MAY 2, 2016

BIDRUFF

BRAXTON'S BOUNTY FOR BONES

RYAN KRUEGER & BEN SHARP
BUTLER UNIVERSITY – EPICS 2016 – PANOS LINOS

TABLE OF CONTENTS

Summary:

We were given the task of designing a silent auction mobile application that would serve Braxtyn's Bounty for Bones in future auctions to raise money for their cause. The goal of the team was to design a multi-step Silent Auction application on iOS that would allow the organization to have a safe and reliable means to raise money for their non-profit. Braxtyn's Bounty's funds are donated to foundations whose goals include eliminating canine cancer through education, outreach and research. Their hope is that their efforts will help find cures and better treatments to minimize the number of dogs suffering from cancer.

Chapter 1: Introduction

The Organization

Putt from the Ruff is an annual event hosted by Braxton's Bounty for Bones. "Putt from the Ruff was created by Braxtyn's family to raise awareness and money for the National Canine Cancer Foundation as well as local animal not-for-profits. One out of three dogs will get cancer. Our goal is to not only make this statistic a thing of the past, but information gathered researching canine cancer can aid in humans battle with cancer, as well. We also have a list of items that anyone (participating or not) can donate to be given to the local charities to aid in their quest of making animals' lives better." (puttfromtheruff.com)

A large portion of the event and the source of most of the funds that they raise is a silent auction with donated goods.

The Request

Braxton's Bounty for Bones challenged the EPICS team with creating a mobile application so that during the event, bidding can take place electronically. Silent auction applications have also been shown to increase money generated by about 30-50% compared to traditional silent auctions. Existing silent auction applications that are available on the market are far too expensive for small organizations such as Bones; therefore, the EPICS project at Butler was deemed a good fit. Considering previous attempts (such as using Facebook) to move the auction online failed, anything would be an improvement. In the future, there's also hope that they could utilize the application to allow them to host auctions during other times in the year without having a physical event. The application would allow for a silent auction to be open for a predetermined amount of time, register users quickly, view items, place bids, have a starting bid and set bid increment, as well as back end management tools.

The Plan and Process

The Fall 2015 semester is when this project was originally brought to EPICS at Butler University. Previous team members included Alex Steel, Gwen Kozak, Dennis Drew, and Harrison Lingren. They found the BidHub open source code (http://www.bidhub.org) and implemented the Cloud Code on Parse which serves as the app's backend and database. They also modified the android app just enough to work with the Parse server. Unfortunately, though, they did not have the means to create an iPhone app - which was the client's' highest priority due to the large amount of users with iPhones.

This semester's team included Ryan Krueger and Ben Sharp. (Another student was initally part of the team, but left very early in the semester to pursue another EPICS project.) Since both of us had iPhones and Mac computer with XCode and, more importantly, an aptitude to learn and code with Swift. After learning about the organization, both Ben and Ryan knew they had found the right project. While neither of them actually had done any coding in XCode before which posed a serious challenge, they both knew that the project would likely go inactive for the semester if they didn't sign up for that challenge. This

then posed another challenge on figuring out where the last group left off, and how to make our app work with the Android app and Parse Database the previous group started.

Thus, the first task was to meet with the previous group and figure out where they left off. Ryan met with Harrison who provided a useful insight in the project and advice on where to concentrate efforts.

After meeting with the clients, the next task was for both Ben and Ryan to self learn Swift and XCode development. This was far from an easy task and took two weeks just to get antiquated with the programming language.

The next task was to update the open source code as it was written in Swift version 1 and would not compile for newer versions of iOS such as version 9. If not addressed, it would be impossible to list the app in the Apple AppStore as well.

After updating the source code, the application was now fully functioning and worked with the Parse database. This was a huge step for the team. The next step was to remove the branding of the open source code's company and incorporate the Braxton's Bounty for Bones's colors and branding.

At this point, the semester began to wind down. The next feature to be implemented was push notifications, which proved to be tricky with regards to iOS. The team was only able to put in time to research and plan out this requirement, but never made it to the implementation stage.

Regardless, for a team without previous experience with iOS development, the semester was a huge success and the clients were thrilled to have a functioning iOS application running on their iPhones.

The Participants

Ryan Krueger (team leader) is a junior studying Computer Science and Electrical Engineering in the Dual Degree Engineering Program at Butler University. Ryan is involved in many organizations on campus including Delta Tau Delta Fraternity, the Student Government Association and the Engineering Club. He is more experienced with coding and working with outside organizations, so Ryan was the natural choice for Team Leader. Ryan is excited to be able to work with Braxtyn's Bounty for Bones and help them accomplish their goals.

Ben Sharp is a junior Actuarial Science and Computer Science student at Butler University. Ben enjoys volunteering and working with others to better the community. Ben is involved in Delta Tau Delta Fraternity and Butler University Dance Marathon. He is excited to gain real world experience and work to help such a great cause in Braxtyn's Bounty for Bones this semester.

Included in this report is a multi-tiered architecture on how we developed our knowledge base, and how we worked through the problem statement.

CHAPTER 2: REQUIREMENTS SPECIFICATIONS

The client requested an application that would conduct the silent auction for them. Functional requirements included a login for both users and administrators to keep the application secure and user specific. Additionally, each auction item would be displayed complete with picture, description, donor, starting price, required bid increment, current bid, an option to place a new bid, and push notifications when one has been outbid or has won an auction. A requested, but not required, functional feature was the ability to pay for the item on the app; a concern was making sure that the application was secure enough to store and use credit card information.

Non-functional requirements include a cloud database managed by Parse to store all of the auction items as well as each individual item's information. The multi-platform design allows for variety. We chose to develop on iOS because of the client's preference.

CHAPTER 3: ARCHITECTURE

The program is broken down into four main sections. The cloud code backend, the iOS application, the Android application, and the Admin web panel backend. Both the iOS and Android apps are backed by the Parse database. Parse is a free multi-purpose tool for handling the database, while allowing the ability to add server-side logic when actions are taken. After editing the config/global.json file to include Application Name, Application ID and the Master Key found in Parse, Settings > Keys, and installing the Parse command line tools to allow for computer to Parse interaction, the back-end cloud is ready for use.

In order to start working, you need to *Parse deploy* the files to Parse. This will push the cloud/main.js to Parse, allowing you to view it by going to Parse, Core > Cloud Code. In order to run your first job, Initialize the auction by going to Core > Jobs and schedule a job.

Once there is a populated test object item, the items can be edited to create the backend database. The easiest way to add an item is directly from Parse by going to Core > Data > Item and add one single item, row by row, or by using +CSV import.

ANDROID PORTION

After the backend cloud database was established and set-up through Parse, the android repository was cloned and imported into Android Studio. At the core of Android Studio is an intelligent code editor capable of advanced code completion, refactoring, and code analysis. The editor helps you be a more productive Android app developer. Again, the Application ID and Client Key must be taken from Parse. This can be done by going to Parse > Settings > Keys. In the AuctionApplication.java, set APP_ID and CLIENT_KEY to the Parse values. Some of the important things to note: in the folder drawable… /notificationicon.png /appicon.png and /bg.png have all of the specific assets for push notifications, app icon, and background.

iOS Portion

The iOS version is done similarly to the Android version. After the backend cloud database was established and set-up through Parse, the iOS repository was cloned and imported into Xcode. At the core of Xcode is an intelligent code editor capable of advanced code completion, refactoring, and code analysis. The editor helps you be a more productive Apple app developer. Again, the Application ID and Client Key must be taken from Parse. This can be done by going to Parse > Settings > Keys. In the AuctionApplication file, set APP_ID and CLIENT_KEY to the Parse values.

Chapter 4: Design

Android Portion

The application has a simple design with a dark background that has been adjusted for most new end Android devices. While working in Android Studio, there may be certain phones where the format needs to be changed, but that is done when rendering the apk to put on your phone. We discovered that the recommended screen size was imperfect when testing the application on our phones; after toggling this setting on Android Studio, it was easily fixed.

The app icon is the logo for Braxtyn's Bounty for Bones. When opened, there is a prompt for a user to enter their name and email, clicking submit to proceed into the app. This will push to the Parse database and log the users onto the application. There is not a real time feature to display users. Next, a screen will display all items available for bidding.

A menu option is displayed at the top of the screen that when selected displays options to see a user's account information, view the bids that the user has won, as well as all of the user's current bids. Everything is real-time, but a refresh button is present for manual refreshing as well.

Each item page displays the name and picture of the item available. Below the image, the donor name, bid value, number of bids, and item description is listed. All information can be edited through Parse or loaded onto the data script. Once items are loaded into Parse on the cloud database, all available items can be easily scrolled through consecutively. Android Studio handles all errors.

iOS Portion

Much of the design elements were handled in the open source code provided. However, many changes to this code were made in order to brand the application for the Braxton's Bounty for Bones. Colors were changed to be on brand with the organizations. Likewise, the app name was rebranded from BidHub to BidRuff. Most of the rebranding effort was placed on the login screen, however. As this screen is shown during every use of the application, it was deemed important to make sure that the organization's name was shown and that it had a fun feel. A looping video of a dog panting was chosen as the background to meet that requirement.

Chapter 5: Quality Assurance & Testing

Since the group had minimal knowledge and experience programming mobile applications; therefore, a fair amount of time was spent learning how they work. We had to sift through code that had been implemented wrong to find a starting point in our development. HubSpot was the remedy to this problem as they had developed a free open-source auction system that needed minimal tweaking to get it in functional order.

Overall, most of our product testing was debugging the material at hand. Considering that we used a working open-source program that had already been debugged, small tweaks to ensure that our product worked and suited the client's needs were the only adjustments necessary after discovering BidHub. A majority of time was spent working with the program to make sure we understood its full potential to create the best and most user friendly product for the client.

Chapter 6: Project Organization & Management

The team is constructed of two individual members: Ryan Krueger the team lead and Ben Sharp. We decided it would be best for the team members to not define roles, but instead work collaboratively on the project. Ryan worked on primary programming and designing, while Ben gathered information and worked on the documentation. Most of the work was divided by the week and technically necessity. The group tried to meet at least once a week for at minimum a check-in to determine accomplishments and adjust goals. Work was conducted outside of team meetings at individuals' own preferred pace. The main tools used were Parse, which handled the backend cloud database, and XCode, which which worked with the packages and application itself. To see how the program operates, please see **Chapter 3.**

Weekly Status Reports (WSRs):

WSR-4-26-16
WSR-4-19-16
WSR-4-12-16
WSR-4-5-16
WSR-3-29-16
WSR-3-22-16
WSR-3-15-16
WSR-3-1-16
WSR-2-23-16
WSR-2-16-16
WSR-2-9-16
WSR-2-2-16
WSR-1-26-16
WSR-1-19-16

Chapter 7: Future Work

The primary goals for the next team should be push notifications, billing functionality and uploading the app to the App Store. Push notifications are the big thing that needs to be added to the application. We were unable to get these to work due to the fact that we ran out of time in the semester to work on the application. Push notifications are the number one requirement on the client's list.

Also, the application needs to be taken off of Parse because it shutting down in January 2017. The next group will need to find a new server to handle the backend cloud database.

Additionally, the next team will need to work on a secure way to handle payments in the app. This was not required but it was a request by the client. If the application could handle payments it would make the lives of the clients so much easier. Billing functionality would be a nice perk and bonus for the clients if it were to be implemented. We recommend checking out Stripe.

Finally, one of the most important requirements is to upload the app to the App Store. One of the team members will have to get an Apple developer account and the clients have said they will foot the bill for any fees that are necessary.

Parse Login: jocelyn408@gmail.com

Parse Pass: Epics2015

References/Bibliography

http://product.hubspot.com/blog/building-an-auction-app-in-a-weekend

http://www.bidhub.org

https://github.com/HubSpot/BidHub-CloudCode

https://github.com/HubSpot/BidHub-Android

https://github.com/HubSpot/BidHub-iOS

https://github.com/HubSpot/BidHub-WebAdmin

https://parse.com/docs

https://parse.com/docs/rest/guide

https://play.google.com/apps/testing/edu.butler.epics.auction

APPENDIX

SOURCE CODE

Below are some of the main files and classes of our source code.

https://github.com/harrisonlingren/EPICSFall15

https://github.butler.edu/rkrueger/Bones-iOS

CLOUD CODE BACKEND

Cloud > Main.js

```
// Utility to get items unique to either array.
Array.prototype.diff = function(a) {
        return this.filter(function(i) {return a.indexOf(i) < 0;});
};

// This code will be run before saving a new bid.
Parse.Cloud.beforeSave("NewBid", function(request, response) {

        currentBid = request.object;

        // Grab the item that is being bid on.
        itemQuery = new Parse.Query("Item");
        itemQuery.equalTo("objectId", request.object.get("item"));
        itemQuery.first({
                success: function(item) {

                        if (!item)
                         return;

                        var date = Date.now();

                        // Make sure bidding on this item hasn't closed.
                        if (date > item.get("closetime")) {
                                response.error("Bidding for this item has ended.");
                                return;
                        }

                        // Make sure the bid isn't below the starting price.
                        if (currentBid.get("amt") < item.get("price")) {
                                response.error("Your bid needs to be higher than the
item's starting price.");
                                return;
```

```
                                        }

                                        // Make sure the bid increments by at least the minimum
increment
        minIncrement = item.get("priceIncrement");
        if (!minIncrement) {
                minIncrement = 1;
        }
                                        if (currentBid.get("amt") < (item.get("currentPrice") +
minIncrement )) {
                                                response.error("You need to raise the current price by
at least $" + minIncrement);
                                                return;
                                        }

                                        // Sanity check. In-house testing revealed that people love
bidding one trillion dollars.
                                        if (currentBid.get("amt") > 99999) {
                                                response.error("Remind me to apply for your job.");
                                                return;
                                        }

                                        // Retrieve all previous bids on this item.
                                        query = new Parse.Query("NewBid");
                                        query.equalTo("item", request.object.get("item"));
                                        query.descending("amt", "createdAt");
                                        query.limit = 1000; // Default is 100
                                        query.find({
                                          success: function(allWinningBids) {

                                                        item.set("numberOfBids", allWinningBids.length
+ 1);

                                                var quantity = item.get("qty");
                                                var currentPrice = [];
                                                var currentWinners = [];
                                                var previousWinners = item.get("currentWinners");

                                           var allBidders = item.get("allBidders");
                                           if (!allBidders) {
                                                        allBidders = [];
                                           }

                                                // Build an object mapping email addresses to their
highest bids.
                                           var bidsForEmails = {};
                                           allWinningBids.forEach(function(bid) {
                                                                var curBid =
bidsForEmails[bid.get("email")]
                                                                if (curBid) {
```

```
                                                bidsForEmails[bid.get("email")]
= (curBid.get("amt") > bid.get("amt") ? curBid : bid);
                                        }
                                        else {

        bidsForEmails[bid.get("email")] = bid;
                                        }
                                });

                        // Get this bidder's last bid and make sure the new bid is
an increase.
                                // If the new bid is higher, remove the old bid.
                        var previousMaxBid = bidsForEmails[currentBid.get("email")];
                        if (previousMaxBid) {
                                if (currentBid.get("amt") <=
previousMaxBid.get("amt")){
                                        response.error("You already
bid $" + previousMaxBid.get("amt") + " - you need to raise your bid!");
                                        return;
                                }
                                else {
                                delete
bidsForEmails[currentBid.get("email")];
                                }
                        }

                        // Build an array of all the winning bids.
                        allWinningBids = [];
                        for (var key in bidsForEmails) {
                                allWinningBids.push(bidsForEmails[key]);
                        }

                                // Add the new bid and sort by amount,
secondarily sorting by time.
                        allWinningBids.push(currentBid)
                        allWinningBids = allWinningBids.sort(function(a, b){
                                var keyA = a.get("amt");
                                var keyB = b.get("amt");

                                        // Sort on amount if they're
different.
                                        if (keyA < keyB) {
                                                return 1;
                                        }
                                        else if (keyA > keyB) {
                                                return -1;
                                                }

                                var dateKeyA = a.get("createdAt");
                                var dateKeyB = b.get("createdAt");
```

```
                                        // Secondarily sort on time if
the amounts are the same.
                                        if (dateKeyA < dateKeyB) {
                                                return 1;
                                        }
                                        else if (dateKeyA > dateKeyB) {
                                                return -1;
                                                }

                                        return 0;
                                });

                        // Slice off either the top n bids (for an item where the
highest n bids win)
                                        // or all of them if there are fewer than n bids.
                        var endIndex = 0;
                        if (quantity > allWinningBids.length) {
                                        endIndex = allWinningBids.length;
                        }
                                else {
                                        endIndex = quantity;
                        }

                                        var newBidIsWinning = false;
                var currentWinningBids = allWinningBids.slice(0, endIndex);

                // If the new bid is in the list of winning bids...
                if (currentWinningBids.indexOf(currentBid) != -1){
                                        newBidIsWinning = true;

                                                // Update the item's current price and
current winners.
                                                for (var i = 0; (i <
currentWinningBids.length); i++) {
                                                        var bid =
currentWinningBids[i];
        currentPrice.push(bid.get("amt"));
        currentWinners.push(bid.get("email"));
                                                }

                                        // Add this bidder to the list of all bidders...
                                        allBidders.push(currentBid.get("email"));

                                                // ...and remove them if they're already
there.
                                        var uniqueArray =
allBidders.filter(function(elem, pos) {
```

```
                                                                return allBidders.indexOf(elem)
== pos;
                                                        });

                                                item.set("allBidders", uniqueArray);
                                                        item.set("currentPrice", currentPrice);
                                                        item.set("currentWinners",
currentWinners);
                                                        item.set("previousWinners",
previousWinners)

                                                        // Save all these updates back to the
Item.
                                                        item.save(null, {
                                                                success: function(item) {

        response.success();
                                                                },
                                                                error: function(item, error) {

        console.error(error);

        response.error("Something went wrong - try again?");
                                                                }
                                                        });
                                        }
                                                // If it's not, someone else probably outbid you in the
meantime.
                                        else {
                                                        response.error("Looks like you've been outbid!
Check the new price and try again.");
                                                        return;
                                        }


                                         },
                                         error: function(error) {
                                                console.error("Error: " + error.code + " " +
error.message);
                                                response.error("Error: " + error.code + " " +
error.message);
                                         }
                                        });

                        },
                        error: function(error) {
                                console.error("Error: " + error.code + " " + error.message);
                                response.error("Error: " + error.code + " " + error.message);
                        }
                });
```

```
});

// This code is run after the successful save of a new bid.
Parse.Cloud.afterSave("NewBid", function(request, response) {

		currentBid = request.object;

		// Get the item that's being bid on.
		itemQuery = new Parse.Query("Item");
		itemQuery.equalTo("objectId", request.object.get("item"))
		itemQuery.first({
			success: function(item) {

					var previousWinners = item.get("previousWinners");

					// For multi-quantity items, don't bother the people "higher"
than you
					// ex: don't send a push to the person with the #1 bid if
someone overtakes
					// the #2 person.
					var index = previousWinners.indexOf(currentBid.get("email"));
					if (index > -1) {
						previousWinners.splice(index, 1);
					}

					// Grab installations where that user was previously a winner
but no longer is.
					var query = new Parse.Query(Parse.Installation);
					query.containedIn("email",
previousWinners.diff(item.get("currentWinners")));

					// We'll refer to the bidder by their name if it's set...
					var identity = currentBid.get("name").split("@")[0];

					// ...and their email prefix (ex. jtsuji@hubspot.com -> jtsuji) if it's
not.
					if (identity.length < 3) {
						identity = currentBid.get("email").split("@")[0];
					}

					// Fire the push.
					Parse.Push.send({
					  where: query,
					  data: {
						alert: identity + " bid $" + currentBid.get("amt") + " on "
+ item.get("name") + ". Surely you won't stand for this.", // People like sassy apps.
						itemname: item.get("name"),
						personname: identity,
						itemid: item.id,
```

```
                                        sound: "default",
                                        amt: currentBid.get("amt"),
                                        email: currentBid.get("email")
                                    }
                                }, {
                                    success: function() {
                                        console.log("Pushed successfully.")
                                    },
                                    error: function(error) {
                                        console.error("Push failed: " +error)
                                    }
                                });

                        },
                        error: function(error) {
                                console.error("Push failed: " +error)
                        }
                });

});

// Sets up all the tables for you.
Parse.Cloud.job("InitializeForAuction", function(request, status) {
                Parse.Cloud.useMasterKey();

                // Add a test item.
                var Item = Parse.Object.extend("Item");
                var item = new Item();
                item.set("name", "Test Object 7");
                item.set("description", "This is a test object, and you (probably) won't be asked to
donate your bid on this item to charity. Who knows, though.");
                item.set("donorname", "Generous Donor");
                item.set("price", 50);
        item.set("priceIncrement", 1);
                item.set("imageurl", "http://i.imgur.com/kCtWFwr.png");
                item.set("qty", "3");
                item.set("currentPrice", []);
                item.set("numberOfBids", 0);
                item.set("allBidders", []);
                item.set("currentWinners", []);
                item.set("previousWinners", [])
                item.set("opentime", new Date("Dec 05, 2014, 05:00"));
                item.set("closetime", new Date("Dec 06, 2015, 05:00"));
                item.save(null, {
                        success: function(item) {
                                        var NewBid = Parse.Object.extend("NewBid");
                                        var bid = new NewBid();
                                        bid.set("item", "");
                                        bid.set("amt", 0);
                                        bid.set("email", "");
```

```
                                        bid.set("name", "");
                                        bid.save(null, {
                                                success: function(bid) {
                                                        console.log("Initialization complete.");
                                                },
                                                error: function(bid) {
                                                        console.log("Initialization complete.");
                                                }
                                        });
                        },
                        error: function(item, error) {
                                        console.error("Unable to initialize Item table. Have you set your
application name, app ID, and master key in config/global.json?")
                        }
                });


});
```

Web Admin app:

Index.html

```
<!DOCTYPE html>
<html ng-app="auctionApp">
<head lang="en">
        <meta charset="UTF-8">
        <link href='http://fonts.googleapis.com/css?family=Roboto:400,300,100,500' rel='stylesheet'
type='text/css'>
                <link href="style.css" rel="stylesheet" type="text/css">
        <script src="http://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
        <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.0/angular.min.js"></script>
        <script src="controllers.js"></script>
        <script src="client.js"></script>
   <title>Manage Auction</title>

        <script>
        function showImg() {
        $("#newimg").attr("src", $("#imgurl").val());
        }

                        function deleteItem() {

                                        $.ajax({
                                                type: "",
                                                url: "https://api.parse.com/1/classes/Item",
                                                beforeSend: function(xhr){
                xhr.setRequestHeader('X-Parse-Application-Id',
'WhkQetI8nb0HrIykoaNc8LJ9fIHIxOvgaXhFXFxm');
                xhr.setRequestHeader('X-Parse-REST-API-Key',
'G02koccgg9q6RzqwRmpiQDx3QIlASet5iW2XbLob');
```

```
            },
                                                    data: ,
                                                    success: function () {
            location.reload();
            },
        contentType:"application/json; charset=utf-8",
        dataType:"json"
                                            });
                    }

    function createItem() {

    var newItem = JSON.stringify(
            {
            "name": $("#itemname").val(),
            "donorname": $("#donorname").val(),
            "price": parseInt($("#price").val()),
            "imageurl": $("#imgurl").val(),
            "description": $("#descriptionbody").val()
            }
    );

    $.ajax({
            type: "POST",
            url: "https://api.parse.com/1/classes/Item",
        beforeSend: function(xhr){
            xhr.setRequestHeader('X-Parse-Application-Id',
'WhkQetI8nb0HrIykoaNc8LJ9flHIxOvgaXhFXFxm');
            xhr.setRequestHeader('X-Parse-REST-API-Key',
'G02koccgg9q6RzqwRmpiQDx3QIlASet5iW2XbLob');
            },
            data: newItem,
        success: function () {
            location.reload();
             },
        contentType:"application/json; charset=utf-8",
        dataType:"json"
    });
    }
    </script>
</head>
<body ng-controller="ItemCardsCtrl">
<div id="container">
    <div id="header">
    AUCTION MONITORING
    </div>
    <div id="content">

    <div class="card left">
    <div class="header" style="background: #999"></div>
```

```
<div style="height: 3px; background: #777"></div>
<div style="padding: 25px; color: #fff;">
   <table cellPadding="0" cellspacing="10">
      <tr>
         <td style="text-align: right">
            <span class="donor">Raised so far </span></td>
       <td><span class="title">{{totalRaised | noFractionCurrency}}</span></td>
      </tr>
      <tr>
         <td style="text-align: right">
            <span class="donor">Bid count </span></td>
         <td><span class="title">{{bidCount}}</span></td>
      </tr>
   </table>
   <br/>
   <span class="title">{{mostPopularBidCount}}</span><br/>
   <span class="donor">Most popular by bid count</span>
   <br/><br/>
   <span class="title">{{mostPopularPrice}}</span><br/>
   <span class="donor">Most popular by highest bid</span>
   <br/><br/>
   <span class="title">{{highestGrossing}}</span><br/>
   <span class="donor">Highest Grossing</span>

</div>
<div style="height: 3px; background: #777"></div>
<div class="footer" style="background: #999"></div>
</div>

<div class="card right">
<div class="header" style="background: #999"></div>
<div style="height: 3px; background: #777"></div>
<div style="padding: 25px; color: #fff;">
   <table cellPadding="0" cellspacing="10">
      <tr>
         <td style="text-align: right">
            <span class="donor">Number of items </span></td>
         <td><span class="title">{{itemCount}}</span></td>
            </tr>

      <tr>
         <td style="text-align: right">
            <span class="donor">items with no bids </span></td>
         <td><span class="title">{{noBidCount}}</span></td>
      </tr>
   </table>
   <br/>

   <div ng-repeat="item in items | orderObjectBy:'gross'">
```

```
        <span class="title">${{item.gross}}: {{item.name}} <span style="color: #aaa">by
{{item.donorname}}</span></span><br/>

        <div class="donor" ng-repeat="bidObj in item.bidObjs">${{bidObj.amt}} by
{{bidObj.who}}</div>

        <br/><br/>
        </div>

    </div>
    <div style="height: 3px; background: #777"></div>
    <div class="footer" style="background: #999"></div>
    </div>

                <div class="card left">
    <div class="header" style="background: #999"></div>
    <div style="height: 3px; background: #777"></div>

                                <form action="scripts/readForm.py" method="POST">
                                    <div class="image" id="imgcontainer" style="text-align:
center; position: relative">
                                        <img id="newimg"
src="{{item.imageurl}}" style="position: relative" />
                                        <input id="imgurl" name="itemImgURL"
type="text" placeholder="image url" style="position: absolute; margin-left: 10px; margin-bottom: 10px;
left: 0; bottom: 0; right: 0; margin-top: 70px; background: #515359;" onBlur="showImg()" />
                                        <input type="submit" value="submit
item"
                                            style="position: absolute; top:
0; right: 0; margin-top: 10px; margin-right: 10px; background: #669900;
                                            border: 1px solid #fff;" />
                                    </div>
                                    <div style="height: 1px; background: #AAA"></div>
                                    <div class="infocontainer" style="min-height: 70px;">
                                        <div class="names">
                                            <span class="title"><input
id="itemname" name="itemName" type="text" class="title" placeholder="item name. keep it short"
style="width: 230px;" /></span><br/>
                                        <span class="donor"><input
id="donorname" name="itemDonorName" type="text" class="donor" placeholder="your name. keep it
real" style="width: 230px;" /></span></br>
                                        </div>
                                        <div class="names">
                                            <span class="donor"><input
id="openDate" name="itemOpenDate" type="date" class="donor" style="width: 230px;" /><label
for="openDate" style="color:#AAA;">Open Time</label></span>
                                        <span class="donor"><input
id="openTime" name="itemOpenTime" type="time" class="donor" style="width: 230px;"
/></span></br>
```

```
                                                    <span class="donor"><input
id="closeDate" name="itemCloseDate" type="date" class="donor" style="width: 230px;" /><label
for="closeDate" style="color:#AAA;">Close Time</label></</span>
                                                    <span class="donor"><input
id="closeTime" name="itemCloseTime" type="time" class="donor" style="width: 230px;"
/></span></br>
                                                </div>
                                                <div class="bids">
                                                    <span
class="winningbid">$<input id="price" type="text" name="itemPrice" class="winningbid"
placeholder="$$" style="width: 40px; margin-left: 5px;"/></span><br/>
                                                    <span class="donor">LIST
PRICE</span>
                                                </div>
                                            </div>

                                            <div style="height: 1px; background: #AAA"></div>

                                            <div class="description" id="description" style="max-
height: 100px; min-height: 100px;">
                                                <textarea id="descriptionbody"
style="height: 90px;" name="itemDesc" placeholder="talk about your item. this can be as long as you
want. the reader's attention span is the limit!"></textarea>
                                            </div>
                                        </form>
        <div style="height: 3px; background: #777"></div>
        <div class="footer" style="background: #999"></div>
        </div>

                    <!-- Old form for adding items -->
                    <!--
                    <div class="card left">
                            <form action="scripts/readForm.py" method="POST">

                                <label for="itemName">Name:</label>
                                <input type="text" name="itemName"
id="itemName"/>
                                <br />

                                <label for="itemDesc">Description:</label>
                                <textarea name="itemDesc" id="itemDesc"
placeholder="Type item description here..."></textarea>
                                <br />

                                <label for="itemDonorName">Donor:</label>
                                <input type="text" name="itemDonorName"
id="itemDonorName"/>
                                <br />

                                <label for="itemPrice">Opening Price:</label>
```

```
                                        <input type="text" name="itemPrice" id="itemPrice"/>
                                        <br />

                                        <label for="itemImgURL">Image URL:</label>
                                        <input type="text" name="itemImgURL"
id="itemImgURL"/>
                                        <br /> <br />

                                        <input type="submit" value="Add Item" />
                                        <input type="reset" value="Reset" />
                                    </form>

                                    <div style="height: 3px; background: #777"></div>
                            <div class="footer" style="background: #999"></div>
                    </div> -->

            <div class="card left" ng-repeat="item in items" ng-class-even="'left'" ng-class-odd="'right'">
            <div class="header"></div>
            <div style="height: 3px; background: #558800"></div>
            <div class="image"style="text-align: center; position: relative">
               <img src="{{item.imageurl}}" style="position: relative" />
               <input type="button" value="remove item" class="removebutton"
                    style="position: absolute; top: 0; right: 0; margin-top: 10px; margin-right: 10px;
background: #990000;
                    border: 1px solid #fff;"
                    ng-click="deleteItem(item)"/>
            </div>
            <div style="height: 1px; background: #aaaaaa"></div>
            <div class="infocontainer">
               <div class="names">
                  <span class="title">{{item.name}}</span><br/>
                  <span class="donor">{{item.donorname | uppercase}}</span>
               </div>
               <div class="bids">
                  <span class="winningbid">${{item.price}}</span><br/>
                  <span class="donor">LIST PRICE</span>
               </div>
            </div>

            <div style="height: 1px; background: #aaaaaa"></div>

            <div class="description">
               {{item.description}}
            </div>

            <div style="height: 3px; background: #558800"></div>
            <div class="footer"></div>
            </div>
            </div>
</div>
```

```
</body>
</html>
```

Controllers.js

```
/**
 * Created by hubspot on 11/16/14.
 */
var auctionApp = angular.module('auctionApp', []);

auctionApp.controller('ItemCardsCtrl', function ($scope) {

        $.ajax({
        url: "https://api.parse.com/1/classes/Item",
        type: "GET",
        beforeSend: function(xhr){
        xhr.setRequestHeader('X-Parse-Application-Id',
'WhkQetI8nb0HrIykoaNc8LJ9fIHIxOvgaXhFXFxm');
        xhr.setRequestHeader('X-Parse-REST-API-Key',
'G02koccgg9q6RzqwRmpiQDx3QIlASet5iW2XbLob');
        },
        success: function(result) {
        $scope.$apply(function(){
            $scope.items = result.results;
                var totalRaised = 0;
                var totalStartPrice = 0;
           var numBids = 0;
                var topBidCount = "";
                var topBidCountCur = 0;
                var topBidAmt = "";
                var topBidAmtCur = 0;
                var highestGrossing = "";
                var highestGrossingCur = 0;

                var noBidCount = 0;

            $scope.items.forEach(function(item) {
               var gross = 0;
               item.currentPrice.forEach(function(bidprice) {
                   totalRaised = totalRaised + bidprice;
                   gross = gross + bidprice;
               });

                  if (item.currentPrice.length == 0) {
                   noBidCount = noBidCount + 1;
                 }

               numBids = numBids + item.numberOfBids;

                  if (item.numberOfBids > topBidCountCur) {
```

```
                topBidCount = item.numberOfBids + " bids - " + item.name + " by " + item.donorname;
                topBidCountCur = item.numberOfBids;
                }

                if (item.currentPrice[0] > topBidAmtCur) {
                topBidAmt = "$" + item.currentPrice[0] + ": " + item.name + " by " + item.donorname;
                topBidAmtCur = item.currentPrice[0];
                }

                if (gross > highestGrossingCur) {
                highestGrossingCur = gross;
                highestGrossing = "$" + gross + ": " + item.name + " by " + item.donorname;
                }

              item.gross = gross;

              item.bidObjs = [];
              for (var i = 0; i < item.currentWinners.length; i++) {
                item.bidObjs.push({"who": item.currentWinners[i], "amt": item.currentPrice[i]});
                }
                });

            $scope.totalRaised = totalRaised;
            $scope.bidCount = numBids;
            $scope.mostPopularBidCount = topBidCount;

                $scope.mostPopularPrice = topBidAmt;

            $scope.highestGrossing = highestGrossing;
            $scope.itemCount = $scope.items.length;
            $scope.noBidCount = noBidCount;
        });

        }
        });

        $scope.buildCSV = function() {
        var headers = ["item name", "donor name", "winner1", "bid1", "winner2", "bid2", "winner3",
"bid3", "winner4", "bid4",
        "winner5", "bid5", "winner6", "bid6", "winner7", "bid7", "winner8", "bid8"];
        var data = [];

      $.each($scope.items, function(idx, item) {
        var name = item.name;
        var donor = item.donorname;

        var winnerName = "";
        var highestBids = "";

        if (item.qty > 1) {
```

```
            $.each(getBidsForItem(item.objectId), function(idx, bid) {
                if (bid && idx < item.qty) {
                winnerName += bid.name + " [" + bid.email + "] " + ((idx == item.qty - 1) ? "" : " & ");
                    highestBids += bid.amt + ((idx == item.qty - 1) ? "" : " & ");
                }
                });
        }
        else {
                var bid = getBidsForItem(item.objectId)[0];
                if (bid) {
            winnerName = bid.name + "[" + bid.email + "]";

                if (bid.amt)
                highestBids = bid.amt;
                }
        }

        data.push([name, donor, winnerName, highestBids]);
        });



        var csvContent = "data:text/csv;charset=utf-8,";
        csvContent += headers.join(",") + "\n";
      data.forEach(function(infoArray, index){
        dataString = infoArray.join(",");
        csvContent += dataString + "\n";
        });

        var encodedUri = encodeURI(csvContent);
      window.open(encodedUri);
        };

});

auctionApp.filter('orderObjectBy', function(){
        return function(input, attribute) {
        if (!angular.isObject(input)) return input;

        var array = [];
        for(var objectKey in input) {
        array.push(input[objectKey]);
        }

      array.sort(function(a, b){
        a = parseInt(a[attribute]);
        b = parseInt(b[attribute]);
        return b - a;
        });
        return array;
```

```
        }
});

auctionApp.filter('noFractionCurrency',
        [ '$filter', '$locale',
      function(filter, locale) {
        var currencyFilter = filter('currency');
        var formats = locale.NUMBER_FORMATS;
        return function(amount, currencySymbol) {
                var value = currencyFilter(amount, currencySymbol);
                var sep = value.indexOf(formats.DECIMAL_SEP);
            if(amount >= 0) {
               return value.substring(0, sep);
                }
                return value.substring(0, sep) + ')';
        };
        } ]);
```

Android app:

src > edu > butler > epics > auction > AuctionApplication.java

```
package edu.butler.epics.auction;

import android.app.Application;
import android.util.Log;

import edu.butler.epics.auction.models.AuctionItem;
import edu.butler.epics.auction.models.Bid;
import com.parse.Parse;
import com.parse.ParseACL;
import com.parse.ParseException;
import com.parse.ParseObject;
import com.parse.ParsePush;
import com.parse.ParseUser;
import com.parse.SaveCallback;

public class AuctionApplication extends Application {

  public static final String APP_ID = "WhkQetI8nb0HrIykoaNc8LJ9flHIxOvgaXhFXFxm";
  public static final String CLIENT_KEY = "uxKjxfKHKj5mIDwAtykNsur3vd1Hlc1Zms1IodEg";

  //Braxtyn account key: Ha6x84QKrSEfs0DMcsZ2w1bGDjddsPtrK7iVzpyN

  @Override
  public void onCreate() {
        super.onCreate();

    ParseObject.registerSubclass(AuctionItem.class);
```

```
    ParseObject.registerSubclass(Bid.class);

        // Add your initialization code here
    Parse.initialize(this, APP_ID, CLIENT_KEY);
    ParsePush.subscribeInBackground("", new SaveCallback() {
        @Override
        public void done(ParseException e) {
        if (e == null) {
       Log.i("TEST", "successfully subscribed to the broadcast channel.");
        } else {
       Log.i("TEST", "failed to subscribe for push", e);
        }
        }
        });

    ParseUser.enableAutomaticUser();
        ParseACL defaultACL = new ParseACL();

        // If you would like all objects to be private by default, remove this line.
    defaultACL.setPublicReadAccess(true);

    ParseACL.setDefaultACL(defaultACL, true);
  }
}
```