```
//
//  AppDelegate.h
//  Garage Sale
//
//  Created by Alexander Hammond on 12/14/16.
//  Copyright © 2016 TripleA. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;


@end
```

```
//
//  AppDelegate.m
//  Garage Sale
//
//  Created by Alexander Hammond on 12/14/16.
//  Copyright © 2016 TripleA. All rights reserved.
//

#import "AppDelegate.h"

@interface AppDelegate ()

@end

@implementation AppDelegate


- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
    (NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    return YES;
}


- (void)applicationWillResignActive:(UIApplication *)application {
    // Sent when the application is about to move from active to inactive
        state. This can occur for certain types of temporary interruptions
        (such as an incoming phone call or SMS message) or when the user quits
        the application and it begins the transition to the background state.
    // Use this method to pause ongoing tasks, disable timers, and invalidate
        graphics rendering callbacks. Games should use this method to pause the
        game.
}


- (void)applicationDidEnterBackground:(UIApplication *)application {
    // Use this method to release shared resources, save user data, invalidate
        timers, and store enough application state information to restore your
        application to its current state in case it is terminated later.
    // If your application supports background execution, this method is called
        instead of applicationWillTerminate: when the user quits.
}


- (void)applicationWillEnterForeground:(UIApplication *)application {
    // Called as part of the transition from the background to the active
        state; here you can undo many of the changes made on entering the
        background.
}


- (void)applicationDidBecomeActive:(UIApplication *)application {
    // Restart any tasks that were paused (or not yet started) while the
        application was inactive. If the application was previously in the
        background, optionally refresh the user interface.
    //creates instance of user defaults (used for storage on device)
```

```
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    //creates liked array from stored data
    NSArray *likedArray = [defaults objectForKey:@"LikedItems"];
    //if liked array is nil, initializes the array and reassigns - to ensure
        that no nil value will be accessed when processing elsewhere in the
        app, only relevant for the first time the app boots
    if (likedArray == nil) {
        likedArray = [likedArray init];
        [defaults setObject:likedArray forKey:@"LikedItems"];
    }

}


- (void)applicationWillTerminate:(UIApplication *)application {
    // Called when the application is about to terminate. Save data if
        appropriate. See also applicationDidEnterBackground:.
}


@end
```

```
//
//  ViewController.h
//  Garage Sale
//
//  Created by Alexander Hammond on 12/14/16.
//  Copyright © 2016 TripleA. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController {

}

//defines username property - used to check if user has username and to store
    one if they don't
@property NSString *username;


@end
```

```
//
//  ViewController.m
//  Garage Sale
//
//  Created by Alexander Hammond on 12/14/16.
//  Copyright © 2016 TripleA. All rights reserved.
//

/*
 Create a mobile application that would allow a platform for a digital yard
     sale to raise funds to
 attend NLC. The app should allow for the donation of items, including picture,
     suggested price,
 and a rating for the condition of the item. The app should allow for
     interaction/comments on
 the items. Code should be error free.

 Required Functionality:
 Donation of items
 Items must have:
    Picture
    Suggested Price
    Rating
    Interact/Comments on the items

 UI
    Home Screen
        See the garage sale details
            Name at the top center (label)
            amount raised on left side (label)
            goal on right side (label)
            completion date beneath (label)
        List of items available for purchase (ScrollView?)
        + button to add item (location tbd) (button)

    Add Item:
        Starts with adding image screen
            Functionality:
                Pulls up camera in top half of screen - camera library in
                    bottom half
                User clicks picture button (circle) and it saves camera image
                    or selects library image which gets highlighted and they
                    press next
            Appearance:
                //ignore this, this will be learned
                UIImageView in top half
                UIButton (shaped to be a circle) in center and toward bottom of
                    this half
        Next is adding details screen
            Functionality:
                Adds the suggested price, rating, and name of the item, as well
                    as an optional comment
            Appearance:
                UIImageView in top half (will display the selected image)
                TextField for Name below
                TextField for Price below
```

```
                Slider for rating
                TextField for comments
                Bottom right should be a "Post button" //this can be moved for
                    appearance's sake, maybe to bottom center
                Confirm Post View should pop up (comprised of a Label and Yes
                    No buttons all on a background imageview that is a
                    transparent grey with a center rounded rectangle outline

    Selected Item
        Functionality:
            Displays name, price, rating along the top
            Displays image
            Displays comments below
            Buy button
        Appearance:
            UILabel for Name top center
            UILabel for Price top right
            UILabel for Rating top left
            UIImageView for image (should fill screen width wise - take up 50%
                of screen)
            UIScrollView for comments

    Buy Item
        TBD


    Classes
        Item
            Fields
                Image (idk)
                Name (String)
                SuggestedPrice (float)
                Rating (int) //out of 10
                Comments (MutableArray //of Strings)
            Functions:
                saveImage
                setName
                getName
                setPrice
                getPrice
                setRating
                getRating
                addComment
                removeComment

        HomeViewController
        AddItemViewController



*/
```

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];


    // Do any additional setup after loading the view, typically from a nib.
}

//run when the view has appeared (so that alerts can pop up)
-(void)viewDidAppear:(BOOL)animated {
    //loads an instance of UserDefaults to check for username
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    //loads username
    _username = [defaults objectForKey:@"username"];
    //NSLog(@"%@", _username);
    //creates alert controller for the username
    UIAlertController *usernameInput = [UIAlertController
        alertControllerWithTitle:@"Input Username" message:@"Please create a
        username." preferredStyle:UIAlertControllerStyleAlert];

    //adds a textfield to the alert controller for username to be typed in
    [usernameInput addTextFieldWithConfigurationHandler:^(UITextField *
        textField) {
     textField.placeholder = NSLocalizedString(@"username", @"Username");
     }];

    //adds an OK button for the user to finalize their username
    UIAlertAction* defaultAction = [UIAlertAction actionWithTitle:@"OK" style:
        UIAlertActionStyleDefault handler:^(UIAlertAction * action) {

        //when ok pressed it saves the input username and writes it to defaults
        _username = usernameInput.textFields.firstObject.text;
        [defaults setObject:_username forKey:@"username"];
        NSLog(@"%@", [defaults objectForKey:@"username"]);
    }];

    //adds the ok button to the controller
    [usernameInput addAction:defaultAction];

    //if no username loaded from defaults (first time user has used the app or
        first time reusing after deleting the app) then it shows the controller
        to get a username
    if (_username == nil) {
        //code to show alert controller
        [self presentViewController:usernameInput animated:YES completion:nil];
    }

    //loads the options for conditions
```

```
    NSArray *conditionOptions = [defaults objectForKey:@"conditions"];

    //if no options set, writes the options below (first time app is loaded)
    if (conditionOptions == nil) {

        //options available for item condition
        [defaults setObject:[NSArray arrayWithObjects:@"--Select--",
            @"Unopened", @"Brand New", @"Exceptional", @"Great Condition",
            @"Used", @"Falling Apart", @"Broken", nil] forKey:@"conditions"];
    }
}



- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}


@end
```

```
//
//  ItemDetail.h
//  Garage Sale
//
//  Created by Alexander Hammond on 1/22/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import <UIKit/UIKit.h>

//imports all of the header files it will use
#import "Item.h"
#import "Comments.h"
//#import <PassKit/PassKit.h>
#import "Items.h"


@interface ItemDetail : UIViewController <UITextViewDelegate> {

    //declares all of the UI elements
    IBOutlet UILabel *displayName;
    IBOutlet UILabel *displayCondition;
    IBOutlet UILabel *displayPrice;
    IBOutlet UITextView *displayDescription;
    IBOutlet UIButton *displayLikeButton;
    IBOutlet UIImageView *displayImage;
    IBOutlet UIButton *comment;
    IBOutlet UIImageView *purchased;
    IBOutlet UIButton *buyButton;


}

//declares the properties, actions, and publicly accessible methods

@property Item *itemOnDisplay;

-(IBAction)like:(id)sender;
-(IBAction)comments:(id)sender;
-(void)updateView;

@end
```

```
//
//  ItemDetail.m
//  Garage Sale
//
//  Created by Alexander Hammond on 1/22/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import "ItemDetail.h"
#import "PurchaseThankYou.h"

@interface ItemDetail () <UITextViewDelegate/*, apple pay stuff:
    PKPaymentAuthorizationViewControllerDelegate*/>

@end

@implementation ItemDetail


//loads the comments page when the comments button is pressed
-(IBAction)comments:(id)sender {
    [self performSegueWithIdentifier:@"toComments" sender:self];
}

//ensures that the description text view (required to show multiple lines in a
    scrollable format) will not react to user interaction
- (BOOL)textViewShouldBeginEditing:(UITextView *)textView
{
    return NO;
}

//runs before segues
-(void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {

    //if going to the comments page it sets the current item to the item for
        the comments page to use when displaying comments
    if ([segue.identifier isEqualToString: @"toComments"]) {
        Comments *destinationViewController = segue.destinationViewController;
        destinationViewController.item = _itemOnDisplay;

    }

    //if going to the thank you page it sends the current item forward so that
        it can be sent back by that page when it segues back to this view
    if ([segue.identifier isEqualToString:@"toPurchaseThankYou"]) {
        PurchaseThankYou *dest = segue.destinationViewController;
        dest.itemStorage = _itemOnDisplay;
    }
}

//updates all parts of the UI for the page (easy method to call when data loads
    or changes (i.e. liked or purchased)
-(void)updateView {
    //hides the purchased button to ensure it won't incorrectly show up
    purchased.hidden = YES;
```

```
    //sets the name text
    displayName.text = _itemOnDisplay.name;

    //gets the int that represents condition (indicative of its position in the
        conditions array)
    int index = (int) _itemOnDisplay.condition;

    //loads the conditions array
    NSArray *conditionOptionsDetail = [[NSUserDefaults standardUserDefaults]
        objectForKey:@"conditions"];

    //sets the condition text based on index value
    displayCondition.text = conditionOptionsDetail[index];

    //displays the price (uses helper method from the Item class to convert the
        int in cents to a string in $)
    displayPrice.text = [_itemOnDisplay getPriceString];

    //displays the description text
    displayDescription.text = _itemOnDisplay.itemDescription;

    //if item liked, sets the heart to solid
    if (_itemOnDisplay.liked) {
        [displayLikeButton setImage:[UIImage imageNamed:@"Instagram-Heart-
            Solid.png"] forState:UIControlStateNormal];
    }
    //if item not liked sets the hear to transparent
    else {
        [displayLikeButton setImage:[UIImage imageNamed:@"Instagram-Heart-
            Transparent.png"] forState:UIControlStateNormal];
    }

    //if the item has an image (server errors may cause it to be missing an
        image) it displays it
    if (_itemOnDisplay.image != nil) {
        displayImage.image = _itemOnDisplay.image;
    }
    else {
        //if missing the image it shows a default image that shows the image is
            missing
        displayImage.image = [UIImage imageNamed:@"missing.png"];
    }

    //if item purchased, it shows the purchased image, if not it hides it
    if ([_itemOnDisplay.itemPurchaseState intValue] == 1) {
        purchased.hidden = NO;
    }
    else {
        purchased.hidden = YES;
    }
}


-(IBAction)buyWithoutApplePay:(id)sender {

    //NSLog(@"%@", _itemOnDisplay.localDictionary);
```

```
//has the item update its internal dictionary before checking purchase
    state
[_itemOnDisplay setItemDictionary];
//NSLog(@"local Dictionary\n%@", _itemOnDisplay.localDictionary);

//loads the purchase state from the object (loads as a dictionary for later
    use converting to json)
NSMutableDictionary *tmpDic = [NSMutableDictionary dictionaryWithObject:
    [_itemOnDisplay.localDictionary objectForKey:@"item_purchase_state"]
    forKey:@"item_purchase_state"];

//NSLog(@"TmpDic%@", tmpDic);

//loads an integer from the previously loaded dictionary to be used for
    evaluation
NSInteger *check = (NSInteger *)[[tmpDic
    objectForKey:@"item_purchase_state"] integerValue];
//NSLog(@"Check: %zd",check);

//evaluates the purchased value - if item has been purchased it proceeds to
    'buy' the item (changing its status in the server)
if (check == 0){
    //creates an alert controller for them to confirm the purchase
    UIAlertController *purchaseConfirmation = [UIAlertController
        alertControllerWithTitle:@"Purchase" message:@"Are you sure?"
        preferredStyle:UIAlertControllerStyleAlert];
    //action to confirm purchase
    UIAlertAction *purchase = [UIAlertAction actionWithTitle:@"Buy" style:
        UIAlertActionStyleDefault handler:^(UIAlertAction * action) {
        //all of the code to do the purchase if the Buy button is pressed

        //recreates the dic with the new purchase state
        NSMutableDictionary *tmpDic = [[NSMutableDictionary alloc]
            initWithObjectsAndKeys:[NSString stringWithFormat: @"%i", 1],
            @"item_purchase_state", nil];

        //creates error handler
        NSError *error;

        //creates a json request from the dictionary
        NSData *jsonData = [NSJSONSerialization dataWithJSONObject:tmpDic
            options:NSJSONWritingPrettyPrinted error:&error];

        //creates url for the request
        NSURL *url = [NSURL URLWithString:[NSString
            stringWithFormat:@"https://murmuring-
            everglades-79720.herokuapp.com/items/%zd.json", _itemOnDisplay.
            itemID]];
        //NSLog(@"%@", url);

        //creates a request from the URL with a 60 second timeout
        NSMutableURLRequest *uploadRequest = [NSMutableURLRequest
            requestWithURL:url cachePolicy:
            NSURLRequestUseProtocolCachePolicy timeoutInterval:60.0];
```

```
        //specifics for the request (it is a PATCH (update) request with
            json content)
        [uploadRequest setHTTPMethod:@"PATCH"];
        [uploadRequest setValue:@"application/json"
            forHTTPHeaderField:@"Accept"];
        [uploadRequest setValue:@"application/json"
            forHTTPHeaderField:@"Content-Type"];
        [uploadRequest setValue:[NSString stringWithFormat:@"%lu",
            (unsigned long)[jsonData length]] forHTTPHeaderField:@"Content-
            Length"];
        [uploadRequest setHTTPBody: jsonData];

        //creates the url session for the url request
        NSURLSession *session = [NSURLSession sessionWithConfiguration:
            [NSURLSessionConfiguration defaultSessionConfiguration]];

        //runs the session with handler code to see if the upload was
            successful
        [[session dataTaskWithRequest:uploadRequest completionHandler:^
            (NSData *data, NSURLResponse *response, NSError *error)
          {
              dispatch_async(dispatch_get_main_queue(), ^{
                  NSString *requestReply = [[NSString alloc] initWithData:
                      data encoding:NSASCIIStringEncoding];
                  NSLog(@"requestReply: %@", requestReply);
                  _itemOnDisplay.itemPurchaseState = [NSNumber
                      numberWithInt:1];
                  [self performSegueWithIdentifier:@"toPurchaseThankYou"
                      sender:self];
              });
          }] resume];
    //this ends the code for the action for the Buy button
    }];

    //creates a cancel button for the confirmation alert
    UIAlertAction *cancel = [UIAlertAction actionWithTitle:@"Cancel" style:
        UIAlertActionStyleCancel handler:^(UIAlertAction * action) {
    }];

    //adds the buy and cancel buttons and presents the confirmation
    [purchaseConfirmation addAction:purchase];
    [purchaseConfirmation addAction:cancel];
    [self presentViewController:purchaseConfirmation animated:YES
        completion:nil];
}
else if (check == (NSInteger *)1){
    UIAlertController *alert = [UIAlertController
        alertControllerWithTitle:@"Cannot Purchase" message:@"Someone has
        already purchased this item" preferredStyle:
        UIAlertControllerStyleAlert];
    UIAlertAction* defaultAction = [UIAlertAction actionWithTitle:@"OK"
        style:UIAlertActionStyleDefault handler:^(UIAlertAction * action)
        {}];
    [alert addAction:defaultAction];
    [self presentViewController:alert animated:YES completion:nil];
}
```

```
    else {
        UIAlertController *alert = [UIAlertController
            alertControllerWithTitle:@"Load Error" message:@"Cannot load item
            for purchase" preferredStyle:UIAlertControllerStyleAlert];
        UIAlertAction* defaultAction = [UIAlertAction actionWithTitle:@"OK"
            style:UIAlertActionStyleDefault handler:^(UIAlertAction * action)
            {}];
        [alert addAction:defaultAction];
        [self presentViewController:alert animated:YES completion:nil];
    }

}

//action for like button - calls the internal method to change the liked status
    of the object then update the UI by changing the button image to reflect
    the new state
-(IBAction)like:(id)sender {
    [_itemOnDisplay changeLiked];
    if (_itemOnDisplay.liked) {
        [displayLikeButton setImage:[UIImage imageNamed:@"Instagram-Heart-
            Solid.png"] forState:UIControlStateNormal];
    }
    else {
        [displayLikeButton setImage:[UIImage imageNamed:@"Instagram-Heart-
            Transparent.png"] forState:UIControlStateNormal];
    }
}

//setup for when the view loads
- (void)viewDidLoad {
    [super viewDidLoad];

    //updates the view (all UI elements)
    [self updateView];

    //sets the delegate for the text view (allows above delegate method to
        ensure users can't interact with it)
    displayDescription.delegate = self;


}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
}


@end
```

```
//
//  PurchaseThankYou.h
//  Garage Sale
//
//  Created by Alexander Hammond on 2/1/17.
//  Copyright © 2017 TripleA. All rights reserved.
//


#import <UIKit/UIKit.h>

//imports header files of other classes it will use (it stores an item and
    access an ItemDetail view controller
#import "Item.h"
#import "ItemDetail.h"

@interface PurchaseThankYou : UIViewController

//declares it will have an Item as a property
@property Item *itemStorage;

@end
```

```
//
//  PurchaseThankYou.m
//  Garage Sale
//
//  Created by Alexander Hammond on 2/1/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import "PurchaseThankYou.h"

@interface PurchaseThankYou ()

@end

@implementation PurchaseThankYou

//programatically called segue so that the prepare for segue method will be
    called
-(IBAction)back:(id)sender {
    [self performSegueWithIdentifier:@"returnToItemPage" sender:self];
}

//sends the stored item back to the ItemDetail view to use to update its UI
-(void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    ItemDetail *dest = segue.destinationViewController;
    dest.itemOnDisplay = _itemStorage;

}

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}


@end
```

```
//
//  Filters.h
//  Garage Sale
//
//  Created by Alexander Hammond on 1/27/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import <UIKit/UIKit.h>


@interface Filters : UIViewController <UIPickerViewDelegate,
    UIPickerViewDataSource, UITextFieldDelegate> {
    //declares all of the UI elements
    IBOutlet UITextField *bestCondition;
    IBOutlet UITextField *worstCondition;
    IBOutlet UITextField *minimumPrice;
    IBOutlet UITextField *maximumPrice;
}

//declares all of the properties within the view
@property (strong, nonatomic) UIPickerView *conditionPicker;
@property UIToolbar *toolBar;
@property NSArray *requestResult;
//1 for worst, 2 for best (this is a property so it can be accessed in delegate
    methods)
@property int conditionFieldBeingEdited;

//all of these fields are properties so that there will be no linker error when
    communicating with other view controllers (this is true for most fields in
    all view controllers that communicate with other view controllers)
@property NSDictionary *filtersInPlace;
@property NSArray *filteredItemsList;
@property NSArray *conditionOptionsFilter;
@property bool downloadSuccessful;
//Note: this holds the highest integer value allowed (since the higher the int
    in this case the worse the condition) = condition_max
@property NSInteger *worstConditionInt;
//Note: this holds the lowest integer value allowed (since the lower the int in
    this case the better the condition) = condition_min
@property NSInteger *bestConditionInt;

@property NSInteger *minPriceInCents;
@property NSInteger *maxPriceInCents;


-(IBAction)sendFilters:(id)sender;

@end
```

```
//
//  Filters.m
//  Garage Sale
//
//  Created by Alexander Hammond on 1/27/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import "Filters.h"
#import "Items.h"

@interface Filters () <UIPickerViewDelegate, UIPickerViewDataSource,
    UITextFieldDelegate>

@end

@implementation Filters

-(IBAction)sendFilters:(id)sender {
    //closes all keyboards
    [self.view endEditing:YES];

    //creates an empty dictionary for data to be added
    NSMutableDictionary *dataDic = [[NSMutableDictionary alloc] init];

    //goes through every field, if it is not empty it adds it to the dictionary
        (empty fields are handled by the database with default values)
    if (_worstConditionInt != nil) {
        NSString *condition_max = [NSString stringWithFormat:@"%zd",
            _worstConditionInt];
        [dataDic setObject:condition_max forKey:@"condition_max"];
    }
    if (_bestConditionInt != nil) {
        NSString *condition_min = [NSString stringWithFormat:@"%zd",
            _bestConditionInt];
        [dataDic setObject:condition_min forKey:@"condition_min"];
    }
    if (_minPriceInCents != nil) {
        NSString *price_min = [NSString stringWithFormat:@"%zd",
            _minPriceInCents];
        [dataDic setObject:price_min forKey:@"price_min"];
    }
    if (_maxPriceInCents != nil) {
        NSString *price_max = [NSString stringWithFormat:@"%zd",
            _maxPriceInCents];
        [dataDic setObject:price_max forKey:@"price_max"];
    }
    //creates a string from all of the fields to be put into an array that will
        go to the main items view controller to store the currently loaded
        filters
    NSString *worstConditionString = [NSString stringWithFormat:@"%zd",
        _worstConditionInt];
    NSString *bestConditionString = [NSString stringWithFormat:@"%zd",
        _bestConditionInt];
    NSString *minPriceString = [NSString stringWithFormat:@"%zd",
        _minPriceInCents];
```

```
    NSString *maxPriceString = [NSString stringWithFormat:@"%zd",
        _maxPriceInCents];
    //stores the current filters to be sent to items page and saved for when
        filters page reloads
    _filtersInPlace = [NSDictionary dictionaryWithObjectsAndKeys:
        worstConditionString, @"worst_condition", bestConditionString,
        @"best_condition", minPriceString, @"min_price", maxPriceString,
        @"max_price", nil];
    //creates a dictionary for sending the request - puts it under the data
        heading to match what the database expects
    NSMutableDictionary *tmpDic = [NSMutableDictionary dictionaryWithObject:
        dataDic forKey:@"data"];
    //NSLog(@"%@", tmpDic);

    //error handler
    NSError *error;

    //creates the json data for the url request
    NSData *jsonData = [NSJSONSerialization dataWithJSONObject:tmpDic options:
        NSJSONWritingPrettyPrinted error:&error];

    //creates url for request
    NSURL *url = [NSURL URLWithString:@"https://murmuring-
        everglades-79720.herokuapp.com/items/filtered_list.json"];

    //creates a URL request
    NSMutableURLRequest *uploadRequest = [NSMutableURLRequest requestWithURL:
        url cachePolicy:NSURLRequestUseProtocolCachePolicy timeoutInterval:60.0
        ];

    //specifics for the request (it is a post request with json content)
    [uploadRequest setHTTPMethod:@"POST"];
    [uploadRequest setValue:@"application/json" forHTTPHeaderField:@"Accept"];
    [uploadRequest setValue:@"application/json" forHTTPHeaderField:@"Content-
        Type"];
    [uploadRequest setValue:[NSString stringWithFormat:@"%lu", (unsigned long)
        [jsonData length]] forHTTPHeaderField:@"Content-Length"];
    [uploadRequest setHTTPBody: jsonData];

    //creates the URLSession to start the request
    NSURLSession *session = [NSURLSession sessionWithConfiguration:
        [NSURLSessionConfiguration defaultSessionConfiguration]];

    //sets the boolean to false to indicate the download has not yet finished
    _downloadSuccessful = false;
    //creates empty array to store the response from the server
    _requestResult = [[NSArray alloc] init];

    //initiates the url session with a handler that processes the data returned
        from the server
    [[session dataTaskWithRequest:uploadRequest completionHandler:^(NSData *
        data, NSURLResponse *response, NSError *error) {
        NSLog(@"Error: %@", error);
        dispatch_async(dispatch_get_main_queue(), ^{
            //if the data is empty it will report an error, if not it will
                process the list of items returned by the filter parameters
```

```
            if (data != nil) {
                //error handler
                NSError *jsonError;
                //stores the response
                _requestResult = [NSJSONSerialization JSONObjectWithData:data
                    options:NSJSONReadingMutableContainers error:&jsonError];

                //NSLog(@"requestReply: %@", _requestResult);
                //NSLog(@"%@", [[_requestResult class] description]);

                //if the response is the valid type it indicates the download
                    is successful and proceeds with the segue back to the main
                    page, if unsuccessful it proceeds while leaving the
                    downloadSuccessful as false (the segue delegate method uses
                    this to determine what data to pass to the Items view
                    controller
                if ([[[_requestResult class] description]
                    isEqualToString:@"__NSArrayM"]) {
                    _downloadSuccessful = true;
                    [self performSegueWithIdentifier:@"loadItemsWithFilters"
                        sender:self];
                }
                else {
                    //server error, do some output
                    [self performSegueWithIdentifier:@"loadItemsWithFilters"
                        sender:self];
                }
            }
            //logs if data is nil and the phone did not connect to a server
            else {
                NSLog(@"NO DATA RECEIVED FROM FILTER REQUEST");
                [self performSegueWithIdentifier:@"loadItemsWithFilters"
                    sender:self];
            }
        });

    }] resume];
}

//hanldes segue back to main view controller based on whether or not the
    filtered items were successfully downloaded
-(void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    if ([segue.identifier isEqualToString:@"loadItemsWithFilters"]) {
        //loads destination view controller
        Items *destinationController = segue.destinationViewController;
        //stores the filters in place to save the user's filter options for the
            next time they return to this page
        destinationController.filters = _filtersInPlace;
        //if download successful it tells the Items page to show the filtered
            content
        if (_downloadSuccessful) {

            destinationController.filteredResults = _requestResult;
            destinationController.showFiltered = true;
        }
        else {
```

```
            //show error saying filters could not load
            destinationController.showFiltered = false;
        }
    }
}


//condition picker view - sets the number of columns
-(NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView
{
    return 1;
}

//sets the number of rows for the picker condition to the number of items in
    the conditions array
- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:
    (NSInteger)component
{
    return _conditionOptionsFilter.count;
}

// The data to return for the row and component (column) that's being passed in
- (NSString*)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row
    forComponent:(NSInteger)component
{
    return _conditionOptionsFilter[row];
}


//runs when a row selected (note: this occurs when user stops scrolling, not
    necessarily their final choice)
-(void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
    inComponent:(NSInteger)component {
    //if row is not 0 (--Select--) logs their condition
    if (row != 0) {
        if (self.conditionFieldBeingEdited == 1) {
            worstCondition.text = _conditionOptionsFilter[row];
            long rowTmp = row;
            _worstConditionInt = (NSInteger *)rowTmp;
        }
        else {
            bestCondition.text = _conditionOptionsFilter[row];
            long rowTmp = row;
            _bestConditionInt = (NSInteger *)rowTmp;
        }
    }
    //if --Select-- chosen - logs their condition as being empty
    else {
        if (_conditionFieldBeingEdited == 1) {
            worstCondition.text = @"";
            worstCondition.placeholder = @"Worst Condition";
            _worstConditionInt = nil;
        }
        if (_conditionFieldBeingEdited == 2) {
            bestCondition.text = @"";
            bestCondition.placeholder = @"Best Condition";
```

```
            _bestConditionInt = nil;
        }
    }

}

//outlet to dismiss keyboards - goes to background button on the UI
-(IBAction)dismissKeyBoards:(id)sender {
    [self.view endEditing:YES];
}


//run if the text field is about to begin editing
-(BOOL)textFieldShouldBeginEditing:(UITextField *)textField {
    //for condition pickers it changes their input view to the scroller instead
        of the keyboard
    if ([textField isEqual:worstCondition]) {

        worstCondition.inputView = _conditionPicker;
        textField.inputView = _conditionPicker;
        _conditionFieldBeingEdited = 1;
    }
    if ([textField isEqual:bestCondition]) {
        bestCondition.inputView = _conditionPicker;
        textField.inputView = _conditionPicker;
        _conditionFieldBeingEdited = 2;
    }
    //adds the toolbar to provide a done button for users
    textField.inputAccessoryView = _toolBar;
    return YES;

}


-(void)textFieldDidEndEditing:(UITextField *)textField {
    //all of the code in this is simply checks on the validity of the input for
        each textField then saving the input for each textField to it's
        respective variable
    if ([textField isEqual: minimumPrice]) {
        NSString *cents;
        NSString *dollars;
        int priceCents;
        int priceDollars;
        long finalPriceInCents;
        for (int i = 0; i < minimumPrice.text.length; i++) {
            if ([[[minimumPrice.text substringFromIndex:i] substringToIndex:1]
                isEqualToString:@"."]) {
                cents = [minimumPrice.text substringFromIndex:i];
                dollars = [minimumPrice.text substringToIndex:i];
                break;
            }
        }
        if (dollars == nil) {
            dollars = minimumPrice.text;
            if ([dollars isEqualToString:@""]) {
                dollars = @"0";
```

```
            }
        }
        if (cents == nil) {
            cents = @".00";
        }
        NSCharacterSet *mySet = [NSCharacterSet
            characterSetWithCharactersInString:@"."];
        cents = [cents stringByTrimmingCharactersInSet:mySet];
        priceCents = (int)[cents integerValue];
        dollars = [dollars stringByTrimmingCharactersInSet:mySet];
        priceDollars = (int)[dollars integerValue];
        NSLog(@"Price: %@%i", dollars, priceDollars);
        finalPriceInCents = (priceDollars * 100) + priceCents;
        _minPriceInCents = (NSInteger *)finalPriceInCents;
        NSLog(@"%zd", _minPriceInCents);
        minimumPrice.text = [NSString stringWithFormat:@"$%@.%@", dollars,
            cents];
        if ([minimumPrice.text isEqualToString:@"$0.00"]) {
            minimumPrice.text = @"";
            _minPriceInCents = nil;
            minimumPrice.placeholder = @"$0.00";
        }
    }
    if ([textField isEqual: maximumPrice]) {
        NSString *cents;
        NSString *dollars;
        int priceCents;
        int priceDollars;
        long finalPriceInCents;
        for (int i = 0; i < maximumPrice.text.length; i++) {
            if ([[[maximumPrice.text substringFromIndex:i] substringToIndex:1]
                isEqualToString:@"."]) {
                cents = [maximumPrice.text substringFromIndex:i];
                dollars = [maximumPrice.text substringToIndex:i];
                break;
            }
        }
        if (dollars == nil) {
            dollars = maximumPrice.text;
            if ([dollars isEqualToString:@""]) {
                dollars = @"0";
            }
        }
        if (cents == nil) {
            cents = @".00";
        }
        NSCharacterSet *mySet = [NSCharacterSet
            characterSetWithCharactersInString:@"."];
        cents = [cents stringByTrimmingCharactersInSet:mySet];
        priceCents = (int)[cents integerValue];
        dollars = [dollars stringByTrimmingCharactersInSet:mySet];
        priceDollars = (int)[dollars integerValue];
        NSLog(@"Price: %@%i", dollars, priceDollars);
        finalPriceInCents = (priceDollars * 100) + priceCents;
        _maxPriceInCents = (NSInteger *)finalPriceInCents;
        NSLog(@"%zd", _maxPriceInCents);
```

```
            maximumPrice.text = [NSString stringWithFormat:@"$%@.%@", dollars,
                cents];
            if ([maximumPrice.text isEqualToString:@"$0.00"]) {
                maximumPrice.text = @"";
                _maxPriceInCents = nil;
                maximumPrice.placeholder = @"$0.00";
            }
        }
        //hides all keyboards
        [textField resignFirstResponder];
        [self.view endEditing:YES];
    }

    //hides picker view (and other keyboards)
    -(void)clearPickerView {
        [self.view endEditing:YES];
    }

    - (void)viewDidLoad {
        //creates a toolbar with a done button to end editing for all keyboards
        _toolBar = [[UIToolbar alloc] init];
        _toolBar.barStyle = UIBarStyleDefault;
        _toolBar.translucent = true;
        [_toolBar sizeToFit];

        UIBarButtonItem *finished = [[UIBarButtonItem alloc] initWithTitle:@"Done"
            style:UIBarButtonItemStylePlain target:self action:@selector
            (clearPickerView)];

        [_toolBar setItems:@[finished] animated: false];
        _toolBar.userInteractionEnabled = true;
        _conditionOptionsFilter = [[NSUserDefaults standardUserDefaults]
            objectForKey:@"conditions"];
        _conditionPicker = [[UIPickerView alloc] init];

        //sets the view controller as the delegate and dataSource for the
            textFields and pickerView (allows use of delegate methods to manage the
            control of these objects)
        _conditionPicker.dataSource = self;
        _conditionPicker.delegate = self;
        _conditionPicker.showsSelectionIndicator = YES;
        worstCondition.delegate = self;
        bestCondition.delegate = self;
        maximumPrice.delegate = self;
        minimumPrice.delegate = self;
        [super viewDidLoad];
        // Do any additional setup after loading the view.

        //sets up the filters in case there were previous filters
        if (_filtersInPlace != nil) {
            _worstConditionInt = (NSInteger *)[[_filtersInPlace
                objectForKey:@"worst_condition"] integerValue];
            _bestConditionInt = (NSInteger *)[[_filtersInPlace
                objectForKey:@"best_condition"] integerValue];
            _minPriceInCents = (NSInteger *)[[_filtersInPlace
                objectForKey:@"min_price"] integerValue];
```

```
        _maxPriceInCents = (NSInteger *)[[_filtersInPlace
            objectForKey:@"max_price"] integerValue];
        if (_worstConditionInt != nil) {
            worstCondition.text = _conditionOptionsFilter[(int)
                _worstConditionInt];
        }
        if (_bestConditionInt != nil) {
            bestCondition.text = _conditionOptionsFilter[(int)_bestConditionInt
                ];
        }
        if (_maxPriceInCents != nil) {
            maximumPrice.text = [self getPriceString:_maxPriceInCents];
        }
        if (_minPriceInCents != nil) {
            minimumPrice.text = [self getPriceString:_minPriceInCents];
        }
    }
}

//helper method to convert an int with price in cents to a string the user will
    understand
-(NSString *)getPriceString:(NSInteger *)priceInCents {
    int tmpPriceInCents = (int)priceInCents;
    //NSLog(@"Price in cents: %zd\nPriceInCents %i", _priceInCents,
        tmpPriceInCents);
    int tmpCentsOnes = tmpPriceInCents %10;
    int tmpCentsTens = ((tmpPriceInCents - tmpCentsOnes)%100)/10;
    NSString *priceCents = [NSString stringWithFormat:@"%i%i", tmpCentsTens,
        tmpCentsOnes];
    NSString *priceDollars = [NSString stringWithFormat:@"%i", (tmpPriceInCents
        /100)];

    NSString *priceString =[NSString stringWithFormat:@"$%@.%@", priceDollars,
        priceCents];
    return priceString;
}


- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}


@end
```

```
//
//  Comments.h
//  Garage Sale
//
//  Created by Alexander Hammond on 1/23/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

//imports
#import <UIKit/UIKit.h>
#import "Item.h"
#import "ItemDetail.h"

//declarig interface outlets
@interface Comments : UIViewController <UITextViewDelegate,
    NSURLSessionDelegate> {
    IBOutlet UITextView *otherComments;
    IBOutlet UITextView *inputComment;
    IBOutlet UIImageView *showImage;
    IBOutlet UIImageView *purchased;
}

//class properties
@property Item *item;
@property NSString *userComment;

-(IBAction)postComment:(id)sender;
-(IBAction)back:(id)sender;

@end
```

```
//
//  Comments.m
//  Garage Sale
//
//  Created by Alexander Hammond on 1/23/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import "Comments.h"

@interface Comments () <UITextViewDelegate>

@end

@implementation Comments

//initiates url session to download comments
-(void)downloadComments {
    NSString *jsonUrlString = [NSString stringWithFormat:@"https://murmuring-
        everglades-79720.herokuapp.com/items/%zd.json", _item.itemID];
    //NSLog(@"URL Request: %@", jsonUrlString);
    NSURL *url = [NSURL URLWithString:jsonUrlString];
    NSURLSessionConfiguration* config = [NSURLSessionConfiguration
        defaultSessionConfiguration];
    NSURLSession *session = [NSURLSession sessionWithConfiguration:config
        delegate:self delegateQueue:[NSOperationQueue mainQueue]];
    NSURLSessionDataTask *dataTask = [session dataTaskWithURL:url];
    [dataTask resume];
}

//runs when data comes in
- (void)URLSession:(NSURLSession *)session
          dataTask:(NSURLSessionDataTask *)dataTask
    didReceiveData:(NSData *)data {

    NSError *error;
    //saves the data to a dictionary
    NSDictionary *result = [NSJSONSerialization JSONObjectWithData:data
        options:NSJSONReadingMutableContainers error:&error];
    //NSLog(@"Result (Length: %zd) = %@",_result.count, _result);
    //creates array from all of the comments in the dictionary
    NSArray *fullComments =  [result objectForKey:@"comments"];
   // NSLog(@"All comments:\n%@", fullComments);
    _item.comments = [[NSMutableArray alloc] init];
    //adds the downloaded comments to the local array
    for (int i = 0; i < fullComments.count; i++) {
        NSDictionary *tmpDic = [fullComments objectAtIndex:i];
        //NSLog(@"Dictionary: %@", tmpDic);
        NSString *tmpString = [tmpDic objectForKey:@"comment_text"];
        //NSLog(@"Comment Text: %@", tmpString);
        [_item addComment:tmpString];
        //NSLog(@"Comment Text in comments: %@", _item.comments);
    }
    //NSLog(@"Item Comments: %@", _item.comments);
    //updates the view
    [self showComments];
```

```
    //closes the url seession
    [session invalidateAndCancel];

}

//programmatially called segue (so that prepare foe segue can be used
-(IBAction)back:(id)sender {
    [self performSegueWithIdentifier:@"returnFromComments" sender:self];
}

//passes on the item info to the ItemDetail view controller
-(void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    if ([segue.identifier isEqualToString:@"returnFromComments"]) {
        ItemDetail *destinationViewController = segue.destinationViewController
            ;
        destinationViewController.itemOnDisplay = _item;
    }
}

//posts the user's comment when they click post
-(IBAction)postComment:(id)sender {
    NSString *tmp = inputComment.text;
    //if comment is empty it does not upload it
    if ([tmp isEqual: @""] || [tmp isEqual: @"Type comment here"]) {
        inputComment.text = [NSString stringWithFormat:@"Type comment here"];
        //show error for "please type comment"
    }
    else {
        NSString *username = [[NSUserDefaults standardUserDefaults]
            objectForKey:@"username"];
        NSLog(@"%@", username);
        _userComment = [NSString stringWithFormat:@"%@:%@", username, tmp];
    }
    //if the comment has been posted it uploads it and resets the input
        textfield
    if (_userComment != nil) {
        [_item uploadComment:_userComment];
        [self showComments];
        inputComment.text = @"Type comment here";
        if ([inputComment isFirstResponder]) {
            [inputComment resignFirstResponder];
        }
        [self.view endEditing:YES];
    }
    //alert for when no comment has been posted
    else {
        UIAlertController *alert = [UIAlertController
            alertControllerWithTitle:@"Missing Comment" message:@"Please input
            a comment to post." preferredStyle:UIAlertControllerStyleAlert];
        UIAlertAction* defaultAction = [UIAlertAction actionWithTitle:@"OK"
            style:UIAlertActionStyleDefault handler:^(UIAlertAction * action)
            {}];
        [alert addAction:defaultAction];
        [self presentViewController:alert animated:YES completion:nil];
    }
}
```

```
-(void)showComments {
    //NSLog(@"Item has comments: %@", _item.comments);
    //creates empty string to append all of the comments on
    NSString *commentsOnThisItem = @"";
    //loops through all of the comments and appends them with the specified
        format
    for (int i = 0; i < _item.comments.count; i++) {
        NSString *username;
        NSString *text;
        NSString *full = [_item commentWithIndex:i];
        for (int i = 0; i < (full.length-1); i++) {
            if ([@":" isEqualToString:[[full substringFromIndex:i]
                substringToIndex:1]]) {
                username = [full substringToIndex:i];
                text = [full substringFromIndex:i+1];
                break;
            }
        }
        //does not use appendString b/c that failed in testing
        //adds the comment in the specified format
        commentsOnThisItem = [NSString stringWithFormat:@"%@%@:\n\t%@\n\n",
            commentsOnThisItem, username, text];
    }
    //if no comments displays No Comments
    if ([commentsOnThisItem isEqualToString:@""]) {
        commentsOnThisItem = [NSString stringWithFormat:@"No Comments"];
    }
    //loading the image on the page
    if (_item.image != nil) {
        showImage.image = _item.image;
    }
    //if images is missing it shows the default image
    else {
        showImage.image = [UIImage imageNamed:@"missing.png"];
    }
    //manages the purchased image
    if ([_item.itemPurchaseState intValue] == 1) {
        purchased.hidden = NO;
    }
    else {
        purchased.hidden = YES;
    }
    //NSLog(@"%@\n%@", commentsOnThisItem, _item.comments);
    otherComments.text = commentsOnThisItem;
}

//if user has input text, saves the text for posting, if they have not,
    replaces the default text
-(void)textViewDidEndEditing:(UITextView *)textView {
    //creates a copy of the string with whitespace removed to check if visible
        text has been entered so they will not lose the text box
    NSString *tmp = [inputComment.text stringByTrimmingCharactersInSet:
        [NSCharacterSet whitespaceAndNewlineCharacterSet]];
    if ([tmp isEqual: @""] || [tmp isEqual: @"Type comment here"]) {
        inputComment.text = [NSString stringWithFormat:@"Type comment here"];
```

```
    }
    else {
        _userComment = inputComment.text;
    }
    [textView resignFirstResponder];
}

//removes the placeholder text
-(void)textViewDidBeginEditing:(UITextView *)textView {
    if ([textView.text isEqualToString:@"Type comment here"]) {
        textView.text = [NSString stringWithFormat:@""];
    }
}

//makes keyboard disappear when return (done) pressed
- (BOOL)textView:(UITextView *)textView shouldChangeTextInRange:(NSRange)range
    replacementText:(NSString *)text {

    if([text isEqualToString:@"\n"]) {

        [textView resignFirstResponder];
        return NO;
    }

    return YES;
}

//makes sure they cant edit the view that displays all of the comments
- (BOOL)textViewShouldBeginEditing:(UITextView *)textView
{
    if ([textView isEqual:otherComments]) {
        return NO;
    }
    else {
        return YES;
    }
}

- (void)viewDidLoad {
    [super viewDidLoad];
    inputComment.delegate = self;
    otherComments.delegate = self;
    [self downloadComments];
    // Do any additional setup after loading the view.
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

```
//
//  Items.h
//  Garage Sale
//
//  Created by Alexander Hammond on 1/16/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

//imports
#import <UIKit/UIKit.h>
#import "Item.h"
#import "ItemCustomCell.h"



//declaring interface outlets
@interface Items : UIViewController <UITableViewDelegate, UITableViewDataSource
    , NSURLSessionDelegate>{

     //needs to be saved to the device


    IBOutlet UITableView *itemsView;
    IBOutlet UISegmentedControl *itemListType;
    IBOutlet UINavigationItem *itemName;



}

//declaring properties
@property NSMutableArray *items;
@property NSArray *likedItems;
//if fails make NSMutableArray
@property NSArray *filteredResults;
@property NSMutableArray *filteredItems;
@property NSArray *result;
@property bool showFiltered;
@property bool showAll;
@property Item *itemToSend;
@property NSDictionary *filters;


//methods that can be accessed when importing this header file
-(void)loadAllItems;
-(void)loadLikedItems;

-(Item *)itemFromDictionaryExternal:(NSDictionary *)dictionary;


@end
```

```
//
//  Items.m
//  Garage Sale
//
//  Created by Alexander Hammond on 1/16/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import "Items.h"
#import "ItemDetail.h"
#import "Filters.h"

@interface Items () <UITableViewDelegate, UITableViewDataSource,
    NSURLSessionDelegate>

@end

@implementation Items

//when an item clicked in the table view it passes on the item info and shows
    the item
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
    *)indexPath {
    ItemCustomCell *tmpCell = [itemsView cellForRowAtIndexPath:indexPath];
    _itemToSend = tmpCell.item;
    [self performSegueWithIdentifier:@"showItem" sender:indexPath];
}

//goes to the filters page
-(IBAction)filters:(id)sender {
    [self performSegueWithIdentifier:@"toFilters" sender:self];
}

//passes on information based on which segue is performed
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    if ([segue.identifier isEqualToString:@"showItem"]) {
        ItemDetail *destViewController = segue.destinationViewController;
        destViewController.itemOnDisplay = _itemToSend;
        [destViewController updateView];
    }
    if ([segue.identifier isEqualToString:@"toFilters"]) {
        Filters *destination = segue.destinationViewController;
        destination.filtersInPlace = _filters;
    }
}

//delegate method used to load table view
-(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
    (NSIndexPath *)indexPath {
    ItemCustomCell *cell = (ItemCustomCell *)[tableView
        dequeueReusableCellWithIdentifier:@"ItemCell" forIndexPath:indexPath];
    if (!_showAll) {
        //loads item from array of liked items if set to not show all
        cell.item = [_likedItems objectAtIndex:indexPath.row];
    }
    else {
```

```
        //loads item from array of all items if set to show all
        if (_showFiltered) {
            cell.item = [_filteredItems objectAtIndex:indexPath.row];
        }
        else {
            cell.item = [_items objectAtIndex:indexPath.row];
        }
    }
    cell.parentTable = itemsView;

    //updates the views in the cell
    [cell updateCell];

    return cell;
}

//provides the number of rows that will be in table view (just a count of the
    array)
-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger
    )section {
    if (!_showAll) {
        return _likedItems.count;
    }
    else {
        if (_showFiltered) {
            return _filteredItems.count;
        }
        return _items.count;
    }
}

//run when the segmented control switched - switches between liked and all (and
    when all decides between filtered and all)
-(void)viewSwitched {
    if (_showAll) {
        _showAll = false;
        [self loadLikedItems];
        [itemsView reloadData];
    }
    else {
        _showAll = true;
        [self loadLikedItems];
        //refreshes the liked list in case a user disliked an item while
            viewing the liked items list
        NSMutableArray *tmpItemArray = [_items mutableCopy];
        for (int i = 0; i < tmpItemArray.count; i++) {
            //must reevaluate every single item in case it was liked while
                filtered
            [[tmpItemArray objectAtIndex:i] setLiked:false];
            for (int j = 0; j < _likedItems.count; j++) {
                if ([[tmpItemArray objectAtIndex:i] getItemID] == [[_likedItems
                    objectAtIndex:j] getItemID]) {
                    [[tmpItemArray objectAtIndex:i] setLiked:true];
                }
            }
        }
```

```
        _items = tmpItemArray;
        //assign a new memory address to prevent accidental copies
        tmpItemArray = [[NSMutableArray alloc] init];
        tmpItemArray = [_filteredItems mutableCopy];
        for (int i = 0; i < tmpItemArray.count; i++) {
            //must reevaluate every single item in case it was liked while
                filtered
            [[tmpItemArray objectAtIndex:i] setLiked:false];
            for (int j = 0; j < _likedItems.count; j++) {
                if ([[tmpItemArray objectAtIndex:i] getItemID] == [[_likedItems
                    objectAtIndex:j] getItemID]) {
                    [[tmpItemArray objectAtIndex:i] setLiked:true];
                }
            }
        }
        _filteredItems = tmpItemArray;

        //no point in reloading data from the server here
        [itemsView reloadData];
        //[self loadAllItems];
        //loadAllItems already calles reloadData
    }
}




//loads all of the items
-(void)loadAllItems {

    //-- Make URL request with server to load all of the items
    if (_items != nil) {
        [itemsView reloadData];
    }
    NSString *jsonUrlString = [NSString stringWithFormat:@"https://murmuring-
        everglades-79720.herokuapp.com/items.json"];
    NSURL *url = [NSURL URLWithString:jsonUrlString];
    NSURLSessionConfiguration* config = [NSURLSessionConfiguration
        defaultSessionConfiguration];
    NSURLSession *session = [NSURLSession sessionWithConfiguration:config
        delegate:self delegateQueue:[NSOperationQueue mainQueue]];
    NSURLSessionDataTask *dataTask = [session dataTaskWithURL:url];
    [dataTask resume];
}


//loads all of the items when the data task is complete
- (void)URLSession:(NSURLSession *)session
          dataTask:(NSURLSessionDataTask *)dataTask
    didReceiveData:(NSData *)data {

    NSError *error;
    _result = [NSJSONSerialization JSONObjectWithData:data options:
```

```
        NSJSONReadingMutableContainers error:&error];
    //NSLog(@"Result (Length: %zd) = %@",_result.count, _result);
    //this interprets the data received a creates a bunch of items from it
    NSMutableArray *tmpItemArray = [[NSMutableArray alloc] init];
    for (int i = 0; i < _result.count; i++) {
        NSDictionary *tmpDic = [_result objectAtIndex:i];
        //NSLog(@"Dictionary %@", tmpDic);
        Item *loadItem = [self itemFromDictionaryExternal:tmpDic];
        [self loadItemImage:loadItem];
        [tmpItemArray addObject:loadItem];
    }
    for (int i = 0; i <tmpItemArray.count; i++) {

    }

    //updates the liked items list to load which of the new items the user has
        liked
    [self loadLikedItems];

    //sets the 'liked' value of the loaded items
    for (int i = 0; i < tmpItemArray.count; i++) {
        for (int j = 0; j < _likedItems.count; j++) {
            if ([[tmpItemArray objectAtIndex:i] getItemID] == [[_likedItems
                objectAtIndex:j] getItemID]) {
                [[tmpItemArray objectAtIndex:i] setLiked:true];
            }
        }
    }
    //if data receieved it saves the interpreted data to the local array
    if (tmpItemArray != nil) {
        _items = tmpItemArray;
    }

    else {
        //if no data received it provides this alert
        UIAlertController *alert = [UIAlertController
            alertControllerWithTitle:@"No Connection\n" message:@"Could not
            load items" preferredStyle:UIAlertControllerStyleAlert];
        UIAlertAction* defaultAction = [UIAlertAction actionWithTitle:@"OK"
            style:UIAlertActionStyleDefault handler:^(UIAlertAction * action)
            {}];
        [alert addAction:defaultAction];
        [self presentViewController:alert animated:YES completion:nil];
    }
    [itemsView reloadData];
    [session invalidateAndCancel];

}

-(void)loadLikedItems {
    //creates and loads the likedItems array
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    NSArray *likedArray = [defaults objectForKey:@"LikedItems"];
    NSMutableArray *tmpLikedItems = [[NSMutableArray alloc] init];
    for (int i = 0; i < likedArray.count; i++) {
        NSDictionary *tmpDic = [likedArray objectAtIndex:i];
```

```
            [tmpLikedItems addObject:[self itemFromDictionaryInternal:tmpDic]];
        }
        _likedItems = [tmpLikedItems copy];
        for (int i = 0; i < _likedItems.count; i++) {
            Item *tmpItem = [_likedItems objectAtIndex:i];
            if ([tmpItem.itemPurchaseState intValue] != 1) {
                [self checkPurchased:[_likedItems objectAtIndex:i]];
            }
        }
    }

    //checks to see if item has been purchased - this runs in the background
        separately from downloading all of the data - used for updating whether or
        not to show the purchased image on items loaded locally (liked items)
    -(void)checkPurchased:(Item *)item {
        //creates bacground queue
        dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0
            ), ^{

           //initiates a urlsession to check the purchase stat
            NSURL *url = [NSURL URLWithString:[NSString stringWithFormat:@"https://
                murmuring-everglades-79720.herokuapp.com/items/%zd.json", item.
                itemID]];
            //NSLog(@"%@", url);

            NSURLSession *session = [NSURLSession sessionWithConfiguration:
                [NSURLSessionConfiguration defaultSessionConfiguration]];
            NSURLSessionDataTask *dataTask = [session dataTaskWithURL:url
                completionHandler:^(NSData *data, NSURLResponse *response, NSError
                *error) {
                if (![[[data class] description]
                    isEqualToString:@"__NSCFConstantString"]) {

                    //interprets the received data to check the purchase state
                    NSDictionary *tmpDic = [NSJSONSerialization JSONObjectWithData:
                        data options:NSJSONReadingMutableContainers error:&error];
                    NSNumber *tmpNum = [tmpDic objectForKey:@"item_purchase_state"]
                        ;
                    //if purchased it updates the saved object for the key
                        LikedItems (where the liked list is stored) to update the
                        purchase state of the item
                    if ([tmpNum intValue] == 1) {
                        //creates instance of defaults to access locally saved
                            objects
                        NSUserDefaults *defaults = [NSUserDefaults
                            standardUserDefaults];
                        NSArray *likedArray = [defaults objectForKey:@"LikedItems"]
                            ;
                        NSMutableArray *tmpLikedItems = [[NSMutableArray alloc]
                            init];
                        NSMutableArray *tmpDicArray = [[NSMutableArray alloc] init]
                            ;

                        for (int i = 0; i < likedArray.count; i++) {
                            NSDictionary *tmpDic = [likedArray objectAtIndex:i];
                            [tmpLikedItems addObject:[self
```

```
                            itemFromDictionaryInternal:tmpDic]];
                        [tmpDicArray addObject:tmpDic];
                    }
                    for (int i = 0; i < tmpLikedItems.count; i++) {
                        Item *tmpItem = [tmpLikedItems objectAtIndex:i];
                        if (tmpItem.itemID == item.itemID) {
                            tmpItem.itemPurchaseState = [NSNumber
                                numberWithInt:1];
                            [tmpItem setItemDictionary];

                            [tmpLikedItems replaceObjectAtIndex:i withObject:
                                tmpItem];

                            [tmpDicArray replaceObjectAtIndex:i withObject:
                                tmpItem.localDictionary];
                        }

                    }
                    [defaults setObject:[tmpDicArray copy] forKey:@"LikedItems"
                        ];
                }
                [itemsView reloadData];
            }
        }];
        [dataTask resume];
    });

}

//reloads all of the items (for the reload button specifically)
-(IBAction)reload:(id)sender {
    [self loadLikedItems];
    [self loadAllItems];
        //refreshes the liked list in case a user disliked an item while
            viewing the liked items list
        NSMutableArray *tmpItemArray = [_items mutableCopy];
        for (int i = 0; i < tmpItemArray.count; i++) {
            //must reevaluate every single item in case it was liked while
                filtered
            [[tmpItemArray objectAtIndex:i] setLiked:false];
            for (int j = 0; j < _likedItems.count; j++) {
                if ([[tmpItemArray objectAtIndex:i] getItemID] == [[_likedItems
                    objectAtIndex:j] getItemID]) {
                    [[tmpItemArray objectAtIndex:i] setLiked:true];
                }
            }
        }
        _items = tmpItemArray;
        //assign a new memory address to prevent accidental copies
        tmpItemArray = [[NSMutableArray alloc] init];
        tmpItemArray = [_filteredItems mutableCopy];
        for (int i = 0; i < tmpItemArray.count; i++) {
            //must reevaluate every single item in case it was liked while
                filtered
            [[tmpItemArray objectAtIndex:i] setLiked:false];
            for (int j = 0; j < _likedItems.count; j++) {
```

```
                if ([[tmpItemArray objectAtIndex:i] getItemID] == [[_likedItems
                    objectAtIndex:j] getItemID]) {
                    [[tmpItemArray objectAtIndex:i] setLiked:true];
                }
            }
        }
        _filteredItems = tmpItemArray;

        //no point in reloading data from the server here
        [itemsView reloadData];
        //[self loadAllItems];
        //loadAllItems already calles reloadData

}

-(void)loadFilteredItems {
    NSMutableArray *tmpItemArray = [[NSMutableArray alloc] init];
    for (int i = 0; i < _filteredResults.count; i++) {
        NSDictionary *tmpDic = [_filteredResults objectAtIndex:i];
        //NSLog(@"Dictionary %@", tmpDic);
        Item *tmpItem = [self itemFromDictionaryExternal:tmpDic];
        [self loadItemImage:tmpItem];
        [tmpItemArray addObject:tmpItem];
    }
    for (int i = 0; i < tmpItemArray.count; i++) {
        for (int j = 0; j < _likedItems.count; j++) {
            if ([[tmpItemArray objectAtIndex:i] getItemID] == [[_likedItems
                objectAtIndex:j] getItemID]) {
                [[tmpItemArray objectAtIndex:i] setLiked:true];
            }
        }
    }
    _filteredItems = tmpItemArray;
}

//loads the images for an item (assumes item has saved url)
-(void)loadItemImage:(Item *)item {
    //creates an asynchronous queue so that loading the item will not interfer
        with the main queue (main queue remains open for ui updates)
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0
        ), ^{
        NSLog(@"Item Url:\n%@", item.url);
        item.image = [UIImage imageWithData:[NSData dataWithContentsOfURL:
            [NSURL URLWithString:item.url]]];
        item.imageLoadAttempted = true;
        //runs commands in the main queue once the loading is finished
        dispatch_sync(dispatch_get_main_queue(), ^{
            //reloads the tableView now that images have been saved
            [itemsView reloadData];
        });
    });

}

//reloads items saved to the phone (only difference is that loading images is
    done locally not from a URL
```

```
-(Item *)itemFromDictionaryInternal:(NSDictionary *) dictionary {
    Item *tmpItem = [[Item alloc] init];
    tmpItem.name = [dictionary objectForKey:@"item_name"];
    tmpItem.condition = (NSInteger *)[[dictionary
        objectForKey:@"item_condition"] integerValue];
    tmpItem.itemDescription = [dictionary objectForKey:@"item_description"];
    NSData *imageData = [dictionary objectForKey:@"item_image"];
    if (imageData != nil) {
        tmpItem.image = [UIImage imageWithData:imageData];
    }
    tmpItem.priceInCents = (NSInteger*)[[dictionary
        objectForKey:@"item_price_in_cents"] integerValue];
    //NSLog(@"%@", [dictionary objectForKey:@"item_description"]);
    tmpItem.liked = [[dictionary objectForKey:@"liked"] boolValue];
    NSInteger *tmpID = (NSInteger*)[[dictionary objectForKey:@"id"]
        integerValue];
    //NSLog(@"%@", [dictionary objectForKey:@"id"]);
    //NSLog(@"%zd", tmpID);
    tmpItem.itemID = tmpID;
    tmpItem.itemPurchaseState = (NSNumber *)[dictionary
        objectForKey:@"item_purchase_state"];
    NSArray *tmpComments = [dictionary objectForKey:@"item_comments"];
    [tmpItem.comments removeAllObjects];
    for (int i = 0; i < tmpComments.count; i++) {
        [tmpItem.comments addObject:[tmpComments objectAtIndex:i]];
    }
    return tmpItem;
}

//Loads items from the dictionary passed by the server (uses URL for image
    grab)
-(Item *)itemFromDictionaryExternal:(NSDictionary *) dictionary {
    Item *tmpItem = [[Item alloc] init];
    tmpItem.name = [dictionary objectForKey:@"item_name"];
    tmpItem.condition = (NSInteger *)[[dictionary
        objectForKey:@"item_condition"] integerValue];
    tmpItem.itemDescription = [dictionary objectForKey:@"item_description"];
    NSDictionary *jsonImageFilepath = [dictionary objectForKey:@"item_image"];

    NSString *imageFilepath = [jsonImageFilepath objectForKey:@"url"];
    tmpItem.url = [NSString stringWithFormat:@"%@", imageFilepath];
    NSLog(@"%@", tmpItem.url);

    NSArray *fullComments =  [dictionary objectForKey:@"comments"];
    //NSLog(@"All comments:\n%@", fullComments);
    tmpItem.comments = [[NSMutableArray alloc] init];
    for (int i = 0; i < fullComments.count; i++) {
        NSDictionary *tmpDic = [fullComments objectAtIndex:i];
        //NSLog(@"Dictionary: %@", tmpDic);
        NSString *tmpString = [tmpDic objectForKey:@"comment_text"];
        //NSLog(@"Comment Text: %@", tmpString);
        [tmpItem addComment:tmpString];
        //NSLog(@"Comment Text in comments: %@", tmpItem.comments);
    }
    //NSLog(@"Item Comments: %@", tmpItem.comments);
```

```
    tmpItem.priceInCents = (NSInteger*)[[dictionary
        objectForKey:@"item_price_in_cents"] integerValue];

    tmpItem.liked = [[dictionary objectForKey:@"liked"] boolValue];
    NSInteger *tmpID = (NSInteger*)[[dictionary objectForKey:@"id"]
        integerValue];
    //NSLog(@"%@", [dictionary objectForKey:@"id"]);
    //NSLog(@"%zd", tmpID);
    tmpItem.itemID = tmpID;
    //NSLog(@"%zd", tmpItem.itemID);
    tmpItem.itemPurchaseState = (NSNumber *)[dictionary
        objectForKey:@"item_purchase_state"];
    return tmpItem;
}

//setup coe for the view - initiates loading the items arrays and sets the
    delegates for the necessary views to the view controller
- (void)viewDidLoad {
    [super viewDidLoad];

    // Do any additional setup after loading the view.
    _items = [[NSMutableArray alloc] init];
    _likedItems = [[NSArray alloc] init];

    [self loadAllItems];
    [self loadLikedItems];
    [self loadFilteredItems];
    //show all items not liked items
    _showAll = true;
    itemsView.delegate = self;
    itemsView.dataSource = self;
    [itemListType addTarget:self
                     action:@selector(viewSwitched)
           forControlEvents:UIControlEventValueChanged];



}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}



@end
```

```
//
//  Item.h
//  Garage Sale
//
//  Created by Alexander Hammond on 1/19/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>


@interface Item : NSObject  {

}

//defines properties
@property NSString *name;
@property NSInteger *condition;
@property NSString *itemDescription;
@property NSInteger *priceInCents;
@property UIImage *image;
@property NSInteger *itemID;
@property NSMutableArray *comments;
@property bool liked;
@property NSDictionary *localDictionary;
@property NSNumber *itemPurchaseState;
@property NSString *url;
@property bool imageLoadAttempted;



//some simpler get statements (default set statements adequate)
-(NSString *)getName;

-(NSInteger *)getCondition;

-(NSString *)getItemDescription;

-(NSInteger *)getPriceInCents;
-(void)setThePriceInCents:(int)price;
-(NSString *)getPriceString;

-(UIImage *)getImage;

-(NSInteger *)getItemID;
-(void)setTheItemID:(int)ID;

-(bool)getLiked;

-(void)changeLiked;

-(void)changeComment:(NSString *)oldComment toComment:(NSString *)newComment;

-(void)addComment:(NSString *)comment;
-(void)uploadComment:(NSString *)comment;
```

```
-(NSString *)commentWithIndex:(int)index;

-(void)setItemDictionary;




@end
```

```
//
//  Item.m
//  Garage Sale
//
//  Created by Alexander Hammond on 1/19/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import "Item.h"

@implementation Item


-(void)uploadComment:(NSString *)comment {
    [_comments addObject:comment];
    NSError *error;

    //creates mutable copy of the dictionary to remove extra keys
    NSMutableDictionary *tmpDic = [NSMutableDictionary dictionaryWithObject:
        comment forKey:@"comment_text"];
    [tmpDic setObject:[NSString stringWithFormat:@"%zd", _itemID]
        forKey:@"item_id"];

    //converts the dictionary to json
    NSData *jsonData = [NSJSONSerialization dataWithJSONObject:tmpDic options:
        NSJSONWritingPrettyPrinted error:&error];
    //logs the data to check if it is created successfully
    //NSLog(@"%@", [[NSString alloc] initWithData:jsonData
        encoding:NSUTF8StringEncoding]);

    //creates url for the request
    NSURL *url = [NSURL URLWithString:@"https://murmuring-
        everglades-79720.herokuapp.com/comments.json"];

    //creates a URL request
    NSMutableURLRequest *uploadRequest = [NSMutableURLRequest requestWithURL:
        url cachePolicy:NSURLRequestUseProtocolCachePolicy timeoutInterval:60.0
        ];

    //specifics for the request (it is a post request with json content)
    [uploadRequest setHTTPMethod:@"POST"];
    [uploadRequest setValue:@"application/json" forHTTPHeaderField:@"Accept"];
    [uploadRequest setValue:@"application/json" forHTTPHeaderField:@"Content-
        Type"];
    [uploadRequest setValue:[NSString stringWithFormat:@"%lu", (unsigned long)
        [jsonData length]] forHTTPHeaderField:@"Content-Length"];
    [uploadRequest setHTTPBody: jsonData];

    //
    NSURLSession *session = [NSURLSession sessionWithConfiguration:
        [NSURLSessionConfiguration defaultSessionConfiguration]];

    [[session dataTaskWithRequest:uploadRequest completionHandler:^(NSData *
        data, NSURLResponse *response, NSError *error) {
        dispatch_async(dispatch_get_main_queue(), ^{
        NSString *requestReply = [[NSString alloc] initWithData:data encoding:
```

```
            NSASCIIStringEncoding];
        NSLog(@"requestReply: %@", requestReply);
        });
    }] resume];
    //add the comment to the server
}

//quick helper method to return specific comments
-(NSString *)commentWithIndex:(int)index {
    return [_comments objectAtIndex:index];
}

//some short getter methods
-(NSString *)getName {
    return _name;
}


-(NSInteger *)getCondition {
    return _condition;
}


-(NSString *)getItemDescription {
    return _itemDescription;
}


-(NSInteger *)getPriceInCents {
    return _priceInCents;
}

-(UIImage *)getImage {
    return _image;
}

-(NSInteger *)getItemID {
    return _itemID;
}

-(bool)getLiked {
    return _liked;
}


//this method converts the item's price into a formatted string to display
-(NSString *)getPriceString {
    int tmpPriceInCents = (int)_priceInCents;
    //NSLog(@"Price in cents: %zd\nPriceInCents %i", _priceInCents,
        tmpPriceInCents);
    int tmpCentsOnes = tmpPriceInCents %10;
    int tmpCentsTens = ((tmpPriceInCents - tmpCentsOnes)%100)/10;
    NSString *priceCents = [NSString stringWithFormat:@"%i%i", tmpCentsTens,
        tmpCentsOnes];
    NSString *priceDollars = [NSString stringWithFormat:@"%i", (tmpPriceInCents
        /100)];
```

```
    NSString *priceString =[NSString stringWithFormat:@"$%@.%@", priceDollars,
        priceCents];
    return priceString;
}

//helper to set the price for item with int instead of NSInteger
-(void)setThePriceInCents:(int)price {
    long tmpPrice = price;
    _priceInCents = (NSInteger *)tmpPrice;
}

//sets the id for the item with an int
-(void)setTheItemID:(int)ID {
    long tmpID = ID;
    _itemID = (NSInteger *)tmpID;
}

//changes the liked status of the item
-(void)changeLiked {
    //loads array of liked items
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    NSArray *likedArray = [defaults objectForKey:@"LikedItems"];

    //creates a copy to edit (all objects loaded from defaults are immutable so
        mutableCopy command does not work)
    NSMutableArray *likedArrayMutable = [[NSMutableArray alloc] init];
    //copies all of the items from the liked array into the mutable array
    for (int i = 0; i < likedArray.count; i++) {
        [likedArrayMutable addObject: [likedArray objectAtIndex:i]];
    }

    //if the item was not liked, sets liked to true and adds the item to the
        array
    if (_liked == false) {
        _liked = true;
        [self setItemDictionary];
        [likedArrayMutable addObject:_localDictionary];

    }
    //if the item was liked, sets to false and removes it from the array
    else {
        _liked = false;
        for (int i = 0; i < likedArrayMutable.count; i++) {
            NSDictionary *tmpDic = [likedArray objectAtIndex:i];

            NSInteger *tmpInteger = (NSInteger*)[[tmpDic valueForKey:@"id"]
                integerValue];

            if (tmpInteger == _itemID) {
                [likedArrayMutable removeObjectAtIndex:i];
            }
        }
    }
    //creates a non-mutable copy of the updated array (cannot save mutable
        arrays to user defaults)
```

```
    NSArray *newLikedArray = [likedArrayMutable copy];
    //saves the updated array
    [defaults setObject:newLikedArray forKey:@"LikedItems"];
    [defaults synchronize];
}

//updates the internal dictionary of the item (helps when editing items from
    other classes)
-(void)setItemDictionary {
    NSData *imageData = UIImageJPEGRepresentation(_image, .6);
    NSNumber *likedData = [NSNumber numberWithBool:_liked];
    NSArray *commentsCopy = [_comments copy];
    NSNumber *purchaseState = _itemPurchaseState;
    //NSLog(@"Purchase State: %@", purchaseState);
    if (imageData == nil)
        imageData = [[NSData alloc] init];
    if (commentsCopy == nil) {
        commentsCopy = [[NSArray alloc] init];
    }
    if (likedData == nil) {
        likedData = [NSNumber numberWithBool:true];
    }
    if (purchaseState == nil) {
        purchaseState = [[NSNumber alloc] initWithInt:-1];
    }
    //NSLog(@"These Can't Be NULL: %@, %@, %@", imageData, commentsCopy,
        likedData);
    _localDictionary = [[NSDictionary alloc] initWithObjectsAndKeys:
    _name, @"item_name",
    [NSString stringWithFormat:@"%zd", _condition], @"item_condition",
    _itemDescription, @"item_description",
    [NSString stringWithFormat:@"%zd", _priceInCents], @"item_price_in_cents",
    likedData, @"liked",
    [NSString stringWithFormat:@"%zd", _itemID], @"id",
    imageData, @"item_image",
    commentsCopy, @"item_comments",
    purchaseState, @"item_purchase_state",

                        nil];
}


//adds a comment
-(void)addComment:(NSString *)comment {
    [_comments addObject:comment];
}

//included in case need, should not be needed
-(void)changeComment:(NSString *)oldComment toComment:(NSString *)newComment {
    for (int i = 0; i < _comments.count; i++) {
        if ([[_comments objectAtIndex:i] isEqualToString:oldComment]) {
            [_comments replaceObjectAtIndex:i withObject:newComment];
        }
    }
}
-(void)removeComment:(NSString *)comment {
```

```
    for (int i = 0; i < _comments.count; i++) {
        if ([[_comments objectAtIndex:i] isEqualToString:comment]) {
            [_comments removeObjectAtIndex:i];
        }
    }
}

@end
```

```
//
//  ItemCustomCell.h
//  Garage Sale
//
//  Created by Alexander Hammond on 1/21/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "Item.h"


//class declarations
@interface ItemCustomCell : UITableViewCell

@property Item *item;
@property UITableView *parentTable;

@property IBOutlet UILabel *name;
@property IBOutlet UILabel *condition;
@property IBOutlet UILabel *price;
@property IBOutlet UIButton *likeButton;
@property IBOutlet UIImageView *image;
@property IBOutlet UIImageView *purchased;

-(void)updateCell;

@end
```

```
//
//  ItemCustomCell.m
//  Garage Sale
//
//  Created by Alexander Hammond on 1/21/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import "ItemCustomCell.h"


@implementation ItemCustomCell


//updates all of the UI elements of the cell to match the item data
-(void)updateCell {
    //hides purchased during loading sequence
    _purchased.hidden = YES;
    NSArray *conditionOptionsCustom = [[NSUserDefaults standardUserDefaults]
        objectForKey:@"conditions"];
    _name.text = _item.name;
    int tmpInt = (int)_item.condition;
    _condition.text = conditionOptionsCustom[tmpInt];
    _price.text = [_item getPriceString];
    if ([_item.itemPurchaseState intValue] == 1) {
        _purchased.hidden = NO;
    }
    else {
        _purchased.hidden = YES;
    }
    //updates like button appearance based on whether or not the object has
        been liked
    if (_item.liked) {
        [_likeButton setImage:[UIImage imageNamed:@"Instagram-Heart-
            Solid@3x.png"] forState:UIControlStateNormal];
    }
    else {
        [_likeButton setImage:[UIImage imageNamed:@"Instagram-Heart-
            Transparent.png"] forState:UIControlStateNormal];
    }
    if ([_image isAnimating]) {
        [_image stopAnimating];
    }
    if (_item.image != nil) {
        _image.image = _item.image;
    }
    else {
        //if image failed to load, shows missing image
        if (_item.imageLoadAttempted == true) {
            _image.image = [UIImage imageNamed:@"missing.png"];
        }
        else {
            //loading animation
            UIImage *image1 = [UIImage imageNamed:@"large1.png"];
            UIImage *image2 = [UIImage imageNamed:@"large2.png"];
            UIImage *image3 = [UIImage imageNamed:@"large3.png"];
```

```
            UIImage *image4 = [UIImage imageNamed:@"large4.png"];
            _image.animationImages = @[image1, image2, image3, image4];
            _image.animationDuration = 1;
            _image.animationRepeatCount = 0;
            [_image startAnimating];
            _image.image = [UIImage imageNamed:@"default.png"];
        }
    }

}

//updates the liked status of the item then updates the view
-(IBAction)likeButtonClicked:(id)sender {
    [_item changeLiked];
    [self updateCell];
    [_parentTable reloadData];
}

- (void)awakeFromNib {
    [super awakeFromNib];
}

- (void)setSelected:(BOOL)selected animated:(BOOL)animated {
    [super setSelected:selected animated:animated];

}

@end
```

```
//
//  Donate.h
//  Garage Sale
//
//  Created by Alexander Hammond on 1/16/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "Item.h"

//variable declarations
NSString *name;
NSInteger *conditionInt;
NSString *condition;
NSInteger *priceInCents;
NSString *description;
UIImage *image;
bool imageUploaded;

NSArray *conditionOptionsDonate;



@interface Donate : UIViewController <UIPickerViewDataSource,
    UIPickerViewDelegate, UITextViewDelegate, UITextFieldDelegate,
    UIImagePickerControllerDelegate, UINavigationControllerDelegate> {
    //outlet declarations (UI elements)
    IBOutlet UITextField *nameTextField;
    IBOutlet UITextField *conditionTextField;
    IBOutlet UITextField *priceTextField;
    IBOutlet UITextView *descriptionTextView;
    IBOutlet UIButton *imageView;
    IBOutlet UIButton *cameraButton;
    IBOutlet UILabel *tmpLabel;
}

//class properties
@property (strong, nonatomic) UIPickerView *conditionPicker;
@property UIToolbar *toolBar;

//Action methods (actions are linked to UI events)
-(IBAction)done:(id)sender;
-(IBAction)selectImage:(id)sender;

@end
```

```
//
//  Donate.m
//  Garage Sale
//
//  Created by Alexander Hammond on 1/16/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import "Donate.h"
#import "Item.h"


@interface Donate () <UITextViewDelegate, UITextFieldDelegate,
    UIPickerViewDelegate, UIPickerViewDataSource,
    UIImagePickerControllerDelegate, UINavigationControllerDelegate>

@end

@implementation Donate

//run when user clicks on the image or the camera button
-(IBAction)selectImage:(id)sender {
    //creates image picker
    UIImagePickerController *picker = [[UIImagePickerController alloc] init];
    picker.delegate = self;
    picker.allowsEditing = YES;
    //lets user choose camera or photo library
    UIAlertController *alert = [UIAlertController alertControllerWithTitle:nil
        message:nil preferredStyle:UIAlertControllerStyleActionSheet];
    //camera action
    UIAlertAction *camera = [UIAlertAction actionWithTitle:@"Camera" style:
        UIAlertActionStyleDefault handler:^(UIAlertAction *action) {
        //initiates picker with camera
        [picker setSourceType:UIImagePickerControllerSourceTypeCamera];
        [self presentViewController:picker animated:true completion:nil];
    }];
    //photo library action
    UIAlertAction *photoAlbum = [UIAlertAction actionWithTitle:@"Photo Library"
        style:UIAlertActionStyleDefault handler:^(UIAlertAction *action) {
        //initiates picker with photo library
        [picker setSourceType:UIImagePickerControllerSourceTypePhotoLibrary];
        [self presentViewController:picker animated:true completion:nil];
    }];
    //cancel action
    UIAlertAction *cancel = [UIAlertAction actionWithTitle:@"Cancel" style:
        UIAlertActionStyleCancel handler:^(UIAlertAction *action) {}];
    //adds all of the actions
    [alert addAction:camera];
    [alert addAction:photoAlbum];
    [alert addAction:cancel];
    //presents the options
    [self presentViewController:alert animated:true completion:nil];
}

//delegate methods for image picker
-(void)imagePickerController:(UIImagePickerController *)picker
```

```
    didFinishPickingMediaWithInfo:(NSDictionary<NSString *,id> *)info {
    //saves the selected image to the image field
    UIImage *tmpImage = info[UIImagePickerControllerEditedImage];
    image = tmpImage;
    imageUploaded = true;
    [imageView setBackgroundImage:image forState:UIControlStateNormal];
    [picker dismissViewControllerAnimated:YES completion:NULL];
}

- (void)imagePickerControllerDidCancel:(UIImagePickerController *)picker {

    [picker dismissViewControllerAnimated:YES completion:NULL];

}

//this is the method run when the users submits an item
-(IBAction)done:(id)sender {
    [self.view endEditing:YES];

    //first need to check that all fields have data
    if ([name isEqualToString:@""] || [condition isEqualToString:@""] ||
        [description isEqualToString:@""] || priceInCents == 0 || !
        imageUploaded) {
        NSString *errorMessage;
        if ([name isEqualToString:@""]) {
            errorMessage = @"Please put in a name for the item";
        }
        else if ([condition isEqualToString:@""]) {
            errorMessage = @"Please select a condition for this item";
        }
        else if ([description isEqualToString:@""]) {
            errorMessage = [NSString stringWithFormat: @"Please provide a brief
                description of your item"];
        }
        else if (priceInCents == 0) {
            errorMessage = @"Please suggest a price";
        }
        else if (!imageUploaded) {
            errorMessage = @"Please take a picture of this item by selecting
                the camera button.";
        }
        //shows error message for missing information
        UIAlertController *alert = [UIAlertController
            alertControllerWithTitle:@"Missing Information" message:
            errorMessage preferredStyle:UIAlertControllerStyleAlert];
        UIAlertAction* defaultAction = [UIAlertAction actionWithTitle:@"OK"
            style:UIAlertActionStyleDefault handler:^(UIAlertAction * action)
            {}];
        [alert addAction:defaultAction];
        [self presentViewController:alert animated:YES completion:nil];
    }
    //if all information present
    else {
        //closes keyboards
        [self.view endEditing:YES];
        //creates the Item object
```

```
Item *newItem = [[Item alloc] init];
newItem.name = name;
newItem.image = image;
newItem.condition = conditionInt;
newItem.priceInCents = priceInCents;
NSLog(@"%zd", newItem.priceInCents);
newItem.itemDescription = description;
//Note: item has not been liked (does nothing except prevent nil from
    stopping dictionary creation)
newItem.liked = false;
//0 means it has not been purchased
newItem.itemPurchaseState = 0;


//Then upload the item to the database
//refreshes the internal item dictionary
[newItem setItemDictionary];

//creates error handler
NSError *error;

//creates mutable copy of the dictionary to remove extra keys
NSMutableDictionary *tmpDic = [newItem.localDictionary mutableCopy];

//removes extra keys (item_image is replaced with a different key for
    the image data)
[tmpDic removeObjectForKey:@"id"];
[tmpDic removeObjectForKey:@"item_image"];
NSData *imageData = UIImageJPEGRepresentation(image, .6);
NSString *imageBase64 = [imageData base64EncodedStringWithOptions:
    NSDataBase64EncodingEndLineWithLineFeed];
//NSLog(@"Upload Data: %@", imageBase64);
[tmpDic setObject:imageBase64 forKey:@"va_image_data"];
[tmpDic setObject:[NSString stringWithFormat:@"%i", 0]
    forKey:@"item_purchase_state"];

//JSON Upload - does not upload the image
//converts the dictionary to json
NSData *jsonData = [NSJSONSerialization dataWithJSONObject:tmpDic
    options:NSJSONWritingPrettyPrinted error:&error];
//logs the data to check if it is created successfully
//NSLog(@"%@", [[NSString alloc] initWithData:jsonData
    encoding:NSUTF8StringEncoding]);

//creates url for the request
NSURL *url = [NSURL URLWithString:@"https://murmuring-
    everglades-79720.herokuapp.com/items.json"];

//creates a URL request
NSMutableURLRequest *uploadRequest = [NSMutableURLRequest
    requestWithURL:url cachePolicy:NSURLRequestUseProtocolCachePolicy
    timeoutInterval:60.0];

//specifics for the request (it is a post request with json content)
[uploadRequest setHTTPMethod:@"POST"];
[uploadRequest setValue:@"application/json"
```

```
            forHTTPHeaderField:@"Accept"];
        [uploadRequest setValue:@"application/json"
            forHTTPHeaderField:@"Content-Type"];
        [uploadRequest setValue:[NSString stringWithFormat:@"%lu", (unsigned
            long)[jsonData length]] forHTTPHeaderField:@"Content-Length"];
        [uploadRequest setHTTPBody: jsonData];

        //create some type of waiting image here
        NSURLSession *session = [NSURLSession sessionWithConfiguration:
            [NSURLSessionConfiguration defaultSessionConfiguration]];

        //runs the data task
        [[session dataTaskWithRequest:uploadRequest completionHandler:^(NSData
            *data, NSURLResponse *response, NSError *error) {
            dispatch_async(dispatch_get_main_queue(), ^{
            NSString *requestReply = [[NSString alloc] initWithData:data
                encoding:NSASCIIStringEncoding];
                NSLog(@"requestReply: %@", [[requestReply class] description]);
                //if the result has the class type __NSCFConstantString then
                    the item failed to upload
                if ([[[requestReply class] description]
                    isEqualToString:@"__NSCFConstantString"]) {
                    //alert for failing to upload
                    UIAlertController *alert = [UIAlertController
                        alertControllerWithTitle:@"No Connection\n"
                        message:@"Could not donate the item. Please check your
                        internet connection and try again." preferredStyle:
                        UIAlertControllerStyleAlert];
                    UIAlertAction* defaultAction = [UIAlertAction
                        actionWithTitle:@"OK" style:UIAlertActionStyleDefault
                        handler:^(UIAlertAction * action) {}];
                    [alert addAction:defaultAction];
                    [self presentViewController:alert animated:YES completion:
                        nil];
                }
                else {
                    [self performSegueWithIdentifier:@"showDonationThankYou"
                        sender:(self)];
                }
            });
        }] resume];


    }
}

// Control methods for each text/image view

//makes keyboard disappear after finished editing
-(BOOL) textFieldShouldReturn:(UITextField *)textField{

    [textField resignFirstResponder];
    return YES;
}

//makes keyboard disappear after finished editing
```

```
- (BOOL)textView:(UITextView *)textView shouldChangeTextInRange:(NSRange)range
    replacementText:(NSString *)text {

    if([text isEqualToString:@"\n"]) {

        [textView resignFirstResponder];
        return NO;
    }

    return YES;
}


//for the text fields that are directly edited (name and price)
-(void)textFieldDidEndEditing:(UITextField *)textField {
    //if editing the name text field it checks to see if the width of name (as
        it would appear on UI) is too wide, if not it adds it, if so it
        provides alert for shorter name
    if ([textField isEqual: nameTextField]) {
        NSString *tmpName = nameTextField.text;
        tmpLabel.text = tmpName;
        tmpLabel.hidden = YES;
        [tmpLabel sizeToFit];
        NSLog(@"%f", tmpLabel.frame.size.width);
        if (tmpLabel.frame.size.width < 180) {
            name = tmpName;
        }
        else {
            nameTextField.text = @"";
            name = @"";
            nameTextField.placeholder = @"Name of Item";
            UIAlertController *alert = [UIAlertController
                alertControllerWithTitle:@"Invalid Name\n" message:@"Please use
                a shorter name" preferredStyle:UIAlertControllerStyleAlert];
            UIAlertAction* defaultAction = [UIAlertAction actionWithTitle:@"OK"
                style:UIAlertActionStyleDefault handler:^(UIAlertAction *
                action) {}];
            [alert addAction:defaultAction];
            [self presentViewController:alert animated:YES completion:nil];
        }
    }
    //if user was editting price it updates the price field and formats the
        input to be displayed in the text field
    if ([textField isEqual: priceTextField]) {
        NSString *cents;
        NSString *dollars;
        int priceCents;
        int priceDollars;
        long finalPriceInCents;
        //formatting input
        for (int i = 0; i < priceTextField.text.length; i++) {
            if ([[[priceTextField.text substringFromIndex:i] substringToIndex:1
                ] isEqualToString:@"."]) {
                cents = [priceTextField.text substringFromIndex:i];
                dollars = [priceTextField.text substringToIndex:i];
                break;
```

```
            }
        }
        if (dollars == nil) {
            dollars = priceTextField.text;
            if ([dollars isEqualToString:@""]) {
                dollars = @"0";
            }
        }
        if (cents == nil) {
            cents = @".00";
        }
        NSCharacterSet *mySet = [NSCharacterSet
            characterSetWithCharactersInString:@"."];
        cents = [cents stringByTrimmingCharactersInSet:mySet];
        if (cents.length == 1) {
            [cents stringByAppendingString:@"0"];
        }
        priceCents = (int)[cents integerValue];
        dollars = [dollars stringByTrimmingCharactersInSet:mySet];
        priceDollars = (int)[dollars integerValue];
        NSLog(@"Price: %@%i", dollars, priceDollars);
        //saving input
        finalPriceInCents = (priceDollars * 100) + priceCents;
        priceInCents = (NSInteger *)finalPriceInCents;
        NSLog(@"%zd", priceInCents);
        //outputting formatted input
        priceTextField.text = [NSString stringWithFormat:@"$%@.%@", dollars,
            cents];
        if ([priceTextField.text isEqualToString:@"$0.00"]) {
            priceTextField.text = @"";
            priceInCents = nil;
            priceTextField.placeholder = @"$0.00";
        }
    }
    //closes keyboard
    [textField resignFirstResponder];
    [self.view endEditing:YES];
}

//closes the picker view
-(void)clearPickerView {
    [conditionTextField resignFirstResponder];
}

//for condition (not directly edited, uses a picker instead)
-(BOOL)textFieldShouldBeginEditing:(UITextField *)textField {
    if (textField.tag == 2 || [textField isEqual:conditionTextField]) {
        _conditionPicker = [[UIPickerView alloc] init];
        _conditionPicker.dataSource = self;
        _conditionPicker.delegate = self;
        _conditionPicker.showsSelectionIndicator = YES;
        conditionTextField.inputView = _conditionPicker;
        textField.inputView = _conditionPicker;
        UIButton *doneButton = [[UIButton alloc] init];
        [doneButton setTitle:@"Done" forState:UIControlStateNormal];
        textField.inputAccessoryView = _toolBar;
```

```
    }
    if ([textField isEqual: priceTextField]) {
        textField.inputAccessoryView = _toolBar;
    }
    return YES;
}

//number of columns - only one for condition
-(NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView
{
    return 1;
}

// The number of rows of data
- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:
    (NSInteger)component
{
    return conditionOptionsDonate.count;
}

// The data to return for the row and component (column) that's being passed in
- (NSString*)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row
    forComponent:(NSInteger)component
{
    return conditionOptionsDonate[row];
}



-(void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
    inComponent:(NSInteger)component {
    //if user selected a row with data in it, it saves that data as the
        selected condition
    if (row != 0) {
        conditionTextField.text = conditionOptionsDonate[row];
        long rowTmp = row;
        conditionInt = (NSInteger *)rowTmp;
        condition = conditionOptionsDonate[row];
    }
    //if user selects --select-- (meant to be an empty option) it saves the
        condition selected as empty
    else {
        conditionTextField.text = @"";
        conditionTextField.placeholder = @"Condition";
        condition = @"";
    }
}

//updates the description with the user input
-(void)textViewDidEndEditing:(UITextView *)textView {
    //creates a copy of the string with whitespace removed to check if visible
        text has been entered so they will not lose the text box
    NSString *tmp = [descriptionTextView.text stringByTrimmingCharactersInSet:
        [NSCharacterSet whitespaceAndNewlineCharacterSet]];
    if ([tmp isEqual: @""]) {
        descriptionTextView.text = [NSString stringWithFormat:@"Insert
```

```
            description here"];
    }
    else {
        description = descriptionTextView.text;
    }
    //closes keyboard
    [textView resignFirstResponder];
}

//if description has placeholder text in there it empties it (no built in
    support for placeholder text for text views)
-(void)textViewDidBeginEditing:(UITextView *)textView {
    if ([textView.text isEqualToString:@"Insert description here"]) {
        textView.text = [NSString stringWithFormat:@""];
    }
}

//dismisses keyboards when large button in background clicked - enables users
    to navigate away from editting in an intelligent way
-(IBAction)dismissKeyBoards:(id)sender {
    [self.view endEditing:YES];
}

- (void)viewDidLoad {
    //creates a 'done' toolbar to be used for all non-keyboard inputs
    _toolBar = [[UIToolbar alloc] init];
    _toolBar.barStyle = UIBarStyleDefault;
    _toolBar.translucent = true;
    [_toolBar sizeToFit];
    //creates the done button for the toolbar
    UIBarButtonItem *finished = [[UIBarButtonItem alloc] initWithTitle:@"Done"
        style:UIBarButtonItemStylePlain target:self action:@selector
        (clearPickerView)];
    //adds the done button
    [_toolBar setItems:@[finished] animated: false];
    _toolBar.userInteractionEnabled = true;

    [super viewDidLoad];
    // Do any additional setup after loading the view.
    //initializes the variables to be empty (prevents needing to account for
        both empty and null when verifying user input)
    name = @"";
    condition = @"";
    priceInCents = 0;
    description = @"";
    //creats the conditions array
    conditionOptionsDonate = [[NSUserDefaults standardUserDefaults]
        objectForKey:@"conditions"];

    //sets the delegate for the text fields
    nameTextField.delegate = self;
    conditionTextField.delegate = self;
    descriptionTextView.delegate = self;
    priceTextField.delegate = self;
    tmpLabel.hidden = YES;
    imageUploaded = false;
```

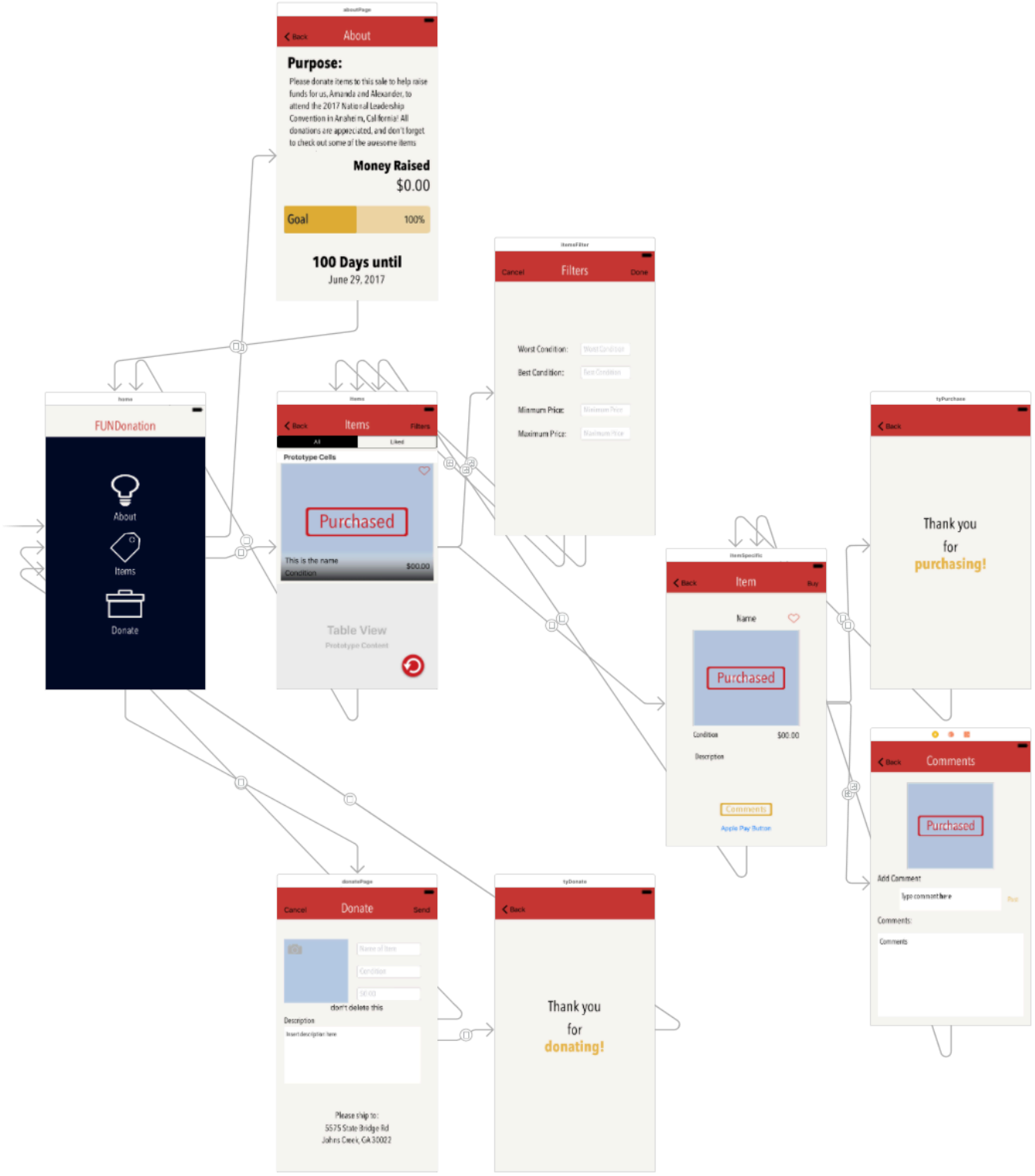
```
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}


@end
```

```
//
//  About.h
//  Garage Sale
//
//  Created by Alexander Hammond on 1/16/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import <UIKit/UIKit.h>



@interface About : UIViewController {
    //interface outlets
    IBOutlet UILabel *amountRaisedText;
    IBOutlet UIImageView *progressBar;
    IBOutlet UIImageView *totalBar;
    IBOutlet UILabel *daysUntilNLC;
    IBOutlet UILabel *percentRaised;
    IBOutlet NSLayoutConstraint *progressBarWidth;


}

//class properties
@property int amountRaisedInCents;
@property int goalInCents;
@property int daysRemaining;

@end
```

```
//
//  About.m
//  Garage Sale
//
//  Created by Alexander Hammond on 1/16/17.
//  Copyright © 2017 TripleA. All rights reserved.
//

#import "About.h"

@interface About () <NSURLSessionDelegate>

@end

@implementation About

/* Overall Methods
 updateAmountRaised
 */

-(void)updateAmountRaised {
    //loads the most recent value for the amount raise
    [self loadAmountRaised];
    //updates the progress bar
    [self setProgressBarWidth];
    //updates the text
    [self setAmountRaisedText];
    [self setPercentRaised];
}

//updates the data then the view
-(void)updateDaysRemaining {
    [self loadDaysRemaining];
    [self setDaysRemaining];
}


//sets the width of the progress bar based on the amount raised and goal
-(void)setProgressBarWidth {
        int barWidth = (_amountRaisedInCents * totalBar.frame.size.width)/
            _goalInCents;
        NSLog(@"Bar Width: %i", barWidth);
    if (barWidth > totalBar.frame.size.width) {
        progressBarWidth.constant = totalBar.frame.size.width;
    }
    else {
        progressBarWidth.constant = barWidth;
    }
        progressBar.hidden = NO;

}

//sets percent raised (class variable to reduce processing time and code
    repetition)
-(void)setPercentRaised {
    int percentRaisedInt = (100*_amountRaisedInCents)/_goalInCents;
```

```
    percentRaised.text = [NSString stringWithFormat:@"%i%%", percentRaisedInt];
}


//updates the UI with the amount raised
-(void)setAmountRaisedText {
    int centsRaised = _amountRaisedInCents % 100;
    int centsRaisedTens = centsRaised / 10;
    int centsRaisedOnes = centsRaised % 10;
    int dollarsRaised = (_amountRaisedInCents - centsRaised)/100;
    amountRaisedText.text = [NSString stringWithFormat:@"$%i.%i%i",
        dollarsRaised, centsRaisedTens, centsRaisedOnes];
    amountRaisedText.hidden = FALSE;
}

//udates the UI with the days remaining
-(void)setDaysRemaining {
    daysUntilNLC.text = [NSString stringWithFormat:@"%i Days Until",
        _daysRemaining];
    daysUntilNLC.hidden = FALSE;
}



//loads the amount that has been raised from the server, if failure to connect
    it leaves it as 0
-(void)loadAmountRaised {
    //sets to default of 0
    _amountRaisedInCents = 0;
    //creates url session to load amount raised
    NSString *jsonUrlString = [NSString stringWithFormat:@"https://murmuring-
        everglades-79720.herokuapp.com/total.json"];
    NSURL *url = [NSURL URLWithString:jsonUrlString];
    NSURLSessionConfiguration* config = [NSURLSessionConfiguration
        defaultSessionConfiguration];
    NSURLSession *session = [NSURLSession sessionWithConfiguration:config
        delegate:self delegateQueue:[NSOperationQueue mainQueue]];
    NSURLSessionDataTask *dataTask = [session dataTaskWithURL:url];
    [dataTask resume];
}

//if data comes back from url session (only session in here is to load amount
    raised) then it updates the amount raised (the view and the data)
- (void)URLSession:(NSURLSession *)session
          dataTask:(NSURLSessionDataTask *)dataTask
    didReceiveData:(NSData *)data {
    NSError *error;
    NSString *totalRaised = [NSJSONSerialization JSONObjectWithData:data
        options:NSJSONReadingAllowFragments error:&error];
    _amountRaisedInCents = [totalRaised intValue];
    [self setProgressBarWidth];
    //updates the text
    [self setAmountRaisedText];
    [self setPercentRaised];
    [session invalidateAndCancel];
```

```
}

//calculates the number of days remaining and updates the variable
    _daysRemaining
-(void)loadDaysRemaining {
    //creates NSDate for today
    NSDate *today = [NSDate date];
    //Creates Date Components for start of NLC (end of fundraiser)
    NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
    [dateFormatter setDateFormat:@"YYYY-MM-dd"];
    NSDate *dateNLC = [dateFormatter dateFromString:@"2017-06-29"];

    //next couple declarations from Apple's reccomendation on how to compare
        dates
    NSCalendar *tempCalendar = [[NSCalendar alloc] initWithCalendarIdentifier:
        NSCalendarIdentifierGregorian];

    NSDateComponents *components = [tempCalendar components:NSCalendarUnitDay
        fromDate:today  toDate:dateNLC options:0];

    NSInteger days = [components day];


    //casts Integer (which stores a long) to int
    _daysRemaining = (int)days;

}

//startup code - here is where goal is set and the updates for the UI are
    called
- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view.

    //hides the text and progress bar until they are loaded (they are revealed
        within the update methods through the set methods)
    //              10000.00
    //              10000 00
    //              1000000
    _goalInCents = 1000000;
    _amountRaisedInCents = 0;
    amountRaisedText.hidden = TRUE;
    daysUntilNLC.hidden = TRUE;
    progressBar.hidden = TRUE;
    [self updateAmountRaised];
    [self updateDaysRemaining];
}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```


About
Purpose:
Please donate items to this sale to help raise funds for us, Amanda and Alexander, to attend the 2017 National Leadership Convention in Anaheim, California! All donations are appreciated, and don't forget to check out some of the awesome items
Money Raised
$0.00
Goal
100%
100 Days until
June 29, 2017
FUNDonation
About
Items
Donate
Items
Filters
All
Liked
Prototype Cells
Purchased
This is the name
$00.00
Condition
Table View
Prototype Content
Filters
Cancel
Done
Worst Condition:
Best Condition:
Minimum Price:
Maximum Price:
Item
Buy
Name
Purchased
Condition
$00.00
Description
Comments
Apple Pay Button
Thank you
for
purchasing!
Comments
Purchased
Add Comment
Type comment here
Post
Comments:
Comments
Donate
Cancel
Send
don't delete this
Description
Please ship to:
5575 State Bridge Rd
Johns Creek, GA 30022
Thank you
for
donating!

FUNDonation

About

Items

Donate

Purpose:

Please donate items to this sale to help raise funds for us, Amanda and Alexander, to attend the 2017 National Leadership Convention in Anaheim, California! All donations are appreciated, and don't forget to check out some of the awesome items

Money Raised

$1915.94

Goal 19%

147 Days Until

June 29, 2017

Car

Great Condition

$9865.99

Description

A 2006 Mini Cooper convertible with a dark interior. It makes for a great day to day car with its small size and agility.

Sprint LTE 1:45 PM 84%

Back

Items

Filters

All | Liked

Car

Great Condition $9865.99

A 2006 Mini Cooper convertible with a dark interior. It makes for a great day to day car with its small size and agility.

Comments

Worst Condition: Worst Condition

Best Condition: Best Condition

Minmum Price: Minimum Price

Maximum Price: Maximum Price

Add Comment

Type comment here

Post

Comments:

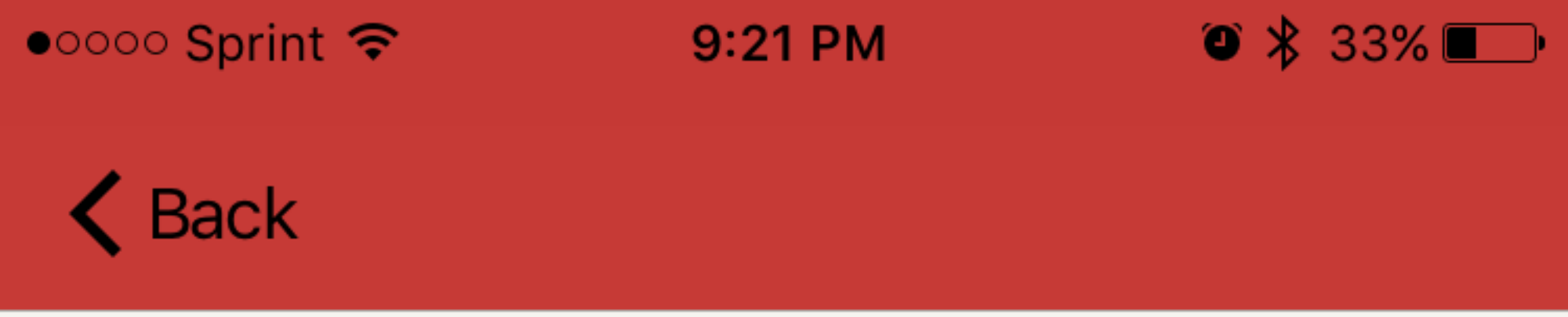
It's very blue🚙

:

The car is super kooooooool and awesome

mandyGang:

Can we negotiate for a lower price💸



Thank you

for

donating!

Thank you

for

purchasing!