# {app}etite

## It’s like Tinder, but for food

A Capstone Project by JT Seger & Brittney Allred

# The Idea

- The concept of {app}etite is two-fold.
  - I’m in a new area and want something to eat RIGHT NOW.
  - I want to use the app to see what’s around me for future meals.
- Just as Tinder is used primarily for searching visually, so is {app}etite.
- An interactive app that seems like playing as opposed to searching.

# Design Goals

- Build a separate front-end for restaurant users to post dishes, update info, and set location.
- Required features for restaurants:
  - Create a visually appealing profile
  - Set location to be found by users
  - Create dish posts
  - Track how many times their posts have been shown and matched by users

# Design Goals

- Build an app mirroring Tinder’s visual-based front-end.
- Required features for diners:
  - Swiping for matches based on image of a dish
  - View information on matched restaurants
  - Set distance of potential restaurant matches
  - Keep history of all “Dish Matches” for a user
  - View restaurant location on a map
  - Take advantage of special offers through the app

# Walkthrough

Hold onto your seats ladies and gentleman.

# Tools

- Swift is a new programming language for iOS and OS X apps that builds on the best of C and Objective-C, without the constraints of C compatibility.
- Xcode is standard iOS IDE
- Github allowed us to work independently, while both contributing to the same code base
- Parse.com

# Implementing Design

- Created backgrounds, buttons, logos and icons using the Adobe Creative Cloud
- Everything was converted into a .png and uploaded into Images.xcassets folder
- Problems

# Implementing Design

- Main storyboard in xcode allowed for dragging and dropping objects
- Auto-Layout:
  - Want {app}etite to work on any iOS device
  - Applied constraints to each component.
  - Ensures that no matter the screen size, everything stays lined up relative to the sides of the screen and in relation to other components.
- Walkthrough Xcode

# Checking out the Data

- How did we store our data? Why?
- Retrieving nearby posts is the most data-consuming operation within {app}etite.
- Solution - Build up a table of restaurant pointers and their locations as GeoPoints.

# Parse.com

https://www.parse.com/apps

# Most difficult challenge?

```
override func viewDidLoad()  {
    super.viewDidLoad()
    self.refreshControl?.addTarget(self, action: "refresh:", forControlEvents: UIControlEvents.
        ValueChanged)
    self.navigationController?.navigationBarHidden = false
    var relation = user.relationForKey("PostList")
    relation.query().findObjectsInBackgroundWithBlock {
        (Posts: [AnyObject]!, error: NSError!) -> Void in
        if error != nil {
            // There was an error
            self.displayError("Failed to Find Posts", error: "Please check your connection")
        } else {
            //success
            self.posts = [PFObject]()
            for post in Posts{
                self.posts.append(post as! PFObject);
            }
        }
        self.numRows = self.posts.count
        self.tableViewPosts.reloadData()
    }
}
```

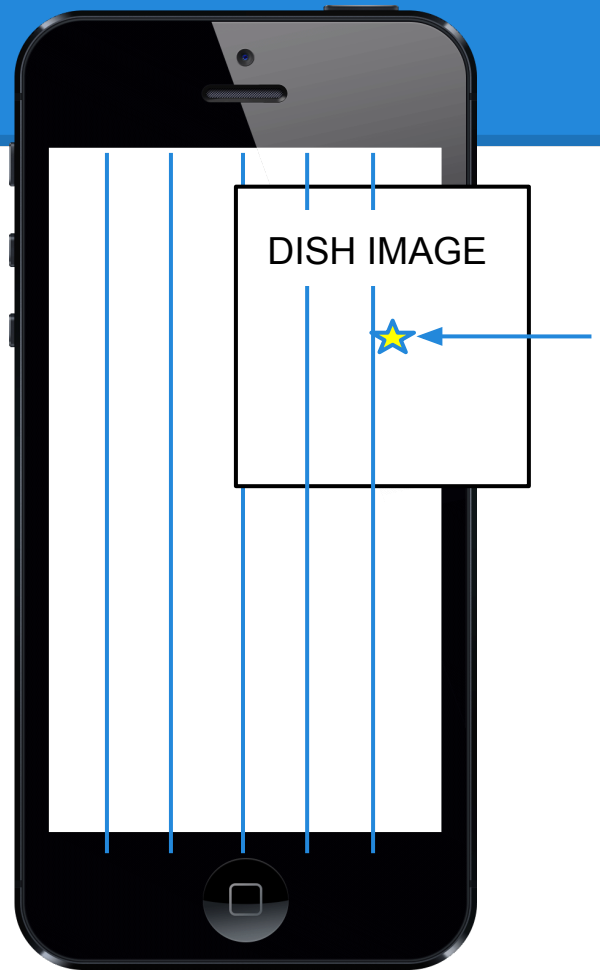
# Populating the UITableView

```
    override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
        UITableViewCell {
        var cell = tableView.dequeueReusableCellWithIdentifier("myCell", forIndexPath: indexPath)
            as? PostsTableViewCell
        if ( cell == nil){
            var cell = PostsTableViewCell(style: UITableViewCellStyle.Default, reuseIdentifier:
                "myCell") as PostsTableViewCell;}
        var post = self.posts[indexPath.row]
        cell!.nameOfDish.text = post["DishName"] as? String
        cell!.priceOfDish.text = post["price"] as? String
        var imageFile = post["imageFile"]as! PFFile!
        imageFile.getDataInBackgroundWithBlock{
            (imageData: NSData!, error: NSError!) -> Void in
            if (error == nil){
         let image = UIImage(data: imageData)
                cell!.cellImage.image = image;
            }
        }
            return cell!
    }
}
```

# Creating the Matching Page

1. Query for user's past matches and the nearby restaurants concurrently.
2. Once this is finished, retrieve the list of dish posts for each restaurant nearby.
3. For each post retrieved that has **not** already been matched with, we append to a list of postsToDisplay.
4. Once postsToDisplay is fully populated, the view is ready to be loaded...
    a. if dishImage == nil && index < postsToDisplay.length()
        i. new dishImage == postsToDisplay[index];
5. If match occurs, save match, increment index++, and remove image.
    a. Revert to step 4.
6. If no more postsToDisplay, add placeholder image.

# How Swiping Works




if image.center.x > self.width - (self.width / 6)
then CHOSEN!

else if
image.center.x < self.width - (self.width * (⅚))
then NOT CHOSEN!

# Future Plans

- Coupon feature
  - After matching, a coupon could be made available to be redeemed at the restaurant.
- Map of user location relative to restaurant
  - If map clicked, open up maps app with directions
- Restaurant card pop-up upon matching
  - Would you like to view this restaurant now? Yes/No
- Ready for App Store Debut!

# Questions?