CS 193A

Remote Databases; 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

JDBC

- **JDBC** (Java Database Connectivity): API for connecting to remote databases in Java code
  - part of the official Java spec since JDK 1.1

- to use JDBC:
  - most of the classes / APIs come with Java JDK
  - but a key component is missing and must be downloaded:
    - **JDBC driver ("connector")** for your particular kind of database
    - (e.g. MySQL, SQLite, Oracle, Microsoft SQL Server, ...)

MySQL connector

- **MySQL**: popular open-source database engine
  - an alternative to SQLite
  - uses the same SQL query language

- Getting a MySQL connector:
  - add this line to `build.gradle` dependencies area:

```
compile 'mysql:mysql-connector-java:5.1.38'
```

  - alternative:
    - https://dev.mysql.com/downloads/connector/j/
    - download connector JAR
    - put into Android Studio project in app/libs/

Querying a database with JDBC

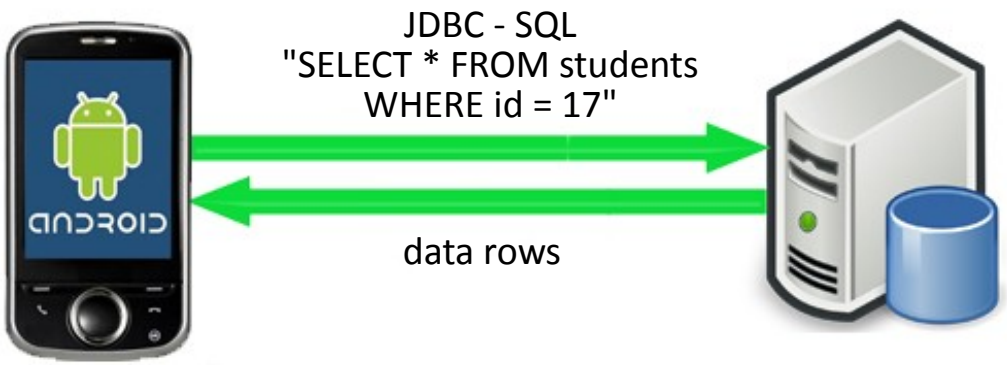
```
// connect to remote MySQL server using JDBC and run a query
try {
    // load JDBC driver and connect to server
    Class.forName("com.mysql.jdbc.Driver");
    String url = "jdbc:mysql://server:port/databaseName";
    String user = "username";
    String pass = "password";
    Connection con = DriverManager.getConnection(url, user, pass);

    // run the query
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("query");   // <-- SQL here!
    while (rs.next()) {
        String result = "";
        int col1 = rs.getInt("columnName");
        String col2 = rs.getString("columnName");
        ...
    }
    rs.close();
} catch (Exception e) {
    Log.wtf("sql", e);
}
```

Problems with remote databases

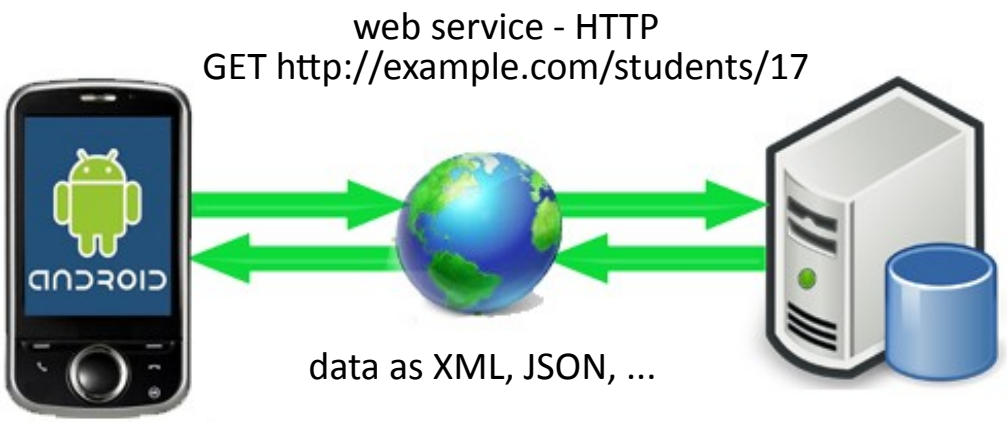
- While running your own remote database can work well, it also has potential drawbacks:
  - Cost: have to pay to get hosting from DreamHost etc.
  - Administration: Have to set up and take care of the database, server, etc. yourself
  - Security: If you aren't careful, anyone with password can connect to your database!
  - Privacy: ...
  - Robustness: Database isn't automatically backed up, protected
  - Scaling: Too many users querying the same server will slow it down
  - ...

Web services to databases

- JDBC as shown connects directly to a remote database.



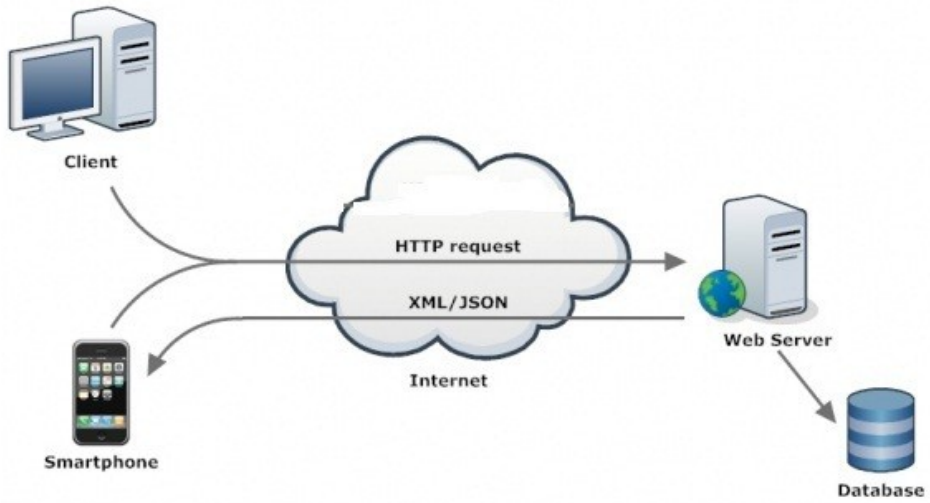
- Many apps instead write a web layer between app and db.
  - Client makes queries by contacting certain specific URLs.
  - Server sends the appropriate database data back.



Web Services

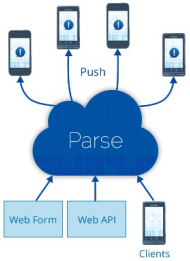
- **web service**: a set of functionality offered over a server using the web, but not web pages / HTML
  - Use the web's HTTP protocol to connect and transfer data.
  - Client connects to specific URLs to request specific data, which is then sent back in some documented format such as XML or JSON.
  - **REST**: Representational State Transfer. Common style of web services.
    - "RESTful web services" or "RESTful APIs"

- Web services are a bit like remote function calls where you can request data via URLs with parameters and get the data returned as a response.



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:
  - 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**: One that does not store data into tables as is done in a standard relational database, and does not use SQL.
  - became popular in early 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, Oracle NoSQL, Voldemort, Dynamo)
  - data structure servers (**Redis**)
  - graph stores (Allegro, Neo4J, Virtuoso, MarkLogic)

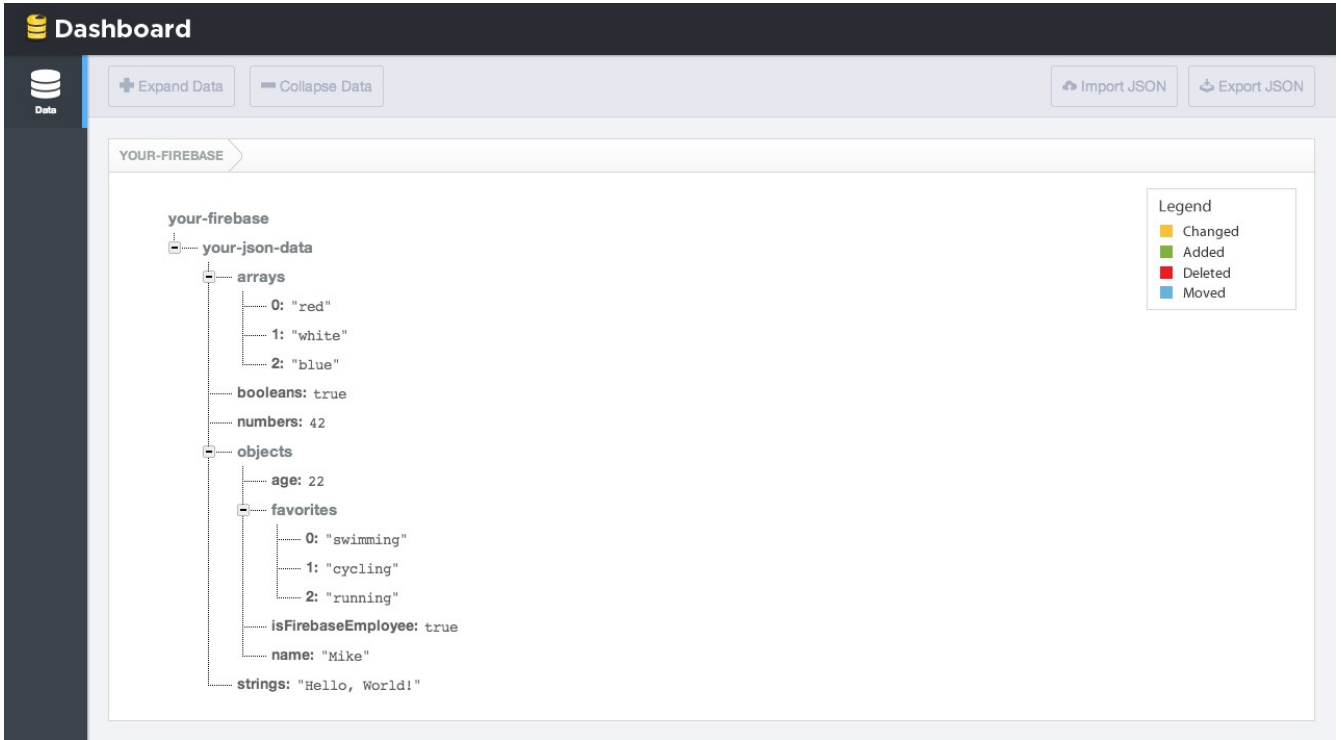
Firebase

- **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, ...

The way Firebase stores data

- The database itself is a giant ***map*** of string keys to values.
  - Values can be text, numbers, boolean, lists, or maps.
  - An object can be thought of as a map from {field name => value}.
  - A list can be thought of as a map from {int index => value}.
  - Overall database is tree-like map structure you can view on the web.



Setting up Firebase

- sign up for free user account

  https://www.firebase.com/login/

- install Firebase into Android Studio project's build.gradle

```
dependencies {
    compile 'com.firebase:firebase-client-android:2.5.2+'
}
```

- add some file exclusions in build.gradle

```
android {
    ...
    packagingOptions {
        exclude 'META-INF/LICENSE'
        exclude 'META-INF/LICENSE-FIREBASE.txt'
        exclude 'META-INF/NOTICE'
    }
}
```

- add INTERNET permission to project

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

Writing Firebase data

- add code to initialize Firebase, once, from some activity

```
public void onCreate() {
    Firebase.setAndroidContext(this);
    ...
}

// create a key/value pairing
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

Firebase methods (link)

Method	Description
`fb.child("name")`	return child data object with given name (creates if it did not exist)
`fb.getKey()`	return key for a given data value
`fb.getParent()`	return data one level up in the map
`fb.getRoot()`	return data at top of map
`fb.push()`	create/return an auto-created new child
`fb.removeValue();` `fb.removeValue(handler);`	delete value associated with this key
`fb.runTransaction(handler);`	run multiple queries in sequence
`fb.setPriority(priority);`	gives this data a 'priority' rating for sorting

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

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

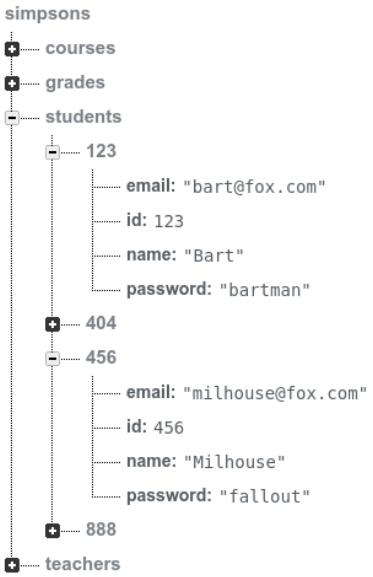
SQL -> Firebase mapping

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

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

students

```
Firebase fb = new Firebase("https://???.firebaseio.com/");
Firebase table = fb.child("simpsons/students");
Firebase bart = table.child(123);   // use db key (id) as FB key
bart.child("id").setValue(123);
bart.child("name").setValue("Bart");
bart.child("email").setValue("bart@fox.com");

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

...
```



Set value with callback

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

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

students

```
Firebase fb = new Firebase("https://???.firebaseio.com/");
Firebase table = fb.child("simpsons/students");
Firebase bart = table.child(123);
bart.child("name").setValue("Bart",
    new Firebase.CompletionListener() {
        public void onComplete(FirebaseError err, Firebase fb) {
            if (err == null) { ...
            }
        }
    });
```

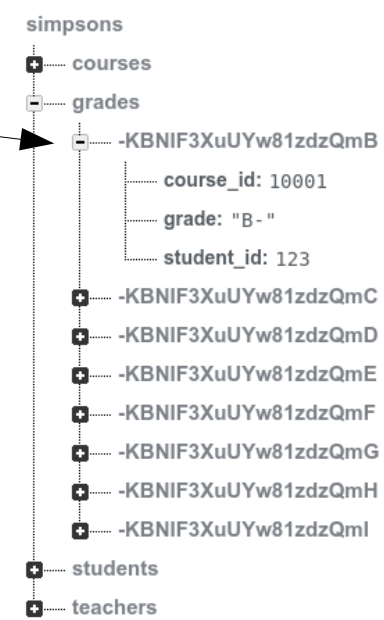
Auto-generated keys

- Some tables don't have a unique "id" column.
  - Firebase can make up unique IDs for you and give auto IDs to new rows using **push**().
  - Also useful in highly parallel situations where many users modify the data at once.

student_id	course_id	grade
123	10001	B-
123	10002	C
456	10001	B+
888	10002	A+
888	10003	A+
404	10004	D+

grades

```
Firebase fb = new Firebase("https://???.firebaseio.com/");
Firebase table = fb.child("simpsons/grades");
Firebase 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:
  - 1) class has a no-params ( ) constructor
  - 2) every field has a get***FieldName***( ) method

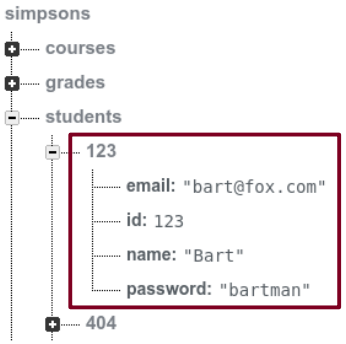
id	name	email
123	Bart	bart@fox.com
456	Milhouse	milhouse@fox.com
888	Lisa	lisa@fox.com
404	Ralph	ralph@fox.com

students

```
public class Student {
    private int id;
    private String name;
    private String email;

    public Student() {}
    public Student(int id, String name, String email) { ... }
    public int getID()       { return id; }
    public String getName()  { return name; }
    public String getEmail() { return email; }
}
...
Firebase fb = new Firebase("https://???.firebaseio.com/");
Firebase table = fb.child("simpsons/students");
Student bart = new Student(123, "Bart", "bart@fox.com");
table.child(123).setValue(bart);
```

Retrieving data

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




```
Firebase fb = new Firebase("https://???.firebaseio.com/");
Firebase bart = fb.child("simpsons/students/123");
bart.addChildEventListener(new ChildEventListener() {
    public void onChildAdded(DataSnapshot dataSnapshot,
                             String s) {
        Log.v("fb", "Bart's " + dataSnapshot.getKey() + ": "
                    + dataSnapshot.getValue());
    }
    ...
});
// Bart's email: bart@fox.com,  Bart's id: 123,   ...
```

Types of data events

Method	Description

`fb.addValueListener(ValueEventListener);`
 	
`- onDataChange(snapshot)`
`- onCancelled(error)`
	listen to changes in a data value

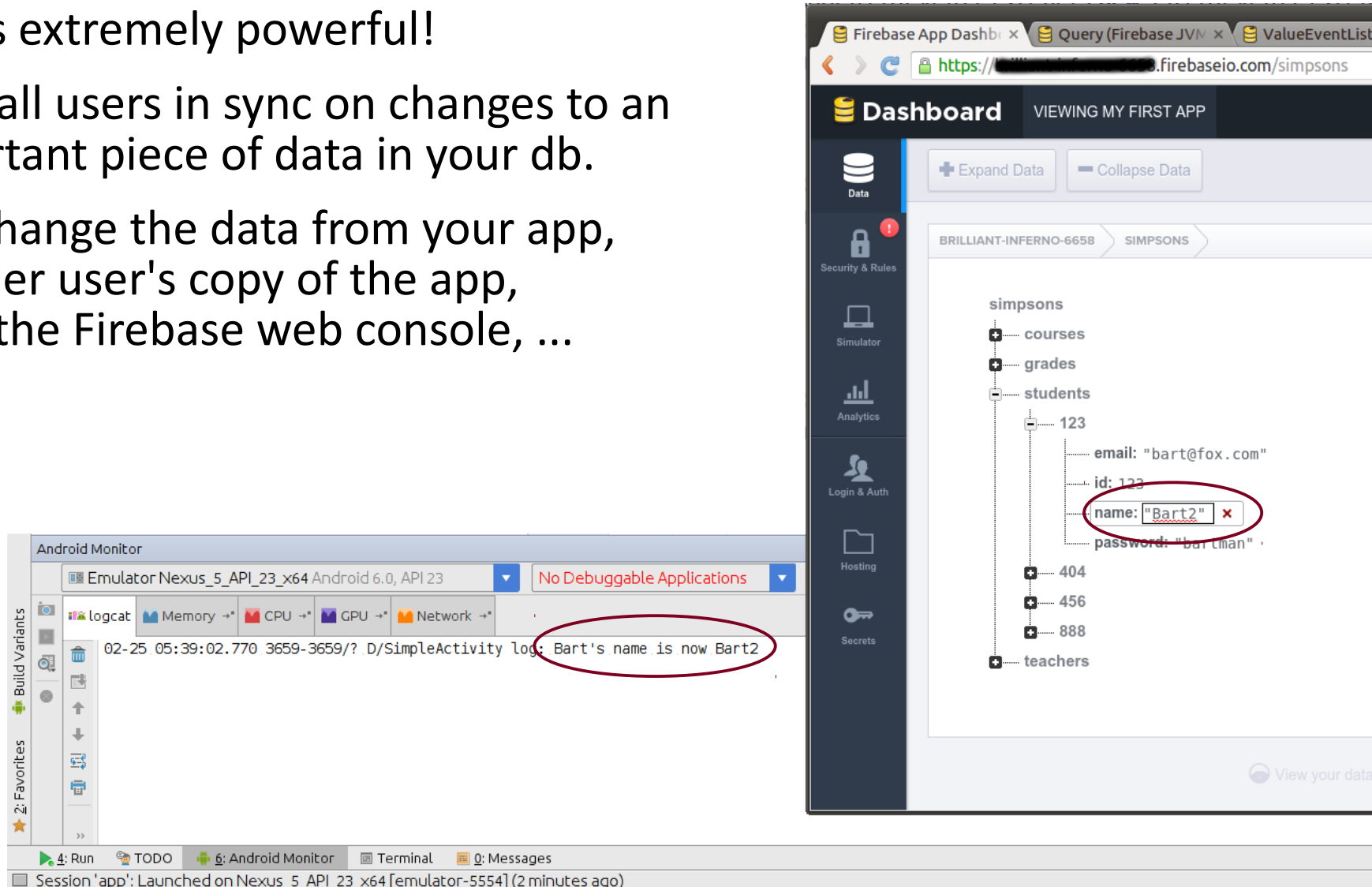
`fb.addListenerForSingleValueEvent(ValueEventListener);`	get initial data and then stop

`fb.addChildListener(ChildEventListener);`
 	
`- onChildAdded(snapshot, name)`
`- onChildChanged(snapshot, oldName)`
`- onChildRemoved(snapshot, oldName)`
`- onChildMoved(snapshot, oldName)`
`- onCancelled(error)`
	listen to changes to the children of a given data value

`fb.removeEventListener(listener);`	removes a listener attached above

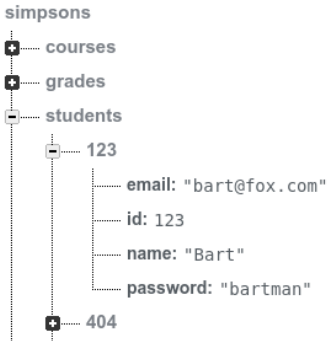
Viewing changes to data

- 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

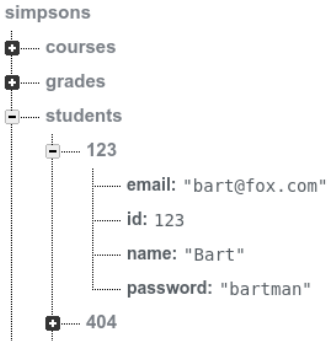
- How do we do queries like we can in SQL?
  - Done using Query, ordering, ranges, etc.
  - Best illustrated by examples:




```
Firebase fb = new Firebase("https://???.firebaseio.com/");
Firebase students = fb.child("simpsons/students");

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

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

Querying data, more detail

```
Firebase fb = new Firebase("???.firebaseio.com/");
Firebase 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) {
        for (DataSnapshot child : dataSnapshot.getChildren()) {
            Log.d("test", "child " + child.getKey() + " => "
                          + child.getValue());
        }
    }
});
```



Query methods (link)

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.endAt()`	sort high -> low	`DESC`
`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	`>=`
`q.startAt()`	sort low -> high *(default)*	`ASC`

DataSnapshot methods (link)

- aoeu

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

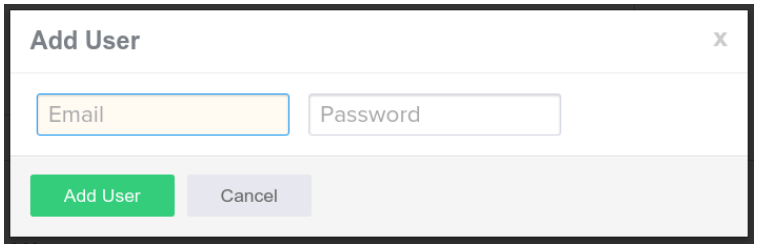
Security and authentication

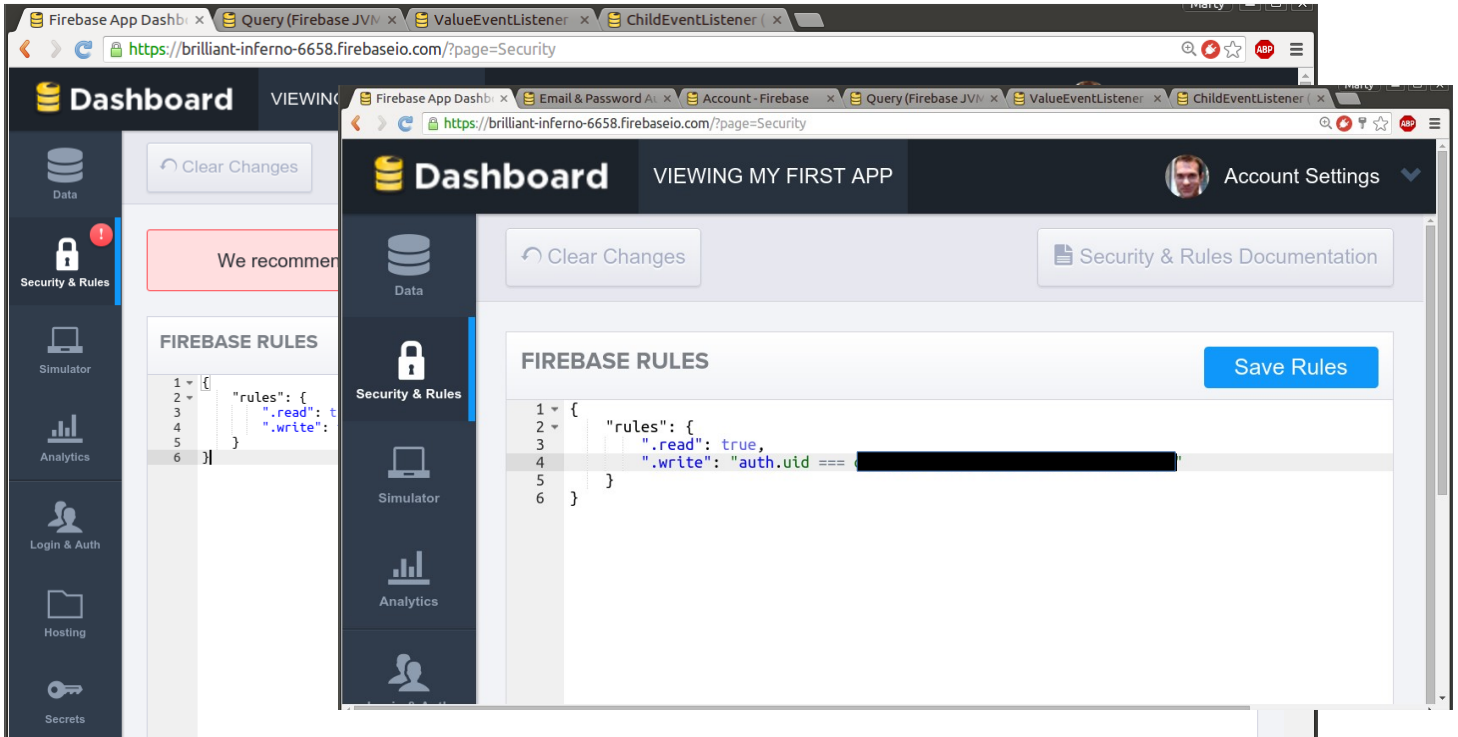
- By default, anyone can read and write your database (!)
  - Use Firebase web UI to add email/password user accounts
  - Modify code to log in with email and password:

```
Firebase fb = new Firebase("https://???.firebaseio.com/");
fire.authWithPassword("email", "password",
    new Firebase.AuthResultHandler() {
        public void onAuthenticated(AuthData authData) {
            // do your queries now!
        }

        public void onAuthenticationError(
                FirebaseError firebaseError) {
            Log.d("auth error! " + firebaseError);
        }
    });
```

Adding user accounts

- several types of logins supported (Google, FB, OAuth, email/password, etc.)
  - enable Email/Password authentication
  - add account(s)
  - copy/paste user ID into a rule in the Firebase Rules area





Firebase auth. methods (link)

Method	Description
`fb.authAnonymously(handler);`	log in anonymously

`fb.authWithCustomToken("token", handler);`
`fb.authWithOAuthToken("provider", "token", handler);`
`fb.authWithPassword("email");`
	log in with various credentials

`fb.changeEmail("old", "password", "new", handler);` `fb.changePassword("email", "old", "new", handler);`	change account's email address/password
`fb.createUser("name", "password", handler);`	make new user account
`fb.getAuth()`	return current auth.data
`fb.resetPassword("email", "handler");`	change account's password
`fb.unauth();`	log out