();
dbr.removeValue(*handler*);
	delete value associated with this key

dbr.runTransaction(*handler*);	run multiple queries in sequence
dbr.setPriority(*priority*);	gives this data a 'priority' rating for sorting

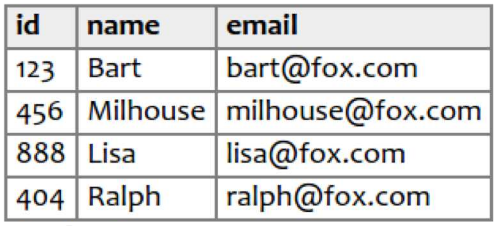
dbr.**setValue**(*value*);
dbr.setValue(*value*, *handler*);
dbr.setValue(*value*, *priority*, *handler*);
	sets new data value, with optional listener to be notified when sync is complete

dbr.updateChildren(*map*);
dbr.updateChildren(*map*, *handler*);
	updates some of object's fields ("children") using the key/value data in the given map

SQL → Firebase mapping

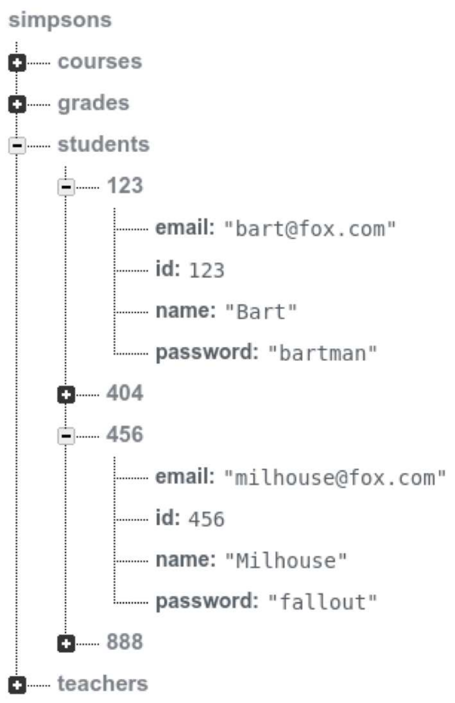
- Recall the **simpsons** database's **students**table.
  - Firebase key/value data might look like this:

```
DatabaseReference fb = FirebaseDatabase
                    .getInstance().getReference();
DatabaseReference table = fb.child("simpsons/students");

DatabaseReference bart = table.child("123");
bart.child("id").setValue(123);
bart.child("name").setValue("Bart");
bart.child("email").setValue("bart@fox.com");

DatabaseReference milhouse = table.child("456");
milhouse.child("id").setValue(456);
milhouse.child("name").setValue("Milhouse");
milhouse.child("email").setValue("milhouse@fox.com");
```



id	name	email
123	Bart	bart@fox.com
456	Milhouse	milhouse@fox.com
888	Lisa	lisa@fox.com
404	Ralph	ralph@fox.com

students



```
simpsons
  + courses
  + grades
  - students
      - 123
          email: "bart@fox.com"
          id: 123
          name: "Bart"
          password: "bartman"
      + 404
      - 456
          email: "milhouse@fox.com"
          id: 456
          name: "Milhouse"
          password: "fallout"
      + 888
  + teachers
```

Set value with callback

- When you call setValue, the data may not update immediately.
  - Your data might be distributed across many servers; it takes time to sync them.
  - To be notified when the data is fully written:

```
DatabaseReference fb = ...;
DatabaseReference bart = fb.child("simpsons/students/123");
bart.child("name").setValue("Bart");

// or, if you want to be notified when it completes

bart.child("name").setValue("Bart",
    new DatabaseReference.CompletionListener() {
        public void onComplete(DatabaseError err, DatabaseReference ref) {
            if (err == null) { ... }
        }
    });
```

Retrieving data

- Must grab the Firebase object for that data, and **bind** an event handler to it.
- Will be **notified** initially and on state changes.

```
DatabaseReference fb = ...;
DatabaseReference bart = fb.child("simpsons/students/123");
bart.addValueEventListener(new ValueEventListener() {
     @Override
     public void onDataChange(DataSnapshot data) {
        // do something with data
        // using data.getKey() data.getValue()
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
    // report/log the error
    }
});
```

Auto-generated keys

- Some tables don't have an "**id**" column.
  - Firebase can make up IDs with push().
  - Also useful in highly parallel situations where many users modify the data at once.

```
DatabaseReference fb = ...;
DatabaseReference table = fb.child("simpsons/grades");
DatabaseReference newGrade = table.push();
newGrade.child("student_id").setValue(123);
newGrade.child("course_id").setValue(10001);
newGrade.child("grade").setValue("B-");
```

Save/load your own classes

- If you write your own Java classes, you can store their objects in Firebase as long as:
    - class has a no-params () constructor
    - every field has a get*FieldName*() method

```
public class Student {
    private int id;
    private String name;
    private String email;
    public Student() {}
    public Student(int id, String name, String email) { ... } //constructor
    public int getID() { return id; }
    public String getName() { return name; }
    public String getEmail() { return email; }
    }

    DatabaseReference fb = ...;
    DatabaseReference table = fb.child("simpsons/students");
    Student bart = new Student(123, "Bart", "bart@fox.com");
    table.child("123").setValue(bart);
```

DataSnapshot methods

Method	Description
ds.child("*path*")	returns child for given key
ds.exists()	true if this data value is non-null
ds.getChildren()	returns iterable list of children (use with for-each loop)
ds.getKey()	returns key used to fetch this data snapshot
ds.getPriority()	priority of this data's root node
ds.getRef()	returns reference to Firebase object
ds.getValue()	returns data associated with this snapshot's key
ds.getValue(*class*)	returns data, converted into the given class (must have a ()constructor and public get methods)
ds.hasChild("*path*")	true if the given child node/path exists in this data
ds.hasChildren()	true if this snapshot contains any data
ds.toString()	text representation of all the data

Types of data events

Method	Description
fb.addValueListener(*ValueEventListener*);	listen to changes in a data value
- onDataChange(*snapshot*)	
- onCancelled(*error*)	
fb.addListenerForSingleValueEvent(*ValueEventListener*);	get initial data and then stop
fb.addChildListener(*ChildEventListener*);	listen to changes to the children of a given data value
- onChildAdded(*snapshot*, *name*)	
- onChildChanged(*snapshot*, *oldName*)	
- onChildRemoved(*snapshot*, *oldName*)	
- onChildMoved(*snapshot*, *oldName*)	
- onCancelled(*error*)	

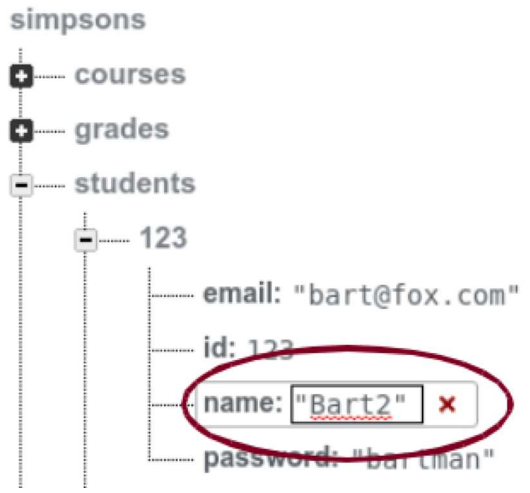
View data as a class

- examine the data value as an object rather than each piece separately

```
DatabaseReference fb = ...;
DatabaseReference bart = fb.child("simpsons/students/123");
bart.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot data) {
      Student stu = data.getValue(Student.class);
      Log.v("fb", stu.name + ", " + stu.email);
    }
    ...
});

public class Student {
      String email;
      int id;
      String name;
      String password;
      public Student() {}
}
```

Viewing changes to data

- When your app binds to a piece of data, it will be notified any time that data is changed from anywhere in the world.
  - This is extremely powerful!
  - Keep all users in sync on changes to an important piece of data in your db.
  - Can change the data from your app, another user's copy of the app, from the Firebase web console, ...



Querying data

- How do we do queries like we can in SQL?
  - Done using Query, ordering, ranges, etc.
  - Best illustrated by examples:

```
DatabaseReference fb = ...;
DatabaseReference students = fb.child("simpsons/students");

// SQL: SELECT * FROM students WHERE id >= 500;
Query query1 = students.orderByKey().startAt(500);
query1.addListenerForSingleValueEvent(
     new ValueEventListener() {...});

// SQL: SELECT * FROM students WHERE name LIKE "B%";
Query query2 = students.orderByChild("name")
                         .startAt("B").endAt("Bz");
query2.addListenerForSingleValueEvent(
                    new ValueEventListener() {...});
```

Querying data, more detail

```
DatabaseReference fb = ...;
DatabaseReference students = fb.child("simpsons/students");

// SQL: SELECT * FROM students WHERE id >= 500;
Query query1 = students.orderByKey().startAt(500);
query1.addListenerForSingleValueEvent(new ValueEventListener() {
    public void onDataChange(DataSnapshot dataSnapshot) {
      // print each k/v pair as a log message
      for (DataSnapshot student : dataSnapshot.getChildren()) {
           Log.v("fb", "student " + student.getKey() + " => "
                + student.getValue());
           }
      }
 });
```

Query methods

Method	Description	SQL
q.endAt(*value*) *q*.endAt(*value*, "*key*")	specify last value to include, or last value for a given key to include	<=
q.equalTo(*value*)*q*.equalTo(*value*, "*key*")	specify only value to include	=
q.limitToFirst(*count*)*q*.limitToLast(*count*)	only show first/last *N* results	LIMIT *N*
q.orderByChild("*name*")	sort/filter results by given child key	ORDER BY
q.orderByKey()	sort/filter results by their key	
q.orderByPriority()	sort/filter by priorities set manually	
q.orderByValue()	sort/filter by their own values	
q.startAt(*value*)*q*.startAt(*value*, "*key*")	specify last value to include, or last value for a given key to include	>=

Security and authentication

- here are several ways to allow/deny access to your database
  - Firebase-specific accounts; Google accounts; etc.
  - Use **Firebase web UI** to add email/password user accounts
  - Modify code to **sign in** with email and password:

```
private FirebaseAuth mAuth;
mAuth = FirebaseAuth.getInstance();
mAuth.signInWithEmailAndPassword("username", "password");
// optional: addOnCompleteListener, addAuthStateListener
```

FirebaseAuth methods

Method	Description
createUserWithEmailAndPassword("*email*", "*pw*")	create new account
signInWithEmailAndPassword("*email*", "*pw*")	log in a standard user
signInWithCredential(*auth*)	log in using access creds
signInAnonymously()	log in as anon. user
signInWithCustomToken("*token*")	log in with an auth token
signOut()	disconnect
getCurrentUser()	return active user account