Deal Finder

MBA 614 Spreadsheet Automation

Final Project

Ryan Cluff

April, 2014

Introduction

Three years ago, my good friend Aaron showed me a website that would forever change the way I shopped online: dealnews.com. This website is dedicated to scowering the internet in search of the best deals and lists them according to "hotness" (rating system for how good the deal is, from 1-lowest to 5-highest). Here, you'll find deals ranging from 80% off clearance at Kohls.com to a free Panda Express entrée, and everything in between.

The purpose of project deal-finder was to go one more step from what dealnews.com does. For years, it became part of my morning routine to go to dealnews.com and scan the new deals for the day; however, as of late, I've noticed my days have been so busy that I often don't have time for my online deal fix. Project deal-finder does what I now cannot – search for the best deals that I care about and publish them in a way that I can quickly look at them. Essentially, deal-finder scans dealnews.com's 300 daily deals and filters them according to keywords that I set. If deal-finder finds a keyword match on a deal that has a "Hotness" above my threshold, it sends me a message – either via text or email. Now I can focus on my studies while deal-finder is on the lookout for the best online deals that matter to me.

Implementation Documentation

Sending Messages

The first functionality that I needed to set up was the ability to send an email. Fortunately, we had a class lecture that showed how to set up an email function, which is what I followed. For privacy reasons, I created a new gmail account who's credentials I use to send the emails. Those credentials are stored in cells B2 and B3 on the main "data" sheet. The purpose for that was to allow the user to be able to use whichever email account they'd like.

The function for sending messages is named "sendMessage" and it requires 3 arguments: 1) the address of the person receiving the message, 2) the message itself, and 3) the subject of the message. The function will return a true or false whether or not it was able to successfully send off the message.

One more step was needed in order to send text messages. Text messages can be sent via email if you know the correct format that the phone number needs. I created a new sheet and named it "phone_carriers". This sheet has only 2 columns: the names of all the phone carriers and their respective MMS mail address with a "<number>" inserted where the phone should be. I later implemented a check in the main sub routine that either uses the phone carrier's email format or the user's email address as the address for sending the message. We'll go over that later.

Parsing the HTML

The next functionality I set up was the system to load and then process the HTML source code for dealnews.com. Initially, I set up a loop that would load the three separate pages (each holding 100 deals), but after looking at the HTML that VBA loaded, I realized that for whatever reason, all 280+ deals were being loaded on the first page, so I removed the looping function.

All the settings that control what the program looks for is saved in the table on the "data" worksheet (See **Appendix A**). This table lists the user's name, phone number and carrier, email address, and information specific for the deal search such as keywords (delimitated by a ";") and the level of hotness.

The function "findDeals" only needs two inputs: the string of keywords and the lowest acceptable level hotness. The HTML is already loaded on a global agent variable. This originally wasn't the case. At first, I was loading the HTML every time the findDeals function was called, however, it takes a good 5-10 seconds to load the HTML and I quickly realized that trying to load (the same) HTML for each user would take forever. I ended up declaring the agent object "a" globally and loading the HTML in a separate function called "loadPage". Now back to the findDeals function.

The findDeals function followed this general process:

1. Search from the current position in the HTML for the phrase "<div class=""article article-". That phrase identifies that start of a new deal .
2. From there, search for the phrase "<div class=""hd"">". This marks the start of the header of the deal. The header contains the main blip describing the deal as well as another that shows the deal's hotness.
3. From this position, it saves all the text until it finds "<div class=""hotness info""" and then again until it finds " ". All this text encompasses the title and hotness of the deal.
4. It searches this text for "hotness: " which is the alt text for the <a> tag that the hotness level is stored in. the very next character is a 1-5 which corresponds to that deal's hotness. If the deal is hot enough, the function continues, otherwise it starts back over at step 1. and searches for a new deal.
5. With the hotness high enough, it now splits the keyword string with the ";" delimiter and for each keyword, searches the title.
6. If there's a match between a keyword and the title, then the function saves the description of the deal by searching the HTML from the current position for the text "<div class=""article-body"">" which marks the beginning of the description of the deal. And then saving all the text until it finds a " " which marks the end of the description.
7. The deal's description is then saved to an array called "deals" and the integer "numDeals" is incremented. Now we go back to step 1 and repeat until there are no more deals.
8. Once there are no more deals, it dims the function's variable (findDeals – which is a variant) to have the same number of elements as that size of "numDeals". Then we set findDeals to deals and that is what is returned from the function.

Creating/Editing Search Records

Now it was time to create a simple way to add, edit, and remove search records. I started by adding two buttons on the worksheet with the table.

1. The first button was labeled "edit search". The macro that this was connected to essentially loaded the form with the data of the row of the currently-selected cell. If the row of the

currently selected cell was the first row (where the headers are) or a row without data, then the last record would be selected.

2. The second button was labeled “add search”. The macro that this was connected to moved the currently selected cell to the end of the list and then loaded the form. Upon saving the form, the data would be added to that row.

We will now discuss each element of the form.

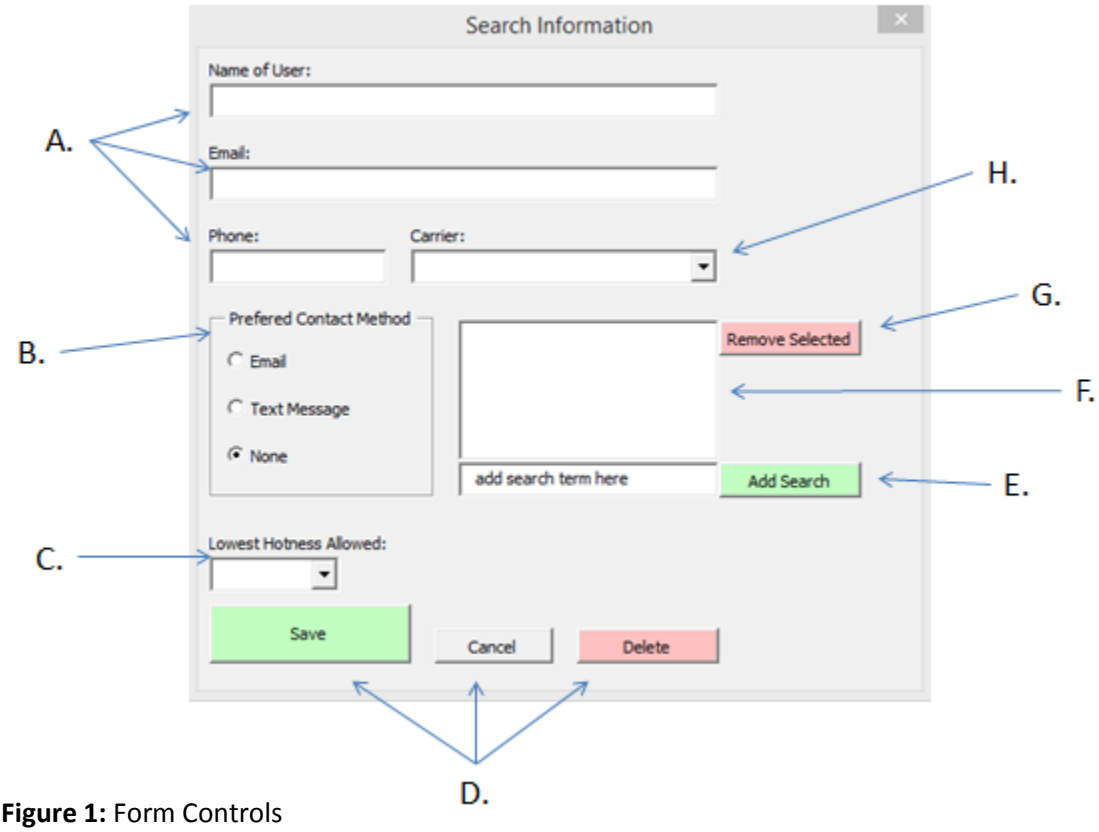


Figure 1: Form Controls

A. The **Name**, **Email**, and **Phone** fields were simple text controls. They were linked directly with their corresponding columns in the “data” table

B. If the **preferred contact method** was set to Email, the form’s saving sub would check to make sure that a valid email was also provided (valid meaning there was at least an “@” in it). Similarly, if “Text Message” was chosen then both a valid phone number (either 7 or 10 digits long) and a phone carrier must be provided.

C. This is just a drop-down menu from 1-5 for the possible hotness levels. Any deal equal to or higher than this value will be accepted.

D. These buttons control the search record and they act just as you’d expect. “Save” saves the form to the current row. “Cancel” closes the form without saving. “Delete” deletes the cells of the current record.

E. This button adds the text in the textbox just left of it to the “keywords” list. Any word in the keywords list (regardless of whether they are selected or not) will be added to the record. I also added a prompting default value in this textbox that says “add search term here”. As soon as the user selects this textbox, I run a sub that clears this value (similar to how some online fields work)
F. This is the **keyword list.** As I mentioned earlier, all the list items here will be saved as keywords to the record.
G. This is the **“Delete Keywords”** button. When the user clicks this button, all the keywords selected in the keyword list will be removed from the list. This is how the user can remove unwanted keywords from their search list.
H. This is the **Carrier dropdown list**. The elements of this list are populated by the cells in the “phone_carrier” worksheet.

Bringing it all Together

Now that we have all the pieces, I needed a main sub that brought them all together. I added a button on the main “data” worksheet – a big button – named “run”. This button runs the “scanSearches” sub. The “scanSearches” sub is what calls the function to load the website, loops through each search record, calls the function “findDeals” function to search for deals, and if deals are found, creates the message and calls the “sendMessage” function to send the message either via email or text. It performs this operation using the following steps:

1. Loads the page by calling the “loadPage” function and passing it the dealnews website url
2. Dims the “users” array. The length of the first dimension of this array will be as long as there are search records in the “data” table. The length of the second dimension is initially set to 300, because that is the maximum number of deals that they could potentially have.
3. Scans through each search record and sets the first “row” of the “users’ array to equal the name of the the user in the search record. After this, all subsequent rows of the “user” array (rows 1-300) are dedicated to holding the descriptions of the deals.
4. Runs the “findDeals” function and sets another dynamic array “deals” to what “findDeals” returns.
5. Loops through each item in “deals” and saves it over to “users”
6. Repeats steps 3-5 until all the search records have been ran
7. Loops through each deal for each user and if there is a deal, it concatenates it to the end of the “message” string.
8. If the “message” string is not empty, then it checks what method of deliver the user wants (text or email) and it calls the “sendMessage” function passing It the correct address (either the email address listed in the search record, or the address corresponding to the phone carrier with the “<number>” part replaced with the user’s actual phone number)

Learning and Difficulties Encountered

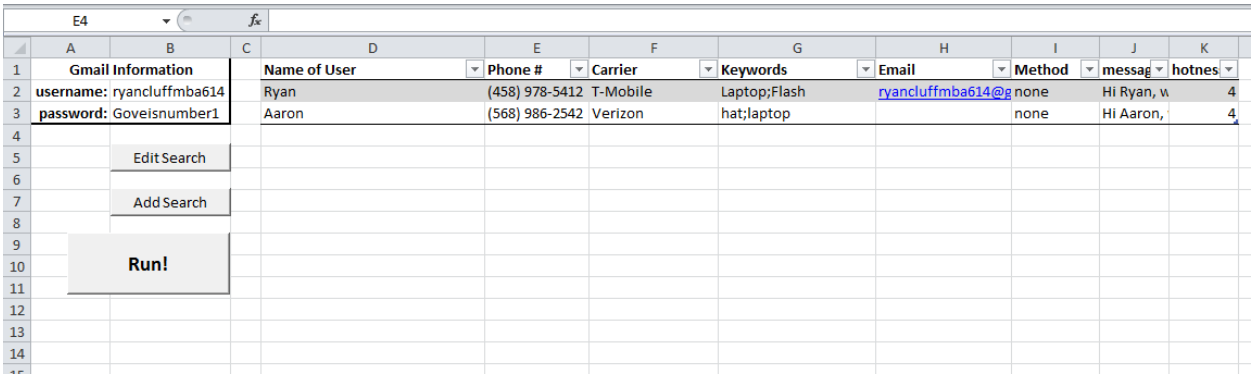
1. I once tested the search using the keyword “hat”. This search returned tons of deals, all of which had nothing to do with hats or clothing at all. I realized that there were many words , like “that”, that contained the substring “hat” in them. This meant that the search function could potentially produce a false positive thinking that there’s a match when really there shouldn’t be.
   Solution: change the search function to look for the keyword surrounded by spaces (eg. “ hat “ instead of “hat”). This solution works, however, it also ensures that some desirable substring matches (such as matching “dresses” with “dress”) would not work. Future versions of this program could potentially fix this by also searching for common suffixes of the keyword (such as “-s”, “-ed”, and “-ing”)
2. There were many looping issues that I needed to debug. Transitioning between looping through arrays with a base 0 and other lists (strings and cells) with a base 1 is always a challenge.
3. As I mentioned earlier, I was originally loading a new HTML every time I called the search function. This resulted in the program taking a dreadfully long time to run.
   Solution: Separate the loadPage function and only call it once at the beginning of the program. Also, I needed to be sure to reset the agent variable back to 1 before looping to the next search record.
4. I few test searches resulted in no matches, yet I would still get a text or email. This was annoying.
   Solution: I added a check at the end of the loading the message that would only call the “sendMessage” function if there was at least one deal found for that search record
5. I thought it would be a little confusing the way I set up the Keyword list in the form.
   Solution: I added the “add search here” default text. In order for it to work correctly, I also had to add an extra check in the the “add search” button so that it wouldn’t add a list item if the textbox contained “” or “add search here”. I also had to add a new event sub for when the textbox was selected that would clear the textbox’s value if its value was “add search here”. And finally, once a term was added to the keyword list, I had to refil the textbox’s value to “add search here”.

Assistance

I had no assistance with this project.

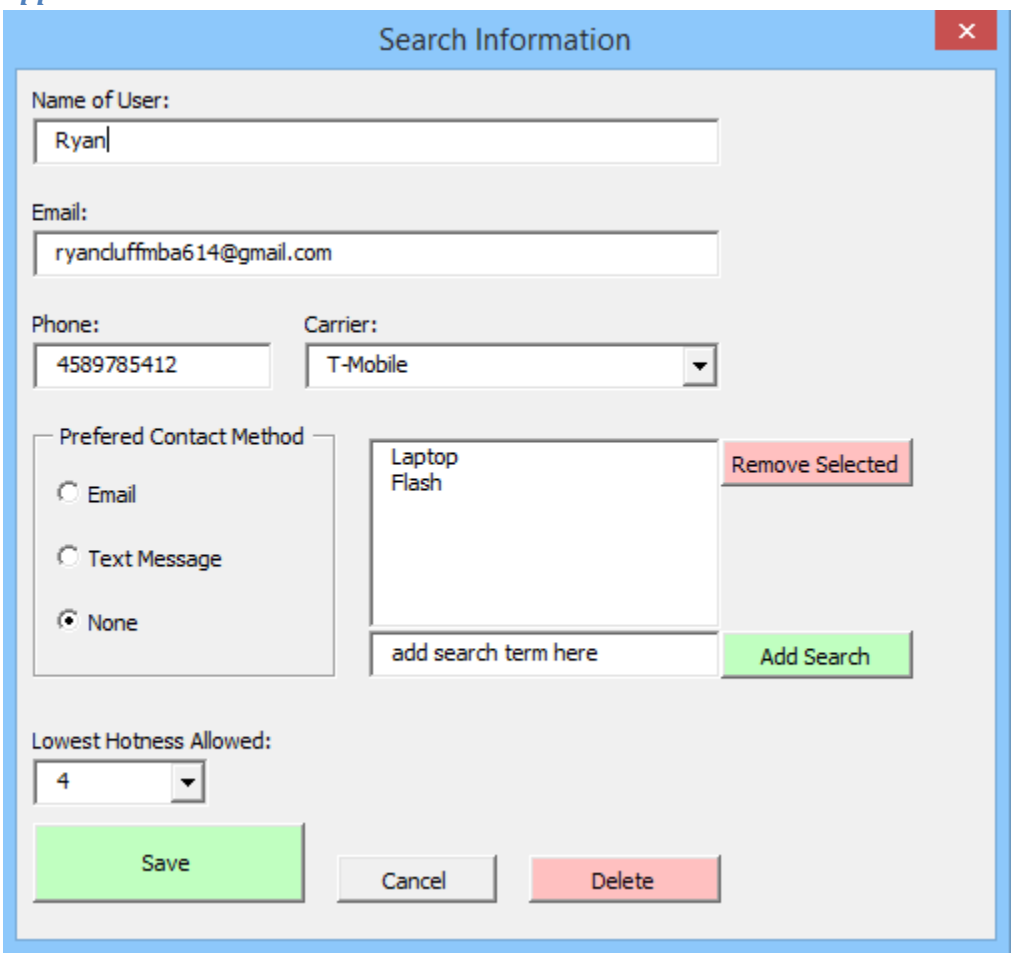
Appendix

Appendix A



Screenshot of the worksheet “data”. This shows the email SMTP settings (username and password), three control buttons (edit search, add search, and run), and the table containing the search recrods.

Appendix B



Sample form with populated data

Appendix C

Hi Ryan, we've found the following deals for you

Manufacturer Certified via eBay offers the refurbished, 4-lb. Toshiba Satellite U845W-S410 Intel Core i5 1.7GHz 21:9 14.4" Ultrabook, model no. PSU5RU-00Q003, for **$399.99** with **free shipping**. That's $25 under our fall mention of a refurb and the lowest total price we've seen for this laptop. (It's a current low for a refurb by $360.) Features include an Intel Core i5-3317U 1.7GHz Ivy Bridge dual-core processor, 14" 1792x768 21:9 aspect ratio LED-backlit ultrawide LCD, 6GB RAM, 500GB hard drive and 32GB SSD, 802.11n wireless, Bluetooth 4.0, webcam, media card reader, USB 3.0, HDMI, 4-cell battery, and Windows 7 Home Premium 64-bit. Note that no optical drive is included.

No warranty information is provided.

Microsoft Store offers the 4.9-lb. Acer Aspire Intel Haswell Core i5 1.6GHz 15.6" Touchscreen Laptop, model no. V5-573P-6896, for **$499** with **free shipping**. That's $100 under our mention from last week and tied with a December deal as the best price we've seen for a new 15.6" 1080p touchscreen laptop with this CPU. Features include an Intel Haswell Core i5-4200U 1.6GHz quad-core processor, 15.6" 1920x1080 (1080p) LED-backlit touchscreen LCD, 4GB RAM, 500GB hard drive, 802.11n wireless, Bluetooth, HDMI, USB 3.0, 4-cell battery, and Windows 8.

The Apple Store offers discounts on its selection of factory-refurbished Apple MacBook Pro laptops, with prices starting at **$999**. With **free shipping** on each, that ties last week's mention and is the lowest total price we could find for refurbished systems with a 1-year Apple warranty (the same as new units). Notables:

- Refurb MacBook Pro MD101LL/A i5 Dual 2.5GHz 13.3" Laptop for **$999** ($200 off)
- refurb MacBook Pro MD212LL/A i5 Dual 2.5GHz 13.3" Retina Laptop for **$1,059** ($440 off)
- refurb MacBook Pro MD103LL/A i7 Quad 2.3GHz 15.4" Laptop for **$1,449** ($350 off)
- refurb MacBook Pro MC976LL/A i7 Quad 2.6GHz 15" Retina Laptop for **$1,899** ($600 off)
- see all refurbished MacBook Pro laptops at The Apple Store

Sample message of deals sent to email