House Course 59-20

Web and Mobile Applications
Week 4: Databases and Services

Attendance: http://bit.ly/1U8XoMM

Davis Gossage
Jesse Hu

Class Webpage

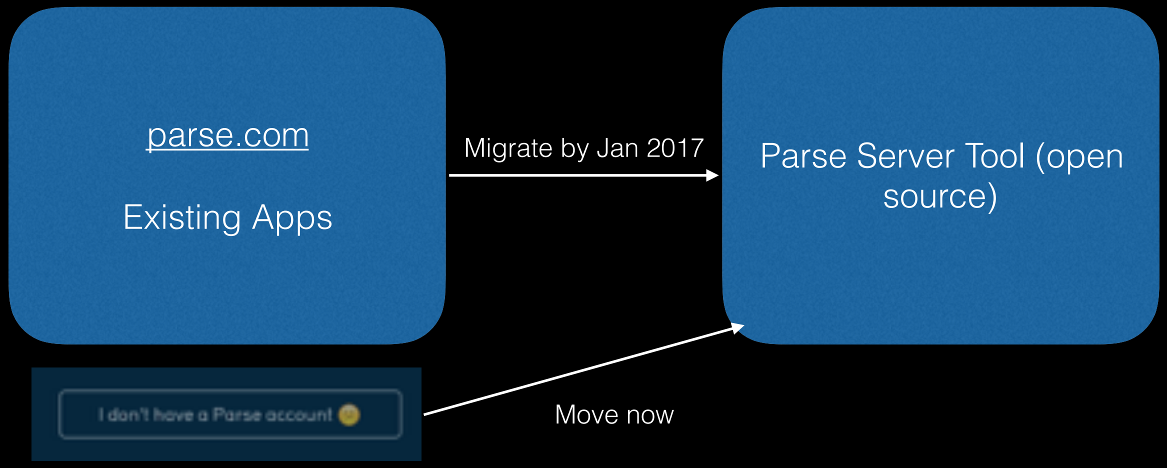
- http://dukeappcourse.com

Attendance

Attendance: http://bit.ly/1U8XoMM

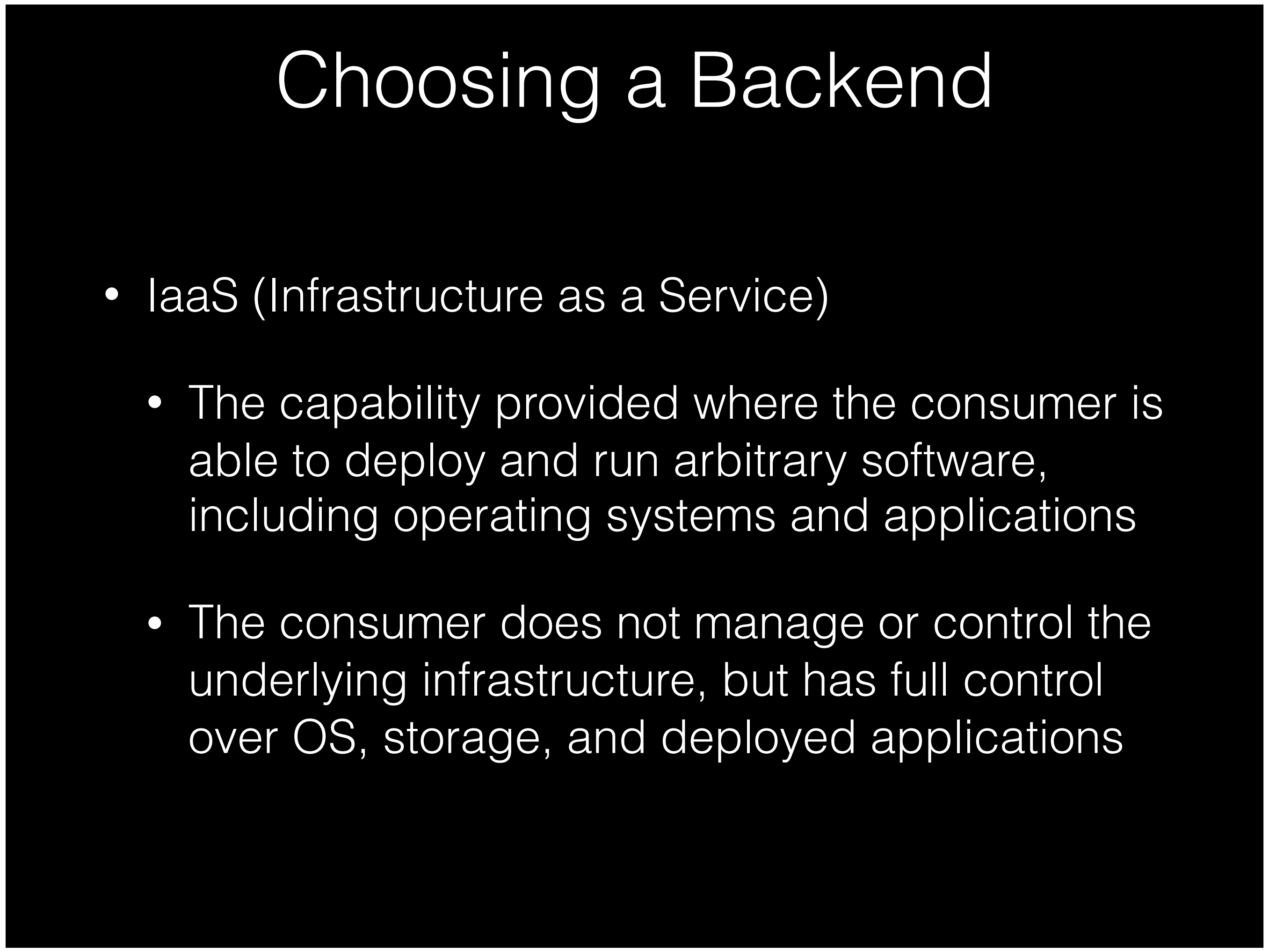
The Parse Saga Continues

- Parse is forcing new users to use the server tool
- We don't like the server tool (yet)



Week 4

- Choosing a Backend
  - IaaS
  - PaaS
- SQL (mySQL) vs. NoSQL (MongoDB)
- Additional iOS Topics
  - CoreLocation
  - Camera Access



Choosing a Backend

- IaaS (Infrastructure as a Service)
  - The capability provided where the consumer is able to deploy and run arbitrary software, including operating systems and applications
  - The consumer does not manage or control the underlying infrastructure, but has full control over OS, storage, and deployed applications

Infrastructure as a service is where a VM is allocated for your use, you own it and have the root password to the entire system. You can choose the operating system you want to run and whatever stack of software you want on it. As a consumer, you don't manage or control the infrastructure. So the service provider is responsible for networking your machine to the outside world and maintaining the hardware and the entire server farm. Examples of IaaS?

Choosing a Backend - IaaS

- Examples
  - https://aws.amazon.com/ec2/pricing/
  - http://vm-manage.oit.duke.edu
- Pros:
  - Cheap or free while you're small
  - Infinite customizability
- Cons:
  - You're responsible for outages, security fixes, random maintenance issues
    - http://heartbleed.com
  - Most of the things you want to do have already been done (user auth, push notifications, etc.)

Choosing a Backend

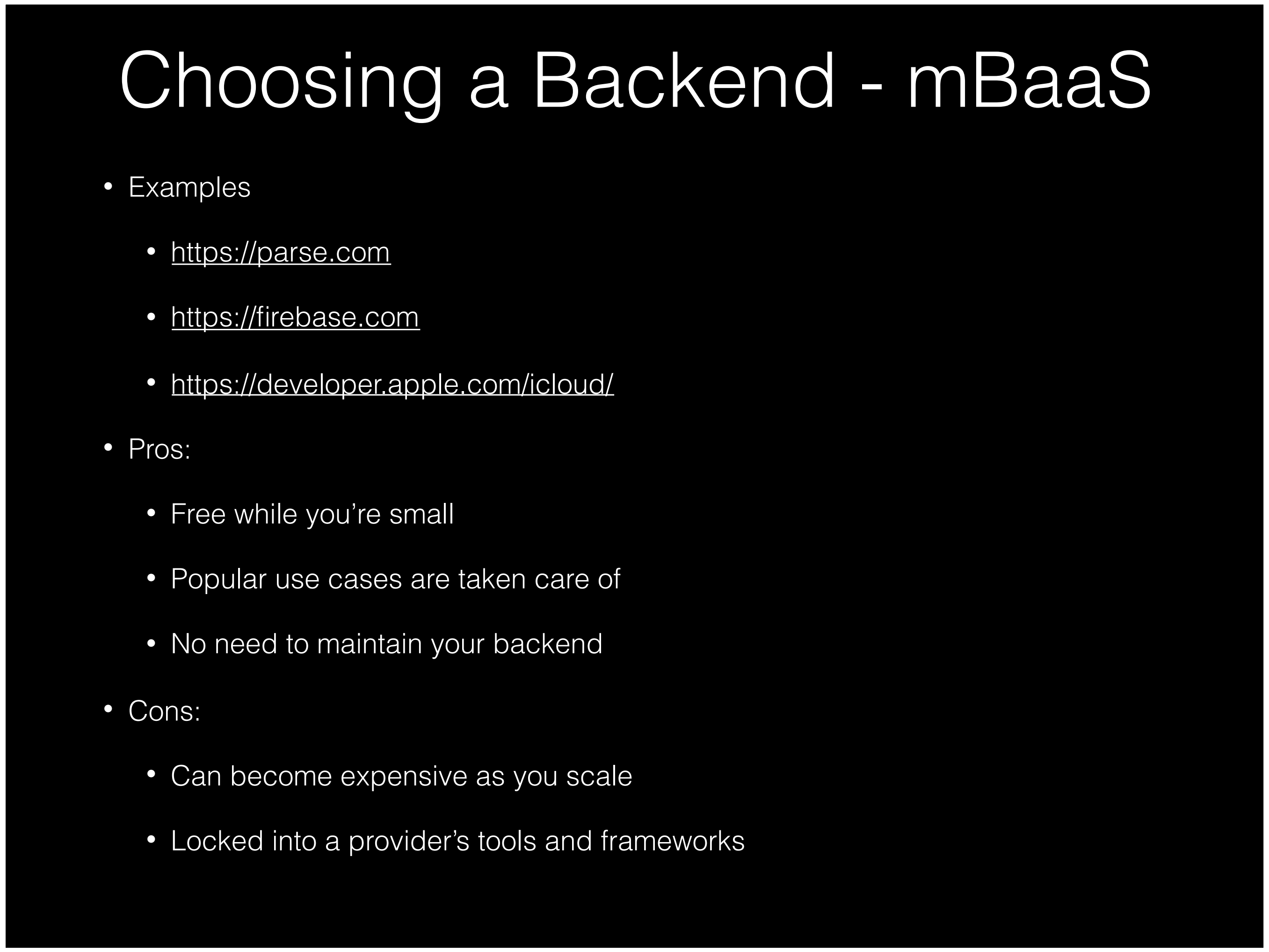
- PaaS (Platform as a Service)
  - The capability provided where the consumer is able to deploy a created application using the programming languages, libraries, services, and tools supported by the provider.
  - The consumer does not manage or control the underlying infrastructure, including OS or storage, but maintains control over deployed applications

Choosing a Backend - PaaS

- Examples
  - https://cloud.google.com/appengine/
  - https://www.heroku.com/
  - http://www.ibm.com/cloud-computing/bluemix/
- Pros:
  - Takes care of infrastructure
  - Quickly build and deploy software
  - Improved scalability, security, integrations
- Cons:
  - Can become expensive as you scale
  - Don't have access to full range of tools

Choosing a Backend

- BaaS (Backend as a Service)
  - Provide web and mobile app developers with features such as backend cloud storage, user management, push notifications, and social network integration
  - Less flexibility than building on top of a PaaS, but offers out-of-the-box functionality for mobile applications.



Choosing a Backend - mBaaS

- Examples
  - https://parse.com
  - https://firebase.com
  - https://developer.apple.com/icloud/
- Pros:
  - Free while you're small
  - Popular use cases are taken care of
  - No need to maintain your backend
- Cons:
  - Can become expensive as you scale
  - Locked into a provider's tools and frameworks

One of the biggest draws to platforms as a service is these backend solutions end up looking very similar, especially for mobile apps. It's almost a given that you'd want to configure things like user authentication, security features like access control (making sure you're in control of who can touch what data), and modules like push notifications (letting you send messages to your app or to the user without the app needing to be active)

Databases

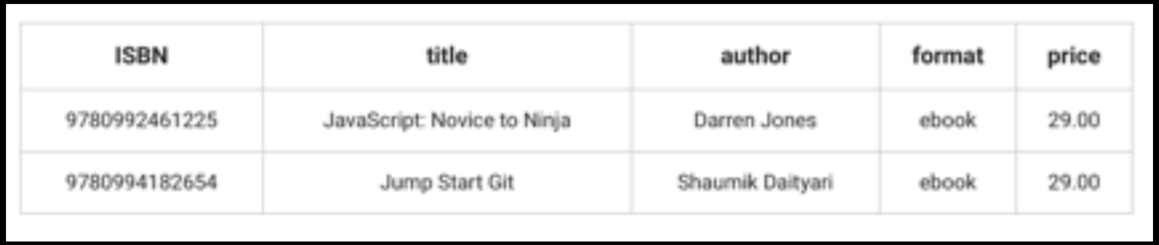
- A means of storing information in such a way that information can be retrieved from it
- Popular types are SQL and NoSQL

SQL

- Structured Query Language
- Usage exploded in the 1990s
- Examples include MySQL, PostgreSQL, SQLite
- LAMP Stack
  - Linux, Apache, MySQL, PHP

SQL

- SQL databases store related data in tables
- Each data item is a row
- The design (schema) is rigid
- Example: table named ‘book’

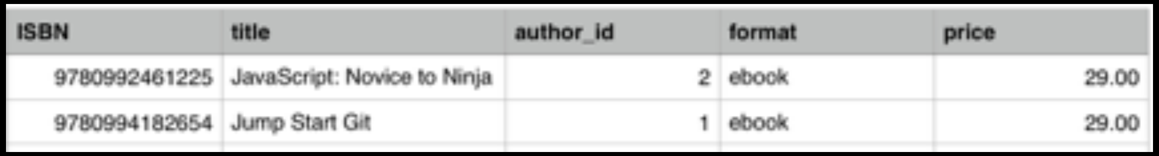


ISBN	title	author	format	price
9780992461225	JavaScript: Novice to Ninja	Darren Jones	ebook	29.00
9780994182654	Jump Start Git	Shaumik Daityari	ebook	29.00

SQL Relations

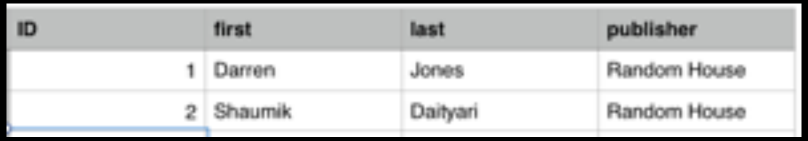
- It makes sense to store repeating or separate information in another table
- These are ‘joined’ by unique IDs

Book



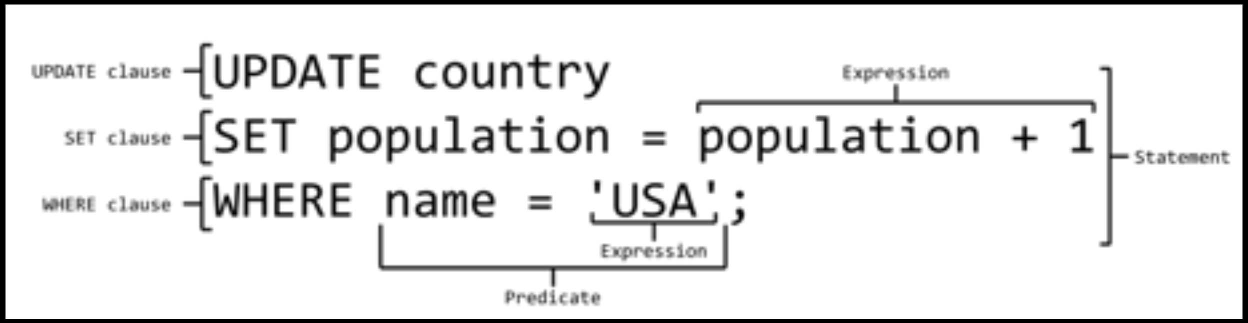
ISBN	title	author_id	format	price
9780992461225	JavaScript: Novice to Ninja	2	ebook	29.00
9780994182654	Jump Start Git	1	ebook	29.00

Author



ID	first	last	publisher
1	Darren	Jones	Random House
2	Shaumik	Daityari	Random House

SQL Query

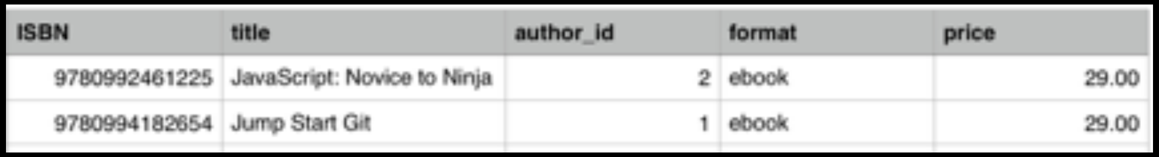


```
UPDATE country
SET population = population + 1
WHERE name = 'USA';
```

SQL Join Query

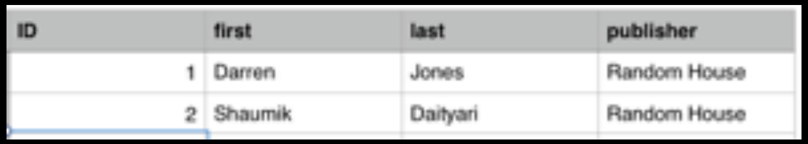
- SELECT * FROM ‘Book’ INNER JOIN ‘Author’ ON Book.author_id = Author.ID

Book



ISBN	title	author_id	format	price
9780992461225	JavaScript: Novice to Ninja	2	ebook	29.00
9780994182654	Jump Start Git	1	ebook	29.00

Author



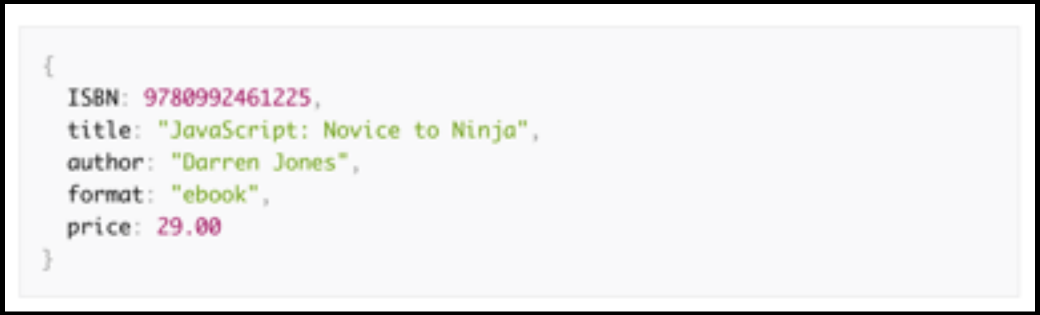
ID	first	last	publisher
1	Darren	Jones	Random House
2	Shaumik	Daityari	Random House

NoSQL

- Been around in some form since the 1960s
- Surge in popularity with services like MongoDB, Redis, Apache Cassandra
- MEAN Stack
    - MongoDB, Express, Angular, Node.js

NoSQL

- NoSQL stores field-value pair **documents** in formats such as XML, YAML, or JSON
- Similar documents are stored in **collections**
  - Analogous to tables, but not enforced



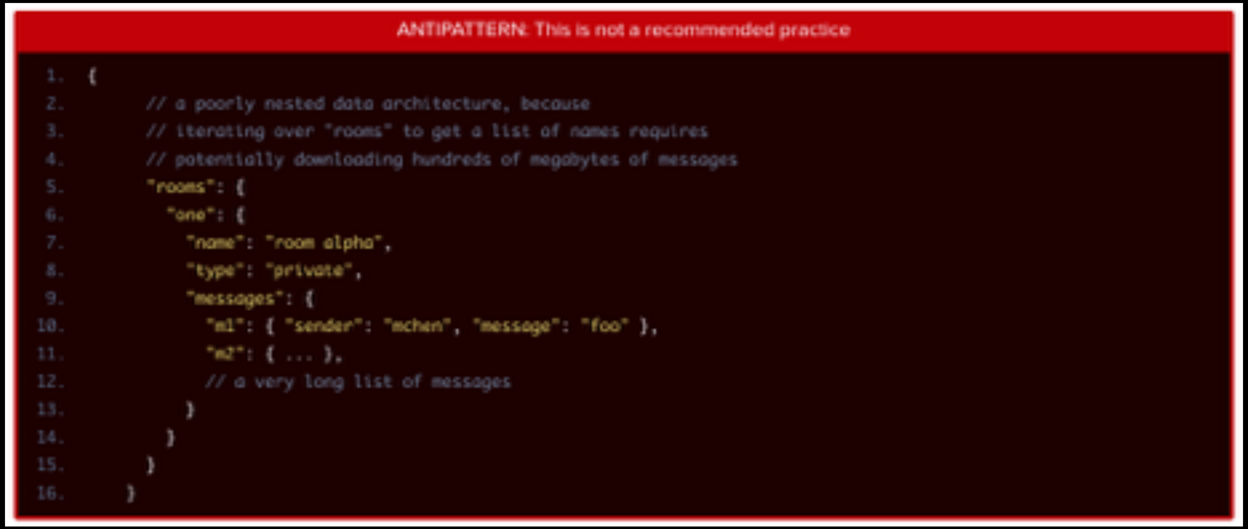
```
{
  ISBN: 9780992461225,
  title: "JavaScript: Novice to Ninja",
  author: "Darren Jones",
  format: "ebook",
  price: 29.00
}
```

JSON = javascript object notation

Since we don’t have to conform to data fields, we need to make sure we are clear about how our data is structured or we risk consistency issues across our app

NoSQL Best Practices

- Flatten data when possible, nested data is rarely ideal…

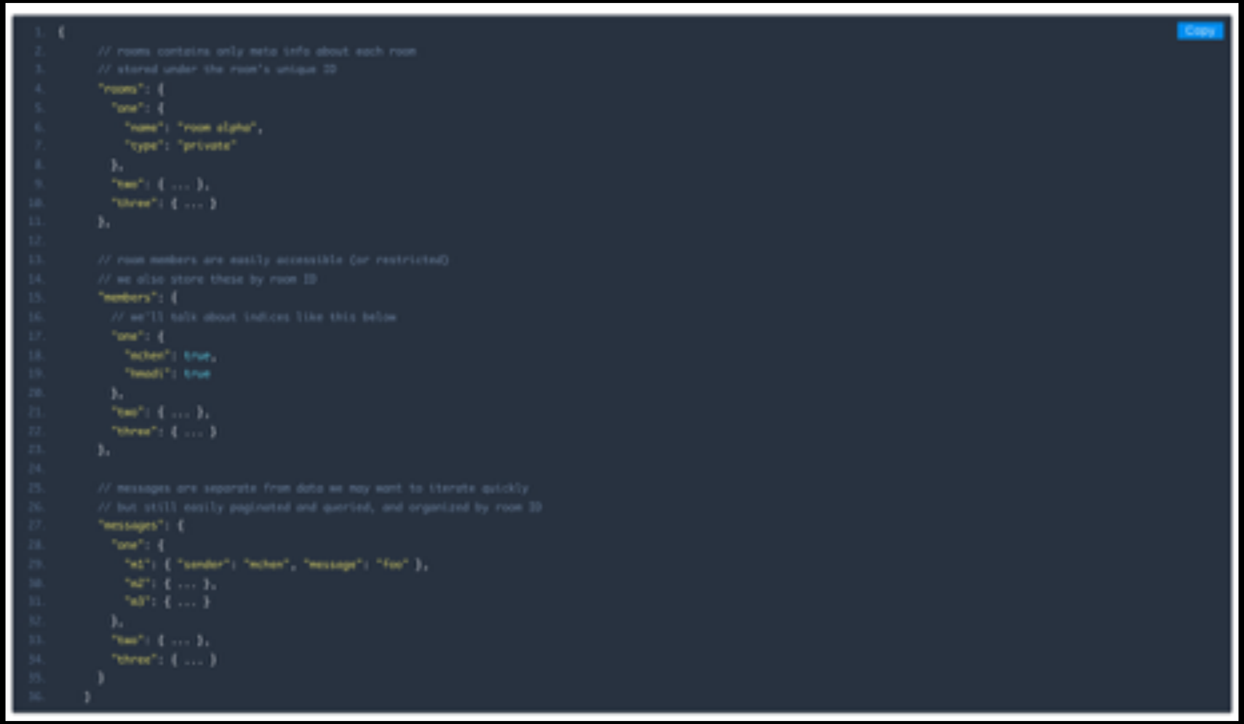


ANTIPATTERN: This is not a recommended practice

```
{
      // a poorly nested data architecture, because
      // iterating over "rooms" to get a list of names requires
      // potentially downloading hundreds of megabytes of messages
      "rooms": {
        "one": {
          "name": "room alpha",
          "type": "private",
          "messages": {
            "m1": { "sender": "mchen", "message": "foo" },
            "m2": { ... },
            // a very long list of messages
          }
        }
      }
    }
```

With our Yik Yak app, we wouldn't want to nest all of the replies in with the actual yak. We would want to flatten this out so that we don't necessarily need to download every reply anytime we want information about a yak.

NoSQL Best Practices

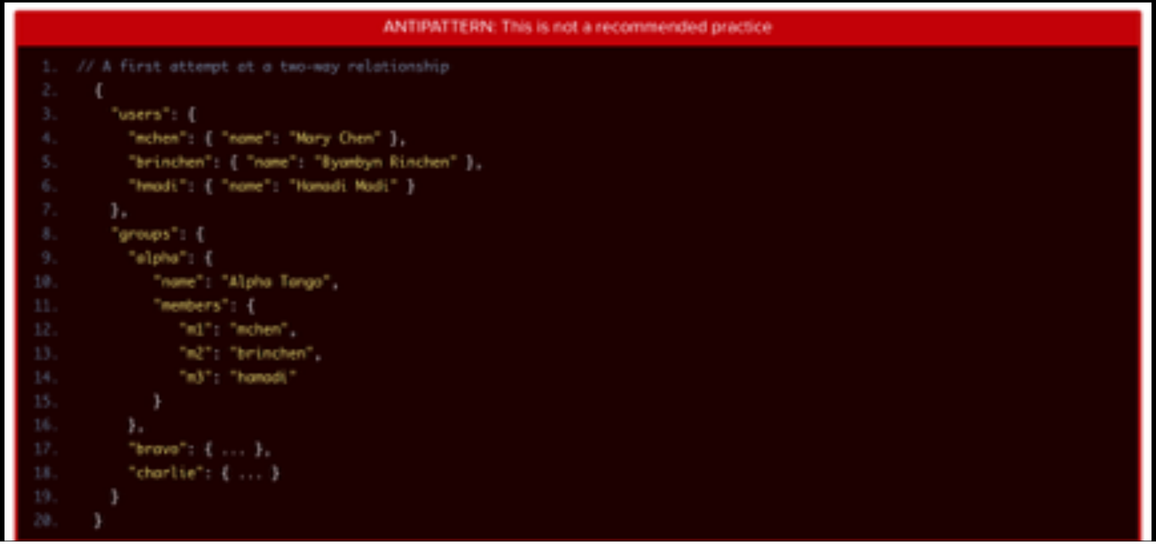

Copy

```
{
      // rooms contains only meta info about each room
      // stored under the room's unique ID
      "rooms": {
        "one": {
          "name": "room alpha",
          "type": "private"
        },
        "two": { ... },
        "three": { ... }
      },

      // room members are easily accessible (or restricted)
      // we also store these by room ID
      "members": {
        // we'll talk about indices like this below
        "one": {
          "mchen": true,
          "hmadi": true
        },
        "two": { ... },
        "three": { ... }
      },

      // messages are separate from data we may want to iterate quickly
      // but still easily paginated and queried, and organized by room ID
      "messages": {
        "one": {
          "m1": { "sender": "mchen", "message": "foo" },
          "m2": { ... },
          "m3": { ... }
        },
        "two": { ... },
        "three": { ... }
      }
    }
```

This data has now been flattened. We have the metadata for the rooms, the room name and type, but all the other information is stored in a different collection and is accessible by the unique room ID.

NoSQL 2-Way Relationships

- When building apps, it is often preferable to download a subset of a list.
- Say we wanted users to be able to belong to a group and groups contain a list of users…



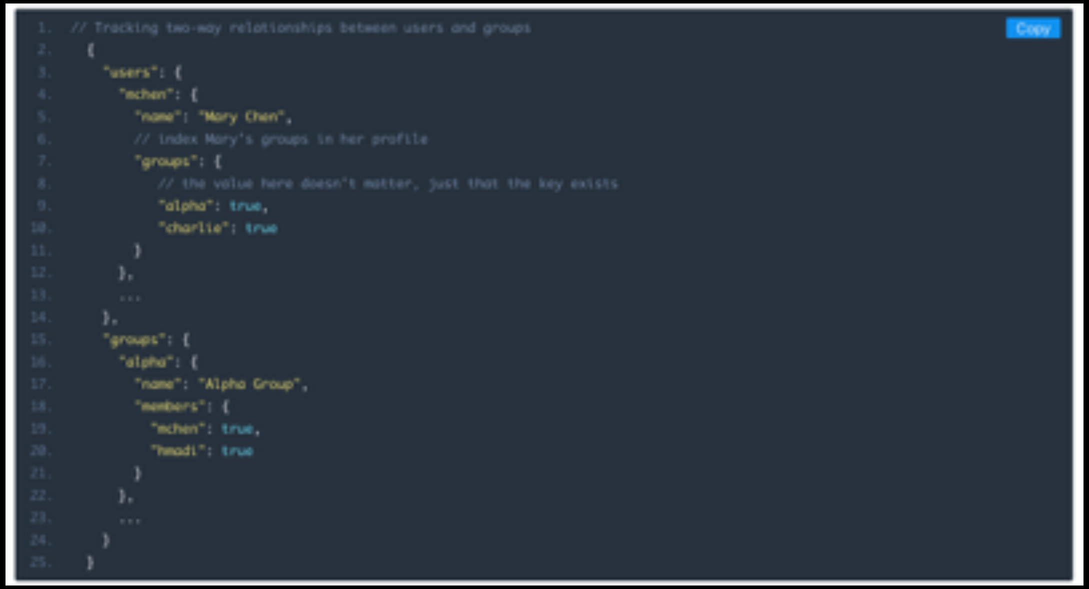
ANTIPATTERN: This is not a recommended practice

```
// A first attempt at a two-way relationship
{
  "users": {
    "mchen": { "name": "Mary Chen" },
    "brinchen": { "name": "Byambyn Rinchen" },
    "hmodi": { "name": "Hamadi Modi" }
  },
  "groups": {
    "alpha": {
       "name": "Alpha Tango",
       "members": {
          "m1": "mchen",
          "m2": "brinchen",
          "m3": "hamadi"
       }
    },
    "bravo": { ... },
    "charlie": { ... }
  }
}
```

This approach is poor because it requires iterating through every group to determine if some user exists in that group

NoSQL 2-Way Relationships

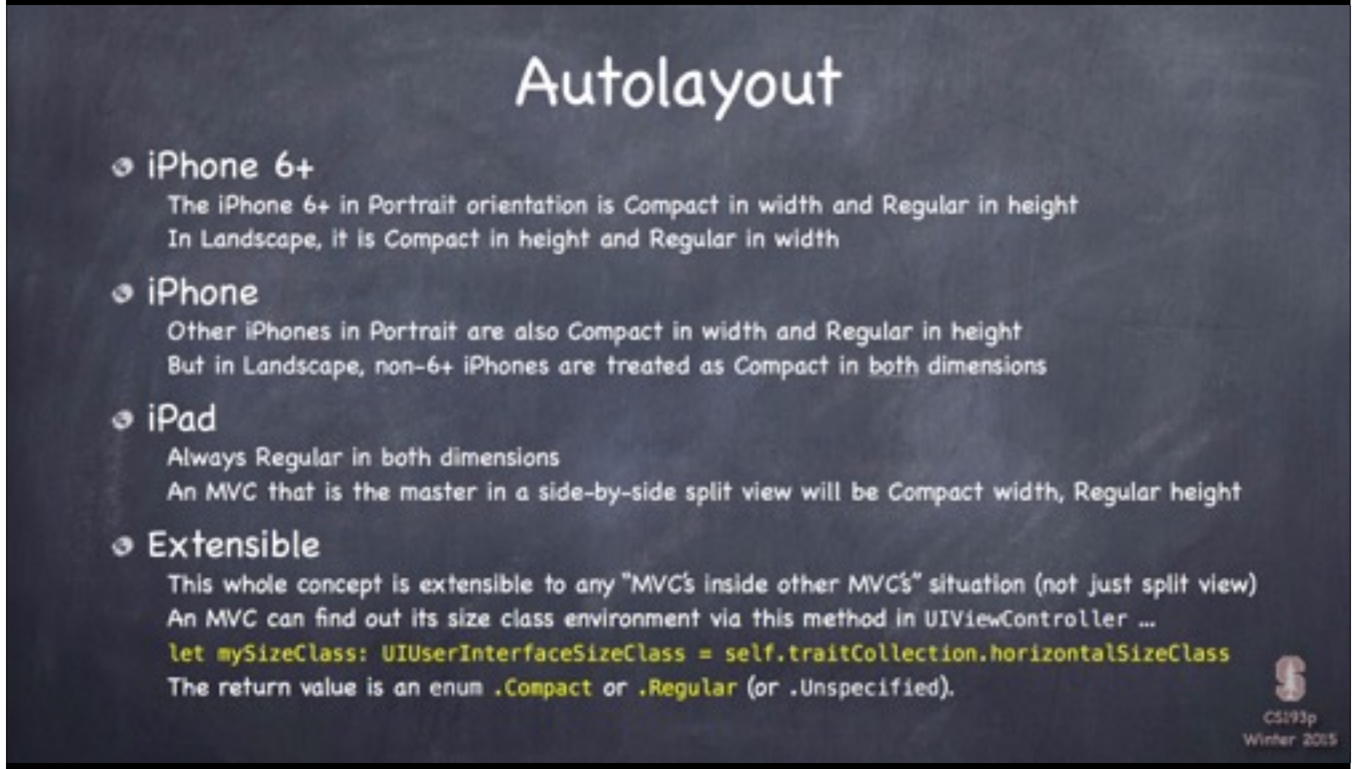
- Need an elegant way to list the groups that Mary belongs to without fetching or looking through all of the groups
- Scalable approach is to use an index, or list of keys, which refer to Mary's groups



Copy

```
// Tracking two-way relationships between users and groups
{
  "users": {
    "mchen": {
      "name": "Mary Chen",
      // index Mary's groups in her profile
      "groups": {
         // the value here doesn't matter, just that the key exists
         "alpha": true,
         "charlie": true
      }
    },
    ...
  },
  "groups": {
    "alpha": {
      "name": "Alpha Group",
      "members": {
        "mchen": true,
        "hmadi": true
      }
    },
    ...
  }
}
```

This redundancy is necessary for two way relationships, it allows us to quickly and efficiently fetch Mary's memberships even when the list of groups grows into the millions. It does mean we have to write to both locations when a membership changes.



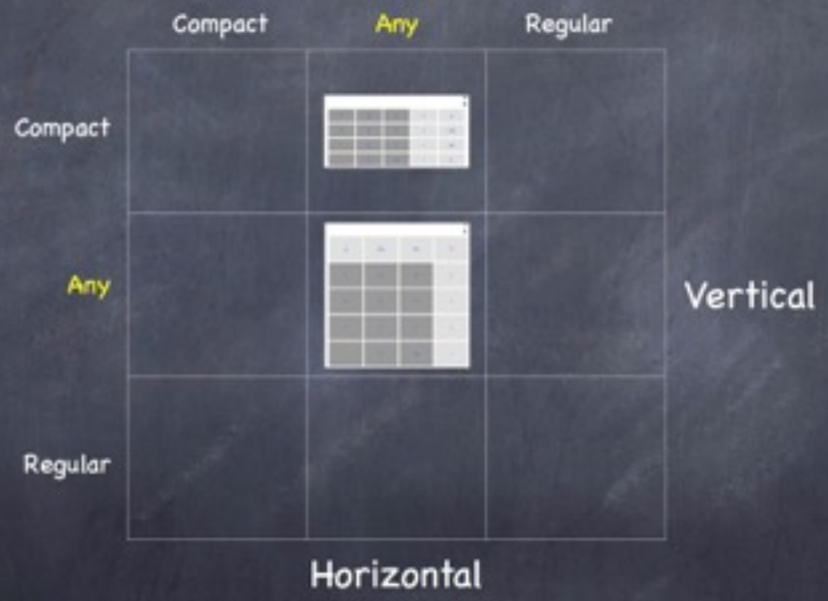
Autolayout

- iPhone 6+
  - The iPhone 6+ in Portrait orientation is Compact in width and Regular in height
  - In Landscape, it is Compact in height and Regular in width
- iPhone
  - Other iPhones in Portrait are also Compact in width and Regular in height
  - But in Landscape, non-6+ iPhones are treated as Compact in both dimensions
- iPad
  - Always Regular in both dimensions
  - An MVC that is the master in a side-by-side split view will be Compact width, Regular height
- Extensible
  - This whole concept is extensible to any "MVC's inside other MVC's" situation (not just split view)
  - An MVC can find out its size class environment via this method in `UIViewController` …

```
let mySizeClass: UIUserInterfaceSizeClass = self.traitCollection.horizontalSizeClass
```

The return value is an enum `.Compact` or `.Regular` (or `.Unspecified`).

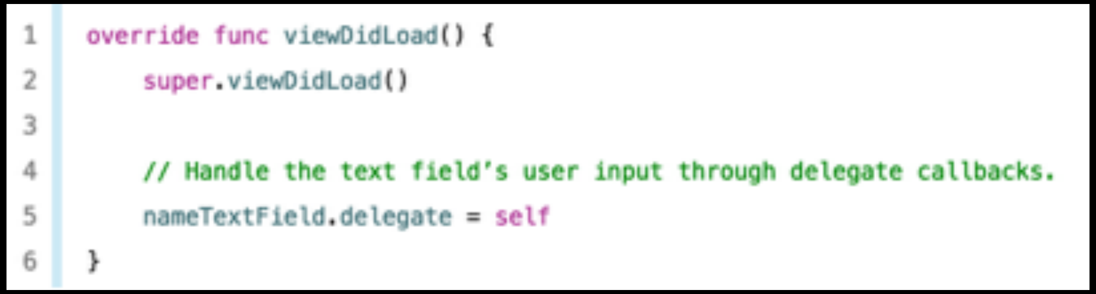
Size Classes



CS193p
Winter 2015

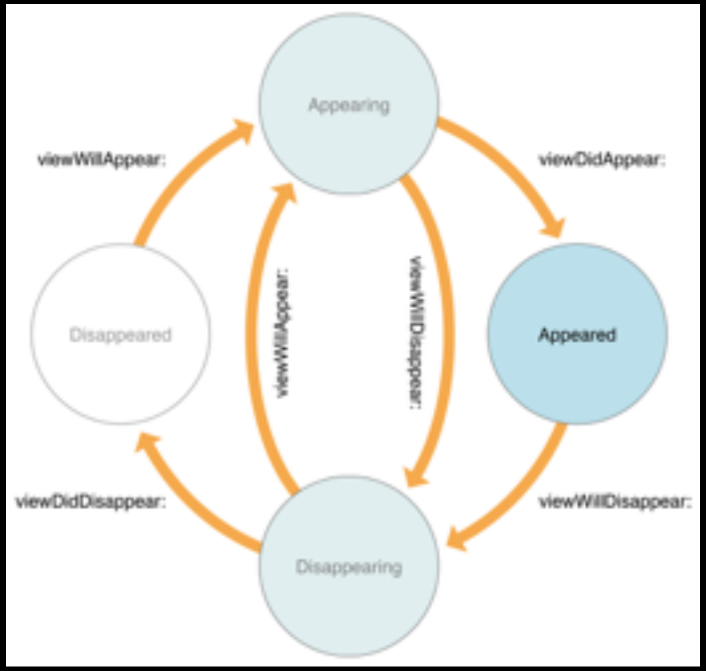
View Controller Lifecycle

- UIViewController objects inherit a set of methods that manage its view hierarchy
- They are called by iOS during transitions.
  You do NOT call them yourself.



```
override func viewDidLoad() {
    super.viewDidLoad()

    // Handle the text field's user input through delegate callbacks.
    nameTextField.delegate = self
}
```

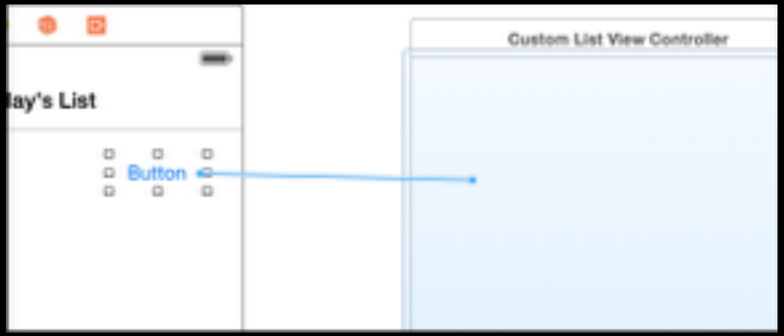
View Controller Lifecycle

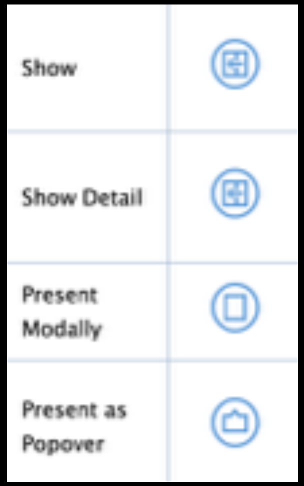


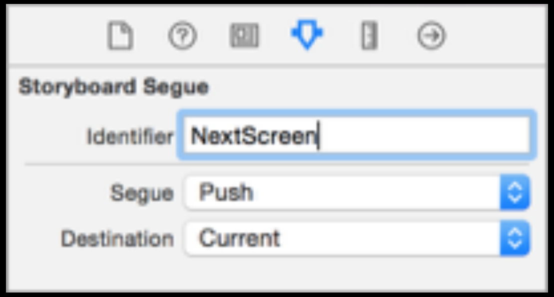
View Controller Lifecycle

- viewDidLoad()—Called when the view controller's content view (the top of its view hierarchy) is created and loaded from a storyboard. Mostly called only once, but not guaranteed.
- viewWillAppear()—Intended for any operations that you want always to occur before the view becomes visible. Called immediately before the content view appears onscreen.
- viewDidAppear()—Intended for any operations that you want to occur as soon as the view becomes visible, such as fetching data or showing an animation.

Segues







Storyboard Segue

Identifier: NextScreen

Segue: Push

Destination: Current

Segues

- During a segue, UIKit calls methods of the current view controller to give you opportunities to affect the outcome of the segue.
- The prepareForSegue:sender: method of the source view controller lets you pass data from the source view controller to the destination view controller. The UIStoryboardSegue object passed to the method contains a reference to the destination view controller along with other segue-related information.

UIImageView

- A view-based container for displaying an image
- self.image to access the UIImage
- self.contentMode to set how the image is fit into the view
    - UIViewContentMode.ScaleAspectFit
    - .ScaleAspectFill
    - .ScaleToFill

CLLocationCoordinate2D

- Structure made up of:
    - latitude: double
    - longitude: double

MKMapView

- Provides an embeddable map interface, similar to Maps.app
- Manipulating
  - Display coordinate
    - setCenterCoordinate:animated:
  - showsTraffic, showsBuildings, showsCompass…
  - zoomEnabled, scrollEnabled

Location Services

- CLLocationManager
  - The central point for delivering location and heading details to your app

CLLocationManager

1. Check manager.locationStatus
    - .NotDetermined = user has not given permission
        - Call manager.requestWhenInUseAuthorization
2. Set manager.desiredAccuracy
    - .TenMeters, .Kilometer, .Best…
3. Call manager.startUpdatingLocation
    - Location comes in via didUpdateLocation delegate method