Team BONES

EPICS15

Panos Linos

Alex Steel, Gwen Kozak, Dennis Drew, Harrison Lingren

Fall 2015

**Table of Contents**

**Chapter 1: Introduction ……….……………….……………………………………………….2**

**Chapter 2: Requirements Specifications……………….………………………………………4**

**Chapter 3: Architecture…………………………………………………………………………5**

**Chapter 4: Design ……………………………………….………………………………………6**

**Chapter 5: Quality Assurance & Testing……………....………………………………………7**

**Chapter 6: Project Organization & Management….….………………………………………8**

**Chapter 7: Future Work…………………………….…..………………………………………9**

**References/Bibliography …………………………….…………………...……………………10**

**Appendix….……………………………………………...……………………………………...11**

**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 and/or Android 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 put on by Braxtyn’s Bounty for Bones to raise awareness and money for the National Canine Cancer Foundation, as well as for local animal not-for-profits. In the memory of their beloved dog, the family has created this organization and event to help fund the research of canine cancer. This fundraiser involves teams of participants playing mini-golf in local bars, as well as a silent auction of many donated items.

*The Request*

Braxtyn’s Bounty requested a silent auction application because silent auction applications have been shown to increase money generated by about 30-50% compared to traditional silent auctions. As Braxtyn’s Bounty’s major source of fundraising, the organization of course wants to increase funds in any way possible. The desired application would provide a quick and easy solution to manage a silent auction taking place in a variety of locations, and could be used for larger auctions if necessary.

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.

Our group decided to pursue this project because we all have a passion for dogs and believe in the project. We all saw a learning experience and sought to benefit from developing a mobile application as a group. Further, we believe in Braxtyn’s Bounty as a great cause, and would love to be a part of their efforts.

*The Plan and Process*

Beginning by drawing out a plan that suited the client’s needs and requests, we started by working on a computer development application called Visual Studio. This allowed us to represent the Graphical User Interface (GUI), or graphic design, of the final product, visually. We designed each screen of the application, creating and linking together every button and feature so that there was a concrete understanding of what we wanted the application to look like to accommodate the desired features. While this was a tedious and overwhelming process, it allowed us to see exactly what we would need to do.

In the time following, team members researched open source applications that could help us create the application that we had designed. We found one called BidHub, an open source application already tailored to silent auctions. After the suitable approval from both client and professor, we began familiarizing ourselves with the source code and developing it into the what the client wanted.

*The Participants*

Alex Steel (team leader) is a senior computer science student that will be pursuing interests in cyber security after college. He has worked with various non-profit organizations and understands the influence that these organizations make. He has very much enjoyed working to aid the National Canine Cancer Foundation.

Gwen Kozak is a junior elementary education major pursuing minors in mild intervention (special education) and computer science. Her goal is to integrate these two passions in the hopes of successfully teaching young students the fundamentals of computer science, to prepare them for the ever changing technological world. Gwen enjoyed applying and expanding her skill set while working with Braxtyn’s Bounty.

Harrison Lingren is a sophomore Computer Science and Interactive Media student preparing to become a web developer. Harrison has spent a lot of time volunteering with several organizations as a student, both in high school and during his undergraduate studies. He loves to learn through real world experiences and loved having the opportunity to help this wonderful organization.

Dennis Drew is a senior computer science and actuarial science student that will likely be pursuing a career in the actuarial field after graduation. Dennis has a passion for dogs and has spent some time volunteering for Indy Humane. He is excited to have been apart of this project.

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**

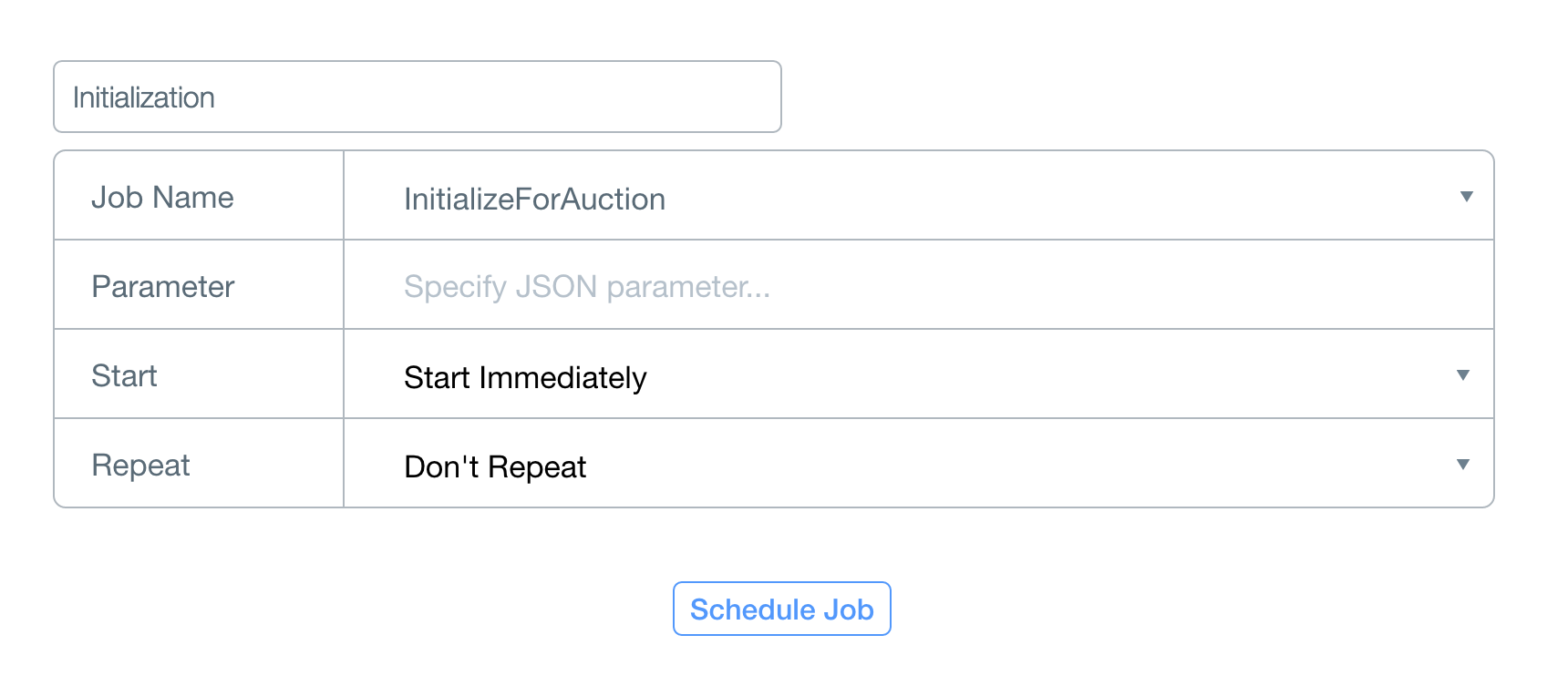
The client requested an application that would conduct the silent auction for them. Functional requirements included a login for both users and administrators to keep the application secure and user specific. Additionally, each auction item would be displayed complete with picture, description, donor, 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 Android because of limitations during the design process. The database provided by parse is slow, so Harrison developed a python script that will load the from an excel spreadsheet into parse allowing for more a large amount of data to be loaded and stored. We chose to do this to make uploading data more convenient for the client, who otherwise would have had to upload data one piece at a time; this did not make sense as the Putt from the Ruff’s silent auction has had upwards of 50 items for auction in the past. Another non-functional requirement is the app store; one must have a development account to allow for the app to be posted onto the app store, as well as a website to host the application itself. Currently, Harrison is hosting and handling this aspect of the application. For more information on this subject, please contact him.

**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. Additionally, Harrison created a database script for importing and population data which can be used to more easily add items.

As the backend has been completed, the group was able to begin work on the application itself. Our group was unfamiliar with xcode and all members were Android users, so we chose to pursue an Android app. The iOS app was put on the back burner for future groups to pursue.\*

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.

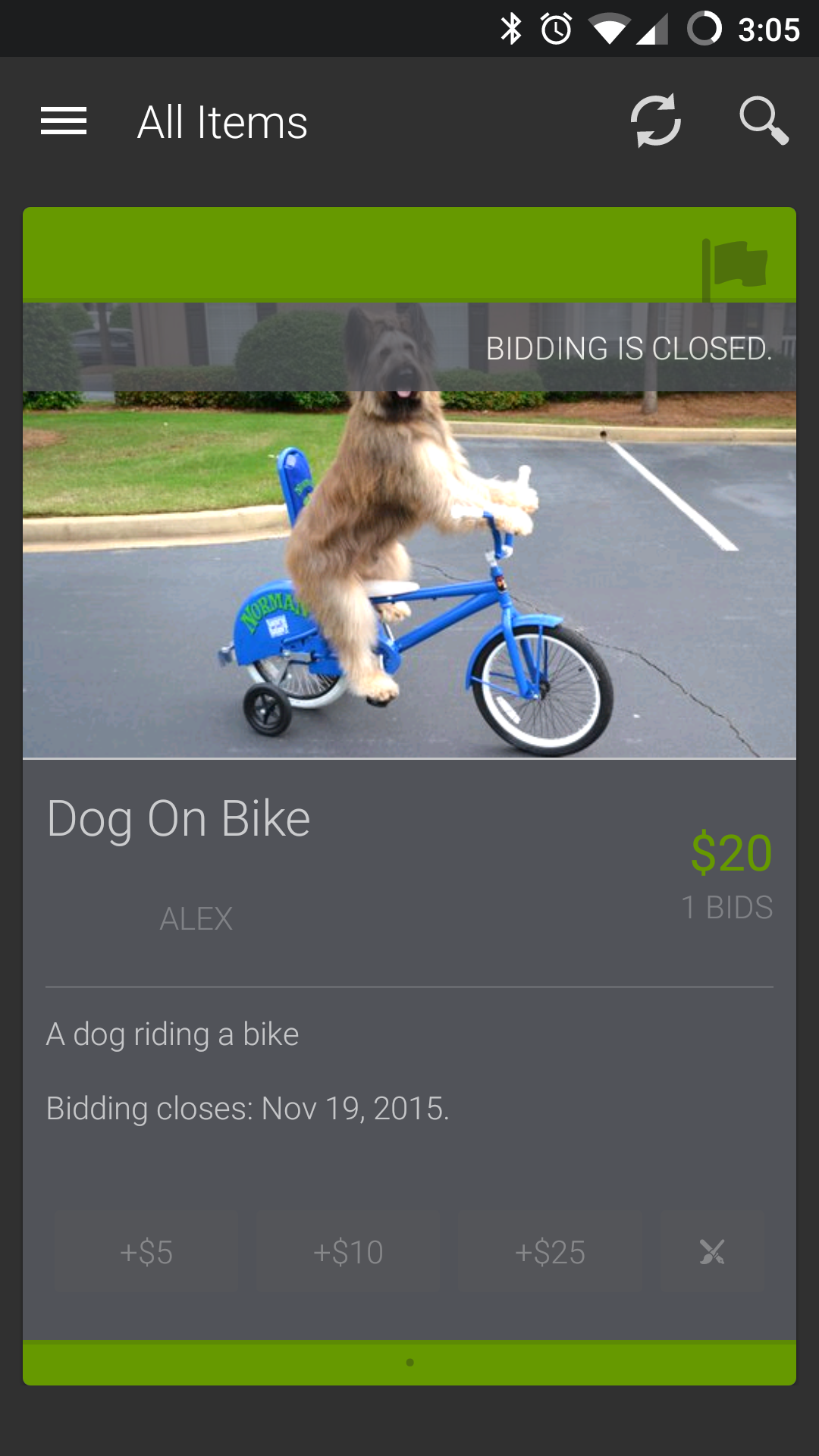
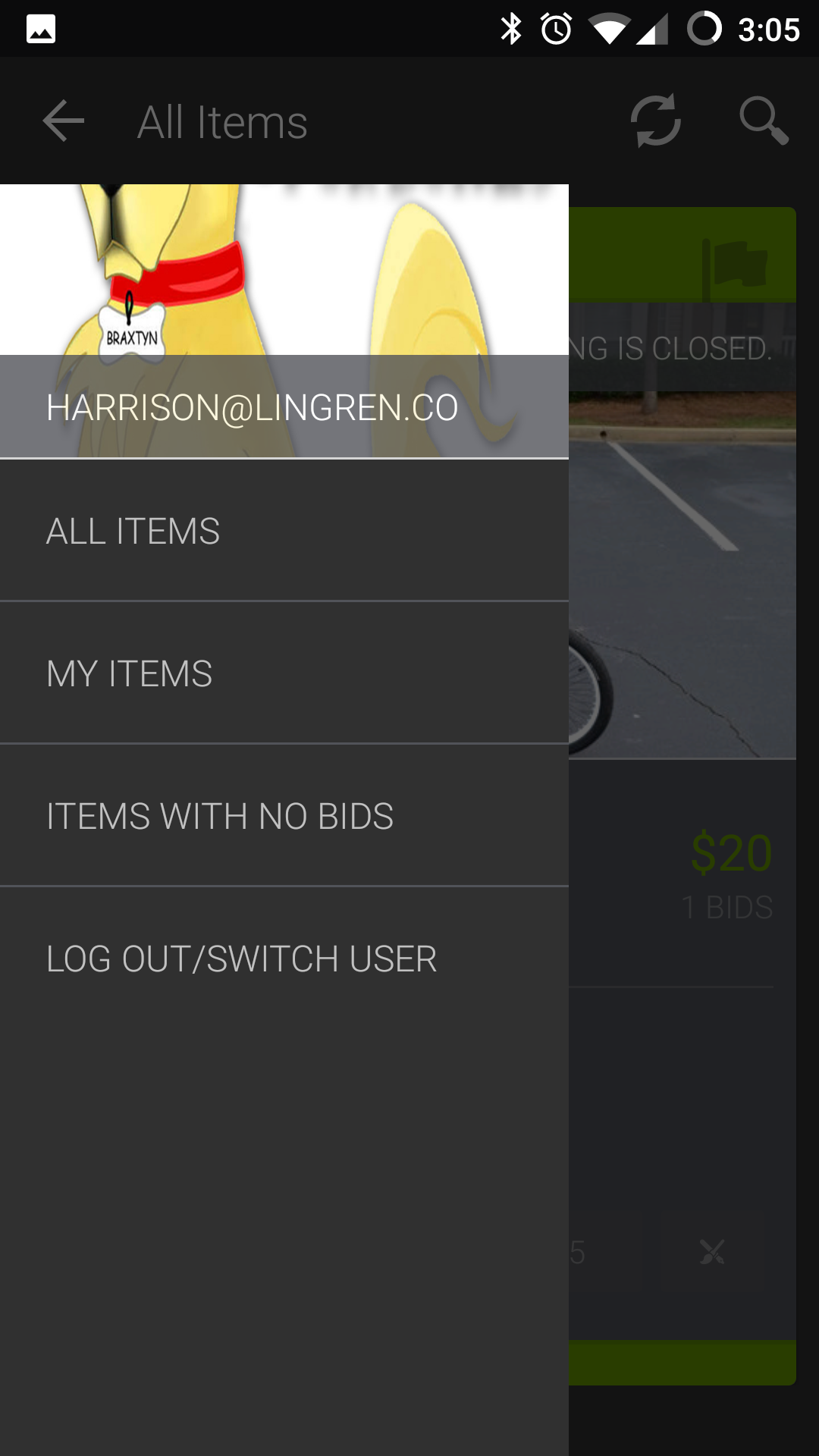
**Chapter 4: Design**

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.



**Chapter 5: Quality Assurance & Testing**

Obstacles that we faced include setting up Android Studio on our computers and the program’s original data load script. It took us some time to troubleshoot working with the Parse mobile cloud development tools and getting those installed on our command line. Once we were able to access the Parse cloud from our computers, we were able to host the database and start a test auction. This test only had a single test item in the auction, and we quickly realized that this would be a poor quality application for the client. The original data load was previously done one cell at a time, making its running time incredibly slow and inefficient. This issue was overcome with Harrison’s data load script written in python, which allowed for an easier way to import the selected data.

The main obstacle was overcoming the mobile application development process. Considering that our combined knowledge on the subject was minimal at the beginning of the semester, learning how to tweak and manipulate the program and debugging the physical application package took some time as well. There were instances where it would not render images in the proper format on the phone, or times when all it really took to fix the problem was an update to the toolkit.

Additionally, 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. After playing with some of the source code, editing some of the images but primarily working with the database load, we ended up with an Alpha/Beta build that can handle mass data loads.

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 four individual members: Alex Steel the team lead, Gwen Kozak, Harrison Lingren, and Dennis Drew. We decided it would be best for the team members to not define roles, but instead work collaboratively on the project. Alex and Harrison worked on primary programming and designing, while Gwen and Dennis 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 Android Studio, which which worked with the packages and application itself. To see how the program operates, please see **Chapter 3.**

**Weekly Status Reports (WSRs):**

9/21<http://epics.butler.edu/wp-content/uploads/2015/09/WSR-Template.pdf>

9/28 <http://epics.butler.edu/wp-content/uploads/2015/09/wsr921927.pdf>

10/4 <http://epics.butler.edu/wp-content/uploads/2015/09/WSRBones104.pdf>

10/12 <http://epics.butler.edu/wp-content/uploads/2015/10/WSR-Template1012.pdf>

10/19 <http://epics.butler.edu/wp-content/uploads/2015/09/WSR-Template1019.pdf>

10/26 <http://epics.butler.edu/wp-content/uploads/2015/09/WSR-Template1026.pdf>

10/31 <http://epics.butler.edu/wp-content/uploads/2015/09/TeamBONESWSR1031.pdf>

11/9 <http://epics.butler.edu/wp-content/uploads/2015/09/TeamBONESWSR119.pdf>

11/16 <http://epics.butler.edu/wp-content/uploads/2015/09/WSRBONES1116.pdf>

11/30 <http://epics.butler.edu/wp-content/uploads/2015/09/WSRBONES1130.pdf>

12/5 <http://epics.butler.edu/wp-content/uploads/2015/09/WSR125.pdf>

**Gannt Chart:**

<http://epics.butler.edu/wp-content/uploads/2015/09/gannt.pdf>

**Use Case Descriptions:**

<http://epics.butler.edu/wp-content/uploads/2015/09/Use-Case-Desc.pdf>

**Chapter 7: Future Work**

The primary goals for the next team should be an iOS application and a web interface for adding, editing, and removing items. <https://github.com/HubSpot/BidHub-iOS> We chose not to pursue the iOS application because none of the team members had iPhones -- because of our lack of comfort and familiarity with this piece of technology, it did not make sense for this to be our primary goal. The client has requested an iOS application, as they believe that a large percentage of their target market will be iPhone users.

We have started work on the web interface and it can be found in the project files, but it is not complete. The only function it provides as of now is the ability to view statistics on the current auction listings and the ability to add items via a form. The team that takes over this project should seek to add more features to this so that the client does not have to deal with the backend of the project to add, edit, or remove items.

Additionally, the Android application that we have created can use some touching up. We believe that the next group should also use the materials to familiarize themselves with the database script made by Harrison, which will prevent them from having to add items to the database one at a time. The next group will also presumably be working at a time nearer to the upcoming auction, and they should use this information to their advantage. We suggest that they be in contact with the client to use the real items as test subjects for the applications. It is particularly important for this group to learn how to find and import current and necessary images, so that the client will be able to do this on their own in the future. There also needs to be an implementation for a payment feature.

The following group will also need to be responsible for finding out how to upload the app to the Play Store with a development account. Currently that is being done with Harrison’s personal play store account. Once the play store dev account is obtained, all you really need to do is login to Parse using the credentials below and use the sources from Github (downloaded as a zip). Everything needed to develop the iOS side is linked above by HubSpot, and our work can be found in the Appendix below. Thanks to HubSpot and their Open-Source application, we had a base application to start from. Their Github should have anything needed to get it working, but the data load will need to be done by the next group. HubSpot’s application in its current state does not support loading data, it just inputs through Parse, once cell at a time. See our source for some data load scripts.

**Parse Login:** [jocelyn408@gmail.com](mailto:jocelyn408@gmail.com)

**Parse Pass:** Epics2015

**References/bibliography (include all references and/or websites you used)**

<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. For all sources, please refer to GitHub: <https://github.com/harrisonlingren/EPICSFall15>

### 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', 'WhkQetI8nb0HrIykoaNc8LJ9flHIxOvgaXhFXFxm');

xhr.setRequestHeader('X-Parse-REST-API-Key', 'G02koccgg9q6RzqwRmpiQDx3QllASet5iW2XbLob');

},

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', 'G02koccgg9q6RzqwRmpiQDx3QllASet5iW2XbLob');

},

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', 'WhkQetI8nb0HrIykoaNc8LJ9flHIxOvgaXhFXFxm');

xhr.setRequestHeader('X-Parse-REST-API-Key', 'G02koccgg9q6RzqwRmpiQDx3QllASet5iW2XbLob');

},

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 = "uxKjxfKHKj5mlDwAtykNsur3vd1Hlc1Zms1lodEg";

//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);

}

}