Mobile Application Store Authentication Service API Design Document

Date: 2013-11-10
Author: David Killeffer <rayden7@gmail.com>
Reviewer(s): Jonathan Nichols <jn42887@gmail.com>

Introduction

This document defines the design for the Mobile Application Store Authentication Service API, which is a core component of the Mobile Application Store. This document is organized into the following sections:

1. **Overview:** summarizes the problem that is being solved and why
2. ***TODO: Summary of CollectionServiceAPI and ProductAPI Changes**: explains changes that were required to the CollectionServiceAPI to accommodate refactoring based on TA feedback from Assignment 3, as well as changes that were made to accommodate the new AuthenticationServiceAPI's enforcement over restricted interfaces*
3. **Requirements:** enumerates the various requirements for the AuthenticationServiceAPI
4. **Use Cases:** enumerates the use cases that the AuthenticationServiceAPI must support
5. **Class Diagram:** graph of the primary classes and their relationships that are in use in the AuthenticationServiceAPI
6. **Sequence Diagram:** shows an example of how a user interacts with the AuthenticationServiceAPI to login, and then carry out a restricted interface action on the ProductAPI
7. **Activity Diagram:** shows the process of creating roles, permissions, and users by the AuthenticationServiceAPI
8. **Class Dictionary:** a catalog of the classes that collectively comprise the AuthenticationServiceAPI and the various methods, properties, and attributes that define each
9. **Implementation Details:** special considerations and explanatory observations about how the classes relate to each other and work
10. **Changes from Original Design & Requirements:** explanations of how this implementation varies slightly from the stated requirements in MobileApplicationStoreAuthenticationServiceRequirements.pdf
11. **Testing:** explanation of how the classes in AuthenticationServiceAPI are to be tested, and how the TestDriver exercises the functionality
12. **Risks:** identifies and enumerates any potential pitfalls or deficiencies in the current implementation and issues that might arise from this implementation
13. **Instructions for Compiling and Running Application:** explains how to compile and run the application

Overview

The Mobile Application Store Application requires ongoing administrator action to keep it up to date, to initially populate it with Content items and Collections, etc. In order to enable

Administrators and Application Developers to carry our certain restricted interfaces (such as creating Content, creating Collections, etc.), a system for authenticating users and authorizing users to conduct such actions is required. The AuthenticationServiceAPI will allow the creation of User accounts, the ability to define Services, Permissions, and Roles, and further grant Roles and Permissions to those User accounts.

"Permissions" define a restricted interface action that a User may perform; for example, "create_device" is a permission that allows Users that are granted it to call the ProductAPI.importDevices() method. Permissions are *"what you can do"*.

"Roles" are used to aggregate sets of Permissions into a logical group. These logical groups of related Permissions collectively define *"who you are"*, when a Role (or Roles) are granted to a User. For example, the Permissions "create_country", "create_device", and "create_product" all belong to a Role called "product_admin_role", so whomever is granted the "product_admin_role" may be considered to be a Product Administrator.

"Services" define the major functional areas of the Mobile Application Store; currently the Services are the ProductAPI, CollectionServiceAPI, and the AuthenticationServiceAPI.

TODO: Summary of CollectionServiceAPI and ProductAPI Changes

TODO: take feedback from TA's on Assignment 3 design document to make changes

TODO: document how current ProductAPI and CollectionServiceAPI needs to be updated to be used in conjunction with the AuthenicationServiceAPI

AuthenticationServiceAPI Requirements

This section defines the requirements for the AuthenticationServiceAPI system component. In previous sprints, the ProductAPI and CollectionServiceAPI were stubbed out to allow for a GUID token to be passed as an authorization measure before allowing users to call restricted interface methods. The AuthenticationServiceAPI will allow for Users to log into the system, be granted Permissions and Roles, allow administrators to define those Roles and Permissions as well as Services, and allows for a robust way of ensuring that only allowed users carry out certain actions in the Mobile Application Store.

The AuthenticationServiceAPI supports two main types of users: Administrators and RegisteredUsers. Only Administrators should be allowed to:

- Create Services
- Create Permissions
- Create Roles
- Create Users

Administrators and RegisteredUsers should both be allowed to:

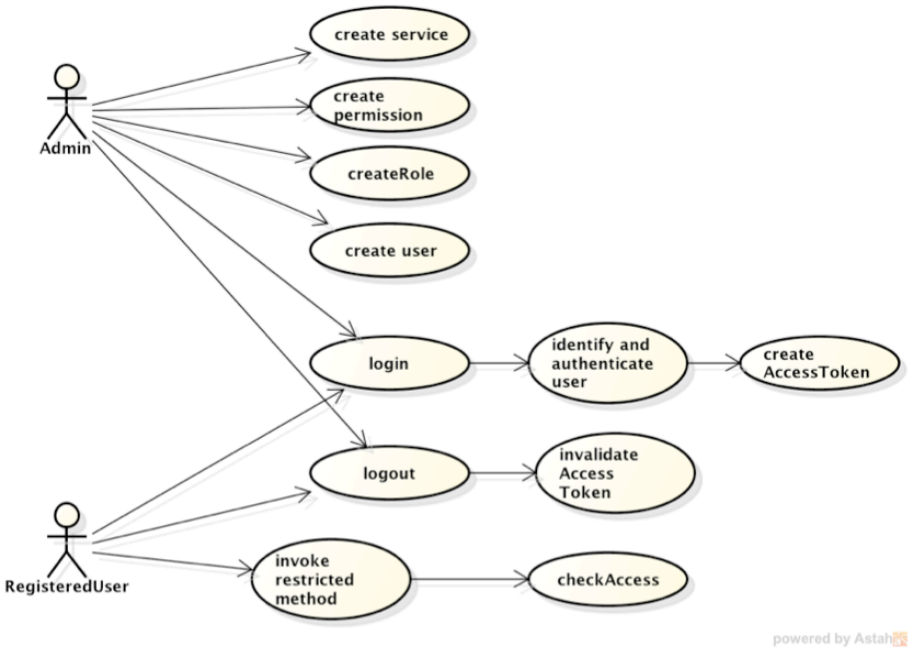
- Login

- Logout
- Invoke Restricted Access Method

Users shall be allowed to have multiple sets of credentials; that is to say, they may have more than 1 set of username/password combinations to use the service. Additionally, when a user calls a method that is a restricted interface on either the ProductAPI, CollectionServiceAPI, or AuthenticationServiceAPI, an AccessToken should be passed to the called method and validated against the AuthenticationServiceAPI to ensure that only authorized users are allowed to call restricted interface methods. More specific details on the requirements are found in the *MobileApplicationStoreAuthenticationServiceRequirements.pdf* document.

Use Cases

The following use case diagram shows the major functional use cases that each type of user of the Authentication Service can perform.



Create Service: only Administrators can create and define new Services. Services define the major discrete areas of the Mobile Application Store that users may interact with (such as the ProductAPI, CollectionServiceAPI, and AuthenticationServiceAPI). This is a restricted interface and requires a valid Administrator be logged in before new Services may be defined.

Create Permission: only Administrators can create and define new Permissions. Permissions define the restricted actions that only authenticated Users may perform in the entire system. This is a restricted interface and requires a valid Administrator be logged in before new Permissions may be defined.

Create Role: only Administrators can create and define new Roles. Roles may contain Permissions, and/or other Roles. Roles aggregate Permissions (and other Roles) to create virtual definitions of what Users may be when they are granted a "Role"; for example, if several Permissions specific to the creation of Content items on the ProductAPI are gathered up into a Role called "ProductAdmin", any Users that are granted the "ProductAdmin" role can be considered administrators of the ProductAPI. This is a restricted interface and requires a valid Administrator be logged in before new Roles may be defined.

Create User: only Administrators can create and define new User accounts. Users may be granted specific Roles and Permissions. This is a restricted interface and requires a valid Administrator be logged in before new Users may be defined.

Login: both Administrators and RegisteredUsers may login, which then will allow them to call the restricted interface methods in the Mobile Application Store that they have access to. Logging in involves identifying and authenticating the username and password, and also obtaining an AccessToken for the User; the AccessTokens may be passed to any of the restricted interface methods the logged in user calls. Once logged in, the user's AccessToken is only good for an arbitrarily predetermined amount of time; if the user does not call any restricted interface methods before that time, then the AccessToken expires.

Logout: both Administrators and RegisteredUsers may logout of the system, which invalidates any of their current AccessTokens, and then would not allow them to call any restricted interface methods until successfully logging in again.

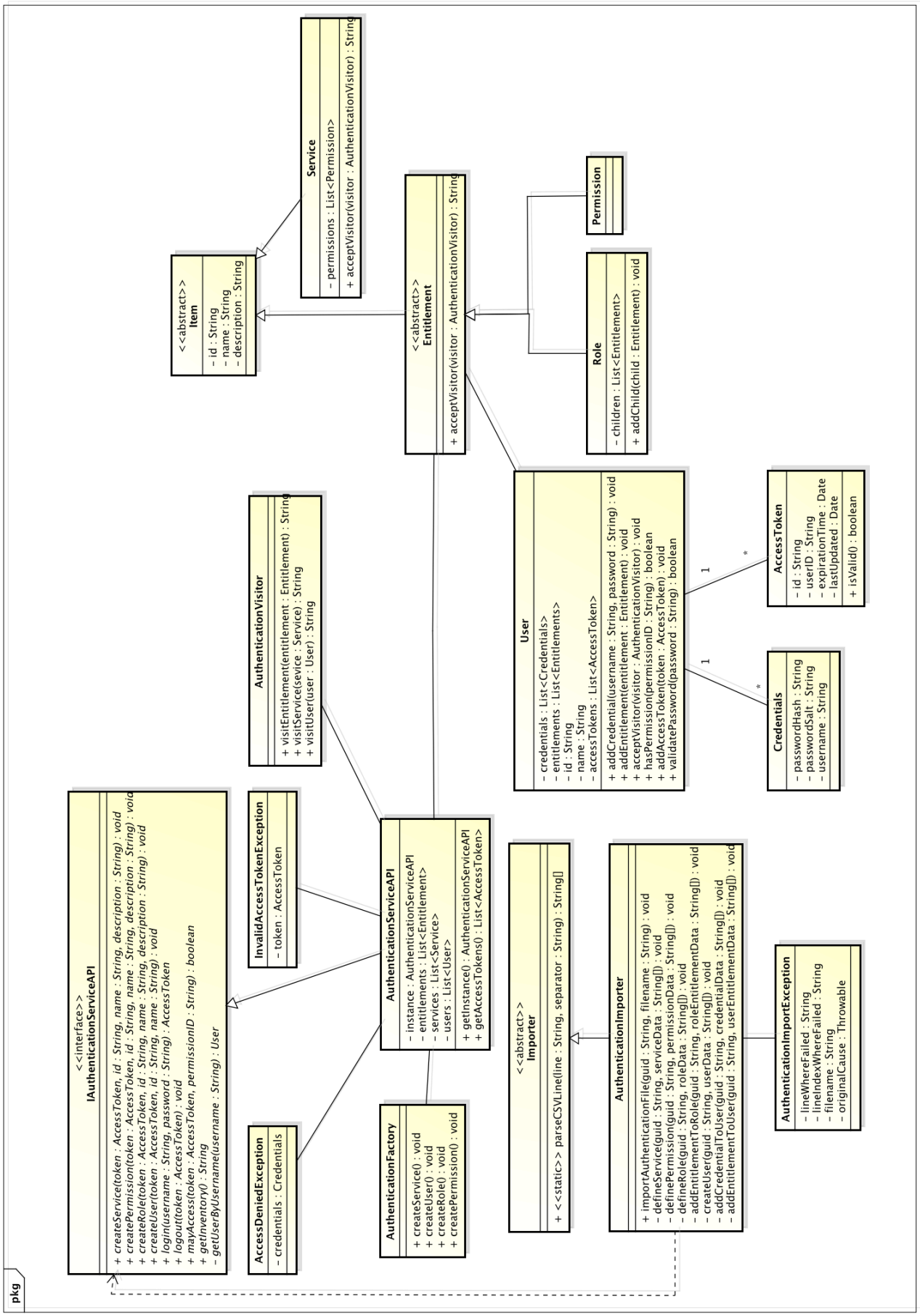
Class Diagram

The following class diagram defines the major classes defined in this design. A summary of the classes and their interrelationships follows the diagram.

Mobile Application Store Product API Design Document

CSCI E-97 Assignment 4

David Killeffer <rayden7@gmail.com>



IAuthenticationServiceAPI / AuthenticationServiceAPI: The primary class for Administrators to interact with the Authentication catalog is through the AuthenticationServiceAPI class. IAuthenticationServiceAPI provides an interface to define the methods that the service must support (programming to interfaces rather than concrete classes is a good general design principle). This is the primary way to create new Users, Services, Roles, and Permissions.

AuthenticationFactory: helper class for the AuthenticationServiceAPI that follows the Factory design pattern. The primary purpose of the usage of the Factory pattern is to ensure that when authentication items are created (Roles, Users, Permissions, Services) that they all have a globally unique ID that is a GUID; each ID should be unique across all authentication objects in the system.

AuthenticationImporter: this is the primary mechanic by which Authentication items (Roles, Permissions, Users, and Services) are created and imported into the catalog, and an inventory of the authentication items is displayed. The AuthenticationImporter itself only exercises the functionality of the AuthenticationServiceAPI. The AuthenticationImporter is called via the TestDriver, which loads CSV files for operation; a single CSV file called authentication.csv is the primary way that all major functionality is exercised. The AuthenticationImporter may throw customized exceptions AuthenticationImportException when an error occurs either creating new authentication items or parsing the CSV file.

AuthenticationVisitor: helper class that follows the Visitor pattern and is used for constructing a inventory of authentication items for display.

Service: denotes and identifies those "services" that the AuthenticationServiceAPI is in charge of administering authentication/authorization services for. Currently this will include the ProductAPI, CollectionServiceAPI, and the AuthenticationServiceAPI itself. Service objects must be created first before the other types can be used because roles and permissions should be related to a service.

Permission: defines a restricted interface task that users may be granted access to perform in the system. Permissions can be added to Services, so that a Service can list out all those permisisons that it supports, as well as be added to Roles, so that a group of Permissions collectively under the umbrella of a Role can define the actions that any User granted that Role may conduct. Permissions define "what users can do".

Role: a container class for Permissions and other Roles, Roles are therefore tree structures and can be several levels deep. *For the purposes of this initial implementation, an arbitrary depth limit of 5 may be incurred so as to not allow for creating overly-complex Role structures that are difficult to maintain.* Roles define "who users are".

User: a registered user of the MobileApplicationStore, users may be able to log into the system and then call API methods on any of the Service restricted methods to which they have been given access. Users may have multiple sets of Credentials, so that they may login to the system with different usernames/passwords.

Credentials: a set of credentials for a user. Does not contain the plain-text password, but rather the passwordHash and the password "salt". When checking a supplied plain-text username and password, the password salt is applied along with a hashing algorithm to create the hashed password – this is compared against the "passwordHash" attribute to determine whether the user has successfully authenticated or not.

AccessToken: a time-based token that grants the logged in user the ability to carry out restricted interface methods to which they have been given access via the user's entitlements.

Item: abstract base class for Service, Role, and Permission to encapsulate the shared attributes common to all (id, name, description).

Entitlement: abstract base class for Role and Permission, inheriting from Item. Declares the acceptVisitor() method common to Role and Permission.

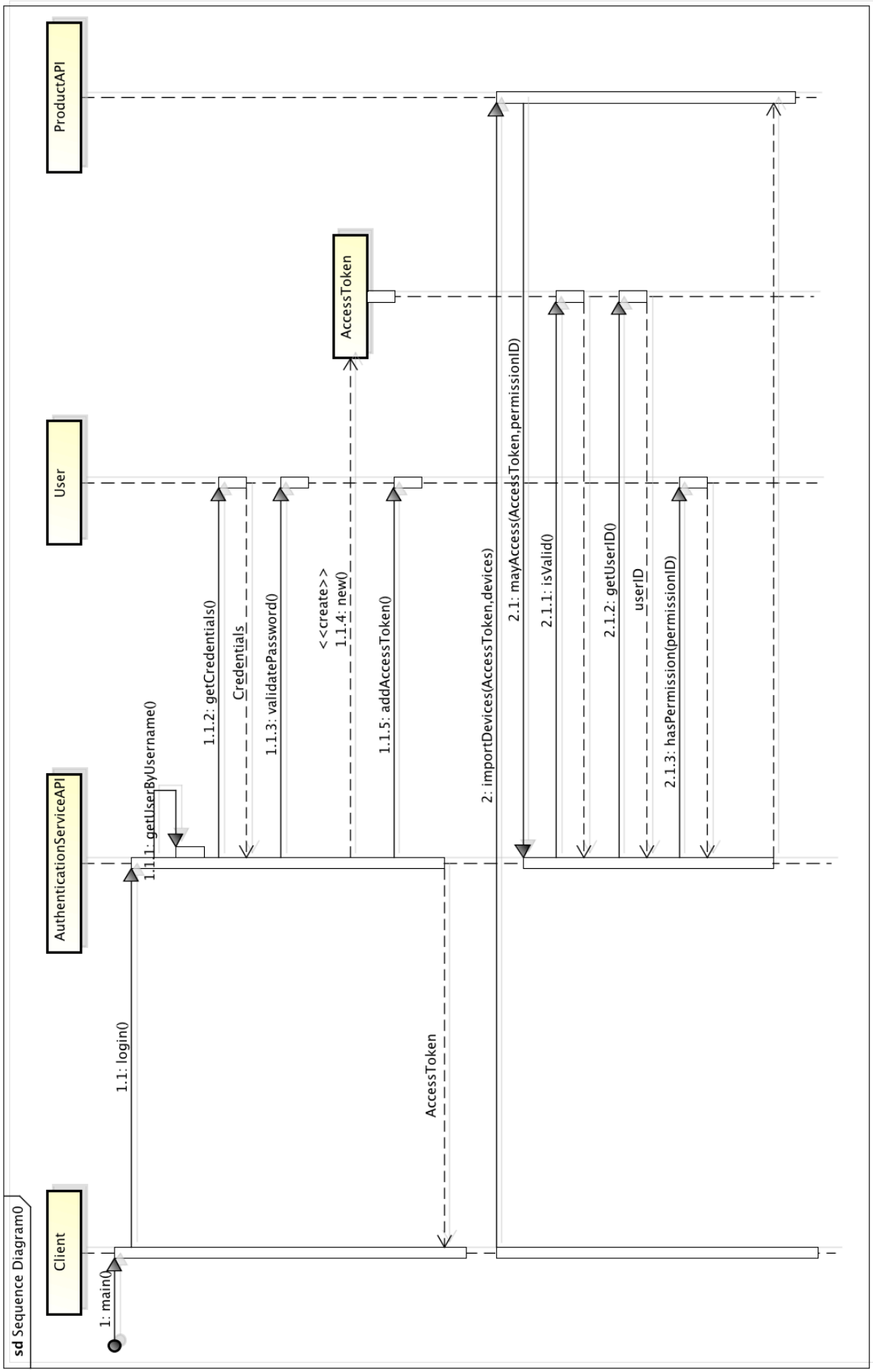
Sequence Diagram

This sequence diagram shows how a client might interact with the AuthenticationServiceAPI to login to the system and then call a restricted interface method on the ProductAPI.

Mobile Application Store Product API Design Document

CSCI E-97 Assignment 4

David Killeffer <rayden7@gmail.com>

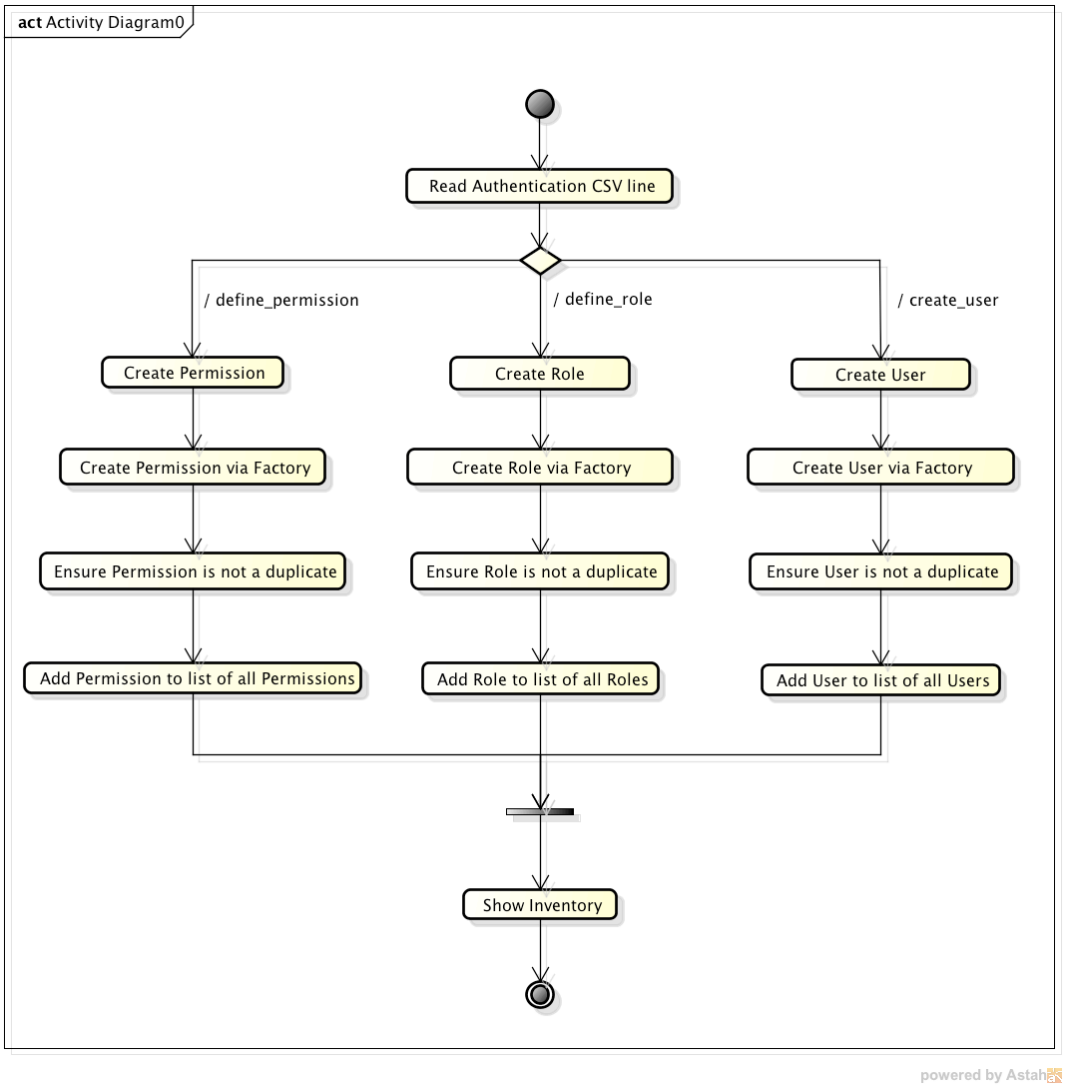


The diagram illustrates that when a user attempts to login to the system, first the AuthenticationServiceAPI is checked for the existence of a matching User that has the same username. When the User is found, the User's credentials are checked and the password is validated. Once authenticated, a new AccessToken for the User is created, added to the User, and then returned to the calling method (login).

The bottom part of the diagram shows how the AccessToken is passed when calling restricted interface methods and the user's credentials are validated and checked against the User to ensure that only allowed users are able to call the "importDevices()" method.

Activity Diagram

The following activity diagram shows the logical process by which users, roles, and permissions may be created in the Authentication system by their loading via the CSV file method in the AuthenticationImporter.



Class Dictionary

This section specifies the class dictionary for the AuthenticationServiceAPI. The primary classes should be defined within the package "cscie97.asn4.ecommerce.authentication". Per the changes outlined previously under the "Summary of ProductAPI and CollectionServiceAPI Changes" section, exception classes should be defined within the package "cscie97.asn4.ecommerce.exception", and CSV-handling classes should be defined within the package "cscie97.asn4.ecommerce.csv". For the sake of brevity, simple accessor/mutator methods will not be documented here.

Package: **cscie97.asn4.ecommerce.authentication**

IAuthenticationServiceAPI <<interface>>

This interface defines the primary service contract for any classes that intend to implement these methods and act as the concrete implementation of the AuthenticationServiceAPI. Administrators may create Services, Users, Roles, and Permissions. Both Registered Users and Administrators may login and logout, and the service also ensures that AccessTokens are checked before granting access to restricted interface methods in other services (ProductAPI, CollectionServiceAPI) and in the AuthneticationServiceAPI itself. Also provides a method for getting a string representation of the unique authentication service inventory of Services, Roles, Permissions, and Users.

TODO: list all methods, properties, and associations of this and all other classes

Implementation Details

The AuthenticationServiceAPI is implemented as a Singleton, as returned by the static getInstance() method. Following good basic design principles, the AuthenticationServiceAPI's public methods are enumerated in the interface that it inherits from, IAuthenticationServiceAPI. Anywhere in the code that a reference to the service is expected, the interface type is returned rather than a concrete implementation.

Role, Permission, and Entitlement together follow the Composite design pattern.

The individual object class types Role, Permission, Service, and User all support static factory methods inside the AuthenticationFactory for object creation to ensure unique ID values for each. The id's of each of the objects in the Authentication catalog should be unique GUIDs.

The AuthenticationVistor class follows the Visitor design pattern for visiting each of the specific objects in the methods it supports; this is used for the purpose of building up and declaring an inventory of all the "items" (e.g., unique Users, Roles, Permissions, Services) in the authentication catalog for display.

Changes from Original Design & Requirements

All CSV based content loading classes were organized under a new package: cscie97.asn4.ecommerce.csv. Additionally, the Importer from Assignment 2 was refactored into a more general purpose, abstract utility class that may be used by all other classes that deal with CSVs – it's primary shared static method parseCSVLine() is used for uniformly parsing the CSV lines, which SearchEngine, ContentImporter, CollectionImporter, and AuthenticationImporter all use. This logical grouping of classes with similar responsibilities under a common package separate from both cscie97.asn4.ecommerce.authentication, cscie97.asn4.ecommerce.collection, and cscie97.asn4.ecommerce.product will allow for further re-use amongst shared code logic in future sprints.

Similarly, all exception classes were broken out into a shared package as well: cscie97.asn4.ecommerce.exception. A more generic top-level exception class, MobileAppException, was created as a superclass for more specific custom exception types.

InvalidAccessTokenException, AccessDeniedException, and AuthenticationImportException classes were added as children of MobileAppException.

Testing

The TestDriver class will exercise importing users, roles, permissions, and services into the Authentication catalog via public methods on the AuthenticationServiceAPI, defining the Permissions and Roles that collectively comprise aggregate Roles, as well as generating an inventory of all the Users, Permissions, Services, and Roles in the Authentication catalog.

A static main() method will interactively prompt the user for Countries, Devices, Content, ContentSearches, Collections, and Authentication CSV files. The Authentication CSV should contain definitions of services, permissions, and roles, as well as adding permissions to roles, creating users, and adding entitlements (e.g., permissions and roles) to users. The main() method should call public import methods on the AuthenticationServiceAPI to load authentication items into the Authentication catalog and define their properties.

In addition to the sample countries.csv, devices.csv, products.csv, queries.csv, and collections.csv files that were provided, several new devices, products, content queries, and collection data were added to further test the functionality and correctness of the implementation. To exercise the tree structure of Collections, test adding both StaticCollection and DynamicCollection objects onto Collections, test the depth-first iteration, etc., a series of collections (named "staticA", "staticC", and "dynamicB") were added to the collections.csv, and easily-identifiable content items named A-K were added to

products.csv. To see some of the logic on the expected results of iterating over the "staticA" collection, refer to the collections.csv file.

Additoinally, to test actually logging into the AuthenticationServiceAPI, the TestDriver will attempt to log into the AuthenticationServiceAPI and query for an inventory of Authentication items, as well as try to carry out some of the restricted interface methods on both ProductAPI and CollectionServiceAPI.

Risks

As risks are uncovered in the development process they will be documented here.

Instructions for Compiling and Running Application

Because there are two external JAR dependencies and the package organization has been slightly reorganized from the prior sprint, the way to compile and run the application is slightly different from the original specifications.

To compile the code, run the following:

```
javac -cp ".:commons-collections4-4.0-alpha1.jar:commons-lang3-
3.1.jar" cscie97/asn4/ecommerce/authentication/*.java
cscie97/asn4/ecommerce/collection/*.java
cscie97/asn4/ecommerce/csv/*.java
cscie97/asn4/ecommerce/exception/*.java
cscie97/asn4/ecommerce/product/*.java cscie97/asn4/test/*.java
```

To run the application, run the following:

```
java -cp ".:commons-collections4-4.0-alpha1.jar:commons-lang3-
3.1.jar" cscie97.asn4.test.TestDriver countries.csv devices.csv
products.csv queries.csv collections.csv authentication.csv
```