**Savvy Shopper – Android / Web (Backend)**

**Requirements Analysis and System Workflow Definitions**

**Revision:** D.0.8

**Date:** 19-Apr-2016

**Index**

[1. Introduction 4](#_Toc444696306)

[1.1. Purpose 4](#_Toc444696307)

[1.2. Referenced Documents 4](#_Toc444696308)

[1.3. Revision History 4](#_Toc444696309)

[1.4. Conventions 4](#_Toc444696310)

[2. System Overview 4](#_Toc444696311)

[2.1. Synopsis 4](#_Toc444696312)

[2.2. Roles 5](#_Toc444696313)

[2.3. General Constraints 5](#_Toc444696314)

[2.4. General Workflows 5](#_Toc444696315)

[3. Workflows 5](#_Toc444696316)

[3.1. Splash screen 5](#_Toc444696317)

[3.2. Sign up/Sign in 5](#_Toc444696318)

[3.3. Email verification 7](#_Toc444696319)

[3.4. Rate the app 8](#_Toc444696320)

[3.5. Navigation Menu 8](#_Toc444696321)

[3.6. Home Screen 8](#_Toc444696322)

[3.7. Clicking on Buy button from anywhere in the app 9](#_Toc444696323)

[3.8. Loading top brand names 10](#_Toc444696324)

[3.9. Create New Search 14](#_Toc444696325)

[3.10. Search Engine for Amazon 18](#_Toc444696326)

[3.11. Search Engine for FlipKart 22](#_Toc444696327)

[3.12. SnapDeal – Gather data 25](#_Toc444696328)

[3.13. Search Engine for SnapDeal 31](#_Toc444696329)

[3.14. My Searches 35](#_Toc444696330)

[3.15. Search Results 36](#_Toc444696331)

[3.16. View More Results – Tracking. 37](#_Toc444696332)

[3.17. My Savvy Deals 38](#_Toc444696333)

[3.18. Send Feedback 38](#_Toc444696334)

[3.19. If the user account is deleted from server 39](#_Toc444696335)

[3.20. Cron for Email verification notification 39](#_Toc444696336)

[3.21. Cron to send search notifications to user 40](#_Toc444696337)

[3.22. Cron to send price changed notifications 40](#_Toc444696338)

[3.23. Cron for Email verification not completed – block account 41](#_Toc444696339)

[3.24. Cron to fetch the Deals of the Day 41](#_Toc444696340)

[3.25. Cron to delete all searched data for inactive searches 42](#_Toc444696341)

[3.26. Cron to delete all searched data for a deleted search request 42](#_Toc444696342)

[4. Phase II Wishlist 43](#_Toc444696343)

# Introduction

## Purpose

This document provides the following details:

* System Workflow Definitions of the various modules as per the project scope.

## Referenced Documents

|  |  |  |
| --- | --- | --- |
| Ref. No. | Document Title | Author |
|  |  |  |

## Revision History

|  |  |  |
| --- | --- | --- |
| **Release Number** | **Date** | **Description** |
| D0.1 | 05-Feb-2016 | Initial Draft |
| D0.2 | 10-Feb-2016 | Revisions based on the 9-Feb call |
| D0.3 | 15-Feb-2016 | Revisions based on the 11-Feb call |
| D0.4 | 18-Feb-2016 | Defined the technical architecture of the search algorithms. |
| D0.5 | 22-Feb-2016 | Revised the document as per Satyen’s comments and further search criteria |
| D0.5 | 2-Mar-2016 | Revised the document as per Balaji’s comments during the screen design walk through.  Revised the document based on some more findings on the API analysis. |
| D0.6 | 17-Mar-2016 | Revised the document based on the FlipKart findings for colour and size variations. |
| D0.7 | 17-Mar-2016 | Revised the document based on the search engine fine tuning. |
| D0.8 | 19-Apr-2016 | Phrase search enhancement of SnapDeal |

## Conventions

|  |  |
| --- | --- |
| Red | Queries |
| Green | Assumptions |
| Blue | Notes |
| **Yellow** | Revisions to the document with the new release. |

# System Overview

## Synopsis

* This is a native **Android application** to find the best product matches for a user by scanning 3 major online Indian shopping portals – Amazon India, FlipKart and SnapDeal.

## Roles

* Customer (Android app)
* Admin end (Out of scope for Phase I)

## General Constraints

* Server Environment: Linux, Apache Web Server, PHP, MySQL database.
* Protocol used: HTTP.
* Mobile users need to be online to access the app features.

## General Workflows

* Date format followed will be mm/dd/yyyy.
* Show appropriate progress indicators while making any web service or internet calls.
* Image optimization and lazy loading of images on users’ mobiles.

# Workflows

## Splash screen

|  |  |
| --- | --- |
| Req. | * Splash screen that will appear for few seconds during the launch of the application. This will be static image. * We shall propose a graphic for the same when we start on the screen designs. |

## Sign up/Sign in

|  |  |
| --- | --- |
| Req. | * There will be only 1 method to sign up:   + Sign in via facebook. This will require an email verification process. |
| Sign up form | * Option to “Sign in with Facebook” will appear by default if the user has not yet signed up or is not yet logged-in to the app. * If the user has already signed in, then the logged-in state will be maintained, till the user explicitly logs out of the app. * Tapping on “Sign in with Facebook” will prompt the user to enter his/her Facebook credentials. * On successful Facebook authentication, we receive an authentication token which is unique for a user. A user account will be created in the SavvyShopper database with this unique authentication token and isEmailVerified = false. If this is the first time that the user has signed in using Facebook, the app will take the user to a screen to enter his email address, which is a mandatory step and the user won’t be able to access the app unless he completes this step. If this is a Repeated sign in using Facebook   + Authenticate using Facebook authentication token.   + If successful, update the last logged in date/time on the server.   + Check for isAccountBlocked = true. If yes, display “Sorry! Access denied.”   + Else, login the user into the app and take him to the home screen.  |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Field** | **Control/** **Label** | **Mandatory?** | **Validations** | **Notes** | | Just one more step - please confirm your email address. If you're ever locked out of your account, this will help us get you back in. | | | | | | Email | Text box | Y | Maxlen = 100 characters  Email format validation | Pre-loaded if we are able to get the email from Facebook, else blank.  The user can change the email address or enter a new email address.  This email address should be a unique email address throughout the system. If not, prompt the user so. | | **Confirm Email** | Button |  |  | We shall record this email address with the user account in the SavvyShopper database so that we don’t prompt the user to enter his email again on subsequent sign ins.  Redirect the user to the Home screen of the app i.e. auto-login him into the app. | |
| “Update email” web service call – detailed workflow | * If the email address is not unique then inform the user that the email should be unique. * If the email address is unique, then   + Update the record in the users table with email and set IsEmailStepCompleted = 1   + Generate a unique key that will be bound to the verification link in the email.   + Send email to the user with a verification link to verify his email address.   + On successful email update on the server end, the app will make another call to the GCM for registering the device to receive push notifications.   + Verifying the email account is not a mandatory process and the user can login to the app and continue to access it.   + Display a message to the user “Thank you! You are now logged and have full access to this app. We have sent you an email to confirm your email address. Please ensure you confirm your email address at the earliest for continued access to this app.” |
| My Searches on Login | * The data for “my searches” will be downloaded on the mobile device immediately on login. * Only active searches and up to 15 deactivated searches (which are not yet deleted) will be downloaded on the device. |

## Email verification

|  |  |
| --- | --- |
| Req. | * Verifying the email account is not a mandatory process and the user can login to the app and continue to access it. But the app will keep reminding the user to verify his email. |
| Email | Subject: <name>, please confirm your email address  Welcome to Savvy Shopper, <name>!  Thank you for registering with us. Just one more step - please confirm your email address. If you're ever locked out of your account, this will help us get you back in.  Confirm [email](mailto:email@domain.com) address <button> |
| Verification process | * If the user account is not found show a message “Sorry! Your account has been deleted as you did not verify your email address with us. You’ll need to sign up again to gain an access to the app.” * If user account if found, check if isAccountBlocked is false. Then update the user record to isEmailVerified = true. Display the message saying “Thank you for confirming your email address with us.” * Else, if isAccountBlocked = true, show a message “Sorry! Your account has been deactivated.”. |

## Rate the app

|  |  |
| --- | --- |
| Req. | * A prompt will be implemented that encourages the user to rate the app on the play store. * This is a third party plugin that’ll be integrated as is. |

## Navigation Menu

|  |  |
| --- | --- |
| Menu | * Navigation menu will have the following options   + Logged in as <user name> (the user name as retrieved from Facebook)   + App version number   + Home   + My Searches   + My Savvy Deals   + Send Feedback   + Logout |

## Home Screen

|  |  |
| --- | --- |
| Req. | * This screen will be split into 2 section:   + 4 targeted Categories   + Best Deals * Display an icon for “My Searches” on the top right corner. |
| Categories | * Section to display the notification if the email has not been verified yet. * Some indicator if there are new deals of the day to catch the user’s attention. We’ll check for the possibility later after we check for the user experience. * There will 4 fixed categories i.e. Beauty, Apparels, Bags, Mobile. * Image of the category (Fixed category images which will be bundled with the app) and the category name will be displayed. During the design phase, we shall propose some paid imagery (from <https://stock.adobe.com/>) to make this section look attractive. The images will need to be purchased and provided to us. * Tapping on category box / create search button:   + Will take the user to the search screen with the selected category name preloaded in the search form so that the user can start a new search in that category. * Tapping on the right hand side corner button:   + Display the top brands within a category so that it is easy for the user to create a search for those brands.   + These will be fixed top brands for each category bundled with the app. If you need to change them, then we will need to submit an update to the play store.   + Tapping on the brand will open the create search form with:     - Category selected     - Advanced search form opened     - Brand selected |
| Best Deals | * Make an API call to return random 10 deals of the day from the SavvyShopper database. If this returns empty, don’t show this section. * This will allow us to display:   + Title (this is something like “Minimum 60% off” instead of the product name)   + Description / Product title   + Image of the product: Lazy loading of images for better user experience and performance   + Buy button   + Create Search button * Tapping on Buy button:   + Will open up a web view redirecting the user to appropriate website as per the URL returned by the API. * Create Search:   + Will take the user to Search screen   + Following search form fields will be pre-loaded with the deal details selected by the user:     - Category     - Keywords (from the description of the deal) |

## Clicking on Buy button from anywhere in the app

|  |  |
| --- | --- |
| Sample Product URLs | Affiliate ID has been highlighted in blue below:  **Amazon**  <http://www.amazon.com/Kensington-K62533US-Contour-Notebook-Notebooks/dp/B0011YAOBE%3Fpsc%3D1%26SubscriptionId%3DAKIAJIGJAW7OF4KO5TIA%26tag%3Dseffcon-21%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3DB0011YAOBE>  *Seffcon-21 is our affiliate id for Amazon that will be used during development. Later this will need to be replaced by your account.*  **Filpkart**  <http://dl.flipkart.com/dl/american-tourister-flint-backpack/p/itmea59yghjcgkvc?pid=BKPEA59YANFGRYGN&affid=satyenkan>  **Snapdeal**  <http://www.snapdeal.com/product/lenovo-black-laptop-bag/656142880794?utm_source=aff_prog&utm_campaign=afts&offer_id=17&aff_id=74452> |
| Tapping on Buy button – common workflow | * The product URL will not be displayed anywhere within the app. * We shall open the product URL within an embedded webview instead the app, but will also provide an option to open the URL in the device’s browser. * On tapping the buy button, a web service call will be made on the server to track the tap event. It is possible that the user may not complete the purchase on the e-commerce site, but it is important for us to know when the user clicked on “buy”. The web service will record the following details:   + User id   + Date/time   + Ecommerce Site (Amazon or FlipKart or SnapDeal)   + SearchID (using which the user found this product. This will not be available the DOTD)   + IsDOTD? (If yes, then Ecommerce site will be FlipKart, as we are able to get DOTD only from FlipKart)   + ProductID (this is the product id of the ecommerce site fetched from the searched results)   + URL (DOTD URL or Product URL) |

## Loading top brand names

|  |  |
| --- | --- |
| Req. | * List of brands needs to be built manually. * These brands will be fixed within the app and if you want to change this list, a new app upgrade will be required. |
| Process | * We can build this manually for now for each category. * Use a sitelike:<http://business.mapsofindia.com/top-brands-india/top-lipstick-brands-in-india.html> to get only the top 10 brands. |
| **For Beauty** | 1. Lakme 2. Revlon 3. Avon 4. M.A.C 5. Maybelline 6. L'Oreal 7. Color Bar 8. Avon 9. Elle 18 10. Chambor 11. Nivea 12. Olay 13. Reviva 14. Garnier 15. Shahnaz Husain 16. Himalaya 17. VLCC 18. Biotique 19. Jovees 20. Amway |
| **For Apparels** | 1. Aaboli 2. Biba 3. Diya 4. EK 5. Fabindia 6. Melange 7. Rangriti 8. Allen Solly 9. Arrow 10. Wrangler 11. Provogue 12. Killer 13. Madura Garments 14. Raymond 15. Pantaloon 16. Levi Strauss 17. Levi’s 18. DJ&C 19. Spykar 20. Peter England 21. John Miller 22. Wills 23. Mufti 24. Oxemberg 25. United colors of Benetton 26. Pepe Jeans 27. Lee 28. Van Heusen 29. Barbie 30. Disney 31. Gini & Jony 32. Orange and Orchid 33. Goodway 34. Cool Baby  Cool Baby 35. NeedyBee  NeedyBee 36. Easy Feel  Easy Feel 37. PinkXenia  PinkXenia 38. Lifestyle - You  Lifestyle - You 39. DSB  DSB 40. Naughty Baby  Naughty Baby 41. House of Quirk  House of Quirk 42. MYKID  MYKID 43. Aglare  Aglare 44. Beau & Belle Littles  Beau & Belle Littles 45. Lilliput 46. Weekender 47. Zapp 48. The Gud Look 49. DeDe'S 50. DivyaEmporio 51. Vibes 52. silkone 53. A V Fashion 54. Salwar Studio 55. Clickedia 56. Vaamsi 57. Anu Clothing 58. Leading Lady 59. mrig 60. Paheli 61. FABFIRKI 62. Ustaad |
| **For Bags** | 1. 10th Planet 2. 1Bolzo 3. 120 Spice 4. 2ndmay 5. 3 coins plus 6. Adidas 7. Baggit 8. Hidesign 9. Lavie 10. Caprese 11. Ladida 12. The House of Tara 13. Da Milano 14. Lino Perros 15. Peperone 16. Kara |
| **For Mobiles** | 1. Samsung 2. Sony 3. Apple 4. HTC 5. Microsoft 6. Nokia 7. LG 8. Lenevo 9. Micromax 10. Panasonic 11. Karbonn 12. Intex 13. Asus 14. InFocus 15. Motorola |

## Create New Search

|  |  |
| --- | --- |
| Req. | * Search screen can be reached via the navigation menu or home screen when tapped on category or best deals. |
| Pre-Requisite. | * A user can create a new search only if the existing active searches < 5. The max. number of active searches that a user can create is as per the setting returned in the Login webservice call (MaxActiveSearchesForAUser and MaxTotalSearchesForAUser). * Tap on Create New Search button:   + Check for the count of total searches and if it is < MaxTotalSearchesForAUser, then:     - Check for the count of active searches and if it is < MaxActiveSearchesForAUser, then take the user to the create search form. On this form, there’ll be only 1 button “Create Search”.     - But if this count >= MaxActiveSearchesForAUser, then show him a warning “You’ve reached the limit for number of active searches (5). You’ll need to deactivate a search in order to activate this one”. But let him continue to the create search form and create a search. On this form, there’ll be only 1 button “Create Search”.   + Else, show him a message “You cannot create more than MaxTotalSearchesForAUser searches. You may delete a search and try again.” |
| Search screen | * Category drop down: This may be preloaded, if category or best deal from the home screen was tapped on. Else the user will need to select the category. Based on the selected category, user will be presented with any one of the below search forms:   **1] Beauty:**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Field** | **Control/** **Label** | **Mandatory?** | **Validations** | **Notes** | | Keyword | Text box | Y  N when advanced search is clicked and something is filled, else Y | Maxlen = 100 characters | E.g.: Revlon Lipstick | | Carry out search for | Drop down (1 to 14) for days | Y  Default 7 days |  | The search will run on a daily basis for the selected number of days.  The limit to the days – should be category specific and changeable. | | Advanced Search – This will be hidden and will be appear when the user opts for advanced search. | | | | | | Brand | Multi-selection allowed | N |  | Max 5 allowed. | | Color | Text box | N |  | To enter the color | | Enter max price | Text box | N | Only numbers allowed |  | | Submit | Button |  |  |  |   **2] Bags:**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Field** | **Control/** **Label** | **Mandatory?** | **Validations** | **Notes** | | Keyword | Text box | Y  N when advanced search is clicked and something is filled, else Y | Maxlen = 100 characters |  | | Carry out search for | Drop down (1 to 14) for days | Y  Default 7 days |  | The search will run on a daily basis for the selected number of days. | | Advanced Search – This will be hidden and will be appear when the user opts for advanced search. | | | | | | Type | Drop Down List | N |  | Fixed list of common types will be fixed with the app – laptop bag, backpack, rope bag, duffel bag, laundry bag, school bag, messenger bags, luggage, briefcases, handbags, sling bags, gym bags, cabin baggage (all categories at: <http://www.amazon.in/s/ref=nb_sb_noss?url=search-alias%3Dluggage&field-keywords>=)  We suggest to soft code these bag types at the database level. There will be no script developed to manage these bag types to begin with. | | Brand name | Multi-selection allowed | N |  | Max 5 allowed. | | Color | Text box | N |  | To enter the color | | Enter max price | Text box | N | Only numbers allowed |  | | Submit | Button |  |  |  |   **3] Mobile:**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Field** | **Control/** **Label** | **Mandatory?** | **Validations** | **Notes** | | Keyword | Text box | Y  N when advanced search is clicked and something is filled, else Y | Maxlen = 100 characters | E.g.: of the product. | | Carry out search for | Drop down (1 to 14) for days | Y  Default 7 days |  | The search will run on a daily basis for the selected number of days. | | Advanced Search – This will be hidden and will be appear when the user opts for advanced search. | | | | | | Accessory Manufacturer | Text box | N |  |  | | Phone Manufacturer | Text box | N |  |  | | Color | Text box | N |  | To enter the color | | Enter max price | Text box | N | Only numbers allowed |  | | Submit | Button |  |  |  |   **4] Apparels:**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Field** | **Control/** **Label** | **Mandatory?** | **Validations** | **Notes** | | Keyword | Text box | Y  N when advanced search is clicked and something is filled, else Y | Maxlen = 100 characters |  | | Sub-category | List | Y |  | Drop down with *boys clothing, girls clothing, women ethnic wear, infant wear, women clothing, men clothing. This will be a fixed list packaged with the app.* | | Carry out search for | Drop down (1 to 14) for days | Y  Default 7 days |  | The search will run on a daily basis for the selected number of days. | | Advanced Search – This will be hidden and will be appear when the user opts for advanced search. | | | | | | Brand name | Multi-selection allowed | N |  | Max 5 allowed. | | Color | Text box | N |  | To enter the color | | Size | Text box | N |  | E.g.: Small, Medium, Large, X-Large | | Enter max price | Text box | N | Only numbers allowed |  | | Submit | Button |  |  |  |   Note: Many of fields are free text boxes, if there are typos made by the user then the results will not be correct. |
| Tapping on Submit | * Make a webservice call to record the search requested by the user on the server. The search request will be also have an expiry date based on the number of days selected for the search to be carried out. Calculate UserDeactivatedOn Date as Expiry Date + 14 days so that after 14 days of expiry the search results are automatically deleted. * The user will not receive the search results immediately. A message will be displayed to the user as “Thank you for creating the search. You can activate it immediately or activate it later from the listing.” Clicking on “Activate Now” will take you to the listing with latest created search on the top and activated. Clicking on “Got it!” will take him to the listing with latest created search on the top and inactive. |

## Search Engine for Amazon

|  |  |
| --- | --- |
| Algorithm | * The search engine would be a scheduled job that will run every night and find all new search requests that are not yet deleted by the user, not yet deactivated by the user, and the current date <= expiry date of the search request. * Amazon Search API would be called with the correct search criteria for each category. This has been referred from: <http://docs.aws.amazon.com/AWSECommerceService/latest/DG/LocaleIN.html>. The search criteria for each category are defined in this section. * The search attempt will be tracked in the database and the search date/time and the number of results retrieved will be tracked for each search request. * After the search results are obtained they will be stored in the database against the requested search. * Rules:   + Record the unique product id as the [ASIN] field that is returned by Amazon.   + If the product with this ASIN for this search request already exists in the search results table:     - If yes, update the product details (price must have got changed) and IsAvailableWithCurrentSearch = 1.     - If this product is a bookmarked product, then       * Update this product in MySavvyDeals collection with the new details (name, price, description, productURL).       * If the existing price (when the product was bookmarked) and the new price from the latest search are different, then update MySavvyDeals.isPriceChanged to 1.       * (This would help the separate cron to send out price changed notifications for bookmarked items).   + If the product with this ASIN for this search request DOES NOT exist in the search results table:     - This is a new product.     - Insert the new product found with IsAvailableWithCurrentSearch = 1   + For all other existing ASINs for this same search request:     - Check if IsBookmarked = 1       * This was bookmarked by the user so it should not be deleted but update IsAvailableWithCurrentSearch = 0.     - Else if IsBookmarked = 0       * This was not bookmarked by the user so this should be deleted.   + Amazon provides many image product size variants. We have chosen Medium size variant. |
| Amazon API for beauty – search criteria. | * Search criteria to be passed to Amazon Search API * The results are sorted by BestSeller ranking, but we shall sort is as lowest price on the top when displaying the results to the user. * If the API call does not find the most relevant products as per the search criteria provided by us, in most cases, Amazon automatically returns related brand results. * Response group should be "ResponseGroup" => "Medium,Offers,Variations" as for apparels the price is available at the size variation level and not at the parent level  |  |  | | --- | --- | | **Search Criteria collected from user** | **Amazon Searchable Field** | | Category | SearchIndex = “Beauty” | | Brand Name | ItemSearch.Brand  For each brand a separate API call will need to be made. | | Keyword | Keyword = Keyword + Color  Search in ItemSearch.Keywords  E.g.: Revlon Lipstick Red | | Color  Output parameter: [color] | | Max Price | We’ll need to build this logic instead of passing this as the search criteria.  ItemSearch.SalesPrice <= Max Price | | In Stock | ItemSearch.Availability = ‘Available’ | |
| Amazon API for bags – search criteria. | * Search criteria to be passed to Amazon Search API  |  |  | | --- | --- | | **Search Criteria collected from user** | **Amazon Searchable Field** | | Category | SearchIndex = “Luggage” | | Brand Name | ItemSearch.Brand  For each brand a separate API call will need to be made. | | Keyword | Keyword = Keyword + Type + Color  Search in ItemSearch.Keywords | | Type | | Color  Output parameter: [color] | | Max Price | We’ll need to build this logic instead of passing this as the search criteria.  ItemSearch.SalesPrice <= Max Price | | In Stock | ItemSearch.Availability = ‘Available’ | |
| Amazon API for mobile– search criteria. | * Search criteria to be passed to Amazon Search API  |  |  | | --- | --- | | **Search Criteria collected from user** | **Amazon Searchable Field** | | Category | SearchIndex = “Electronics” | | Keyword | Keyword = Keyword + Color  Search in ItemSearch.Keywords | | Color  Output parameter: [color] | | Phone manufacturer / Accessory manufacturer | |  |  |  | | --- | --- | --- | | **Phone manufacturer** | **Accessory Manufacturer** | **Search in?** | | Y | N | ItemSearch.Manufacturer | | N | Y | ItemSearch.Manufacturer | | Y | Y | ItemSearch.Manufacturer = Accessory Manufacturer  ItemSearch.Keywords = Phone manufacturer | | | Max Price | We’ll need to build this logic instead of passing this as the search criteria.  ItemSearch.SalesPrice <= Max Price | | In Stock | ItemSearch.Availability = ‘Available’ | |
| Amazon API for apparels– search criteria. | * Search criteria to be passed to Amazon Search API  |  |  | | --- | --- | | **Search Criteria collected from user** | **Amazon Searchable Field** | | Category | SearchIndex = “Apparel” | | Keyword | Keyword = Keyword + Sub-category + Brand + Color + Size  Search in ItemSearch.Keywords | | Sub-category  Though ItemSearch.Department is an output from Amazon but in Indian locale it is not allowed to directly search using Department. So, we’ll need to use:  ItemSearch.Keywords | | Brand Name  Though ItemSearch.Brand is an output from Amazon but in Indian locale it is not allowed to directly search using Department. So, we’ll need to use:  ItemSearch.Keywords | | Color  Output parameter: [color] | | Size  Output parameter: [ClothingSize] | | Max Price | Not supported in the IN locale  But we can build this logic by comparing the lowest price with the max price requested by the user. | | In Stock | ItemSearch.Availability = ‘Available’ | |
| API limitations | * Returns only 10 results for 1 ItemPage. Max ItemPage allowed is 10. Therefore, 10x10 = 100 results for search criteria can be fetched. Reference: <http://docs.aws.amazon.com/AWSECommerceService/latest/DG/PagingThroughResults.html>   We now extract only 2 pages i.e. 20 results because there are a lot of irrelevant results from Amazon.   * Default sort values information at: <http://docs.aws.amazon.com/AWSECommerceService/latest/DG/localevalues.html> |
| Schedule | * The search engine will run only once a day at 03:30 hrs. |
| More findings | * For some products the prices are not being returned. On the website the price is displayed but in the API it is not being returned. We have found a way to pick up the correct price of a product. It has been documented separately in **Amazon-Price-Not-Found-Analysis-3Mar2016.docx** * Amzon does not return related products like FlipKart API – it simply says “products not found”. However, we have observed that for top mobile brands, even if there is a typo in the name, it returns correct results e.g.: samsang galxy, micromx * Max price will need to be built by us instead of passing this as an input search criteria to the amazon API. |
| Fine tuning | * When we search for “mens clothing” OR “boys clothing”, results of women were also added to it. So now we explicitly added a filter that will not save search results if they contain the word “women” or “woman” or “womens” in this case. |
|  | * When we search for “girls clothing” OR “women clothing”, results of men were also added to it. So now we explicitly added a filter that will not save search results if they contain the word “men” or “man” or “mens” or “boys” in this case. |

## Search Engine for FlipKart

|  |  |
| --- | --- |
| Algorithm | * The search engine would be a scheduled job that will run every night and finds all new search requests that are not yet deleted by the user, not yet deactivated by the user, and the current date <= expiry date of the search request. * FlipKart Search API would be called with the correct search criteria for each category. The search criteria for each category are defined in this section. * The search attempt will be tracked in the database and the search date/time and the number of results retrieved will be tracked for each search request. * After the search results are obtained they will be stored in the database against the requested search. * Rules:   + Record the unique product id as the [productId] field that is returned by FlipKart.   + If the product with this productId for this search request already exists in the search results table:     - If yes, update the product details (price must have got changed) and IsAvailableWithCurrentSearch = 1.     - If this product is a bookmarked product, then       * Update this product in MySavvyDeals collection with the new details (name, price, description, productURL).       * If the existing price (when the product was bookmarked) and the new price from the latest search are different, then update MySavvyDeals.isPriceChanged to 1.       * (This would help the separate cron to send out price changed notifications for bookmarked items).   + If the product with this productId for this search request DOES NOT exist in the search results table:     - This is a new product.     - Insert the new product found with IsAvailableWithCurrentSearch = 1   + For all other existing productIds for this same search request:     - Check if IsBookmarked = 1       * This was bookmarked by the user so it should not be deleted but update IsAvailableWithCurrentSearch = 0.     - Else if IsBookmarked = 0       * This was not bookmarked by the user so this should be deleted.   + FlipKart provides many product image size variants. We have used 200x200 version. |
| FlipKart API for beauty – search criteria. | * Search criteria to be passed to FlipKart Search API * If the API call does not find the most relevant products as per the search criteria provided by us, in most cases, FlipKart automatically returns related brand results  |  |  | | --- | --- | | **Search Criteria collected from user** | **FlipKart Searchable Field = query** | | Category | Don’t need to pass. | | Brand Name  If multiple brands are selected, we will need to make multiple calls – one for each brand to get the results.  Output parameter: [productBrand] | Query = brand name + keyword + color  Pass in query. | | Keyword | | Color  Output parameter: [color] | | Max Price | Not supported in query | | In Stock | [inStock] => 1 | |
| FlipKart API for bags – search criteria. | * Search criteria to be passed to FlipKart Search API  |  |  | | --- | --- | | **Search Criteria collected from user** | **FlipKart Searchable Field = query** | | Category | Don’t need to pass | | Keyword | Query = brand name + type + keyword + color  Pass in query. | | Type | | Brand Name  If multiple brands are selected, we will need to make multiple calls – one for each brand to get the results.  Output parameter: [productBrand] | | Color  Output parameter: [color] | | Max Price | Not supported in query | | In Stock | [inStock] => 1 | |
| FlipKart API for mobile– search criteria. | * Search criteria to be passed to FlipKart Search API  |  |  | | --- | --- | | **Search Criteria collected from user** | **FlipKart Searchable Field = query** | | Category | Don’t need to pass | | Keyword | Query = Phone manufacturer + Manufacturer + keyword + color  Pass in query. | | Phone manufacturer | | Manufacturer | | Color  Output parameter: [color] | | Max Price | Not supported in query | | In Stock | [inStock] => 1 | |
| FlipKart API for apparels– search criteria. | * Search criteria to be passed to FlipKart Search API  |  |  | | --- | --- | | **Search Criteria collected from user** | **FlipKart Searchable Field = query** | | Category | Don’t need to pass | | Keyword | Query = Sub-category + keyword + color + size  Pass in query. | | Sub-category | | Brand Name  If multiple brands are selected, we will need to make multiple calls – one for each brand to get the results.  Output parameter: [productBrand] | | Size  Output parameter: [size] | | Color  Output parameter: [color] | | Max Price | Not supported in query | | In Stock | [inStock] => 1 | |
| API limitations | * Maximum of 10 products are retrieved for each search using the Product Search API. * Flipkart’s API URLs are valid for 10 hours only. Results cannot be cached. |
| Schedule | * The search engine will run only once a day at 04:30 hrs. |
| More findings | * Replace the word Apparels with Apparel as else it does not return search results. * FlipKart returns related products automatically if the exact match is not found. E.g.: If Lakme Foundation is not found, it returns us Lame eyeslashes. * FlipKart does not group the products – i.e. if a product is available in different colour or size variations, then FilpKart has different product records for each.   E.g.: A kurta is available in S, M, L, XL sizes. So for a single product, there are practically 4 products found.  <http://www.flipkart.com/rangmanch-pantaloons-printed-women-s-anarkali-kurta/p/itme92jh3zxg3zyg?pid=KTAE8MVWHBHHJMQU&affid=satyenkan>    <http://www.flipkart.com/rangmanch-pantaloons-printed-women-s-anarkali-kurta/p/itme92jh3zxg3zyg?pid=KTAE8MVWFAWHZPZD&affid=satyenkan>    <http://www.flipkart.com/rangmanch-pantaloons-printed-women-s-anarkali-kurta/p/itme92jh3zxg3zyg?pid=KTAE8MVW5VHM5HCY&affid=satyenkan>    <http://www.flipkart.com/rangmanch-pantaloons-printed-women-s-anarkali-kurta/p/itme92jh3zxg3zyg?pid=KTAE8MVWFAWHZPZD&affid=satyenkan>    As you can see – it’s the same product with 4 different pids.  So let’s say if the user searches for “Pantaloons Kurta” (without specifying the size), when we search using FlipKart API, we get duplicate products. As per the above example, 4 out of 10 products will be same. This will get frustrating for the user as when he clicks on the result found by SavvyShopper, he will land to the same FlipKart product (though technically they are individual products stored by FlipKart).  Same is true for Colour variations as well.  Solution:  FlipKart returns us sizeVariants (as a group of product Ids having different sizes of the same product) for every product.  We can ignore the repeated product ids that are already present as sizeVariants of a product. This way we’ll show only one product to the user instead of duplicates.  FlipKart also returns us colorVariants (as a group of product Ids having different colours of the same product) for every product.  Thus we can ignore duplicate colour variations as well.  When the user visits the site, he’ll be able to select the colour / size on the website itself and can check for the required variations.  However, the user might still see duplicate products if it is being sold by different dealers. In this case the selling price is different – so that should be fine. |
| Fine tuning | * We prepend the brand name / phone manufacturer to the keyword before firing the API. * E.g.: User added “Samsung mobile” in keyword and also selected Samsung in phone manufacturer. Thus the keyword string passed to the API becomes “Samsung Samsung mobile” which does not yield any search results. To overcome this, before prepending the brand name / phone manufacturer to the keyword, we’ll check if they are already not a part of the keyword. |

## SnapDeal – Gather data

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Gathering data | * This ecommerce platform does not provide the search API. So for each category, we need to gather all the results for each category on a daily basis and populate the SavvyShopper database. This will need to be done once a day(00:00 hrs) for each of 11 SnapDeal categories.  |  |  | | --- | --- | | **Our website’s category** | **SnapDealcatgeory** | | Beauty | Beauty & Personal Care | | Mobiles | Mobiles\_Tablets | | Apparels | Boys Clothing(2-8 yrs)  Boys Clothing(8-14 yrs)  Girls Clothing(2-8 yrs)  Girls Clothing(8-14 yrs)  Womens\_Ethnic\_Wear  Infant\_Wear  Mens\_Clothing  Womens\_Clothing | | Bags | Bags\_Luggage | |
| Ways to gather the products from SnapDeal | * There are 3 ways of gathering products from SnapDeal:   + API Call   + FTP Download   + Scraping data on the snapdeal website * The below sections explain the pros and cons of each method. |
| API call | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **No.** | **Category** | **MongoDB Collection** | **Total Records Imported** | **Records on website** | **Total Time Required** | | 1. | Beauty | beauty | 18,357 | 19,188 | 2 m 16s | | 2. | Bags | bags | 68,483 | 78,450 | 4 m 33s | | 3. | Mobiles | mobiles | 39,08,343 | 66,55,444 | 5h 38m 31s | | 4. | **Apparels** |  | **Total: 5,15,687** |  |  | | *Men’s clothing* | mens\_clothing | 1,16,155 | 1,76,993 | 1m 10s | | *Boys Clothing(2-8 yrs)* | boys\_clothing\_2\_8yrs | 11,397 | 15,931 | 45s | | *Boys Clothing(8-14 yrs)* | boys\_clothing\_8\_14yrs | 6,686 | 9,393 | 27s | | *Girls Clothing(2-8 yrs)* | girls\_clothing\_2\_8yrs | 14,954 | 22,627 | 57s | | *Girls Clothing(8-14 yrs)* | girls\_clothing\_8\_14yrs | 7,403 | 10,596 | 28s | | *Infant wear* | infant\_wear | 13,473 | 19,556 | 49s | | *Women’s Clothing* | womens\_clothing | 95,237 | 1,43,838 | 5m 56s | | *Women’s  Ethnic wear* | womens\_ethnicwear | 2,50,382 | 4,00,586 | 17m 12s |   Process:   * The product feed returns only 500 products at a time. At the end of the product listing, there is a NextURL parameter. We’ll have to crawl all the NextURLs till we don’t find one. * All this data for each category will be dumped into separate mongodb collections – separate for each category.   Pros:   * The results for Beauty, Bags and each of the Apparels category is retrieved fairly quickly. * Being an API call the data will be fresh. * Since we get all the data in a category, we can meet most of the search criteria requested by a user.   Cons:   * SnapDeal does not provide a search API. Instead it provides API to fetch all the products in a single category. * We cannot make more than 20 API requests per second so we cannot make too many simultaneous calls. * The mobile category takes a long time to execute. |
| FTP Download | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Sr. No** | **Categories** | **CSV Name** | **using FTP csv** | **using API call** | | 1 | Beauty & Personal Care | topFashion | 6,731 | 18,357 | | 2 | Bags & Luggage | topFashion | 9,393 | 68,483 | | 3 | Mobiles & Tablets | topElectronics | 68,354 | 39,08,343 | | 4 | **Apparels** | **topFashion** | **1,02,093** | **5,15,687** | | *Mens clothing* | topFashion | 33,769 | 1,16,155 | | *Womens clothing* | topFashion | 23,564 | 95,237 | | *Womens Ethnic wear* | topFashion | 33,165 | 2,50,382 | | *Boys clothing(2-8yrs)* | topFashion | 2,383 | 11,397 | | *Tween Boys(8-14yrs)* | topFashion | 1,694 | 6,686 | | *Girls clothing(2-8yrs)* | topFashion | 3,235 | 14,954 | | *Girls clothing(8-14yrs)* | topFashion | 1689 | 7,403 | | *Infant Wear* | topFashion | 2,594 | 13,473 |   Process:   * The product feed is available at the FTP location. We can download that programmatically – we need to download only 2 CSV files – topFashion.csv (about 26MB) and topElectronics.csv (about 11MB). Except for mobiles category, all other categories are found in the topFashion.csv. Data for following extra categories also get downloaded:   TopFashion-   * Men's Footwear * Watches * Sports & Fitness * Fashion Accessories * Jewellery * Fragrances * Musical Instruments * Baby Care * Eyewear * Women's Footwear * Kids Footwear * Gifting & Events   TopElectronics-   * Computers & Peripherals * Stationery * Appliances * TVs, Audio & Video * Office Equipment * Cameras & Accessories * Online Education * Gaming * Movies & Music * All this data for each category will be dumped into 2 separate mongodb collections: TopElectronics and TopFashion.   Pros:   * The results for all required categories are retrieved and imported fairly quickly. * Only the top selling products are returned. Therefore, mobile data is now about 68K records instead of 39 lakhs. * SnapDeal mentions that they upload fresh data every hour. We can retrieve the latest file from FTP and populate the mongodb collections before our search queries are fired.   Cons:   * If the user searches for products that are not top selling on SnapDeal, then the results won’t be returned. How frequent would this be – will users not really be searching for popular products? |
| Scraping data using the search feature implemented on the snapdeal website | Process:   * When a search query is run on SnapDeal website, in the background it makes a call to the following GET query: <http://www.snapdeal.com/acors/json/product/get/search/0/144/48?q=&sort=plrty&brandPageUrl=&keyword=samsung%20galaxy&vc=&webpageName=searchResult&campaignId=&brandName=&isMC=false&clickSrc=go_header> * Using a CURL call we have implemented a sample POC that is able to retrieve the products from the URL.   Pros:   * This will work as a real time API call instead of gathering all the data at once and then querying the SavvyShopper database.   Cons:   * Being a paginated web service call this web service does not return all the records in one go. We’ll need to navigate through every page to crawl the data. This is time consuming. * Not all products being returned provide the image URL. This is essential data which should be returned. So we’ll have to limit the page size to only 8 so that we get the images for all the products in the response. * We get the HTML data and will need to rely on DOM parsing. If SnapDeal changes the HTML format, we’ll need to rewrite our script. * This is not an official way provided by SnapDeal to get the data (unlike the above 2 methods) – there is a probability that they may change the web service implementation and accordingly we may need to change our implementation. |
| DAR | |  |  |  |  | | --- | --- | --- | --- | | **Factor** | **API Call** | **CSV download** | **Web scraping** | | Authenticity | SnapDeal provides to all affiliates | SnapDeal provides to all affiliates | Not an exposed method.  Risk: Subject to change | | Speed | Good for all categories except Mobiles (5.5 hrs+) and Women Ethnic wear (17m)  500 products are returned at once.  An API is always optimized. | Good  topFashion.csv (about 26MB) and topElectronics.csv (about 11MB)  This needs to be done once in day.  Download and Import of these CSVs into Mongo collections happens in seconds. | Slower because:   * It returns results in HTML format. * It allows fetching not more than 8 products at a time with images. If we try to return more than 8, then they won’t have images in it. * This will take comparatively more time to retrieve all the data in chunks of 8 each by crawling the website. * Moreover user specific data is being scraped. * If SnapDeal discovers this they may block our IPs. * We had to tamper the user agent to get the results using the CURL call – otherwise it returned access denied. | | Results data format | JSON format | TSV format | HTML format.  DOM Parsing overhead.  If the HTML structure of the website changes, we’ll need to change our logic. | | Quality of results | All products within a category returned and we can search for the requested search criteria of all users – But we cannot identify the popularity of the data. | Only top selling products are returned. | Results are sorted by popularity. | | Effectiveness in meeting the users’ search criteria | Very good because we get all data. | Can be rare case that a user searches for something that is not returned as a part of top selling items in SnapDeal.  But this user search request can still be fulfilled using the other ecommerce vendors – Amazon and FlipKart. | Very good because that’s what the user gets on the website too. | | Development complexity (more complex – more time required for dev & testing) | We need to retrieve data in chunks of 500. If the script breaks, we need to build the resume functionality.  Medium level complexity. | Fairly quick implementation with the mongoimport command. | We need to retrieve data in chunks of 8. If the script breaks, we need to build the resume functionality. We also need to build the DOM parser.  Medium level complexity. |   Conclusion: Based on the above evaluation, we rank FTP CSV download as the highest and recommend implementation of the same. |
| Schedule | * The snapdeal data table will be truncated and the data will be re-inserted into them whenever the script is run. |

## Search Engine for SnapDeal

|  |  |
| --- | --- |
| Algorithm | * The search engine would be a scheduled job that will run every night and   + Download the latest CSV and import it into mongo (as per the above section).   + find all new search requests that are not yet deleted by the user, not yet deactivated by the user, and the current date <= expiry date of the search request. * SnapDeal Search Engine will search the Savvyshopper database with the correct search criteria for each category. The search criteria for each category are defined in this section. * The search attempt will be tracked in the database and the search date/time and the number of results retrieved will be tracked for each search request. * After the search results are obtained they will be stored in the database against the requested search. * Rules:   + Record the unique product id as the [id] field that is returned by SnapDeal.   + If the product with this id for this search request already exists in the search results table:     - If yes, update the product details (price must have got changed) and IsAvailableWithCurrentSearch = 1.     - If this product is a bookmarked product, then       * Update this product in MySavvyDeals collection with the new details (name, price, description, productURL).       * If the existing price (when the product was bookmarked) and the new price from the latest search are different, then update MySavvyDeals.isPriceChanged to 1.       * (This would help the separate cron to send out price changed notifications for bookmarked items).   + If the product with this id for this search request DOES NOT exist in the search results table:     - This is a new product.     - Insert the new product found with IsAvailableWithCurrentSearch = 1   + For all other existing ids for this same search request:     - Check if IsBookmarked = 1       * This was bookmarked by the user so it should not be deleted but update IsAvailableWithCurrentSearch = 0.     - Else if IsBookmarked = 0       * This was not bookmarked by the user so this should be deleted.   + Snapdeal provides only 1 product image URL and no size variants. |
| Algorithm | * ~~Combine the title and description fields into the description column itself.~~ * This has been removed since a lot of irrelevant search results were being returned. We now search only in the title column for phrase search and then for individual terms. |
| SnapDeal for beauty – search criteria. | * Search criteria to be passed to SnapDeal DB Search  |  |  | | --- | --- | | **Search Criteria collected from user** | **SnapDeal Database Column** | | Category | Search in the snapdealtopfashioncsv mongo collection in the SavvyShopper database. | |  |  | | Keyword | Split each word of the Keyword and search in the columns [Description]   1. Keyword = (Word1 AND Word2…) AND Color 2. If results = 0, search for   Keyword = (Word1 OR Word2…) AND Color   1. If results = 0, search for   Keyword = (Word1 OR Word2…) | | Color | | Brand Name | [Brand] | | Max Price | [Offer Price]  Search for max price <= [Offer Price] | | In Stock | [availability] => in stock | |
| SnapDeal for bags – search criteria. | * Search criteria to be passed to SnapDeal DB Search  |  |  | | --- | --- | | **Search Criteria collected from user** | **SnapDeal Database Column** | | Category | Search in the snapdealtopfashioncsv mongo collection in the SavvyShopper database. | | Keyword | Split each word of the Keyword and search in the columns [Description]   1. Keyword = (Word1 AND Word2…) AND Type AND Color 2. If results = 0, search for   Keyword = (Word1 OR Word2…) AND Type AND Color   1. If results = 0, search for   Keyword = (Word1 OR Word2…) AND Type | | Color | | Type | | Brand Name | [Brand] | | Max Price | [Offer Price]  Search for max price <= [Offer Price] | | In Stock | [availability] => in stock | |
| SnapDeal for mobile– search criteria. | * Search criteria to be passed to SnapDeal DB Search  |  |  | | --- | --- | | **Search Criteria collected from user** | **SnapDeal Database Column** | | Category | Search in the snapdealtopelectronicscsv mongo collection in the SavvyShopper database. | | Keyword | Split each word of the Keyword and search in the columns [Description]   1. Keyword = (Word1 AND Word2…) AND Phone manufacturer AND Manufacturer AND Color 2. If results = 0, search for   Keyword = (Word1 OR Word2…) AND Phone manufacturer AND Manufacturer AND Color   1. If results = 0, search for   Keyword = (Word1 OR Word2…) AND Phone manufacturer AND Manufacturer   1. If results = 0, search for   Keyword = (Word1 OR Word2…) AND Phone manufacturer | | Phone manufacturer | | Manufacturer | | Color | | **--OR--** |  | | Max Price | [Offer Price]  Search for max price <= [Offer Price] | | In Stock | [availability] => in stock | |
| SnapDeal for apparels– search criteria. | * Search criteria to be passed to SnapDeal DB Search  |  |  | | --- | --- | | **Search Criteria collected from user** | **SnapDeal Database Column** | | Category | Search in the snapdealtopfashioncsv mongo collection in the SavvyShopper database. | | Sub-category | [category\_name]   |  |  | | --- | --- | |  | [category\_name] | | Boys Clothing | Boys Clothing (2-8 Yrs) **OR**  Tweens Boys | | Girls Clothing | Girls Clothing (2-8 Yrs) **OR** Girls Clothing (8-14 Yrs) | | Women Ethnic Wear | Women's Ethnic Wear | | Mens Clothing | Men's Clothing | | Womens Clothing | Women's Clothing | | Infant Wear | Infant Wear | | | Brand Name | [Brand] | | Keyword | Split each word of the Keyword and search in the columns [Description]   1. Keyword = (Word1 AND Word2…) AND Size AND Color 2. If results = 0, search for   Keyword = (Word1 OR Word2…) AND Size AND Color   1. If results = 0, search for   Keyword = (Word1 OR Word2…) | | Color | | Size  There are no defined sizes – they could be in terms of age, or L/XL, or numbers. Thus the accuracy cannot be guaranteed. | | Max Price | [Offer Price]  Search for max price <= [Offer Price] | | In Stock | [availability] => in stock | |
| Schedule | * The search engine will run only once a day at 05:30 hrs. |
| More findings | * SnapDeal search results are searched from the Mongo database instead of a live API call as we use the FTP download method. We have built a full text search query in Mongo to search for the results.   E.g.: User entered “White Long Kurta” and submitted the search criteria.  Now the Mongo Full Text search searches for every word “White”, “Long” and “Kurta” in the products database and returns irrelevant results too, like White Shirt, Long Skirt.  >> To cater to this, we further researched and found that the search query also returns us a relevance score for each searched result. Average score for relevant results is observed to be greater than 1. Thus, in order to capture relevant results, we store only those search results with relevance score greater than 1.  If the keyword is a single word then we ignore the relevance score check. |
| Phrase search implementation | * 1. SnapDeal data is imported into mongo and then we search the database instead of the live product search API call.   2. For this, as the first step, we have implemented “phrase search” i.e. the exact phrase should be found in the title of the product. E.g.: If the user enters “nail brush”, then we shall search for all those products who have “nail brush” contained in the title.   3. MongoDB provides text indexes to support query operations that perform a text search of string content. We have used this feature from mongo and implemented **phrase search**. (For details you can check the section “Phrases” in <https://docs.mongodb.org/manual/reference/operator/query/text/>)   4. As the second step, we then query the database for the second time. This time we search on **individual terms** rather than the complete phrase. For the same example “nail brush”, now the words “nail” and “brush” are individually searched for. But this time we get results like “**nail** rounded **brush**”, **nail** polish” and “hair **brush**” too which might be less relevant or even irrelevant.   5. Mongo also provides an excellent feature where it provides you a TextScore. The better the search result (on individual terms), higher is the TextScore. (For details you can check the section “TextScore” in <https://docs.mongodb.org/manual/reference/operator/query/text/>)   6. The TextScore are decimal numbers ranging from 0.1 to 2 (from what we have seen so far). So in the above example, you can consider “nail rounded brush” has a score of 2, where are “nail polish” and “hair brush” have a score of 1.   7. Therefore, we further fine-tuned the algorithms with the following approach:      1. In step b above, the search results that are obtained, considering that they are the most accurate search results, we forcibly update the **TextScore to a high value 10**.      2. Also, FlipKart and Amazon (since they provide a search API already) should also have a **TextScore of 10**.      3. In step d above, the searches are on individual terms in SnapDeal, and therefore we keep the TextScore that is obtained as it is.      4. That means, FlipKart, Amazon and SnapDeal exact phrase search will always be the most relevant results and should be on the top, followed by less relevant results from SnapDeal.   **These search engine side changes have already been implemented as only after this fine tuning we could test and confirm.** |

## My Searches

|  |  |
| --- | --- |
| Req. | * My searches can be reached via the navigation menu. * This will list all the search requests created by the user. * On the top right corner of this screen will be a button to create new search and reload the search requests. |
| My Searches | * My Searches listing will show the following:   + Image of the category (Fixed category icons which will be bundled with the app)   + Searched Keyword appended by advanced search parameters like max price, type, brand color etc. if the user has created an advanced search, else we’ll display only the searched keyword.   + Number of search results bubble.   + Expires on E.g.: Expires in 14 days. (We’ll calculate this at the mobile end as the web service will return us the expirydate). This will be shown only for active searches.   + Option to delete the search request (this can come on a swipe to delete action).   + Option to deactivate an active search request / Option to reactivate a deactivated search request.   + On long press – allow the user to edit the search. Even an active search can be edited. * The search listing will show the active search requests on the top (recently created on the top) followed by the deactivated ones (max. 15). * Deleted search requests will not be displayed. * Each search request will be tap-able to display the actual search results. |
| Refresh My Searches | * Using the refresh icon on the top right corner, the search requests can be refreshed from the server. * Each refresh will truncate the old search requests and reload the search requests from the server. * Truncate all the search results. Update flag to fetch results from server to 1 for all searches. |
| Auto-refresh | * Only if isUpdateBubbleCount = 1, then the bubble count will be automatically refreshed when this screen is loaded if the net is available, else will show the bubble count from the local database. |
| Activate/deactivate | * If an inactive search is activated, do the following:   + Make a web service call to activate this searchid on the server.   + Display a message as “We shall do our best to find the best deals matching your search criteria for the next <selected> days. You’ll be notified on a daily basis as we find new matches for you.”   + Note the user has a wait period till our search algorithms for each search engine run at the midnight and the next day morning send a notification for new deals found. * If an active search is deactivated, do the following:   + Make a web service call to deactivate this searchid on the server.   + Display a message as “Your search has been deactivated. You’ll no longer receive new search results. The search results will be deleted after 14 days.” |
| Delete search request | * Prompt the user as “This search request and all the associated search results will be permanently deleted. Are you sure you want to continue?” * On confirmation, make a web service call to mark the search request as deleted on the server. Remove the search request from the listing screen in the app. |

## Search Results

|  |  |
| --- | --- |
| Req. | * Tapping on an active or deactivated search request should display the search results. Deactivated will display search results only for 14 days. |
| Tapping on a search request will display the search results | * If the mobile is in offline mode, then show the old results. * Else, make a web service call to fetch the search results for the selected search request if the flag to fetch results from server is 1. * The old results will be truncated from the local database and the new data will be inserted. This data will have new products as well as old products that were bookmarked earlier. * First 20 search results will display:   + Product image: Lazy loading of images for better user experience and performance.   + Product title   + Price   + Date when the search result was provided: Yesterday, Today, 2 Days back etc. This will show up as Today for all items generally, except the bookmarked ones which can be older.   + Buy [Button]   + Icon of the ecommerce vendor   + Tagged as **New** if new products are found: All products having the latest created date will be tagged as new.   + Tagged as **Bookmarked**: If the product was previously bookmarked, it will show as bookmarked.   + Tagged as **Available / Not available**: If the product was bookmarked earlier, but not retrieved in the latest API search, then it will be marked as Not Available, else, Available. |
| Refresh My Search Results | * Using the refresh icon on the top right corner, the search results can be refreshed from the server. * Each refresh will truncate the old search results and reload the search results from the server. * Update flag to fetch results from server to 0 for that search request. |
| Tapping on Buy | * Will open up a web view redirecting the user to appropriate website as per the URL returned by the APIs. |
| Tapping on Bookmark icon | * This icon has 2 states:   + If already bookmarked, tapping will allow the user to remove the product from bookmarks.   + If not bookmarked, tapping will allow the user to add the product into bookmarks. * The bookmarked status is recorded on the server using a web service call. |
| Sort Results | * Sort products by price (high to low) (low to high - default) * For a better user experience at the Android end, we propose the following:  1. Whenever the search results are obtained, the default order should be “NEW” on the top, followed by most relevant results on the top, followed by lowest price on the top. 2. Currently, in the Android scope of work, we have considered only 2 sort options: 1.) Price (low to high) and 2.) Price (high to low). 3. But now, we propose having 3 sort options 1.) Relevance – default sort option 2.) Price (low to high) and 3.) Price (high to low). |

## View More Results – Tracking.

|  |  |
| --- | --- |
| Req. | * By default 20 search results will be returned. The user can request for more. * On each request a web service call to fetch the results will be made. |
| Tracking | * To track the interest of users, on each request to view more results, we’ll capture the following:   + UserID   + SearchID   + SearchCategoryID   + Datetime |

## My Savvy Deals

|  |  |
| --- | --- |
| Req. | * Tapping on My Savvy Deals from the left menu should display the bookmarked savvy deals. * This will refresh the results from the server and show the latest price. In case if the mobile device is offline, it will display the bookmarked deals that are locally cached. |
| Favourites | * View all favourite products. For each favourite product it will display   + Product image: Lazy loading of images for better user experience and performance.   + Product Title   + Price   + Icon of the ecommerce vendor   + Buy [Button]   + Tagged as **Bookmarked**: Tapping will remove product from favourites.   + Tagged as Price Changed. |

## Send Feedback

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Req. | * Tapping on Send Feedback from the left menu should display a form.  |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Field** | **Control/** **Label** | **Mandatory?** | **Validations** | **Notes** | | Message | Text area | Y | Maxlen = 500 characters |  | | If your feedback is specific to a search, let us know | | | | | | Select Search | List from which you can select one of the searches created by the user | N |  |  | | **Send Feedback** | Button |  |  | Web service call to record the feedback on the server at the database level. | |

## If the user account is deleted from server

|  |  |
| --- | --- |
| Req. | * If the user account is automatically deleted from the server as a result of not verifying the email address within 3 days, and if the same user is logged in to the app, then we need to have a real time check in the following web services, such that the user is forcibly logged out of the app and is displayed the message as “Sorry! Your account has been deleted as you did not verify your email address with us. You’ll need to sign up again to gain an access to the app.”   + Send feedback   + Get Deals of the Day   + Create search   + Update search status   + Edit search request   + Delete search request   + View Search Results   + Add to My Savvy Deals   + View My Savvy Deals   + Remove from My Savvy Deals   + Tracking Buy Button   + My Searches   + Update GCM |

## Cron for Email verification notification

|  |  |
| --- | --- |
| Req. | * If the user has not verified his email address, we’ll need to send him notifications in the app so that he completes that step. |
| Background job | * This job will check for all users with isEmailVerified = false and IsEmailStepCompleted = 1 and difference between currentdate and the createdondate is <= 3. * A push notification will be sent to all such users in their app like “Reminder – Please confirm your email address with SavvyShopper.” * This notification message will appear within the app on the home screen (can be displayed at the top). * On clicking this notification, a dialog box will appear as “We request you to confirm your email address with us”, followed by 2 buttons: 1. Dismiss 2. Send me the verification email again.   + The user will also be allowed to change his email address here.   + If the user taps on Dismiss, the dialog box and the notification message will disappear.   + If the user taps on “Send me the verification email again”, a web service call will be made to send the email verification email again to the user (This verification email will be valid only for the 3 days). The message will disappear from the display.   + Note: the user can tap on “Send me the verification email again” for a notification only 3 times, after which a message will be displayed as “Sorry! You’ve exhausted the number of requests.” |
| Schedule | * Once a day: 10:30 am |

## Cron to send search notifications to user

|  |  |
| --- | --- |
| Search notifications | * **On a daily basis**, a cron will run and identify for all active search requests (i.e. search requests that are not yet deleted by the user, not yet deactivated by the user, and the current date <= expiry date of the search request) and find out in the search results table the count of new results. * If there are new results, send a push notification as “x new deals found for <keyword>/<item name>.” * Also create a badge to indicate that there is some update in this app. |
| At the mobile end | * Once the push notification is received, the SP for isUpdateBubbleCount will be set to 1 so that when the user is displayed the “My searches” screen, the number of search results for each existing active search request get updated with the new count (to avoid complete reload). Also update the flag for search request to fetch results from server when that search request is tapped on. |
| Cron schedule | * The cron can run in the morning at 7:30 am daily. |

## Cron to send price changed notifications

|  |  |
| --- | --- |
| Cron | * Get all MySavvyDeals where isPriceChanged=1.   + Compose and send a push notification to the user informing him “The product <name> bookmarked by you has a changed price.”   + Update isPriceChanged = 0 for this product.   + Tapping on this notification will open the “My Savvy Deals” page on the device. |
| Note | * If the user receives multiple such notifications, we shall club them   + We can count the number of products that have changed price and show him a message as “x of your bookmarked products have changed prices.” |
| Schedule | * The cron can run in the morning at 8:30 am daily. |

## Cron for Email verification not completed – delete account

|  |  |
| --- | --- |
| Req. | * A cron will scan for all user records with isEmailVerified = False and difference between currentdate and the createdondate is > 3. Delete the user account and clean up all his data (permanent deletion from search results, searches and my savvy deals). Move the user record from user table to DeletedUser table. |
| Cron schedule | * Once a day: 6:30 am |

## Cron to fetch the Deals of the Day

|  |  |
| --- | --- |
| Algorithm | * Truncate all the existing deals before you insert the fresh deals. * FlipKart Deals of the Day:   + We’ll pick up the deals using Flipkart “Deal of the Day” (DOTD) API. FilpKart API does not return category wise deals. We have worked on a technical POC that will retrieve these deals of the day and allow us to filter only for the intended categories. Have a look at: <http://www.seffcon.com/ecommerce-api/flipkart/dotd.php>. This may return us only a few records and not all may belong to the categories that we plan to support. Record the following:     - Title (this is something like “Minimum 60% off” instead of the product name)     - Description     - ProductImageURL     - ProductURL * Amazon Deals of the Day:   + Amazon API to query 3 categories (Bag, Mobile and Beauty) with the following search parameters   MinPercentageOff=> "50", (value in %)  ResponseGroup => ItemAttributes,Offers   * + This will return only those products that have at least 50 % off.   + Note: For apparels, the search parameter MinPercentageOff is not available for the IN Locale.   + Record the following     - Title (We’ll create this as x% off)     - Product title     - ProductImageURL     - ProductURL * SnapDeal Deals of the Day:   + Download the fresh data from the FTP location.   + SnapDeal returns us the actual MRP and the effective price – you’ll always see MRP as crossed out and the effective price on display. The higher the difference, the more the discount.   + Query all the category tables (snapdealTopElectronics, snapdealTopFashion) where (MRP - Effective Price)/MRP \* 100 >= 50.   + Record the following     - Title (We’ll create this as x% off)     - Product title     - ProductImageURL     - ProductURL |
| Schedule | * Hourly |

## Cron to delete all searched data for inactive searches

|  |  |
| --- | --- |
| Cron | * Get all the inactive search requests for which date of deactivation <= current date.   + Delete all the search results data for this search id. (permanent deletion from search results)   + ~~So if the user taps on an inactive search after 14 days on his device, he won’t be able to see the search results unless he must have bookmarked them. Until 14 days, he’ll be able to see them.~~ |
| Schedule | * Once a day |

## Cron to delete all searched data for a deleted search request by the user

|  |  |
| --- | --- |
| Cron | * Get all the deleted search requests (deleted by the user)   + Delete all the search results data for this search id. (permanent deletion from search results) |
| Schedule | * Every 10 mins |

## Cron to change the status of the search request to deactivate that has expired.

|  |  |
| --- | --- |
| Cron | * Get all the expired search requests (current date > expiry date)   + Update the status of the search request to inactive. |
| Schedule | * Once a day. |

Creation of index from the mongo shell

Created index for the first time when you create search results table:

* Create SearchResults Collection : db.createCollection(''SearchResults'');
* Create Index for search ID: db.SearchResults.createIndex( { SearchID: 1 } );

# Phase II Wishlist

1. Include Jewelry as the 5th category.
2. Evaluate paytm down the line.
3. Suggest products if search results are not found: if search result not found, then show him other closest search. E.g.: Price – easy one to do. Adv. Algo. for suggesting brands will be done down the line. This link may be helpful: <http://docs.aws.amazon.com/AWSECommerceService/latest/DG/SimilarityLookup.html>
4. Auto log-out feature – after every 15 days. Priority: Low.
5. Check real time availability for products
6. Look up same product by UPC code to identify if it is the same product on each of the vendors. We can then club that as a single product.
7. Once the user registers logs in on a device, a registration on GCM for his device is done. If the same user using the same login details would login from another device then his GCM id is different. The notifications will only be sent to the latest device from which the user is logged in as that device’s GCM id is stored with us. In such cases, usually when a user logs in from another device, we warn him that he is switching his device and henceforth all push notifications will be received on the new device and not the old one.
8. Implement the count of new search results on the App icon badge.
9. Apart from push notifications on new search results also send email notifications – send only top 10 results. Consolidated email for all active searches.
10. Currently we have identified and documented the workflows of only the search algorithm using the APIs available. No machine learning aspects have been considered. As discussed, this will be a part of the phase II.