Technology Review
UniversityGear

November 13, 2016

Devin Foulger
Hector Trujillo
Bryan Liauw

✦

Contents

1   Introduction
2   Technologies

2.1   Technology: Checkout

Checking out is a feature that is essential to any application or website where you purchase some item. This means that our application will need to allow the user to check out the items in some fashion.

2.1.1   Options

Option 1: Checkout with sign-in

Allowing the user to sign in to checkout items gives us the benefit of allowing members to purchase from our application. In order to implement this feature, we would have to use eBay's Order API. This API would allow for the user to purchase items using their existing eBay and PayPal accounts. This also allows the user to view the shipping address provided and any fulfillment information.

Option 2: Checkout as guest user

Allowing the user to checkout items through a guest check out would allow any user to purchase from our application. In order to implement guest checkout, we would need to use eBay's Order API with GUEST CHECKOUT SESSION. This is different than using the Order API normally. With guest checkout, the user is able to see that item that they have purchased and the status of the payment and order.

Option 3: Checkout is either guest user or with sign-in

Allowing the user with the option to either sign-in or use guest checkout gives us the option to offer which ever method makes the user most comfortable when checking out. This would be that we would have to use eBay's Order API to the fullest by implementing both sign-in and guest checkout. This means that the user would be able to use PayPal, if they have an account, or checkout via credit card.

2.1.2   Use in design

This is a fundamental use in our design. Without this feature, or application would simply be a searching application. The user would only be able to search for specific items that exist within eBay's stores, but not actually purchase them.

2.1.3   Cost, availability, speed, security

Option 1: Checkout with sign-in

The cost of checkout with sign-in is the development time it would take to implement. This feature is tricky and would complicates the use of eBay's APIs. However, the availability is limited to those that have an eBay account. The speed is also largely dependent on internet connectivity and the status of eBay's servers. Finally, the security is very tight. This is because the user must authenticate before making a purchase. On top of that, all information that is used is sent via HTTPS.

Option 2: Checkout as guest user

The cost of guest checkout is relatively low. It would take the least amount of development time and resources. The availability is also very high allowing us to reach anyone who would like to use our app. Again, the speed is dependent on internet connectivity and the status of eBay's servers. However, the security would be great. None of the user's information should be saved to the device. That includes things like credit card information and shipping address. Also, any information that is used is sent via HTTPS requests.

Option 3: Checkout is either guest user or with sign-in

The cost of the guest checkout coupled with sign-in is very high. It would require a significant amount of development time to complete. However, the availability would be at its greatest because it would allow for any and all users to purchase items. The speed of the checkout would, again, depend on the status of eBay's servers and internet connectivity. The security is the same as the above options.

TABLE 1

Comparison Table of Options for Checkout

	Time Cost	Availability	Speed	Security
Option 1	High	To only members of eBay	Fast	Best, because it offers an extra layer of authentication
Option 2	Low	To everyone as a guest	Best	Great
Option 3	Very high	To members of eBay and guests	This could either be best or fast	This could be best or great

2.1.4   Evaluation

Each option has their advantages and disadvantages. If we were to allow users to checkout only using sign-in, then we might only be targeting a smaller audience. The complexity for implementing such a feature is much harder than that of guest checkout as well. Using guest checkout would allow for all users to use our application with no complicated sign in. It would also mean that users would not have to create an account to user our application. If we were to implement both sign-in and guest checkout, the complexity would be too great. Given the small amount of development time it might be hard to implement both features. However,this option would give us the ability to reach every user possible.

2.1.5   Best Choice

The best choice for our application is implementing a guest checkout. This is because with a short development time, guest checkout would be the fastest to implement. This would also allow any user to purchase from our application, which means we would not be alienating someone who does not have an eBay account.

2.2   Technology: Checkout UI

Checking out needs to have a UI that is easy to use. If we only stub out a UI, it might be complicated, or worse, ugly. The UI needs to be attractive and functional.

2.2.1   Options

Option 1: Layout Editor (Native Android Studio)

The native android layout editor doesn't work very well for prototyping. However, it allows for easy UI creation while being able to map the buttons to correct functions. This tool doesn't offer much in terms of quick design, but it offers a lot of functionality. It uses XML files to create the UI. It does allow you to alter any aspect of your UI, which is very good as well. It also allows you to create your own UI objects very easily.

Option 2: Indigo Studio

Indigo Studio is a wire framing tool. With this tool I would be able to create great looking wireframes. This doesn't allow for much in terms of actual development though. It is just a tool for quick prototyping and design. However, it does offer a lot in terms of customization and they also offer a lot of images for buttons, list views, and things of that sort.

Option 3: Just In Mind

Just In Mind is a comprehensive prototyping tool for many types of development, including Android development. This tool offers a lot in terms of development. There are many UI libraries for Android. This tool would also allow me to use templates that they have already built. On top of that, any of the designs or UIs I make can be transferred into already existing projects. This tool also offers integration with other tools as well, like Photoshop. It also features interactive images and animations for use in your projects.

2.2.2   Use in design

Having a functional and beautiful looking UI keeps users attracted to your app. So, having a nice UI is a must in this project. The user will also constantly be interacting with the application.

2.2.3   Cost

These tools are simply for creating clean and responsive UIs. That being said, the native Android Studio Layout Editor is free. Indigo Studio costs 25 dollars a month and Just In Mind costs 19 dollars a month.

TABLE 2

Comparison Table of Options for Checkout UI

	Cost	Learning Curve
Option 1	Free	There is somewhat of a learning curve as it requires you to also develop at the same time
Option 2	Free with limited features, then 25 dollars a month	Low
Option 3	Free with limited features, then 19 dollars a month	Low

2.2.4 Evaluation

Each tool has clear disadvantages and advantages when compared to each other. Just In Mind and Indigo Studio have relatively similar functions. However, Just In Mind offers a bit more such as UI libraries that are compatible with Android Studio. Indigo Studio does allow for faster creation of wireframes when compared to Just In Mind. These two tools can be much better than the native layout editor in Android Studio. The editor layout also offers its own features as well. It gives developers the ability to customizing anything they like about their UI.

2.2.5 Best Choice

The best choice for this project would be to use the native Android Studio Layout Editor. Although the other tools offer a lot more in terms of prototyping, they have a cost. If you don't pay for the tools, you don't get everything that is included. However, Android Studio does offer the ultimate tool for any type of customization you would like to make. You have full control over what you are trying to do. You also don't have to worry about integration with other tools, because it is built in.

2.3 Technology: Getting data for Single Item View

The data that is received from eBay's APIs comes in the form of an JSON file. This JSON data needs to be parsed correctly so that the information is portrayed to the user in a meaningful manner. Each item needs to correctly be displayed to the user so that they can view the description or images of a particular item.

2.3.1 Options

Option 1:

GSON allows for Java objects to be converted into JSON objects, and vice versa. This is a tool that is not native to Android Studio, but offers a lot in terms of use. It has many methods that allow for array creation which holds JSON information. These arrays can then be used to fill the UI with relevant information for the user to view.

Option 2:

JsonReader is a light weight API that would allow me to easily read JSON files. It allows me to create JSON objects and arrays. The arrays would hold information that could easily be presented to the user.

Option 3:

JSON Parser is native to Android Studio and is very comprehensive. It would allow me to create objects using JSON files. I would have the ability to create JSON arrays, object, and key-value pairs using JSON Parser. It includes many methods for easy creation of JSON objects and parsing.

2.3.2   Use in design

A JSON parser is absolutely a must if we want to make sense of our data. This is because eBay's APIs return the data in JSON format. In order to display the information correctly to a user, we have to use a JSON parser.

2.3.3   Cost and speed

All three APIs do the same thing in the same fashion. They all create readable JSON arrays that hold information to be used by the developer.

2.3.4   Comparisons

All three APIs do the same thing in the same fashion. They all create readable JSON arrays that hold information to be used by the developer.

2.3.5   Evaluation

It is important to note that theses APIs perform almost identically. On top of that, they all produce the same output. There is no clear advantage or disadvantages over the others. The only disadvantage to the JsonReader, is that you don't have the ability to create new JSON objects.

2.3.6   Best Choice

The best choice to use for our application would be the built in Android parser, JSON Parser. This is because all the parsers perform similarly, but with the built in parser, I get to avoid the headache of importing a new one. It also has the ability to create new JSON objects if that is needed.

2.4   Task:Correcting User's Search Keyword

Sometimes, when a user wants to find an item, he/she will mistype what they want to search. This is a more prevalent issue in mobile apps since the keyboard input is much smaller in smart phones. If we do not address this issue, the return value from the search will not be optimal. In order for our app to understand what the user wants to find despite the misspelled keyword, we need a fuzzy search algorithm. There is 3 fuzzy search algorithms that comes into consideration: Levenshtein Distance, Hamming Distance and Metaphone.

2.4.1   Option 1: Levenshtein Distance

Levenshtein Distance calculates the similarity of two strings and assign a numerical value that determine how different the two strings are. This algorithm, given a string s and a target string t, will return the number of alterations to s so that it will be the same as t. The algorithm will construct a matrix m which has the size m[s.stringlength][t.stringlength]. The matrix will be built on the minimum from this rule:

```
1. If s[n] == t[n] where n is != t.stringlength

        m[n][n] = 0

        else if s[n] != t[n]

            m[n][n] = 1


2. m[a][b] where a < s.stringlength and b < t.stringlength

        =  m[a-1][b]+1 or m[a][b-1]+1
whichever is smaller


3. m[a][b] where a < s.stringlength and b < t.stringlength

        =  m[a-1][b-1]+1
```

2.4.2   Option 2: Hamming Distance

Hamming Distance is another algorithm that compares the difference between two strings and assign a numerical value on how close one string is from another. However, unlike Levenshtein Distance, the Hamming assigns value similar to a binary. This algorithm will create an array of integer with the size of the length of the string. This arrays value at a given position n is 1, if the character of the first string and the second string at n is the same, or 0 otherwise. The arrays value is totaled up and it will give a value on how different one string is from another.

2.4.3   Option 3: Metaphone

Metaphone, on the other hand, does not assign a numerical value to check the difference. It mainly checks if some character is substituted with its homophone. For example, the algorithm will consider K and C similar and so is kall and call.

2.4.4   Criteria of Evaluation

The main criteria that we evaluate is the reliability of the algorithm to find what the user might mean when they accidentally mistyped a word. Secondary criterias include maintainability and speed.

TABLE 3

Comparison Table for the Fuzzy Search Algorithms

	Reliability	**Maintainability**	**Speed**
Option 1	Applicable to most cases	Easy to maintain	Slower
Option 2	Applicable to most cases except non-equal string	Easy to maintain	Fast
Option 3	Not applicable in most cases	Harder to maintain	Fast

2.4.5   Discussion

For our application, Levenshtein Distance is the best option. Metaphone does not take into account if users input is not homophone to what the user wants to find. This might be a problem if user is searching for certain types of brand. Also, adding language support in the future means that we need to change some parts of the code to cater to the new language

since there will be different pronunciation. We will need to change how our algorithm detect homophone if it happens and it will take unnecessary time. On the other hand, the Hamming Distance is a faster algorithm than Levenshtein since we only need to construct one-dimensional array as compared to Levenshteins two. But, Hamming Distance cannot give an accurate number if the length of string is not equal. This can happen in the application if the user accidentally sends in an incomplete keyword or simply do not know the exact keyword he/she is looking for. Despite the trade-off in speed, we believe that Levenshteins reliability is more suitable for our application in order for user to have an optimal session.

2.5   Task: Search Item in Ebay's Database

User will be able to find certain items in the application. The user will be able to enter keyword and the app will return items that matches, or approximately matches, users query. This search result will be used alongside filters in order to get the exact item that user wants. The 3 ways we can go around this search is: Binary Search, Linear Search, Ebays Finding API

2.5.1   Option 1: Binary Search

Binary search involves dividing the array into multiple halves until a match is found. If the item to be searched is larger than the middle value, then the top half will be the half to be searched. Else, the lower half will be searched. This search algorithms complexity is log n since the array size is halved every iteration. It is considerably fast, especially when used in a larger database. However, the main drawback is that the list must be sorted for this to work. Without a sorted list, dividing the halves would not help finding the item since the top half and lower half might not be larger or smaller respectively.

2.5.2   Option 2: Linear Search

Linear Search is one of the most basic search. This involves checking every item in the list and find if it matches the item. This algorithms complexity is n since it will look for every item. Considering the complexity, the speed of the algorithm is slow especially in larger files. While smaller database would mean unnoticeable difference in speed, it is very unlikely that our application will utilize small database. The advantage of linear search is that there is no need to manipulate the list of data we are looking in. Since the search will iterate over every single item and match it with the keyword, there is no need to sort like binary search. Therefore, despite the trade-off in speed, linear search makes it up with reliability.

2.5.3   Option 3: Ebay Finding API

Ebay has a released API that search their database. It has a lot of function that helps finding based on the query. Furthermore, it has several functions that helps narrow down users selection through filters. Despite not being transparent in how their search works, Ebay does widen the way a query can be searched, such as by product or by keywords. As a bonus. the Ebay API helps in manipulating the way search results are returned. Since we are going to use EBays sell and buy API, this API will integrate well with other APIs.

2.5.4   Criteria of Evaluation

The main criteria are speed and reliability. Some other consideration would be integrating the result with other functions later.

TABLE 4

Comparison Table for the Search Algorithms

	Reliability	Speed
Option 1	Low	Fast
Option 2	High	Slow
Option 3	High	Fast(?)

2.5.5 Discussion

We believe that the best approach would be using the Finding API. Since we are using Ebays database, we have no guarantee that it would be sorted, therefore making binary search a risk. On the other hand, if it is sorted, using a linear search would make it run slower. We believe that the Finding API would be optimized by Ebay and will find the item fast and accurately. Furthermore, giving a lot of options such as filtering will help us in the future. The integration with other API that we will use is also a convenience that we should take advantage of.

2.6 Task: Saving Search for Future Purposes

Our application will store users search history internally in their phone. This is so that we can give an accurate recommendation of items based on users past search history. The search history would be saved in a .txt format in the application folder. There is two ways we can go around this: we can save the item id and search it in the database in order to understand what the users preferred item is or we can save the keywords and determine users preference through those keywords. There is 3 ways we can save this txt file: Internal storage, External Storage, or Server

2.6.1 Option 1: Internal

Internal storage means that the application would save users preference inside their mobile phone. This means that the user has total security from other source. Android makes sure that the file saved internally is, under normal circumstances, only accessible by the application itself. Furthermore, since the files are saved internally, the applications read and write speed is faster by a certain margin; most likely to be noticeable in lower-end hardware. However, the internal storage of most Android smart phone is rather limited compared to external ones, therefore, saving a huge file might frustrate the user.

2.6.2 Option 2: External

Relying on external Storage using SD card or other similar hardware means we are going to operate under the assumption that the user has such external storage. While it is not an uncommon situation, some user might neglect having an external storage, so it is not quite ideal. However, using external storage means high load of data is possible. We would not worry about utilizing too much users data. Although its read and write speed will be slower, but having an external storage means that it is possible to offload some processes to the memory card instead. However, security can be easily compromised if the external storage is detached and lost. The data can easily be retrieved once someone manage to get into the external storage.

2.6.3   Option 3: Server

Using server will require internet connection. Given our application rely heavily on internet, we can safely assume that user will have some kind of connection that allows us to utilize an online database. By saving it into the server we do not need to worry about the data load, since it is going to be handled elsewhere. The trade-off would be the speed to process things, which is highly dependent on the internet connection the user has. This can range from being very fast to very slow depending on where the user is geographically.

2.6.4   Criteria of Evaluation

The speed on which the file can be taken would be very important, because we might need it for different purposes. If the speed is too slow, user might get the feeling that the application is not responding fast enough, which is something we must avoid. Secondary criteria would be security and storage space.

TABLE 5

Comparison Table for the Storage Medium

	Security	Speed	Space
Option 1	Very Secure	Depends on the phone	Limited
Option 2	Not secure	Depends on the phone but slower than internal Storage	Depends on the hardware but generally larger than internal storage
Option 3	Mostly very secure	Depends on the phone and internet connection	Very huge but comes with a cost.

2.6.5   Discussion

Our application will use the internal storage with the option to allow user to use external storage. We believe that the speed on which the information is retrieved is very important. The data we are going to put in is not going to be very large; if the data gets too large, we will have to narrow it down by removing past searches to indicate the users changed preferences. Using a server database online would add a different layer of complication and unnecessary because the data would be small in size. Furthermore, we wont have any other operation on the data which makes it unnecessary for it to be saved on a server. We respect the users and their security, which is why we would try to use the most secure method possible. However, if the user does not have enough space internally, we would give an option to trade that security and speed for more space.

2.7   Optional Task: Machine Learning to Study User's Preference

Once we have users history of searches, our application should make use of that by constructing users preferences based on the information it stores. Additionally, we can also use data that might be provided from the Ebay server to add more data in order to get higher accuracy. This preference will be used for multiple purposes, which can range from showing items on the home screen of the application, to suggesting the user what to buy at the buying page. We construct this preference using machine learning. The 3 approach we can use to determine users preference is Bayesian Estimate, Apriori Algorithm, FP-Growth

2.7.1   Option 1: Bayesian

Bayesian algorithm involves comparing whether one thing is going to happen following a certain event or if another event is likelier. This is calculated by learning the pattern to what is considered as true or false. In terms of this application, true would be users preference. The algorithm counts the combination of prior events and its results and construct a possibility based on a given event. Given high possibility, we can assume that the user will want that item. This operation, however, might be time-consuming because there is a lot of mathematical calculations as well as receiving data.

2.7.2   Option 2: Apriori

Apriori algorithm calculates the most probable item or combination of items. The algorithm involves counting the number of occurrences where each item is selected in the data. Then, the number below a given threshold, which is usually half of the number of data, is removed. Then each item is then paired together and we count the number of occurrence where the pair is selected. Again, the number below the threshold is removed. This keeps repeating until we decide to stop and we will get the most probable item that user might want. Considering this, the running time might be long because we need to iterate through the data every time we want a larger size. Since we might also need a temporary storage for the lists, memory consumption might cause lower-end smartphones to lag, something we want to avoid the user from experiencing.

2.7.3   Option 3:FP-Growth

Similar to Apriori algorithm, the algorithm will count the number of occurrences for each item. However, instead of iterating and growing the items, FP-Growth will create a tree based on the data given. Each item will be represented by a node. The node will have a value on it which depends on how many occurrences. The node is going to be connected based on the data; so given data set 1,2,3 and 1,5,6, the tree will branch out at item 1 and the value of the node at item 1 will be 2. The route with the highest value will be the most probable outcome. This algorithm is significantly faster than Apriori although it will not create as much possibilities.

2.7.4   Criteria of Evaluation

The speed and the accuracy of the algorithm. Since slow algorithm will look unresponsive, we would very much like to avoid that situation.

TABLE 6

Comparison Table for the Machine Learning Algorithms

	Accuracy	Speed
Option 1	Accurate and can give lots of probable items	Slow, since it might need to run through prior data several times
Option 2	Accurate and can give a lot of probable items	Slow, exponential depending on how many iteration is needed
Option 3	Accurate but limited items	Fast, only requires at most 2 passes

2.7.5 Discussion

We believe that FP-Growth might be the best choice. We value speed in our application. User will feel that the app has too many loading times or unresponsive if the algorithm takes much time. Considering the file might be larger as user uses more of it, it is not recommended to use exponentially long algorithm.

3   Conclusion