Mobile Application Store Product API Design Document

Date: October 8, 2013

Author: Fanxing Meng

Reviewer(s): Weiye Wang

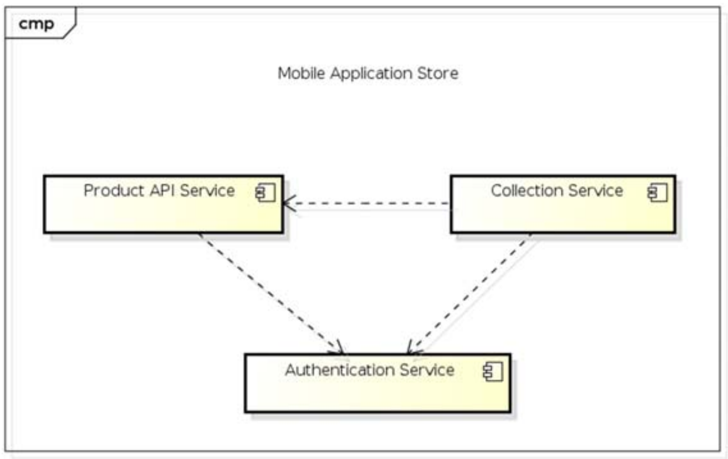
Introduction

This document defines the design for the Mobile Application Store Product API. Mobile application stores provide contents including applications, ringtones and wallpaper for mobile device users. A couple of factors contribute to categorizing and dispatching contents due to potential country-specific export controls, localization issues, and device compatibility.

Overview

The product catalog portion of the app store is being developed in this phase. The API service of products is responsible for management of the product inventory, including the ability to search and page through the contents of the product catalog. This part would constitute with collection service and authentication service to maximize the managing capabilities of the store.

Application developers have to submit meta-data about their products to the store's catalog for customer to search and view. Such data would include the product name, description, type, thumbnail, price, etc. With a well-populated mobile application store, developers benefit from better exposure and customer focus as well as feedback for sustainable development.



Graph 1. Component diagram showing how the product API service portion would fit into the complete store implementation, with dependency shown among various services.

Requirements

Three primary functionalities are to be supported concerning data item entry/import:

1. Adding countries

   Country information contains a 2 letter code, country name, and export status (open or closed). This should be a restricted function for which an access token should be provided for validation.
2. Adding devices

   Device information contains device id, name, and manufacturer. This function is also password-protected.
3. Adding products

   When product information is entered, all required fields must be verified. This includes content name, description, author, rating (0 to 5), categories (0 or more), countries downloadable to, compatible devices, price (float point number), languages supported (5 letter code), an image URL. Content type must also be declared: application, ringtone, or wallpaper. For an application, its size in bytes should also be provided. This interface is also restricted.

One final function, which is unrestricted, is provided for any user to search for contents using criteria including category, text search in name and description field, minimum rating, maximum price, language, country, device id, and content type. Any of the search fields may be left empty.

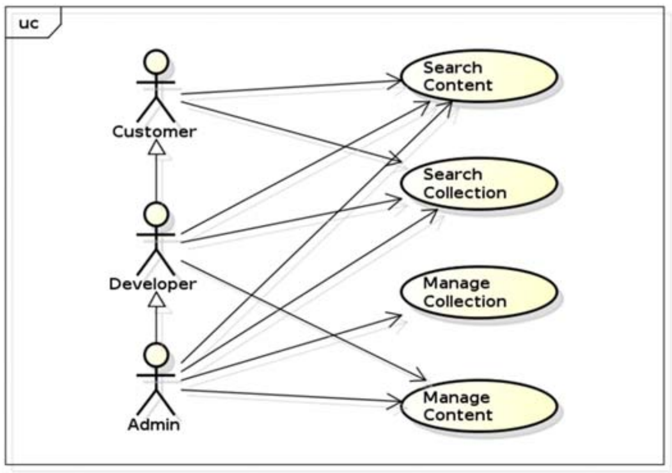
Use Cases

Current design supports the following use cases:

Customers can use the service provided to search for content meta-data and potentially download the product or collections.

Administrators are capable of managing the store in forms of search for contents and collections and update any entries from the country list, device list, or product catalog, with password.

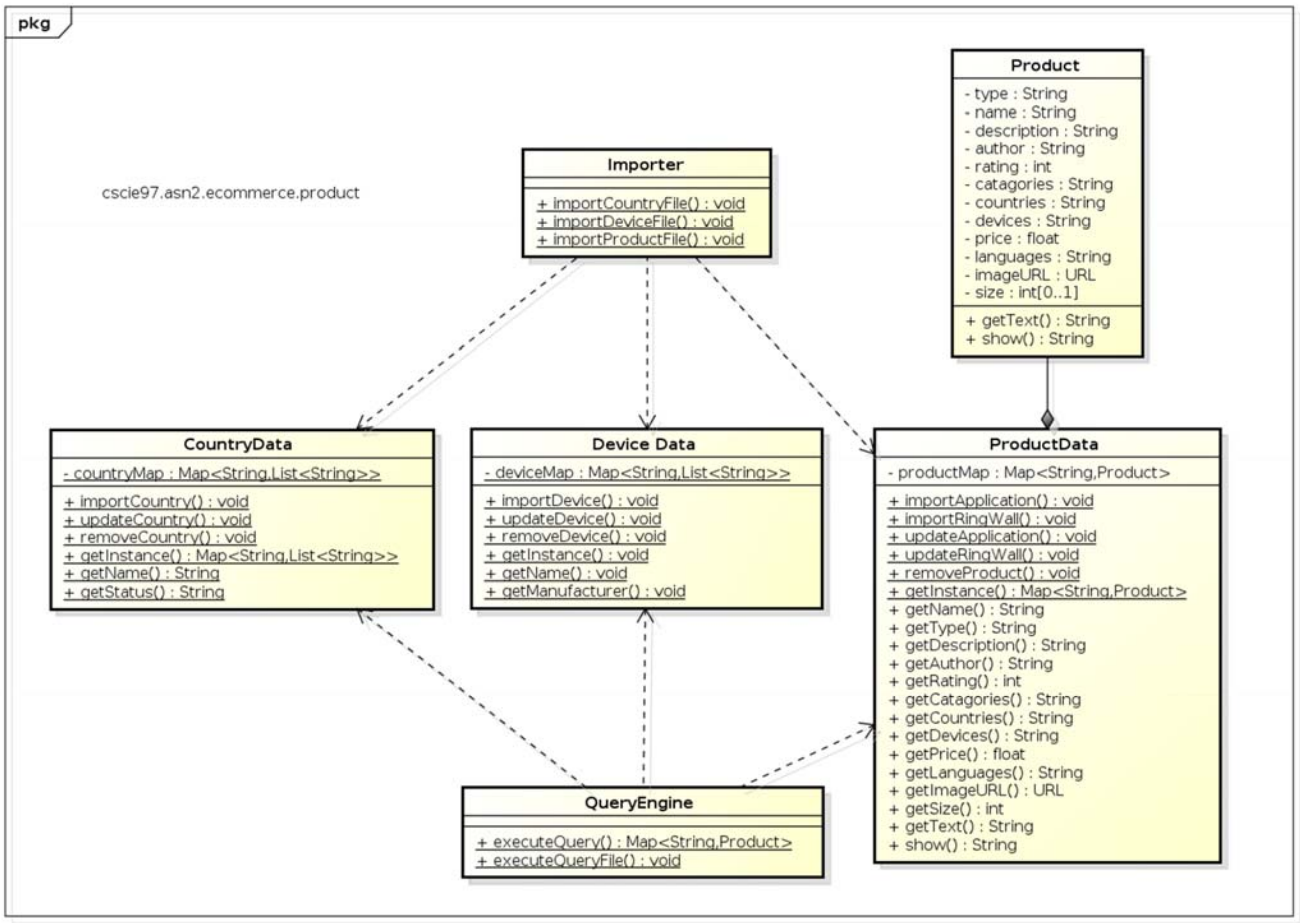
Application developers can manage or update the content they have previously created.



Implementation

Class Diagram

The following class diagram defines the classes defined in this design.



Class Dictionary

This section specifies the class dictionary for the product catalog API service. The classes are defined within the package “cscie97.asn2.ecommerce.product”.

Importer

The importer is responsible for reading country, device, and product information from .csv files. The importer calls import functions in each respective information class to convert verified line of information into data maps.

Methods

Method Name	Signature	Description
importCountryFile	(filename:String, pass:String):void	Public method for reading country information from .csv file line by line and

		calls importCountry function in class CountryData to import into a country map. Checks for country code format. Throws IO Exception on error accessing the file.
importDeviceFile	(filename:String, pass:String):void	Public method for reading device information from .csv file line by line and calls importDevice function in class DeviceData to import into a device map. Throws IO Exception on error accessing the file.
importProductFile	(filename:String, pass:String):void	Public method for reading product information from .csv file line by line and calls importApplication or importRingWall function in class ProductData to import into a product map. Checks for input parameters of products and content type. Throws an exception if the country downloadable to has export restrictions. Throws IO Exception on error accessing the file.

QueryEngine

The QueryEngine class executes queries to product information. Queries are specified in .csv files with empty fields as non-restricted fields and non-empty fields specifying search criteria. All matching results for each query are returned in a map and printed out, preceded by the query string. The Query Engine supports two methods, one accepting a single query string, another supports a list of query strings from a file.

Methods

Method Name	Signature	Description
executeQuery	(queryLine:String, country:Map<String, List<String>>, device:Map<String, List<String>>, product:Map<String, Product>):void	Public method for executing a single query given the input country map, device map and product map.

executeQueryFile	(fileName:String, country:Map<String, List<String>>, device:Map<String, List<String>>, product:Map<String, Product):void	Public method for executing a set of queries read from a file. Checks for valid file name. Delegates to executeQuery for processing individual queries.

CountryData

This class manages the map of all country information stored in a countryMap association. It is a singleton and has to use a static method getInstance() to provide access to the single CountryData instance.

Methods

Method Name	Signature	Description
importCountry	(newCode:String, newName:String, newStatus:String, token:String):void	Public method for adding a set of country information to the countryMap. This function is password-protected for authorized access only. This method checks for existing country code and shows error message on such occasions.
updateCountry	(newCode:String, newName:String, newStatus:String, token:String):void	Public method for changing a set of country information in the countryMap and is also restricted. This method checks for non-existing country code and shows error message in such case.
removeCountry	(newCode:String, token:String):void	Public method for removing a country from the countryMap and is also restricted. This method checks for non-existing country code and shows error message.
getInstance	(void): Map<String, List<String>>	This method returns a reference to the single static instance of the countryMap.
getName	(code:String):String	Returns the country name given the country code. If the

		country code does not exist, show an error message.
getStatus	(code:String):String	Returns the export status given the country code. If the country code does not exist, show an error message.

Associations

Association Name	Type	Description
countryMap	Map<String, List<String>>	Private association for maintaining the country information. The country code serves as the key, while name and export status are put into a list of strings and serve as the value of the map

DeviceData

This class manages the map of all device information stored in a deviceMap association. It is a singleton and has to use a static method getInstance() to provide access to the single DeviceData instance.

Methods

Method Name	Signature	Description
importDevice	(newId:String, newName:String, newManufacturer:String, token:String):void	Public method for adding a set of device information to the deviceMap. This function is password-protected for authorized access only. This method checks for existing device ID and shows error message on such occasions.
updateDevice	(newId:String, newName:String, newManufacturer:String, token:String):void	Public method for changing a set of device information in the deviceMap and is also restricted. This method checks for non-existing country code and shows error message in such case.
removeDevice	(newId:String, token:String):void	Public method for removing a device from the deviceMap and is also restricted. This

		method checks for non-existing country code and shows error message.
getInstance	(void): Map<String, List<String>>	This method returns a reference to the single static instance of the deviceMap.
getName	(id:String):String	Returns the device name given the device ID. If the ID does not exist, show an error message.
getManufacturer	(id:String):String	Returns the manufacturer given the device ID. If the ID does not exist, show an error message.

Associations

Association Name	Type	Description
deviceMap	Map<String, List<String>>	Private association for maintaining the device information. The device ID serves as the key, while name and manufacturer are put into a list of strings and serve as the value of the map

ProductData

This class manages the map of all product information stored in a productMap association. It is a singleton and has to use a static method getInstance() to provide access to the single productData instance.

Methods

Method Name	Signature	Description
importApplication	(newType:String, newId:String, newName:String, newDescription:String, newAuthor:String, newRating:Int, newCategories:String, newCountries:String, newDevices:String, newPrice:float, newLanguages:String, newImageURL:URL, newSize:int, token:String):void	Public method for adding a set of application information to the productMap. This function is password-protected for authorized access only. This method checks for existing product ID and shows error message on such occasions.

importRingWall	(newType:String, newId:String, newName:String, newDescription:String, newAuthor:String, newRating:Int, newCategories:String, newCountries:String, newDevices:String, newPrice:float, newLanguages:String, newImageURL:URL, token:String):void	Public method for adding a set of ringtone or wallpaper information to the productMap. This function is password-protected for authorized access only. This method checks for existing product ID and shows error message on such occasions.
updateApplication	(newType:String, newId:String, newName:String, newDescription:String, newAuthor:String, newRating:Int, newCategories:String, newCountries:String, newDevices:String, newPrice:float, newLanguages:String, newImageURL:URL, newSize:int, token:String):void	Public method for changing a set of application information in the productMap and is also restricted. This method checks for non-existing product ID and shows error message in such case.
updateRingWall	(newType:String, newId:String, newName:String, newDescription:String, newAuthor:String, newRating:Int, newCategories:String, newCountries:String, newDevices:String, newPrice:float, newLanguages:String, newImageURL:URL, token:String):void	Public method for changing a set of ringtone or wallpaper information in the productMap and is also restricted. This method checks for non-existing product ID and shows error message in such case.
removeProduct	(newId:String, token:String):void	Public method for removing a product from the productMap and is also restricted. This method checks for non-existing product ID and shows error message.
getInstance	(void): Map<String, List<String>>	This method returns a reference to the single static instance of the productMap.

Associations

Association Name	Type	Description

productMap	Map<String, Product>	Private association for maintaining the product information. The product ID serves as the key, while all other information are stored as an instance of the Product class and serve as the value of the map

Product

This class represents a unique product information to be linked into the productMap.

Methods

Method Name	Signature	Description
getName	(void):String	Returns the product name.
getType	(void):String	Returns the type of content.
getDescription	(void):String	Returns the description.
getAuthor	(void):String	Returns the author.
getRating	(void):int	Returns the rating.
getCatagories	(void):String	Returns the categories.
getCountries	(void):String	Returns the countries downloadable at.
getDevices	(void):String	Returns the devices supported.
getPrice	(void):float	Returns the price.
getLanguages	(void):String	Returns the languages.
getImageURL	(void):URL	Returns the image URL.
getSize	(void):int	Returns the size. If not an application, returned value is -1.
getText	(id:String):String	Returns the name and description.
show	(id:String):String	Returns a formatted output string of the product.

Properties

Property Name	Type	Description
name	String	Private name field for the product.
type	String	Private type field for the product.
description	String	Private description field for the product.
author	String	Private author field for the product.
rating	int	Private rating field for the product.
categories	String	Private categories field for the product.
countries	String	Private countries field for the product.
devices	String	Private devices field for the product.
price	float	Private price field for the product.
languages	String	Private languages field for the product.
imageURL	URL	Private imageURL field for the product.
size	int	Private size field for the product. Only valid for applications.

Implementation Details

The importer has been designed to have three separate import functions to avoid pattern-matching of input files and relies primarily on user input, while data format verification is only performed with respect to each input format. Determination of product type (application or ringtone, wallpaper) is also performed in the importer so that the correct product class constructor could be called.

The country, device and product instances are all managed in singleton Hash Maps to provide fast lookup for uniqueness verification, together with straight-forward update and deletion call. While country name and export status, as well as device name and manufacturer can all be put into lists of strings, the product fields have multiple data types and were thus placed in a separate product class. All three types of data possess three managing methods: import, update and delete to force users to specify their intention and avoid unintentional updates.

Searching is done through a negative-match filtering fashion to reduce the “join” operation needed in case of multiple search fields. A duplicate map of products is first made for each query, then for or each non-empty search criteria, an iterator goes through the map and deletes any entry that does not satisfy the current search field. Finally this trimmed map is returned to the query file executer for the result to be printed. All string have been made a lowercase copy prior to matching.

Testing

Implement a test driver class called TestDriver that implements a static main() method. The main() method should accept 4 parameters: an input countries data file, a devices data file, a products data file, and a query file. The main method will call the Importer. importCountryFile(),Importer.importDeviceFile(), and Importer.importProductFile() method, passing in the name of the file and a password.

After loading the input files, the main() method will invoke the QueryEngine.executeQueryFile() method passing in the provided query file name. The TestDriver class should be defined within the package “cscie97.asn2.ecommerce.test”.

Risks

The importer does not check for any extra columns than the required format specified, nor does it check for duplicate country names, especially device data fields. The query engine does not check to see if an otherwise valid search criteria has been put into a wrong column.