House Course 59-20

Web and Mobile Applications
Week 5: Yik Yak’s Backend

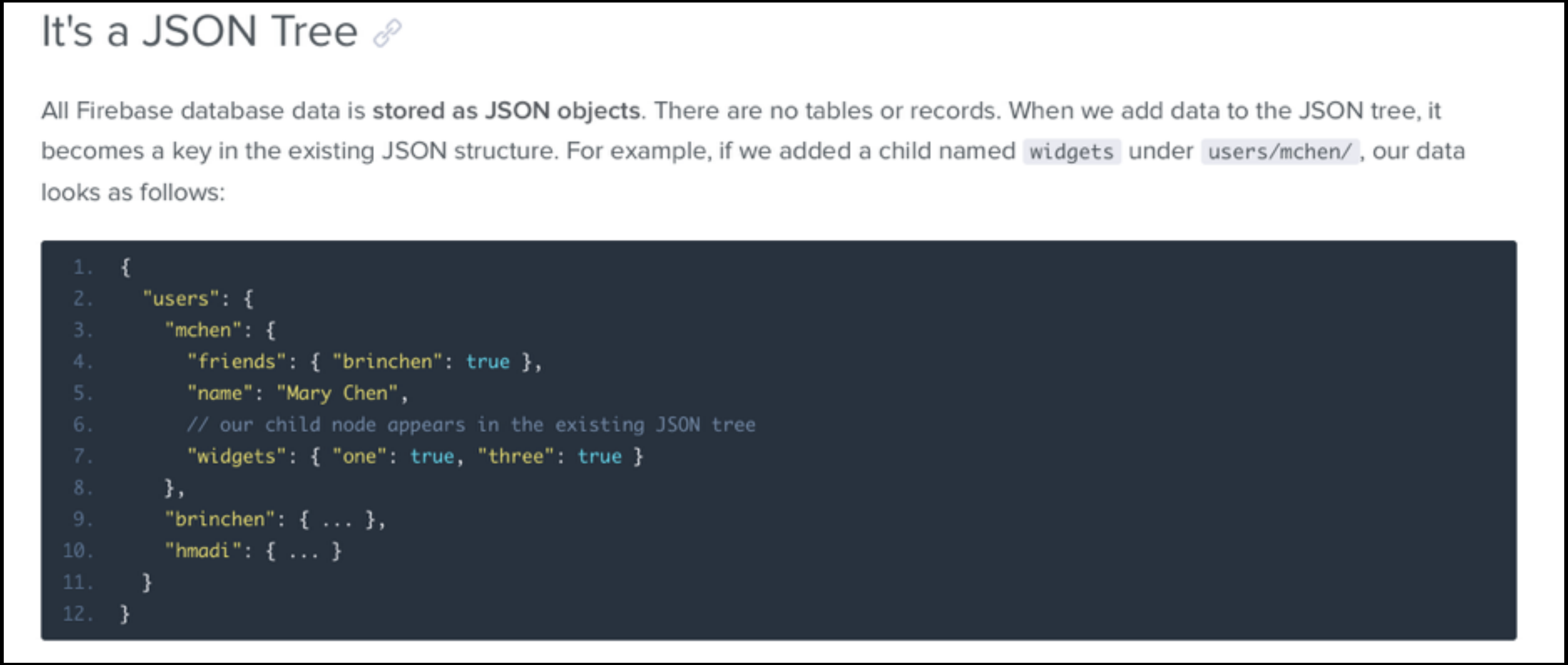
Attendance: TBD

Davis Gossage
Jesse Hu

Week 4

- Firebase
  - Retrieving Data
  - Saving Data
- .xcworkspace

Firebase - Data Structure

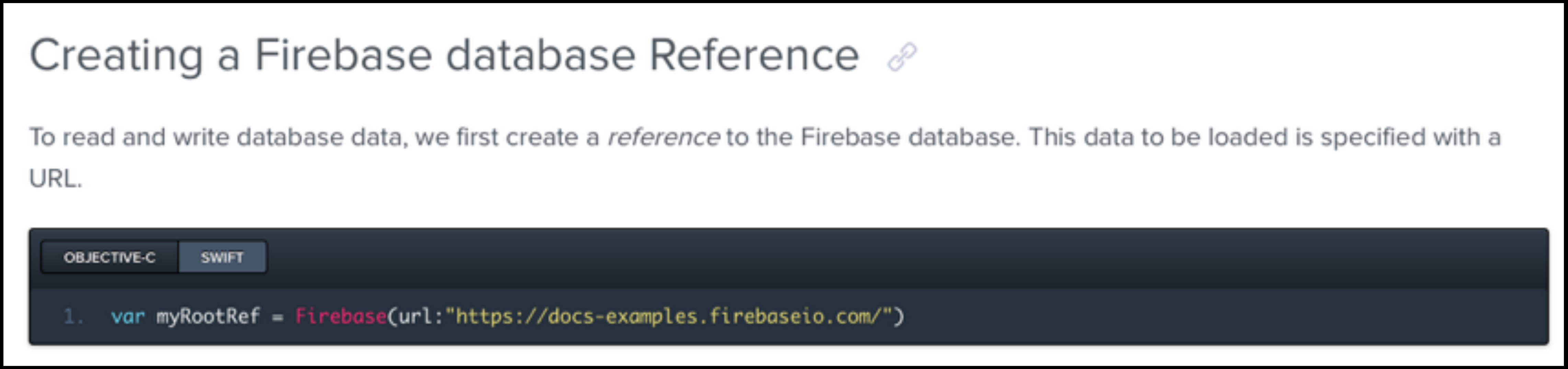


It's a JSON Tree

All Firebase database data is **stored as JSON objects**. There are no tables or records. When we add data to the JSON tree, it becomes a key in the existing JSON structure. For example, if we added a child named `widgets` under `users/mchen/`, our data looks as follows:

```
{
  "users": {
    "mchen": {
      "friends": { "brinchen": true },
      "name": "Mary Chen",
      // our child node appears in the existing JSON tree
      "widgets": { "one": true, "three": true }
    },
    "brinchen": { ... },
    "hmadi": { ... }
  }
}
```

Firebase - Data Structure

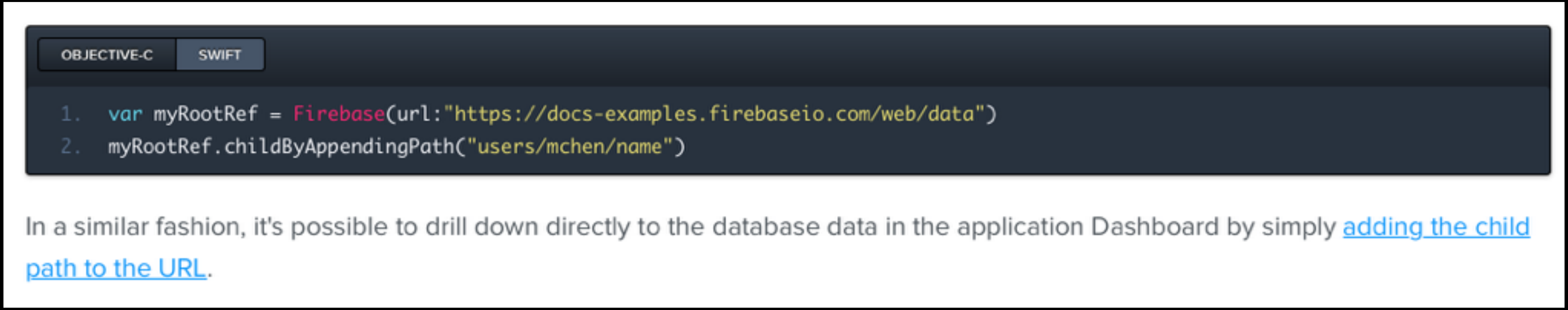


Creating a Firebase database Reference

To read and write database data, we first create a *reference* to the Firebase database. This data to be loaded is specified with a URL.

```swift
var myRootRef = Firebase(url:"https://docs-examples.firebaseio.com/")
```

Drill into data using ‘childByAppendingPath’

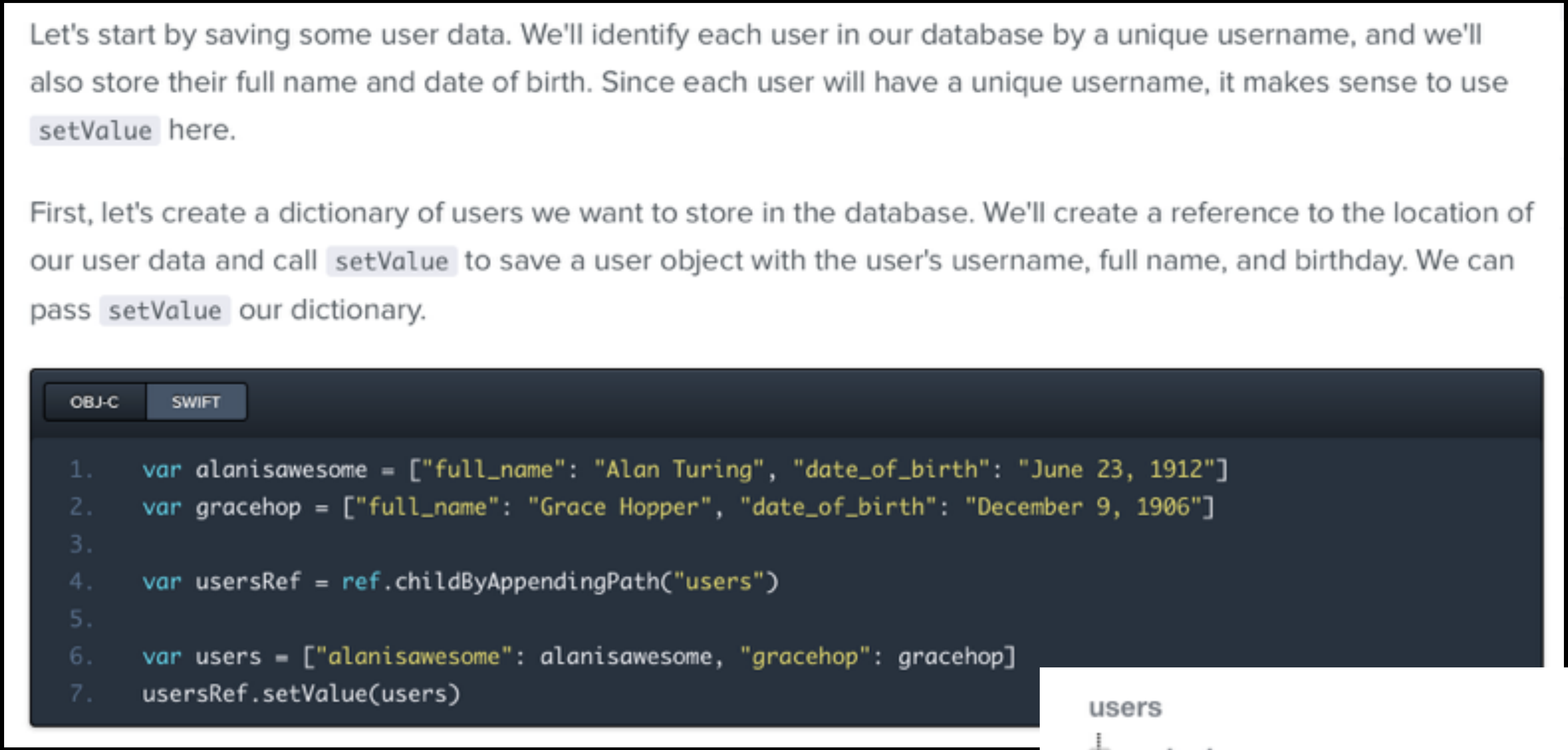


```swift
var myRootRef = Firebase(url:"https://docs-examples.firebaseio.com/web/data")
myRootRef.childByAppendingPath("users/mchen/name")
```

In a similar fashion, it's possible to drill down directly to the database data in the application Dashboard by simply adding the child path to the URL.

Firebase - Saving Data

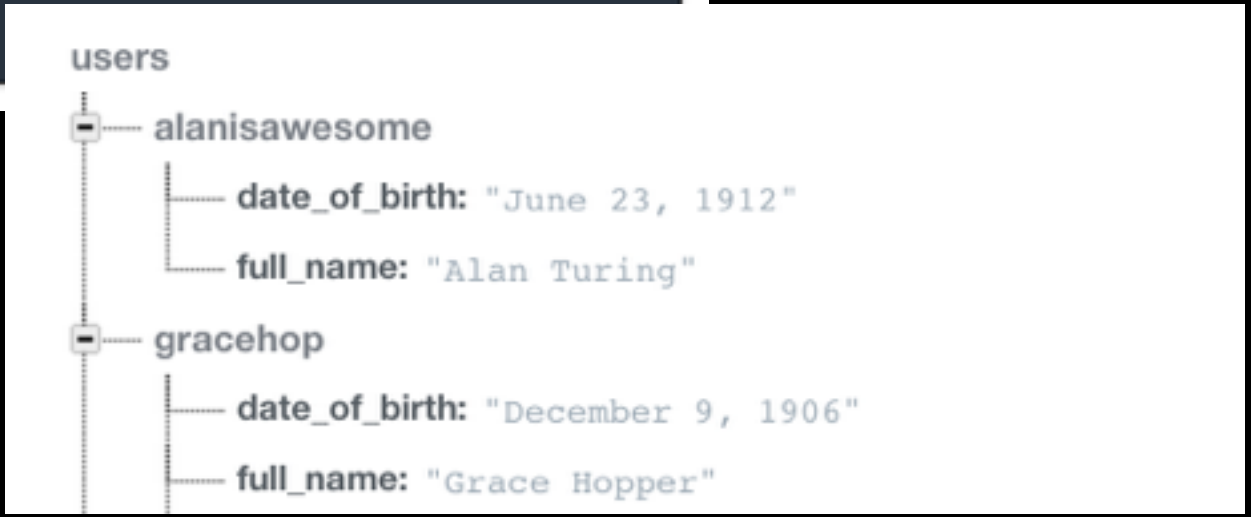
Use ‘setValue’ to save data



Let's start by saving some user data. We'll identify each user in our database by a unique username, and we'll also store their full name and date of birth. Since each user will have a unique username, it makes sense to use `setValue` here.

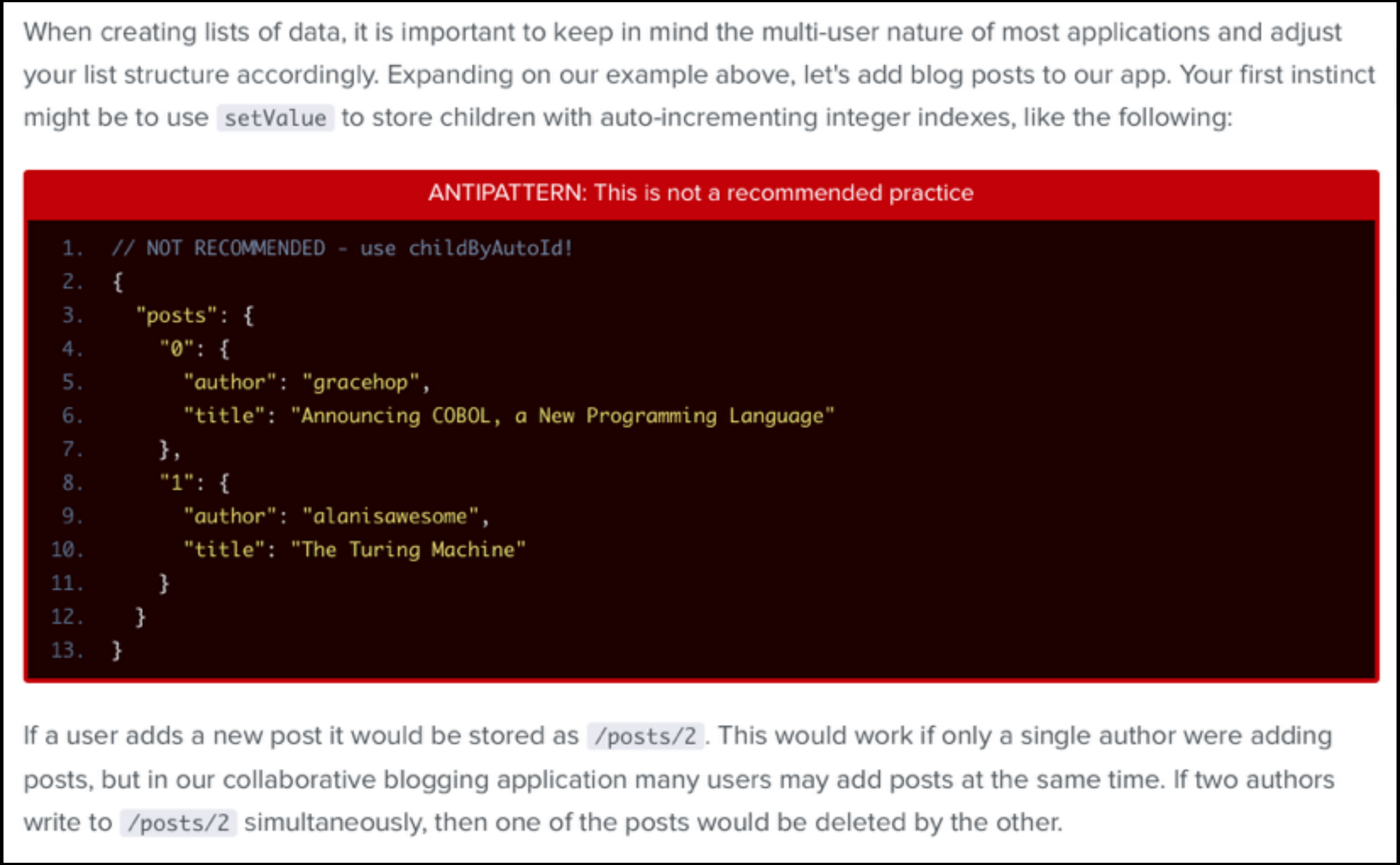
First, let's create a dictionary of users we want to store in the database. We'll create a reference to the location of our user data and call `setValue` to save a user object with the user's username, full name, and birthday. We can pass `setValue` our dictionary.

```swift
var alanisawesome = ["full_name": "Alan Turing", "date_of_birth": "June 23, 1912"]
var gracehop = ["full_name": "Grace Hopper", "date_of_birth": "December 9, 1906"]

var usersRef = ref.childByAppendingPath("users")

var users = ["alanisawesome": alanisawesome, "gracehop": gracehop]
usersRef.setValue(users)
```



```
users
  alanisawesome
    date_of_birth: "June 23, 1912"
    full_name: "Alan Turing"
  gracehop
    date_of_birth: "December 9, 1906"
    full_name: "Grace Hopper"
```

Firebase - Saving Data



When creating lists of data, it is important to keep in mind the multi-user nature of most applications and adjust your list structure accordingly. Expanding on our example above, let's add blog posts to our app. Your first instinct might be to use `setValue` to store children with auto-incrementing integer indexes, like the following:

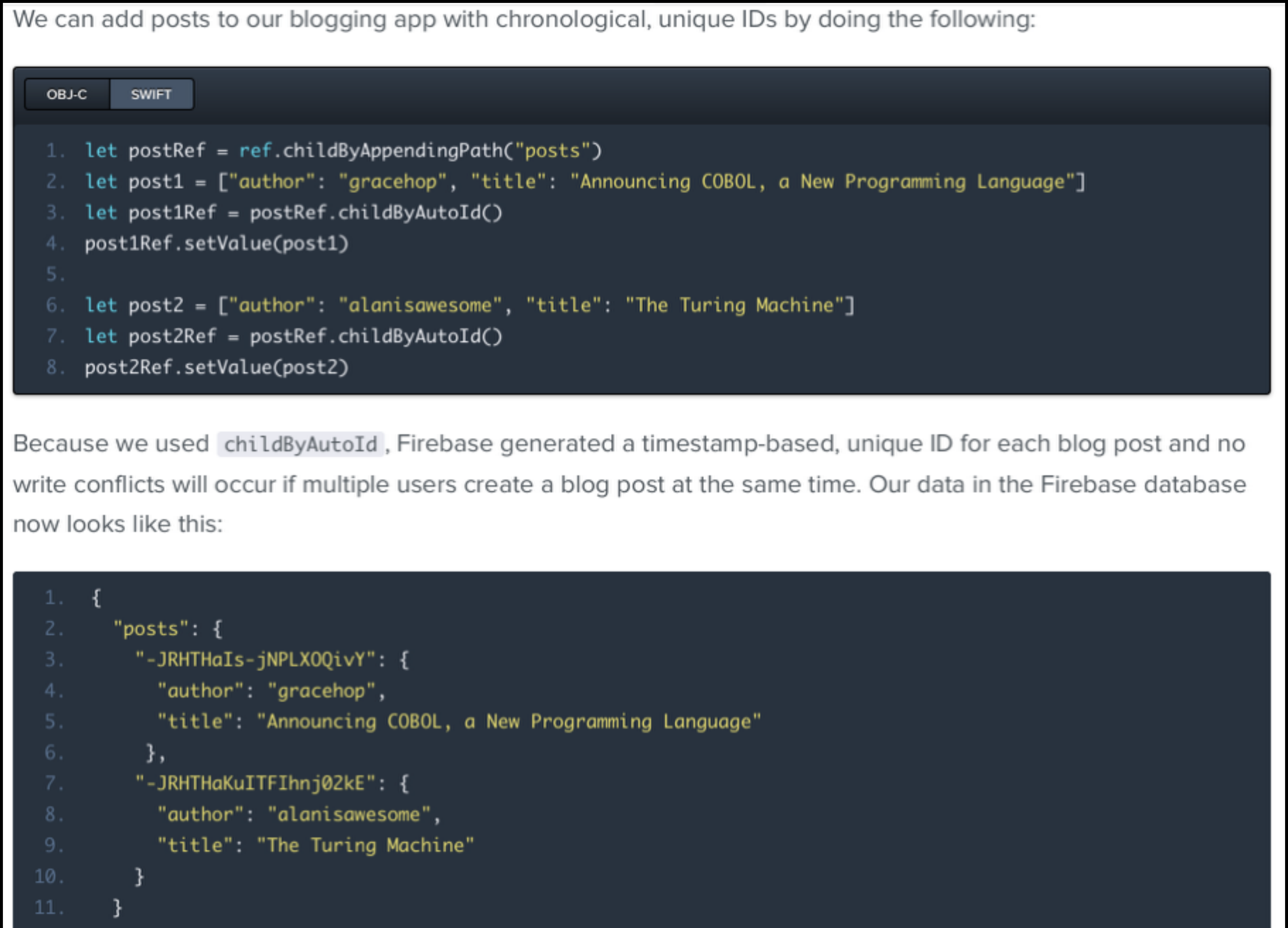
ANTIPATTERN: This is not a recommended practice

```
// NOT RECOMMENDED - use childByAutoId!
{
  "posts": {
    "0": {
      "author": "gracehop",
      "title": "Announcing COBOL, a New Programming Language"
    },
    "1": {
      "author": "alanisawesome",
      "title": "The Turing Machine"
    }
  }
}
```

If a user adds a new post it would be stored as `/posts/2`. This would work if only a single author were adding posts, but in our collaborative blogging application many users may add posts at the same time. If two authors write to `/posts/2` simultaneously, then one of the posts would be deleted by the other.

We need a way to keep track of posts using unique IDs

Firebase - Saving Data



We can add posts to our blogging app with chronological, unique IDs by doing the following:

```swift
let postRef = ref.childByAppendingPath("posts")
let post1 = ["author": "gracehop", "title": "Announcing COBOL, a New Programming Language"]
let post1Ref = postRef.childByAutoId()
post1Ref.setValue(post1)

let post2 = ["author": "alanisawesome", "title": "The Turing Machine"]
let post2Ref = postRef.childByAutoId()
post2Ref.setValue(post2)
```

Because we used `childByAutoId`, Firebase generated a timestamp-based, unique ID for each blog post and no write conflicts will occur if multiple users create a blog post at the same time. Our data in the Firebase database now looks like this:

```json
{
  "posts": {
    "-JRHTHaIs-jNPLXOQivY": {
      "author": "gracehop",
      "title": "Announcing COBOL, a New Programming Language"
     },
    "-JRHTHaKuITFIhnj02kE": {
      "author": "alanisawesome",
      "title": "The Turing Machine"
    }
  }
```

.xcworkspace

- Up to this point, we've only dealt with .xcodeproj files
  - .xcodeproj opens a single project in Xcode
- When we have a project dependent on other projects, we use .xcworkspace to display all of them
  - Cocoapods is a tool we use to manage these dependencies
  - Yik Yak is dependent on the Firebase SDK and a custom textview library

Yik Yak Demo

- Let’s look at Main.storyboard
- UITabBarController with 3 tabs. Each has its own navigation controller
  - Post Scene TableViewController
  - Herd Scene TableViewController (not implemented)
  - Profile Scene TableViewController (not implemented)

Post Scene

- The initial screen where the Yak feed is shown
- Owned by PostTableViewController.swift
- Presents a model segue (Compose Scene) when the compose button is tapped
- Pushes to the Detail Scene when any cell from the feed is tapped
  - We use prepareForSegue to let the Yak Scene know what Yak it needs to display

Compose Scene

- Allows a Yak to be composed
- Owned by ComposeViewController.swift
- Has actions for the cancel button and the send button
- Does some cool stuff with the textViewDelegate
  - Detects the Send button from the keyboard
  - Limits length of Yak

Detail Scene

- Shows Yak details and comments
- Owned by DetailViewController.swift
- Has outlets for info about the Yak
- Has a tableview to show comments
- Subscribes to keyboard show and hide notifications to slide the comment box up and down

Yak.swift

- Holds info about a Yak including:
    - text
    - timestamp
    - replies
    - netVoteCount
    - location

Reply.swift

- Holds info about a reply, or comment
    - text
    - timestamp
    - netVoteCount
    - location
- Reply and Yak are very similar, there is probably an opportunity to have one inherit from the other