Paper Prototyping & Schema’s for Firebase Realtime DB
Not much time left

- **Tonight:**
  - Technology demonstration of novel feature
  - Use cases

- **April 18th:**
  1. Paper prototype of user interface
  2. Real-time database **schema** and **tests**

- **April 25th:**
  - GUI implementation

- **May 2nd:**
  - Final integrated app
Paper Prototyping

- **Cheap, low fidelity prototype**
  - Enables rapid iteration

- **Fresh subjects**
  - No preconceptions

- **Task prompt(s)**
  - Written down (avoid variation in communicating task)

- **Subject performs task (Wizard of Oz)**
  - Human simulates system (*doesn’t explain!*)
  - Ideally, separate observer monitors subject; takes notes

- **Post-interview**
  - Any confusion? What is missing?
Paper Prototype for Next Week

- Make a paper prototype of your App
  - Identify tasks for your participants to do using your App
    - Drawn from use cases in your specification
  - Have friends/acquaintances try to do those tasks
  - Ask follow-up questions about confusions/problems
  - Write 2 paragraph description of findings from prototype
    - Discuss you 2 main findings and what UI changes you’ll make
    - Submit to Compass2g by Tuesday night

- Make a revised paper prototype of your App
  - Bring it to code review
  - Be prepared to run it for a peer
Real-time DB Schemas

- Schema = Organization of Data

- A users table: This table is world readable, but only writable by the user and contains their current alias.
  - users: [ userID: user ]
  - user: { "alias": String }

```
/users
  |--user1
  |    |--alias: "George"
  |
  |--user2
  |    |--alias: "Wilma"
  |
  |
  |--frewy33
  |    |--alias: "Bob"
```
Real-time DB Schemas

■ Important Firebase Behavior for Organization Rules
  ▪ Think of the database as a tree
  ▪ Can listen to sub-trees: values or children
    ▪ Every DB reference either holds an object or an array
  ▪ When we listen, we’re notified if anything below changes
  ▪ When we’re notified, the DataSnapshot contains sub-tree

■ We want to minimize:
  ▪ Notification frequency
  ▪ Size of DataSnapshot
Example Chat App

- Activity 1: list chat room names
- Activity 2: show all of the messages in that chat

Problematic Candidate Schema:

```
/chats
  | places to eat in Chambana
  |   | KWE-IUawhmH5HoPBBz
  |   |   | --author: "user1"
  |   |   |   --message: "I like Cravings even though they keep getting sanitation violations"
  |   | KWE-IUawhmH5Ho801
  |   |   | --author: "user2"
  |   |   |   --message: "Yes, their Fish Masala is notoriously good!"
  |   | Good CS classes
  |   |   | KWE-IUawhmH5Ho833
  |   |   |   | --author: "user1"
  |   |   |   |   --message: "CS 233 teaches all about how computers work. It is cool!"
```

Surely not CS 126... lol...
Better Chat Schema

Act 1
/directory
- KWDl7xvsgpU4FcGzo0y
  | --name: "Places to eat in Chambana"
  | --key: "KWDl7xvsgpU4FcGzo0y"
- KWDl7xvsgp3Mz71rSov
  | --name: "Good CS classes"
  | --key: "KWDl7xvsKguPe1wzruw"

Act 2
/chats
- KWDl7xvsgpU4FcGzo0x
  | --KWE-IUawhmH5HoPBBz
  |   | --author: "user1"
  |   | --message: "I like Cravings even though they keep getting sanitation violations"
  | --KWE-IUawhmH5Hof801
  |   | --author: "user2"
  |   | --message: "Yes, their Fish Masala is notoriously good!"
 | --KWDl7xvsKguPe1wzruw
  | --KWE-IUawhmH5Hof833
  |   | --author: "user1"
  |   | --message: "CS 233 teaches all about how computers work. It is cool!"
Avoid Synchronization Issues

- Example bad case: to users incrementing the same integer

- Avoid situations where multiple users modify same value
  - Append only structures
  - Per-user data
Example Solution: Likes/Upvotes

- Instead of keeping a count of likes...
- Keep an array of the users that have liked something
  - Each user appends their name when they like it
  - Removes their name from the list when they un-like it
  - Number of likes = number of entries in the array

- No synch. problems; each user touches only their data
- Also can track whether a user already liked it
Implementing Database Tests

- Put them in AndroidTests
- Need to deal with asynchronous nature of Firebase DB
  - Want to wait until operation completes
  - Can use CountDownLatch

// set up a counter that is expecting 1 down count
final CountDownLatch writeSignal = new CountDownLatch(1);

// prevent forward progress until count is zero or 10 seconds pass
writeSignal.await(10, TimeUnit.SECONDS);

// decrement counter
writeSignal.countDown();
Example Write Test

```java
final CountDownLatch writeSignal = new CountDownLatch(1);

dbRef.setValue("none").addOnCompleteListener(new OnCompleteListener<Void>() {
    @Override
    public void onComplete(@NonNull Task<Void> task) {
        writeSignal.countDown();
    }
});

writeSignal.await(10, TimeUnit.SECONDS);
```
Example Read Test

final CountDownLatch writeSignal = new CountDownLatch(1);

dbRef.addListenerForSingleValueEvent(new ValueEventListener(){
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        assertEquals("expected value", dataSnapshot.getValue(String.class));
        writeSignal.countDown();
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {
    }
});

writeSignal.await(10, TimeUnit.SECONDS);
Your tests:

- You should have one that performs each kind of operation that you’ll need to do to the database
  - i.e., insert a user, get/listen to all chats, update high score
Matchmaking code