Design Document
UniversityGear

February 13, 2017

Devin Foulger
Hector Trujillo
Bryan Liauw

Abstract

This design document will outline and describe the steps we will take to develop our application ”UniversityGear.” It will talk about the purpose of this document as well as many of the different viewpoints. It will also define the classes and methods that we will use in our project.

Luther Boorn	Devin Foulger
Signature	*Signature*
Date	*Date*
Hector Trujillo	Brian Liauw
Signature	*Signature*
Date	*Date*

Contents

1 Introduction

1.1 Scope

The application that is going to be developed will be for Android devices, specifically for N and M OS. It will allow users to purchase college merchandise via eBay's large collection of items. The user will be able to search for items from different colleges or by specific search criteria. The project will begin development in December, 2016 and end in May, 2017.

1.2 Purpose

The purpose of this design document is to outline how the application will be structured to satisfy the requirements we have set. It will go into depth about the design details of our software.

1.3 Intended Audience

The intended audience is for the team ”4Credit” and their client. It is also for the professors of the senior capstone class at Oregon State University.

2 Definitions

TABLE 1
Definitions

Term	Definition
API	Application Programming Interface, allows developers to develop applications that connect to eBay's large inventory of items.
Vendors	Individuals or companies that sell or will sell items on eBay
Filters	Search criteria that will allow the user to more easily find the item that they are looking for.
Third-Party Developers	Individuals outside of eBay that will be using eBay's public APIs to create their own applications.
Application	An Android application that uses eBay's public APIs.
College Merchandise	Items such as school supplies for Higher Ed, fan gear, etc...
User	Any one who will be using the Android Application utilizing eBay's APIs.
Client	eBay
BIN	Buy It Now
N	Android's new Nougat operating system
M	Android's Marshmallow operating system
MVC	A design pattern called Model View Controller

3 Conceptual model for software design descriptions

The basic concepts and context of the design document will be explained in this section of the document.

3.1 Software design in context

An object oriented approach will be taken and we will be using a specific design pattern called MVC. This will allow us to maintain some form of a separation of concerns. In turn, it will make development much smoother. We will also have to keep track of an API layer, as we will be using eBay's new APIs. It is imperative that our application also works on the M and N Android operating systems.

3.2 Software design descriptions within the life cycle

3.2.1 Influences on design document preparation

The software requirements specification (SRS) plays a large role in influencing the design document. This is because the SRS defines all of the functional and interface requirements for the successful completion of the project.

3.2.2 Design verification and design role in validation

Test cases will be developed simultaneously to development phase. This will allow us to validate that our application is successfully working as expected. The last step of validation we will complete if our application successfully met the requirements defined in the SRS.

4 Design description information content

This section of the design document will identify how the application will be implemented and designed.

4.1 Design document identification

4.2 Design stakeholders and their concerns

The stakeholders of the project our the developers and their clients. The biggest concern is that the product will meet all of the requirements specified in the SRS. The application must also contain design that is described in this document.

4.3 Design views

We will be using different processes for representing diagrams. We will be using UML for describing our functions and classes.

4.4 Design viewpoints

The design viewpoints will be described with a combination of UML diagrams and short descriptions. The descriptions could include information regarding the context, composition, logical, or interactive viewpoints. Each viewpoint will also have a specific name.

4.5 Design elements

There are a lot of design attributes we will be discussing in our document. Our design entities will consist of the APIs and classes that we will be using to accomplish our goals. Each entity will have a name, a type, and a purpose. The name attribute is simply the name of the element or entity. The type attributes will consist of what kind of class or API we will be using. The purpose attribute is to give a description as to why an element might exist. Last, an author attribute will be used to identify which designer will be responsible for the element.

4.6 Design rationale

It is important that our application performs well on the N and M Android operating systems. We will make sure that the application is also easily maintainable for future developers. We will also provide documentation of the design process so that other developers will understand the structure of the application.

5 Design viewpoints

The main design viewpoints will be explained here. The exact viewpoints are the context, composition, logical, dependency, interfaction, and algorithm viewpoints.

5.1 Context viewpoint

The context viewpoint will document the functionality between the user and the application. There are two main functionalities the user should be using: searching and purchasing merchandise. This sections serves to define the user interaction with the app. It will also talk about the sub functions that will be used to facilitate the main functions.

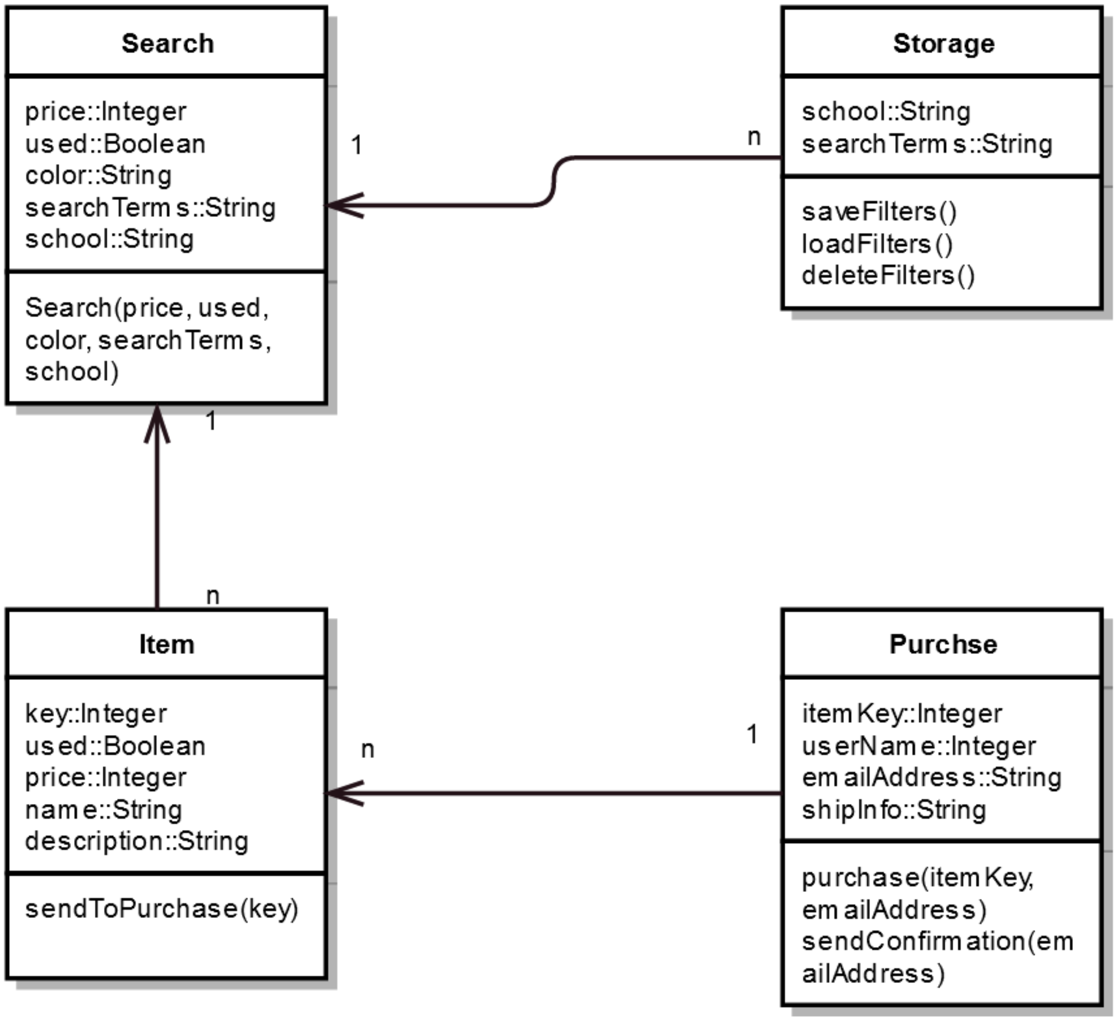
The user of the application will also have to perform other smaller actions in order to get the full use. For example, they will have to download the application from the Google Play store. From there, they would be required to search for something in order to purchase it. They will be searching for specific schools, prices, condition of item, and other search criteria. They will then be presented with a list of items in which they will be able to select and view that item in greater detail. They will also be able to purchase the item and then they will need to enter in their name and shipping information.

When viewing an item in greater detail, the user will be given the opportunity to purchase the item. They will have to provide the application with some user information, such as a credit card number and shipping address. The user will also have to provide other information such as the school they are searching for, what item they are looking, the price of the item, condition of the item, and other filers of that sort.

5.2 Composition viewpoint

The application will be divided into three different parts. The first being the search feature, which will allow the user to find a specific item. The second will be the purchasing feature, which will allow the user to purchase any item they have found. Last, the user should be able to view in detail everything about the item they have searched for. All these pieces need to be connected together and coherently function as one piece.

Fig. 1. UML Diagram for the for main classes



Function attribute

The search class will allow users to search for their items. This class will need several variables, which will be passed into a function that calls an API. The output will be a list of items that are related to the search criteria provided by the user. The author of the search class will be Bryan Liauw. The storage class will allow the user to save their search filers for later use so they don't have to keep entering the same data each time. This requires the same information provided to the search. From there, the class will either save, load, or delete this data. Bryan will also be the author of the storage class. The item class will be used to hold and send the specific item data to the purchase class. Devin Foulger will be the author of the item class. The purchase class will allow the user to purchase the item that they have selected. It will also send a confirmation if the user's purchase has been completed. Devin will also be the author of the item class.

Subordinates

The search class will be the default class that will control the application. This is because the storage and item classes will depend on the search class as you need to search for something in order to save the information and create items. The purchase class will depend upon the item class. This is because if are no items, then there will be nothing to purchase. In order to purchase something, an item must exist.

5.3 Logical viewpoint

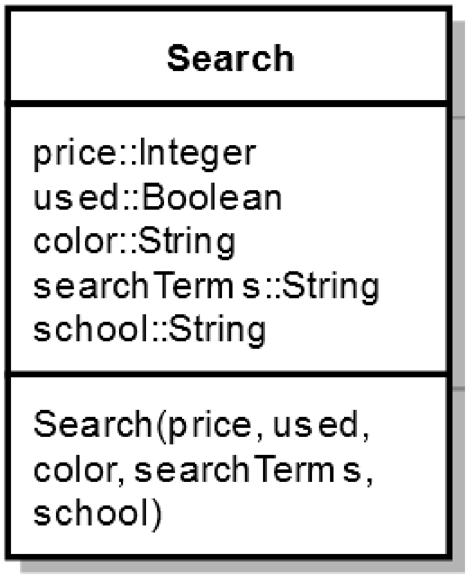
Our application will consist of many classes, but they will either be a model or a controller. These classes will be the Search class, Purchase class, Item class, and the Storage class.

The search class

This class will allow users to search for items that they would like to view or purchase. It will share a relationship with the item and storage classes. It will have a method for searching that needs the search terms that have been provided by the user. The important thing to notice, is the variables provided by the user. These variables are the ”price”, which is a integer. The second is the condition of the item, which is called ”used” and it is a boolean variable. The third variable is the color of the item and is named ”color”, which it is a string value. ”searchTerms” is another variable of type string, and is used to hold the search values the user provides. The last variable is the ”school” and is a string. The search method for this class also heavily relies on an API called ”Browse” which is provided by eBay.

The search class will now encompass the storage class.

Fig. 2. UML for the search class

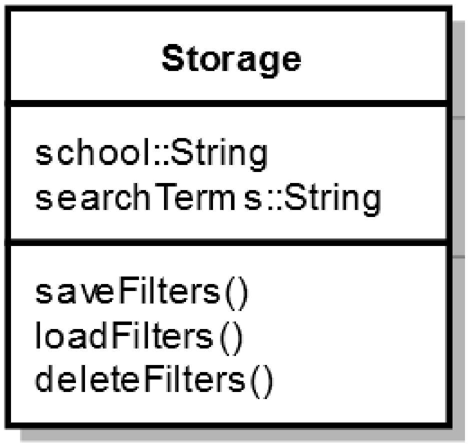


The storage class

This class will be used to save the users search criteria. This is so the user will not have to enter in the same information again, considering that they will most likely search for the same schools over and over again. It will have the ability to save, load, and delete filters. It will also need the information provided by the user such as the school and searchTerms. Those variables will be provided by the search class.

The storage class will no longer be used. This is because the methods involved are small enough that they can be condensed into the search class. The only place that these methods would be used is in the search class and because of this, we decided to condense the two.

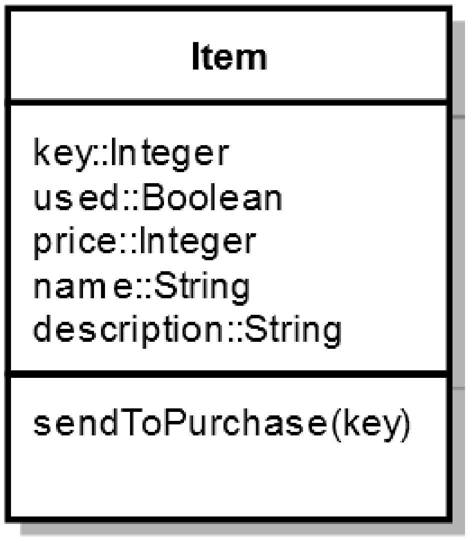
Fig. 3. UML for the storage class



The item class

The item class will hold all the information on a single item. The items will be displayed to the user in either a list view or a single item view. The information will also be sent to the purchase class so that the user may be able to purchase the item. This class heavily relies on what is returned by the search class and the Browse API. Every item will have a unique ”key”, that is an integer value. This variable exists so that every item that is in eBay’s database has a unique identifier. The item will also contain a ”used” variable that will be a boolean to determine if the item is used. The ”price” will be an integer variable that determines how expensive the item is. It will also have a ”name” and a ”description” variables, both will be strings. The ”sendToPurchase” method will allow users to purchase the item that they have chosen. This means that the key will be needed for this transaction to take place. The key will identify which item is being purchased. The item class will also need to use the Android JSON parser. This is because the returned value from the search from eBay’s Order API is a JSON file. The JSON parser will need to be able to parse the file for the correct information.

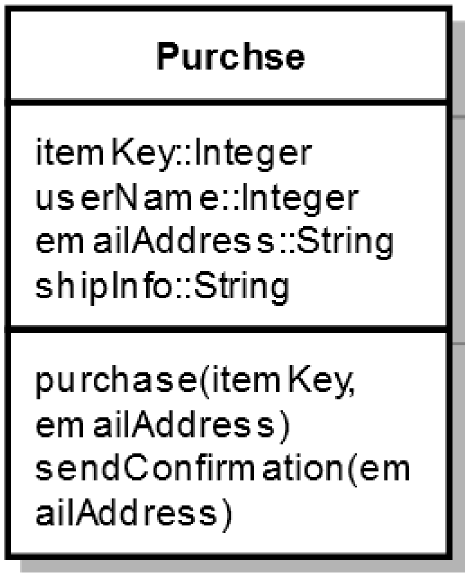
Fig. 4. UML for the item class



The purchase class

The purchase class will allow the user to purchase the item that they have selected. It will need the item key when purchasing an item. It will also notify the user in some way that they know the purchase has been completed. The first variable that is needed is the ”itemKey”, which is a unique key that identifies which item is being purchased. The next variable, is the ”userName”, which will hold the users first and last name. The third variable is a string called ”emailAddress” that will hold the user provided email address. The last variable, ”shipInfo” will hold the users shipping address. The ”purchase” method will require the use of eBay’s ”Order” API. The purchase method will also need the email address.

Fig. 5. UML for the purchase class



5.4 Dependency viewpoint

Each class is somewhat dependent on the search class, as it is the basis of where everything starts. Item and storage can only exist after something has been searched for. The purchase class can only exist if an item exists.

Search class dependencies

The search class does not depend on any other classes in order to function. However, it does depend on eBay's Browse API. This API is what will give us the ability to search for items.

Storage class dependencies

The storage class depends on the search class. It also depends on what information has been provided by the user. The purpose that the storage class depends on the search class, is because it needs to save specific information such as the school and search terms. The search class also contains the information that has been provided by the user, which the storage class will need.

Item class dependencies

The item class depends on the storage class. This is because it will need the information that is returned from the item search. The purpose that this class depends on search is because the item will need the information that is returned from the search method in the search class. It also depends on the Android JSON parser. This is because the returned value from the search method will be in a JSON format. The JSON parser API will need to correctly return the required information for the item class.

Purchase class dependencies

The purchase class depends on the item class. The class will need the information that is in the item class, like the item key. The purpose that this class depends on the item class is because purchasing needs the item key. The purchase class will also depend on eBay's Oder API. EBay's API will allow the class to make its purchase. The API will also need the item key.

5.5 Interaction viewpoint

Home Page

The home page will be a simple design that will include a search bar labeled 'searchText' and a button 'searchButton' for the user to enter text and submit their search. When the user presses the search button, the text within the search text field will be passed as a string parameter to the search method. The user will not be allowed to press the search button without first inputting text into the search field. To submit a search from the home page, the user will also be allowed to use the search button on the on-screen keyboard.

Multi List View

This module will proceed the home page, after the user searches for an item. After the search module completes its search, it will return a list of data structure that contains the search results. The pertinent information in the search results will consist of an image of the item for sale, the item name or title, and the item price. The DisplayResults module will then take these results and present them to the user in a list view.

The image that will be displayed in the list view will be a smaller version of the main image. This will be done by the service that will be used by the search module. The image size reduction will be necessary to reduce loading times and thus performance impacts.

In this same multi list view module, there will be a filter component that will allow the users to filter the search results. The users will be able to filter the results based on predefined criteria. The currently defined search filters include price (Integer), item category (String), or item condition (new or used) (Boolean). These details regarding an item will be part of the data that is returned by the search module but not displayed to the user until they enter the single item view.

No Search Results

If after the user submits their search, or modifies their search criteria to the point that there are no search results to display, this module will present a message stating that there are no search results and to try a different search, or to modify their search criteria. This module will be based off the size of the list returned by search.

5.6 Algorithm viewpoint

The algorithms viewpoints will describe the different algorithms that we will be using to correct users search mistakes. The Levenshtein Distance algorithm that we are going to use is going to be attached to the search bar that is in most pages of the UI. The algorithm will run every time the users finish typing and submitting their search. The algorithm will run regardless whether the keyword needs to be corrected or not. The fix will happen if the user has written something that is not quite correct; that is, the distance between the intended string and the string that user has written does not have a 0 value. Once this case happens, we will find the keyword with the minimum distance from the string user has written and replace user's keyword with that.

Design concerns

There is a huge concern in terms of resources needed. Since the algorithm is going to compare keywords from the user to keywords in the database, this means that the algorithm will run once for every words in the database. Furthermore, the algorithm itself has a complexity of O n-squared, since the number of loops is the string size of one string times the string size of the other. This means that the total complexity would be O n-cubed. One way to get around this is to find a threshold. Once this threshold is reached, the word is considered similar enough. This can reduce a lot of processing time.

Processing attribute

This algorithm will be part of the search method inside the search class. Because of this, it will affect only the variables within the class. Since we are correcting the users input string, we will use the algorithm to manipulate the variable color, school and searchTerms. The fixing price and used variable is unnecessary, because it does not need to be fixed. The only thing we could do to price is find the approximate integer to the user's input, while the used variable should not be modified. The algorithm will find the most similar string based on the differences in the character in each position. This is done by constructing a two-by-two matrix and then filling it with an integer based on a certain rule. This rule will measure the differences between the characters in each string and assign an integer to determine how different it is. As previously mentioned, the loop happens every time user enters a keyword. Once the keyword is entered, the algorithm will run until it reaches a string distance that is below or equal to the threshold.

Examples

```
LevenshteinDis(string1, string2){
        Matrix[string1.length][string2.length]
        for(a = 0 to string1.length){
                for(b = 0 to string2.length){
                        if a==b{
                                if string1 at a = string2 at b
                                   Matrix[a][b] = 1
                                else
                                   Matrix [a][b] = 0
                        }
                        else
                           min(Matrix[a-1][b]+1, Matrix[a][b-1]+1, Matrix[a-1][b-1]+1)
                }
        }
}
```

TABLE 2
Revisions Log

Name	**Revision**
Devin Foulger	Added a description as to why we will no longer be using the storage class