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. **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 provides methods to create User accounts, to create Services, Permissions, and Roles, define the Permissions on Services 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.

Summary of CollectionServiceAPI and ProductAPI Changes

The ProductAPI and CollectionServiceAPIs were modified to fully implement the authentication mechanisms that the AuthenticationServiceAPI implements. Specifically, the validateAccess() methods that used to be in both were replaced by calls to the AuthenticationServiceAPI.mayAccess() method. Additionally, each restricted interface method in the ProductAPI and CollectionServiceAPI also sends along a PermissionType enum value to denote which permission the mayAccess() method should be checking the matching User account for an entitlement on.

Additionally, based on feedback from Assigment 3, the exception classes were refactored and moved into their respective service-specific packages, rather than grouping them together (enhances modularity).

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:

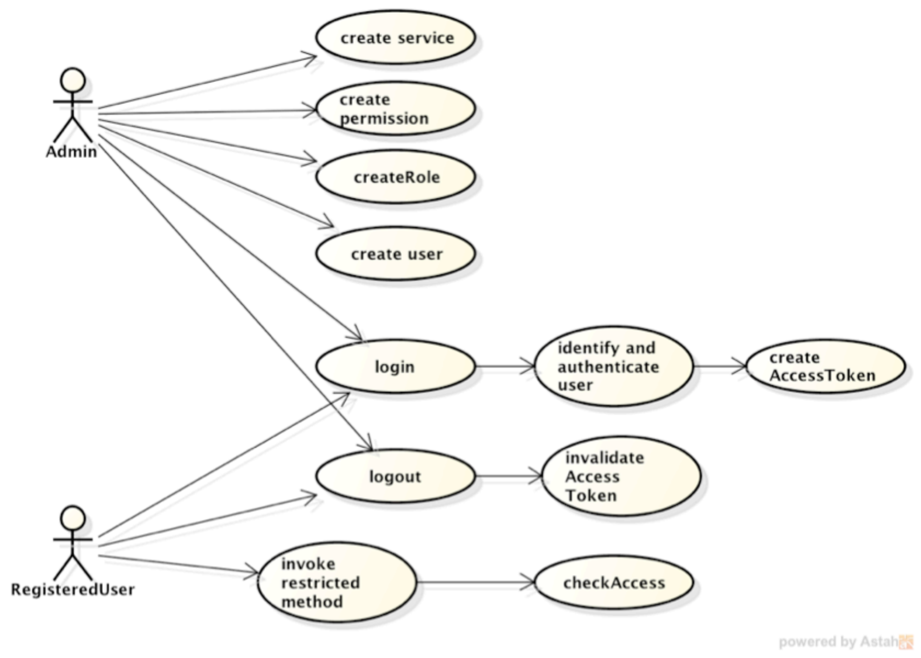
- Login
- Logout
- Invoke Restricted Access Method

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. Users are given “entitlements”, which is how they are given permission to access certain restricted interface methods on the various Application Store APIs. My understanding of the requirements is that although users shall be allowed to login and access restricted interface methods with different credentials, they only have 1 set of “entitlements” (e.g., different username/password combinations do not result in different “entitlements” used to access the services – that would mean that we were allowing users to impersonate others, which did not seem to be the goal of the assignment). If requirements were to change and there was a desire to allow users to associate different sets of entitlements with different credentials, this could be accomplished by associating “entitlements” with the Credentials object, rather than at the User level.

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.

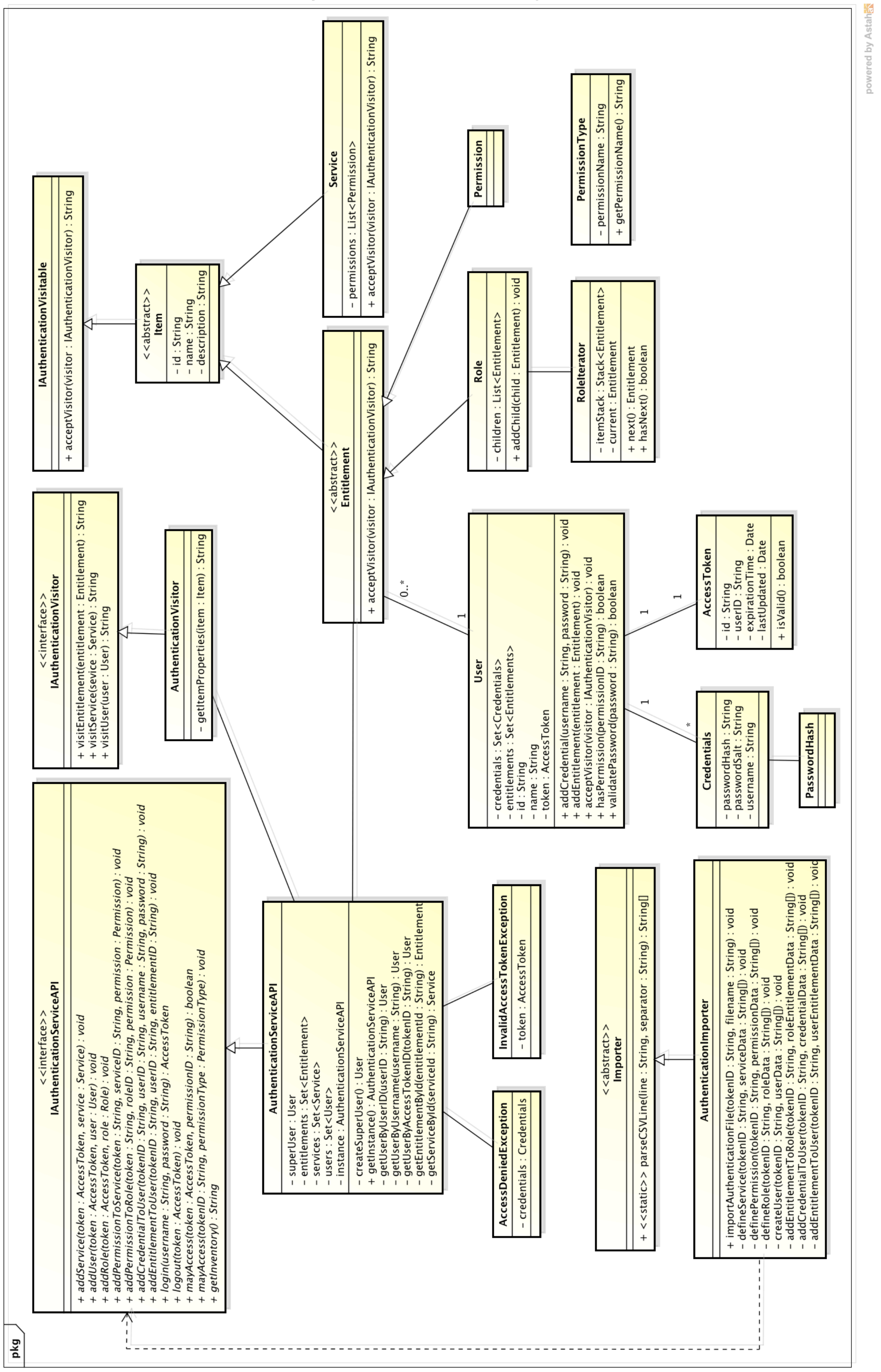
Create User: only Administrators can create and define new User accounts. Users may be granted specific Roles and Permissions. This is a restricted interface and requires a valid Administrator be logged in before new Users may be defined.

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. When the User later makes calls to restricted interface methods on any of the ProductAPI, CollectionServiceAPI, or AuthenticationServiceAPI, they must pass the ID of their AccessToken, at which point their entitlements are checked to ensure they have proper access. 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.


pkg
IAuthenticationVisitable
+ acceptVisitor(visitor : IAuthenticationVisitor) : String
<<abstract>> Item
- id : String
- name : String
- description : String
Service
- permissions : List<Permission>
+ acceptVisitor(visitor : IAuthenticationVisitor) : String
Permission
PermissionType
- permissionName : String
+ getPermissionName() : String
<<abstract>> Entitlement
+ acceptVisitor(visitor : IAuthenticationVisitor) : String
Role
- children : List<Entitlement>
+ addChild(child : Entitlement) : void
RoleIterator
- itemStack : Stack<Entitlement>
- current : Entitlement
+ next() : Entitlement
+ hasNext() : boolean
<<interface>> IAuthenticationVisitor
+ visitEntitlement(entitlement : Entitlement) : String
+ visitService(sevice : Service) : String
+ visitUser(user : User) : String
AuthenticationVisitor
- getItemProperties(item : Item) : String
0..*
1
User
- credentials : Set<Credentials>
- entitlements : Set<Entitlements>
- id : String
- name : String
- token : AccessToken
+ addCredential(username : String, password : String) : void
+ addEntitlement(entitlement : Entitlement) : void
+ acceptVisitor(visitor : IAuthenticationVisitor) : void
+ hasPermission(permissionID : String) : boolean
+ validatePassword(password : String) : boolean
1
1
AccessToken
- id : String
- userID : String
- expirationTime : Date
- lastUpdated : Date
+ isValid() : boolean
1
*
Credentials
- passwordHash : String
- passwordSalt : String
- username : String
PasswordHash
<<interface>> IAuthenticationServiceAPI
+ addService(token : AccessToken, service : Service) : void
+ addUser(token : AccessToken, user : User) : void
+ addRole(token : AccessToken, role : Role) : void
+ addPermissionToService(token : String, serviceID : String, permission : Permission) : void
+ addPermissionToRole(token : String, roleID : String, permission : Permission) : void
+ addCredentialToUser(tokenID : String, userID : String, username : String, password : String) : void
+ addEntitlementToUser(tokenID : String, userID : String, entitlementID : String) : void
+ login(username : String, password : String) : AccessToken
+ logout(token : AccessToken) : void
+ mayAccess(token : AccessToken, permissionID : String) : boolean
+ mayAccess(tokenID : String, permissionType : PermissionType) : void
+ getInventory() : String
AuthenticationServiceAPI
- superUser : User
- entitlements : Set<Entitlement>
- services : Set<Service>
- users : Set<User>
- instance : AuthenticationServiceAPI
- createSuperUser() : User
+ getInstance() : AuthenticationServiceAPI
- getUserByUserID(userID : String) : User
- getUserByUsername(username : String) : User
- getUserByAccessTokenID(tokenID : String) : User
- getEntitlementById(entitlementId : String) : Entitlement
- getServiceById(serviceId : String) : Service
InvalidAccessTokenException
- token : AccessToken
AccessDeniedException
- credentials : Credentials
<<abstract>> Importer
+ <<static>> parseCSVLine(line : String, separator : String) : String[]
AuthenticationImporter
+ importAuthenticationFile(tokenID : String, filename : String) : void
- defineService(tokenID : String, serviceData : String[]) : void
- definePermission(tokenID : String, permissionData : String[]) : void
- defineRole(tokenID : String, roleData : String[]) : void
- createUser(tokenID : String, userData : String[]) : void
- addEntitlementToRole(tokenID : String, roleEntitlementData : String[]) : void
- addCredentialToUser(tokenID : String, credentialData : String[]) : void
- addEntitlementToUser(tokenID : String, userEntitlementData : String[]) : void
powered by Astah

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.

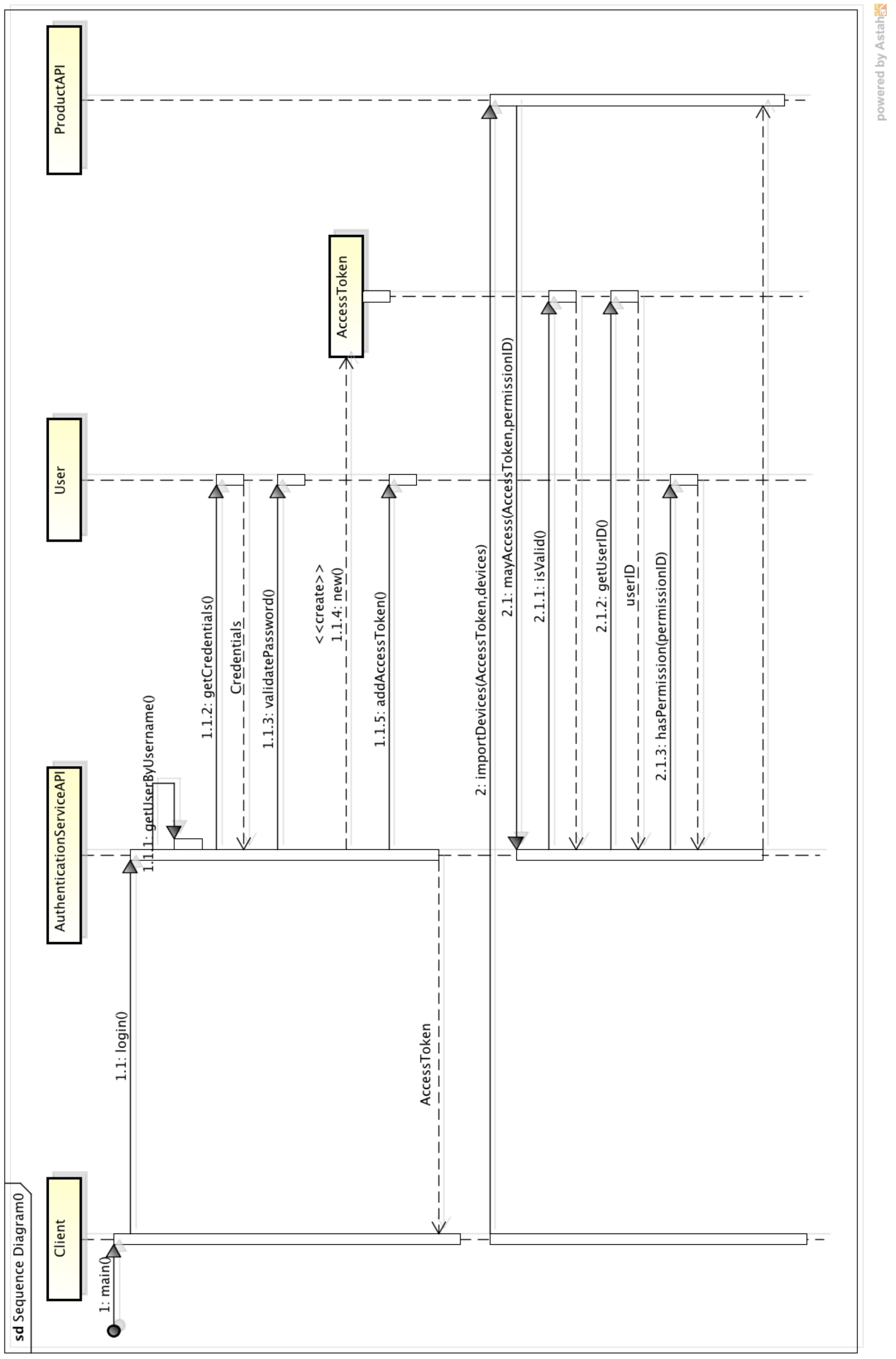
AccessToken: a time-based token that grants the logged in user the ability to carry out restricted interface methods to which they have been given access via the user’s entitlements.

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.

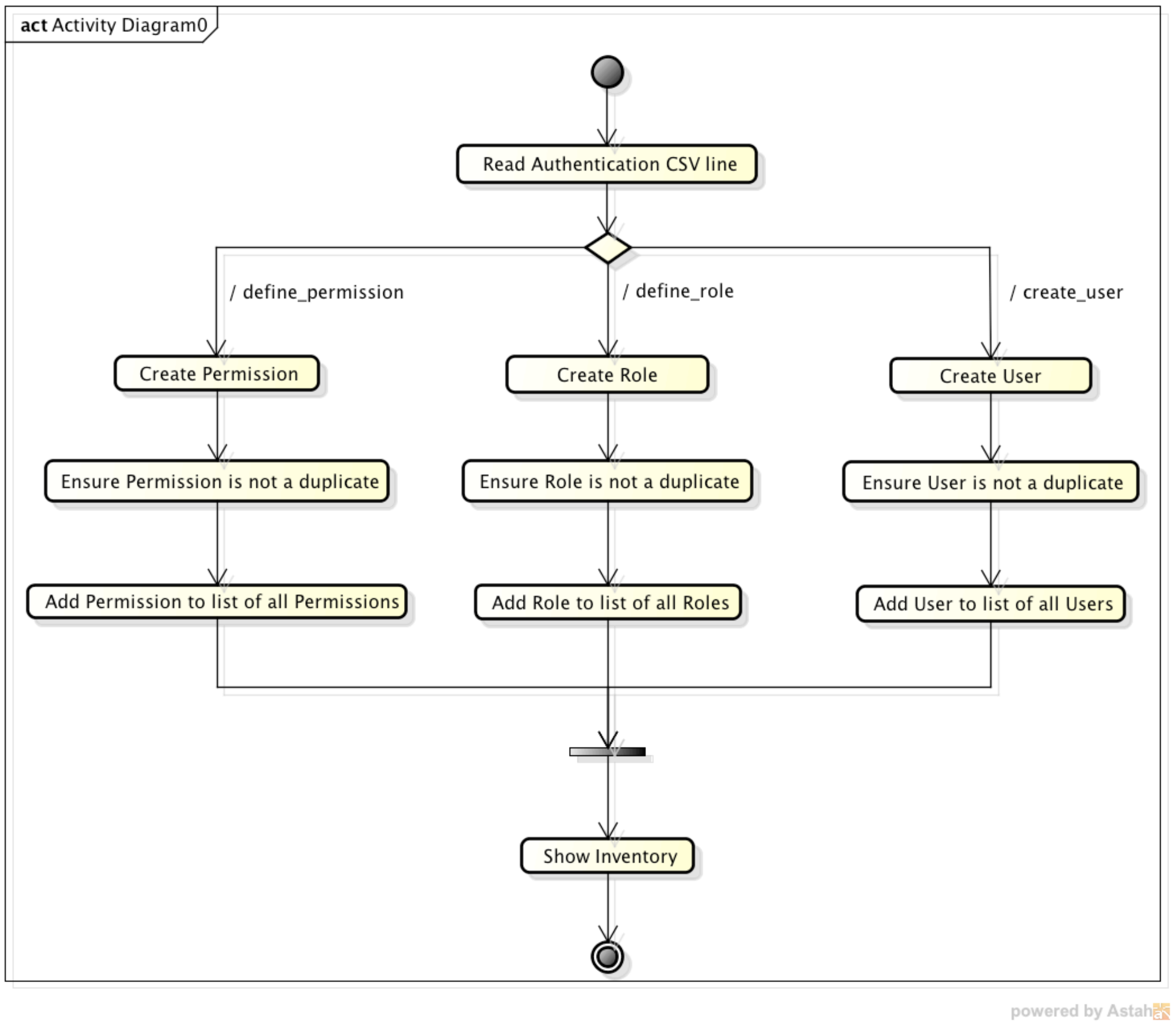


The diagram illustrates that when a user attempts to login to the system, first the AuthenticationServiceAPI is checked for the existence of a matching User that has the same username. When the User is found, the User's credentials are checked and the password is validated. Once authenticated, a new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


act Activity Diagram0
Read Authentication CSV line
/ define_permission
/ define_role
/ create_user
Create Permission
Create Role
Create User
Ensure Permission is not a duplicate
Ensure Role is not a duplicate
Ensure User is not a duplicate
Add Permission to list of all Permissions
Add Role to list of all Roles
Add User to list of all Users
Show Inventory
powered by Astah

Class Dictionary

This section specifies the class dictionary for the AuthenticationServiceAPI. The primary classes should be defined within the package "cscie97.asn4.ecommerce.authentication". For the sake of brevity, simple accessor/mutator methods will not be documented here, nor will overridden methods in classes implementing interfaces or abstract classes.

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.

Methods

Method Name	**Signature**	**Description**
addService	(String : tokenID, Service : service) : void	Adds a new Service definition to the catalog. Services can contain child permissions, but primarily serve as a marker class to logically group permissions into sensible sets.
addUser	(String : tokenID, User : user) : void	Adds a new registered User to the catalog.
addRole	(String : tokenID, Role : role) : void	Adds a new Role to the catalog. Roles may contain other Roles or Permissions to define logical groupings that correspond to types of Users that may use the AuthenticationServiceAPI.
addPermissionToService	(String : tokenID, String : serviceID, Permission : permission) : void	Adds a new Permission to the catalog as a child of the Service.
addPermissionToRole	(String : tokenID, String : roleID, String : permissionID) : void	Adds a new Permission to the catalog as a child of the Role.
addCredentialToUser	(String : tokenID, String : userID, String : username, String : password)	Adds a new Credential to the User.
addEntitlementToUser	(String : tokenID,	Adds a new Entitlement (can be a Role or

	String : userID, String : entitlementID) : void	Permission) to the User.
login : throws AccessDeniedException	(String : username, String : password) : AccessToken	Logs a user into the AuthenticationService. Modifies the AccessToken of the User to expire in 1 hour.
logout	(String : tokenID) : void	Logs the User out that owns the AccessToken with the supplied ID. Modifies the AccessToken to set the expiration time to be now.
mayAccess	(String : tokenID, String : permissionID) : boolean	Checks if the User that owns the AccessToken corresponding to the passed tokenID has permission to use the Permission. Looks up the AccessToken's owning User, and inspects their Entitlements to confirm whether or not the user has the corresponding Permission.
mayAccess	(String : tokenID, PermissionType : permissionType) : boolean	Convenience method; intended to call alternate implementation of mayAccess that accepts a permissionID as the second parameter.
getInventory	() : String	Returns a string representation of the entire Authentication catalog, including Services, Users, Roles, and Permissions. Uses the Visitor pattern to visit each Service, User, Role, and Permission to inquire about the object's salient properties.

AuthenticationServiceAPI, implements IAuthenticationServiceAPI

Implements the IAuthentictionServiceAPI to provide a concrete implementation for client usage. Observes the Singleton pattern, since clients obtain the single shared instance of the object. In order to properly function, requires bootstrapping an initial account for usage in loading all of the catalog items (Users, Roles, Permissions, Services); this is done in the private createUser() method, which is called automatically when obtaining an instance of the AuthenticationServiceAPI class.

Methods

Method Name	Signature	Description
createSuperUser : <<private>>	() : void	Bootstraps the AuthenticationServiceAPI with a "super user" that can be used to initially load the authentication catalog via the AuthenticationImporter} This user will have all the Permissions of the AuthenticationServiceAPI, and so will be able to call all the methods defined in IAuthenticationServiceAPI.
getInstance : <<static, synchronized>>	() : IAuthenticationServiceAPI	Returns a reference to the single static instance of the AuthenticationServiceAPI. Will also create a "super user" account that can be used for loading all the Authentication catalog items.

getUserByUserID : <<private>>	(userID : String) : User	Helper method to retrieve a User by their ID.
getUserByUsername : <<private>>	(username : String) : User	Helper method to retrieve a User by any of their associated usernames.
getUserByAccessTokenID : <<private>>	(tokenID : String) : User	Helper method to retrieve a User by the id of their AccessToken
getEntitlementById : <<private>>	(entitlementId : String) : Entitlement	Helper method to retrieve an Entitlement by its ID.
getServiceById : <<private>>	(serviceId : String) : Service	Helper method to retrieve a Service by its ID.

Properties

Property Name	**Type**	**Description**
SUPER_ADMINISTRATOR_USERNAME : <<private, static, final>>	String	This is the private, hardcoded username that is to be used for bootstrapping the API with an administrative super-user that is used by the TestDriver for loading all the Authentication catalog items
SUPER_ADMINISTRATOR_PASSWORD : <<private, static, final>>	String	This is the private, hardcoded password that is to be used for bootstrapping the API with an administrative super-user that is used by the TestDriver for loading all the Authentication catalog items
superUser : <<private, static, final>>	User	When getInstance() is called, it also calls createSuperUser(), which will store the superUser here
entitlements : <<private>>	Set<Entitlement>	The unique top-level Entitlements contained in the Authentication catalog; each Entitlement may only be declared at the top-level once, but may be nested arbitrarily deeply in other Entitlements.
services : <<private>>	Set<Service>	The unique Services contained in the Authentication catalog.
users : <<private>>	Set<User>	The unique registered Users contained in the Authentication catalog.
instance : <<private, static>>	IAuthenticationServiceAPI	Singleton instance of the AuthenticationServiceAPI; returned by calling getInstance()

AuthenticationImporter, extends Importer

Provides a single public method for handling the creation of new Authentication Service items, which include:

- User

- Role
- Permission
- Service

Requires a CSV file be passed that contains the definitions of all items. Also allows for defining the Permissions that comprise a Role, adding child Roles to Roles, adding Permissions to Services, and adding Credentials to Users and granting Roles to Users.

Methods

Method Name	**Signature**	**Description**
importAuthenticationFile : <<static>>, throws ImportException, ParseException, AccessDeniedException	(tokenID : String, filename : String) : void	Public method for importing Authentication items into the Authentication Service catalog, including Services, Roles, Permissions, and Users, and setting all appropriate attributes on those objects.
defineService : <<private, static>>, throws ParseException	(tokenID : String, authenticationData : String[]) : void	Creates services and adds them to the AuthenticationServiceAPI. The format of each element in authenticationData should be: • ***define_service*** • service ID • service name • service description
definePermission : <<private, static>>, throws ParseException	(tokenID : String, authenticationData : String[]) : void	Creates permissions and adds them to the AuthenticationServiceAPI. The format of each element in authenticationData should be: • ***define_permission*** • service ID • permission id • permission name • service description
defineRole : <<private, static>>, throws ParseException	(tokenID : String, authenticationData : String[]) : void	Creates roles and adds them to the AuthenticationServiceAPI. The format of each element in authenticationData should be: • ***define_role*** • role id • role name • role description
createUser : <<private, static>>, throws ParseException	(tokenID : String, authenticationData : String[]) : void	Creates registered users and adds them to the AuthenticationServiceAPI. The format of each element in authenticationData should be: • ***create_user*** • user id • user name
addEntitlementToRole : <<private, static>>, throws ParseException	(tokenID : String, authenticationData : String[]) : void	Adds Entitlements (which may be actually either Roles or Permissions) to Roles. The format of each element in authenticationData should be: • ***add_entitlement_to_role*** • role id • entitlement id

addCredentialToUser : <<private, static>>, throws ParseException	(tokenID : String, authenticationData : String[]) : void	Adds Credentials to users, which are comprised of a username and password (hashed). Users can have multiple sets of Credentials, which will allow them to login to the AuthenticationServiceAPI with different usernames and password. The format of each element in authenticationData should be: • ***add_credential*** • user id • username • password
addEntitlementToUser : <<private, static>>, throws ParseException	(tokenID : String, authenticationData : String[]) : void	Adds Entitlements to users, which can be either a Permission or Role. The format of each element in authenticationData should be: • ***add_entitlement_to_user*** • user id • entitlement id

IAuthenticationVisitor <<interface>>

Interface for defining the visit methods that the implementing class must overwrite; follows the Visitor pattern.

Methods

Method Name	**Signature**	**Description**
visitEntitlement	(entitlement : Entitlement) : String	Visits an Entitlement object and returns a string representing its properties. For Role objects, will use the RoleIterator to iterate over all the child Roles and Permissions and include them in the returned string.
visitPermission	(permission : Permission) : String	Visits a Permission object and returns a string representing its properties.
visitRole	(role : Role) : String	Visits an Role object and returns a string representing its properties. Will use the RoleIterator to iterate over all the child Roles and Permissions and include them in the returned string.
visitService	(service : Service) : String	Visits a Service and prints out the salient properties of the object.
visitUser	(user : User) : String	Visits a User and prints out the salient properties of the object.
visit	(item : Object) : String	Calls the appropriate visit* method based on the object type passed.

AuthenticationVisitor, implements IAuthenticationVisitor

Concrete implementation of IAuthenticationVisitor. Used to aid in building up a printable inventory of the IAuthenticationServiceAPI to list out all the Services, Entitlements (which are subclassed as Roles and Permissions), and Users. This class is a primary actor in the Visitor pattern usage for building up a printable inventory of Authentication items.

Methods

Method Name	**Signature**	**Description**
getItemProperties <<private>>	(item : Item) : String	Helper method to retrieve the printable properties of all classes that inherit from Item.

IAuthenticationVisitable <<interface>>

Interface that marks implementing classes as able to accept an IAuthenticationVisitor for building up a printable product inventory.

Methods

Method Name	**Signature**	**Description**
acceptVisitor	(visitor : IAuthenticationVisitor) : String	Accept a visiting class for the purpose of building up a printable product inventory.

Item <<abstract>>, implements AuthenticationServiceAPI

Abstract class representing an authentication item (could be a Role, Permission, or Service) that is included in the Mobile Application Store's Authentication catalog. Attributes here are common to all items in the Authentication catalog, regardless of type. All authentication items (regardless of type) have the following attributes:

- must have an **ID** that is a unique GUID
- must have a **name**
- must have a **description**

Certain specific types of Authentication items may have additional required attributes. Each Item that lives in the Authentication catalog is a unique item. Authentication items may be added to the authentication catalog and made returnable by calling IAuthenticationServiceAPI.getInventory().

Methods

Method Name	**Signature**	**Description**
validateItem <<static>>	(content : Item) : boolean	Public static method that checks that all required fields are set, and that all authentication item values are valid.

Properties

Property Name	Type	Description
id	String	A unique string identifier for each authentication item; should be a GUID if implementing object is a User
name	String	Name of the authentication item
description	String	A brief description of what this authentication item is and it's features.

Entitlement <<abstract>>, implements Item

Abstract parent class for Role and Permission. Since Entitlements can be added to java.util.Collections, includes methods for overriding hashCode() and equals(). Parent class of Permission and Role.

Methods

Method Name	Signature	Description
acceptVisitor	(visitor : IAuthenticationVisitor) : String	Accepts a visitor object for the purposes of building up an inventory of items in the AuthenticationService. Returns a string representing the properties of the object.

Service, extends Item, implements IAuthenticationVisitable

Services represent the major functional areas of the Mobile Application Store, which currently includes the AuthenticationServiceAPI, CollectionServiceAPI, and ProductAPI. Primarily marker classes for logical groupings of child Permissions, Services don't currently exhibit much behavior.

Methods

Method Name	Signature	Description
addPermission	(permission : Permission) : void	Adds a single Permission to the Service
addPermissions	(permissions : Set<Permission>) : void	Adds a set of Permissions to the Service
acceptVisitor	(visitor : IAuthenticationVisitor) : void	Accepts a visitor object for the purposes of building up an inventory of items in the AuthenticationService.

Properties

Property Name	Type	Description
permissions	Set<Permission>	Child Permission objects of the Service

User, extends Item, implements IAuthenticationVisitable

Registered Users may call restricted interface methods on each of the IAuthenticationServiceAPI, IProductAPI, and ICollectionServiceAPI services, as long as the User has the appropriate Entitlements. Users may have multiple sets of Credentials, which mean they are able to log in with different sets of usernames and passwords. However, each user may only have a single AccessToken, the ID of which is passed around to the restricted interface methods to ensure the user is authorized to carry out the method being called.

Methods

Method Name	**Signature**	**Description**
addCredential	(credential : Credentials) : void	Adds a Credential to the user account
addCredential	(username : String, password : String) : void	Creates a new Credentials object and adds it to the User.
addEntitlement	(entitlement : Entitlement) : void	Adds the Entitlement (may be a Role or Permission) to the User.
hasPermission	(permissionID : String) : boolean	Checks to see if the User has an Entitlement that has the passed permissionID.
validatePassword	(password : String) : boolean	Validates that the password exists on the User's set of Credentials.

Properties

Property Name	**Type**	**Description**
credentials	Set<Credentials>	The set of Credentials (usernames, passwords) that the user may use for logging in
entitlements	Set<Entitlement>	The set of unique Entitlements (either Roles, Permissions, or both) that the user has access to
token	AccessToken	The AccessToken is what is checked for validity when the user calls restricted interface methods on any of the published APIs

Credentials

User objects may have multiple sets of credentials; a Credential essentially consists of the username and password combination that a User will use to login and logout of the IAuthenticationServiceAPI, which will create an AccessToken that they may use to carry out restricted interface methods. Users may then have multiple sets of usernames/passwords that they can use to log into the system.

Passwords are hashed using the helper class PasswordHash. At no time are the actual plain-text passwords stored on the object; only the password hash is saved on the object.

Properties

Property Name	Type	Description
username	String	The username that is used when logging into the IAuthenticationServiceAPI
passwordHash	String	The hashed password; used for authentication
passwordSalt	String	The unique salt that is applied when hashing a password; must be saved so that subsequent attempts to verify the password will generate the same hash

PasswordHash

This is an external, 3rd party supporting class that is used for the secure hashing of user passwords and is a helper for the Credentials class. This code is referenced at https://crackstation.net/hashing-security.htm which discusses how to securely hash passwords, and provides this free Java implementation. As this is a 3rd party class dependency, it's methods and properties are not enumerated here.

AccessToken

AccessTokens are used for authorization purposes. Each User account will have an AccessToken created for them when they login to the site. Any future calls to restricted interface methods on either the IProductAPI or ICollectionServiceAPI must pass the AccessToken.id of the user's AccessToken in order to proceed. AccessTokens have a 1 hour default timeout, but successive calls to restricted interface methods on any of the supported APIs will increment this expiration time. Based on my understanding of the requirements, each user shall only have 1 AccessToken.

Properties

Property Name	Type	Description
id	String	Unique identifier for the AccessToken; form is a GUID
userID	String	ID of the user object that owns this AccessToken
expirationTime	Date	When the AccessToken expires and is no longer valid for authentication purposes; defaults to 1 hour when creating new AccessTokens
lastUpdated	Date	Each time a restricted interface method is called and the ID of this AccessToken is passed, this value is updated to the current time and the expirationTime is also updated to be 1 hour into the future

Role, extends Entitlement, implements IAuthenticationVisitable

Roles represent aggregations of Permissions and other Roles. A Role defines "who a user is"; for example, a role called "authentication_admin_role" would contain all the Permissions that collectively define all the restricted actions that only an AuthenticationServiceAPI administrator could perform.

Methods

Method Name	Signature	Description
getIterator	() : RoleIterator	Returns the iterator for the current Role. The iterator also follows the Singleton pattern; once the iterator has been declared and initialized, the already-declared one will be returned. If the iterator has not been defined when getIterator() is called, a new RoleIterator will be created and initialized.
addChild	(entitlement : Entitlement) : void	Adds a child Entitlement to the Role
addChildren	(entitlements : Set<Entitlement>) : void	Adds multiple children Entitlements to the Role

Properties

Property Name	Type	Description
iterator	RoleIterator	Defines the iterator for the current role
children	List<Entitlement>	Defines the child elements of the Role, which may be other Roles or Permissions.

RoleIterator, implements Iterator

Allows for the iteration of Role items, which may contain child Roles or Permissions. This iterator will traverse the Collectible depth-first.

Methods

Method Name	Signature	Description
Next	() : Entitlement	Traverse the Role and return the next item.
hasNext	() : Boolean	Does the iterator have any more elements to iterate over?

Properties

Property Name	Type	Description
itemStack	Stack<Entitlement>	Private stack is used to "flatten" the tree structure of the Role and return items in depth-first order.

current	Entitlement	Keeps a reference to the current item that the hidden internal iterator "pointer" is positioned over.

Permission, extends Entitlement, implements IAuthenticationVisitable

Permissions represent actions that registered Users may carry out on the IAuthenticationServiceAPI, IProductAPI, and ICollectionServiceAPI. Permissions ***must*** be added to Services, and may be added to Roles. A User may be granted a Permission either directly or through a Role. Permission IDs should correspond to the enum values in PermissionType. The Permission class actually has no unique methods or properties to it that it does not already inherit or override from Entitlement and IAuthenticationVisitable.

PermissionType

Enumeration of all Permission types that the IAuthenticationServiceAPI uses, as well as the Permission types that are used by the IProductAPI and ICollectionServiceAPI. These types are used in the authentication.csv file to load the authentication catalog, and also define the permissions that are required for carrying out restricted interface methods on the ProductAPI, CollectionServiceAPI, and AuthenticationServiceAPI.

Methods

Method Name	**Signature**	**Description**
getPermissionName	() : String	Returns the String name of the enumeration property.

Properties

Property Name	**Type**	**Description**
DEFINE_SERVICE	String	used by the IAuthenticationServiceAPI; permissionName is: "*define_service*"
DEFINE_PERMISSION	String	used by the IAuthenticationServiceAPI; permissionName is: "*define_permission*"
DEFINE_ROLE	String	used by the IAuthenticationServiceAPI; permissionName is: "*define_role*"
ADD_ENTITLEMENT_TO_ROLE	String	used by the IAuthenticationServiceAPI; permissionName is: "*add_entitlement_to_role*"
CREATE_USER	String	used by the IAuthenticationServiceAPI; permissionName is: "*create_user*"
ADD_CREDENTIAL_TO_USER	String	used by the IAuthenticationServiceAPI; permissionName is: "*add_credential*"
ADD_ENTITLEMENT_TO_USER	String	used by the IAuthenticationServiceAPI; permissionName is: "*add_entitlement_to_user*"

CREATE_COLLECTION	String	used by the ICollectionServiceAPI; permissionName is: "create_collection"
ADD_CONTENT	String	used by the ICollectionServiceAPI; permissionName is: "*add_content*"
DEFINE_COLLECTION_SEARCH_CRITERIA	String	used by the ICollectionServiceAPI; permissionName is: "*define_collection_dynamic_search*"
CREATE_PRODUCT	String	used by the IProductAPI; permissionName is: "*create_product*"
CREATE_COUNTRY	String	used by the IProductAPI; permissionName is: "*create_country*"
CREATE_DEVICE	String	used by the IProductAPI; permissionName is: "*create_device*"

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

The original requirements document called for using a factory method pattern to deal with creating all instances of authentication objects where appropriate; this design did not follow that recommendation. The way in which this implementation addresses authentication object creation is to largely allow clients of the IAuthenticationServiceAPI to construct objects as desired (whether Users, Services, Roles, Permissions), and then pass those objects to the AuthenticationServiceAPI implementation and allow it to handle the management of the objects (add them to the set of users, roles, services, etc.). After watching the class lecture video on Modularity it became clearer to me that this may not be the best design because it forces clients to know and understand what these authentication items (Users, Services, Roles, Permissions) and construct them prior to being able to tell the AuthenticationServiceAPI about them. If there were more time I may have refactored this out and instead used a Factory Method pattern to handle the object creation, but this would involve some complexity because the objects that it would be creating exist at different levels of the inheritance hierarchy.

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 authentication.csv
countries.csv devices.csv products.csv queries.csv collections.csv
```