Mobile Application Store Collection Service Design Document

Date:	11/20/2013
Author:	Lisa "Lex" Loren
Reviewers:	Frank O'Connor

Introduction

This document defines the design for the Mobile Application Store Authentication Service.

Overview

The Mobile Application Store provides developers and content producers with a way to distribute downloadable applications, wallpapers, and ringtones to consumers. The store makes both free and premium content available for download.

The Collection Service offers administrators an easy to way to offer targeted collections of content for download. It makes the Mobile Application Store easier for consumers to navigate, and provides them with new ways of finding content.

Both these services include methods that should not be accessible to casual end users. To protect the integrity of the system while still allowing administrators and software developers to make appropriate changes, both of these services leverage the features of the Authentication Service.

The Authentication service allows its users (primarily the administrators of other services) to log in with a username and password, and then generate user-specific Access Tokens. These Access Tokens can then be passed to other services, which will query the Authentication Service to determine which permissions the Access Token grants.

This document describes the specific requirements of the Authentication Service and summarizes a design which fulfills them. It includes uses case diagrams, a class diagram, a class dictionary, a sequence diagram, and an activity diagram. In addition, it proves a summary of the ways in which the design meets the requirements, a section on possible risks, and a section on testing.

Requirements

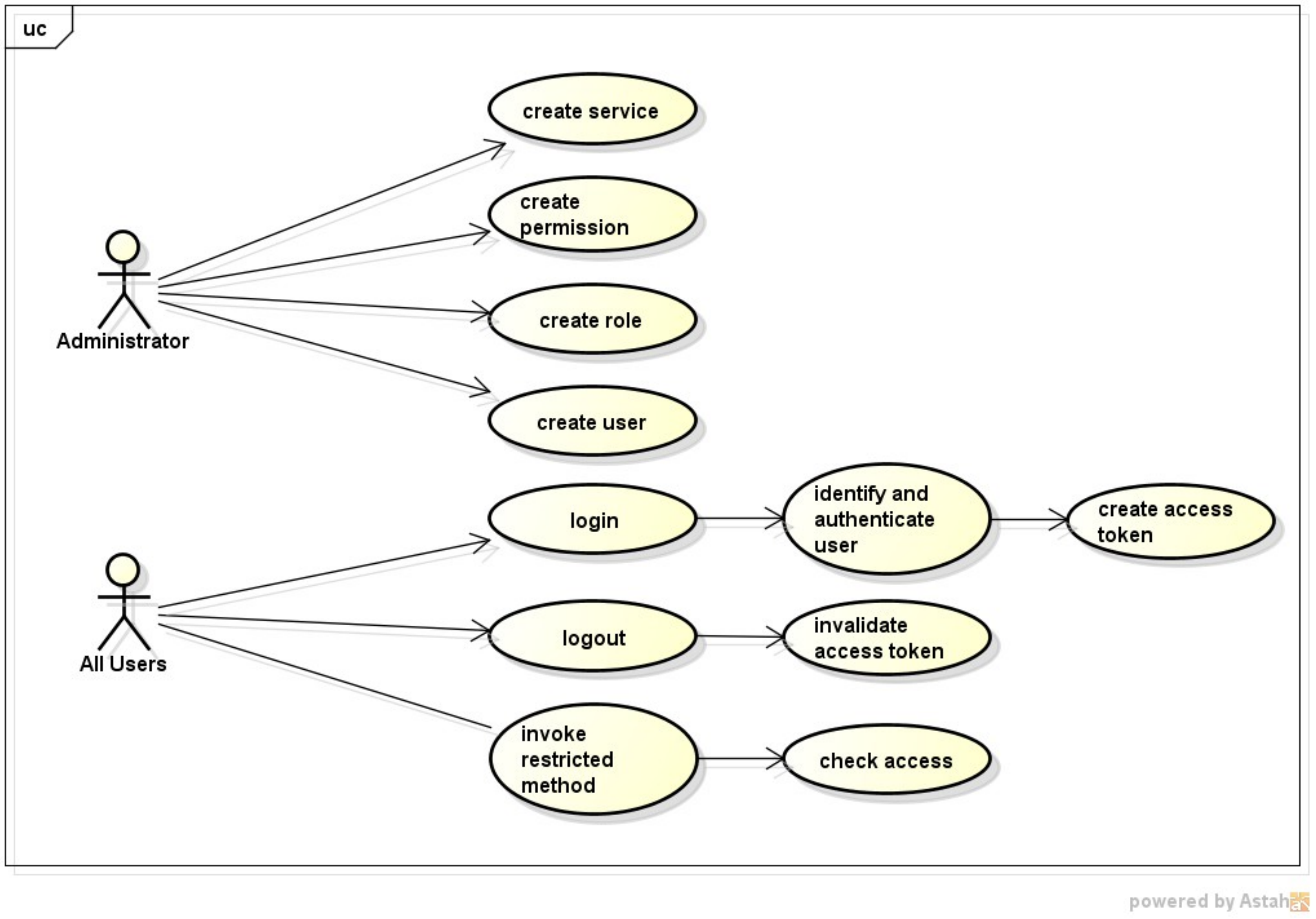
Allow administrators to do the following: (restricted access)
Create representation of Services
Create Permissions specific to a service
Create Roles, which are groups of permissions.
Create Users
Create sets of credentials for users
Assign roles or permissions to users

Allow users to do the following:
Log in to the Authentication Service
Request an AccessToken with their credentials
Log out, deactivating the AccessToken

Allow client services to do the following:
Determine whether an AccessToken gives a certain permission.

Additionally, the design should do the following:
Use the Visitor pattern to provide an inventory of Users, Services, Roles, and Permissions

Use Cases



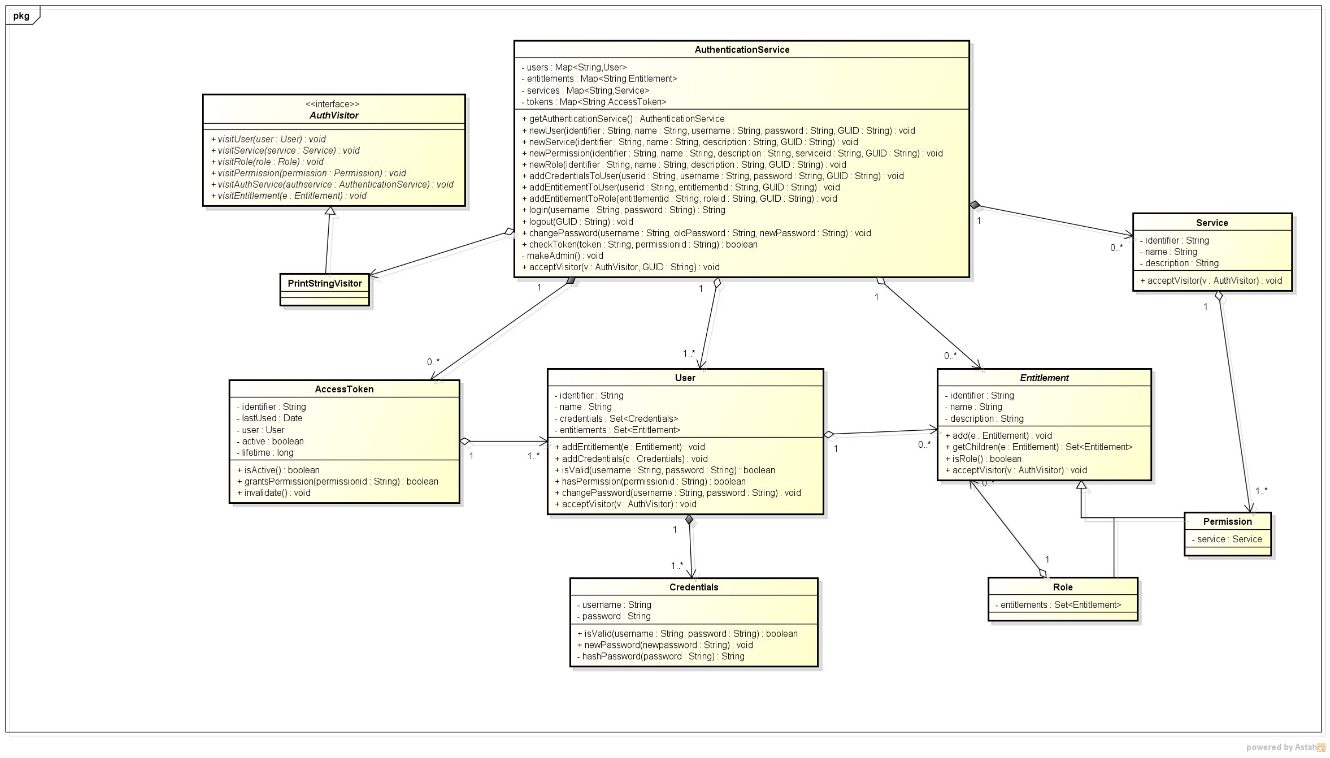
[Figure: use cases for the Authentication Service.]

All users, including Administrators, log in to create access tokens.
All users, including Administrators, log out, invalidating their access tokens.
All users, including Administrators, invoke restricted methods and have their access tokens checked.
Administrators of the Authentication Service create services, create permissions, create roles, and create users.

Implementation

Class Diagram

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


pkg
<<interface>>
AuthVisitor
+ visitUser(user : User) : void
+ visitService(service : Service) : void
+ visitRole(role : Role) : void
+ visitPermission(permission : Permission) : void
+ visitAuthService(authservice : AuthenticationService) : void
+ visitEntitlement(e : Entitlement) : void
PrintStringVisitor
AuthenticationService
- users : Map<String,User>
- entitlements : Map<String,Entitlement>
- services : Map<String,Service>
- tokens : Map<String,AccessToken>
+ getAuthenticationService() : AuthenticationService
+ newUser(identifier : String, name : String, username : String, password : String, GUID : String) : void
+ newService(identifier : String, name : String, description : String, GUID : String) : void
+ newPermission(identifier : String, name : String, description : String, serviceid : String, GUID : String) : void
+ newRole(identifier : String, name : String, description : String, GUID : String) : void
+ addCredentialsToUser(userid : String, username : String, password : String, GUID : String) : void
+ addEntitlementToUser(userid : String, entitlementid : String, GUID : String) : void
+ addEntitlementToRole(entitlementid : String, roleid : String, GUID : String) : void
+ login(username : String, password : String) : String
+ logout(GUID : String) : void
+ changePassword(username : String, oldPassword : String, newPassword : String) : void
+ checkToken(token : String, permissionid : String) : boolean
- makeAdmin() : void
+ acceptVisitor(v : AuthVisitor, GUID : String) : void
Service
- identifier : String
- name : String
- description : String
+ acceptVisitor(v : AuthVisitor) : void
AccessToken
- identifier : String
- lastUsed : Date
- user : User
- active : boolean
- lifetime : long
+ isActive() : boolean
+ grantsPermission(permissionid : String) : boolean
+ invalidate() : void
User
- identifier : String
- name : String
- credentials : Set<Credentials>
- entitlements : Set<Entitlement>
+ addEntitlement(e : Entitlement) : void
+ addCredentials(c : Credentials) : void
+ isValid(username : String, password : String) : boolean
+ hasPermission(permissionid : String) : boolean
+ changePassword(username : String, password : String) : void
+ acceptVisitor(v : AuthVisitor) : void
Entitlement
- identifier : String
- name : String
- description : String
+ add(e : Entitlement) : void
+ getChildren(e : Entitlement) : Set<Entitlement>
+ isRole() : boolean
+ acceptVisitor(v : AuthVisitor) : void
Permission
- service : Service
Credentials
- username : String
- password : String
+ isValid(username : String, password : String) : boolean
+ newPassword(newpassword : String) : void
- hashPassword(password : String) : String
Role
- entitlements : Set<Entitlement>
1
0..*
1
1..*
1
0..*
1
0..*
1
1..*
1
1..*
1
0..*
1
1..*
0..*
1
powered by Astah

[Figure: class diagram for the Authentication Service.]

Class Dictionary

AuthenticationService

The AuthenticationService is a singleton object, holding all data in memory. It is, in part, an interface between client users and the rest of the classes in the authentication package. All actions on any classes in the authentication package go through the AuthenticationService class. (The visitor classes are currently an exception.) This means that permissions need only be checked at a single point.

All restricted methods of the AuthenticationService use the AuthenticationService Access Token and verification methods. When the AuthenticationService singleton is first instantiated, it creates a Service object for its own service, all permissions for its service, and creates a user with those permissions. This user can login with the username "root" and the password "rootpassword". It is recommended that the first user of the service change the password using the changePassword() method.

The service uses three exceptions: AuthenticationServiceException, InvalidAccessTokenException AccessDeniedException, and AuthenticationException. AuthenticationServiceExceptions are thrown only during maintenance of the service, generally when the client gives invalid input for object creation. AuthenticationExceptions are thrown when a user cannot be logged in, generally due to a bad username or password. InvalidAccessTokenExceptions are thrown when the service encounters an access token it does not recognize. AccessDeniedExceptions are thrown when an access token does not grant the requested permission, is expired, or when the user has logged out.

Method	**Signature**	**Description**
getAuthenticationService	() : AuthenticationService	Returns the singleton instance of the AuthenticationService class.
newUser	(identifier : String, name : String, username : String, password : String, GUID : String) : void	Creates a new user and set of credentials. (Restricted Access.)
newService	(identifier : String, name : String, description : String, GUID : String) : void	Creates a new service. (Restricted Access.)
newPermission	(identifier : String, name : String, description : String, serviceid: String, GUID : String) : void	Creates a Permission tied to the service specified by the serviceid String. (Restricted Access.)
newRole	(identifier : String, name : String, description : String, GUID : String) : Role	Creates a new Role. (Restricted Access.)
addCredentialsToUser	(userid : String, username:String, password : String, GUID : String) : void	Adds the specified set of credentials to the user. (Restricted Access.)
addEntitlementToUser	(userid : String, entitlementid : String, GUID : String) : void	Gives the specified user the specified role or permission. (Restricted Access.)

addEntitlementToRole	(entitlementid : String, roleid : String, GUID : String) : void	Adds the specified role or permission to the specified role. (Restricted Access.)
login	(username : String, password : String) : String	If login is successful (the username and password are found/correct) returns a GUID String and creates a corresponding Access Token tied to the user. The login method iterates through the set of Users, calling the isValid() method of each.
logout	(GUID : String) : void	Marks the given Access Token as invalid/inactive.
changePassword	(username : String, oldPassword : String, newPassword : String) : void	If the username and password are found/correct, changes the password for that set of credentials to the new password.
checkToken	(token : String, permissionid : String) : boolean	Returns true if the given Access Token is active AND allows access to the given permissions.
makeAdmin	() : void	Instantiates the root user, service, and service-specific permissions.
acceptVisitor	(v : AuthVisitor) : void	Accepts a visitor.

Association Name	**Type**	**Description**
users	Map<String,User>	All users in the system, mapped to their identifiers
entitlements	Map<String,Entitlement>	All roles and privileges in the system, mapped to their identifiers
services	Map<String,Service>	All services in the system, mapped to their identifiers
tokens	Map<String, AccessToken>	All AcessToken objects in the system, mapped to their GUIDs.

Service

A Service object represents a single service which uses features of the Authentication Service.

Method	**Signature**	**Description**
acceptVisitor	(v : AuthVisitor) : void	Accepts a visitor.

Property Name	Type	Description
identifier	String	A unique identifier, user-specified.
name	String	The name
description	String	The description

Entitlement

The Entitlement class is the abstract superclass of the Permission and Role classes. It uses the composite pattern, allowing a Role object to easily contain both Permissions and other roles. Throws AuthenticationServiceExceptions when Role-specific methods are invoked on Permissions.

Method	Signature	Description
add	(e : Entitlement, GUID:String) : void	Adds a Permission or Role to a Role. (Restricted Access.)
getChildren	(e : Entitlement) : void	Returns a list of all child Entitlements, if applicable.
isRole	() : boolean	Returns true f the object is a Role, false if not.
acceptVisitor	(v : AuthVisitor) : void	Accepts a visitor.

Property Name	Type	Description
identifier	String	A unique identifier, user-specified.
name	String	The name
description	String	The description

Permission

A subclass of the Entitlement class. Each Permission belongs to exactly one Service. A permission can be used by the outside service (corresponding to the permission's Service) to grant the ability to perform one or more restricted methods. In practice, it is best for individual Permissions to correspond to either one method, or a few tightly-linked methods, although this is not enforced by the Authentication Service. Permissions can easily be grouped together into Roles. Throws AuthenticationServiceExceptions when Role-specific methods are invoked

Association Name	Type	Description
service	Service	The service this permission applies to.

Role

A subclass of the Entitlement class. Roles can store any combination of Permissions or other Roles. Roles are not tied to a particular Service, and can contain permissions from multiples services.

Association Name	**Type**	**Description**
entitlements	Set<Entitlements>	A collection of entitlements, potentially including both permissions and other roles.

User

Each User can have any number of Entitlements. Users may also posses multiple sets of credentials. Note: although similar, the user's **name**, the user's **identifier**, and a **username** associated with a credential object are not interchangeable.

Method	**Signature**	**Description**
addEntitlement	(e : Entitlement) : void	Gives the user the specified entitlement.
addCredentials	(username : String, password : String) : void	Adds a set of credentials to the user. Checks to ensure that the username is not already in use.
isValid	(username : String, password : String) : boolean	Returns true if the User has a set of credentials that match the given username and password, false if not. This method simply calls the isValid() method of each credential object associated with it. Throws an AuthenticationException if the username is found but the password does not match.
hasPermission	(permissionid : String) : boolean	Returns true if the user has been granted the permission corresponding to the given permissionid.
changePassword	(username : String, password : String) : void	If the username and password are found/correct, changes the password for that set of credentials to the new password.
acceptVisitor	(v : AuthVisitor) : void	Accepts a visitor.

Association Name	**Type**	**Description**
credentials	Set<Credentials>	Set of all credentials belonging to the user.

entitlements	Set<Entitlement>	Set of all entitlements given to the user.

Property Name	**Type**	**Description**
identifier	String	A unique identifier, user-specified.
name	String	The display name

Credentials

Credentials represent a username/password pair. Each set of credentials is owned by exactly one user. Passwords are passed to the credential object in plaintext and stored as a hash.

Method	**Signature**	**Description**
isValid	(username : String, password : String) : boolean	Returns true if the username is found and a hash of the password matches the stored hash. Returns false if the username is not found. Throws an AuthenticationException if the username is found but the password does not match.
newPassword	(newpassword : String) : void	Changes the password for this set of credentials to the new password.
hashPassword	(password : String) : String	

Property Name	**Type**	**Description**
username	String	The username.
password	String	A hash of the password.

AccessToken

The AccessToken object contains both a String access token/GUID and its associated metadata. Note that the AccessToken does not actually store a list of permissions. Rather, they store a reference to the owning user. When the token is used, the user's permissions are checked dynamically.

Method	**Signature**	**Description**
isActive	() : boolean	Returns true if the token is marked as active and if the time elapsed since it was last used is less than the allowed lifetime. If the token has expired, marks it as

		invalid before returning false.
grantsPermission	(permissionid : String) : boolean	Returns true if the token grants the permission corresponding to the permissionid AND is currently valid. Throws an InvalidAccessTokenException if the Access Token has expired.
invalidate	() : void	Marks the token as invalid.

Association Name	**Type**	**Description**
user	User	The User which owns the token.

Property Name	**Type**	**Description**
identifier	String	A randomly-generated UUID. This identifier is the GUID String referenced elsewhere in this document.
lastUsed	Date	A timestamp of the last time the token was used.
active	boolean	True if the token is active and can be used again.
lifetime	long	In milliseconds, the maximum amount of time that this token can remain unused before it is marked inactive.

AuthVisitor

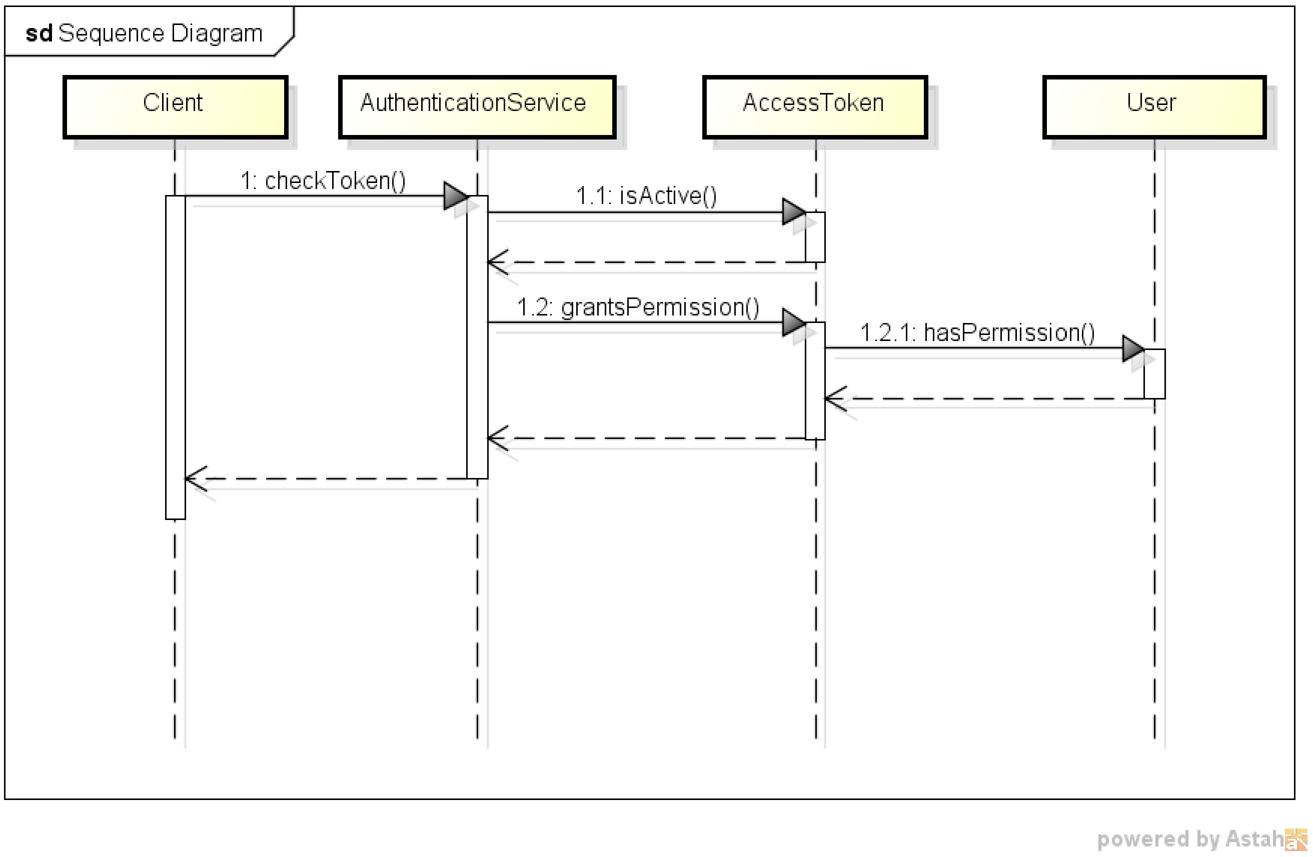
The AuthVisitor class is an interface that plays an important role in the implementation of the visitor pattern.

Method	**Signature**	**Description**
visitUser	(user : User) : void	An abstract method for visiting a User.
visitService	(service : Service) : void	An abstract method for visiting a Service.
visitRole	(role : Role) : void	An abstract method for visiting a Role.
visitPermission	(permission : Permission) : void	An abstract method for visiting a Permission.
visitAuthService	(authservice : AuthenticationService) : void	An abstract method for visiting the AuthenticationService.

PrintStringVisitor

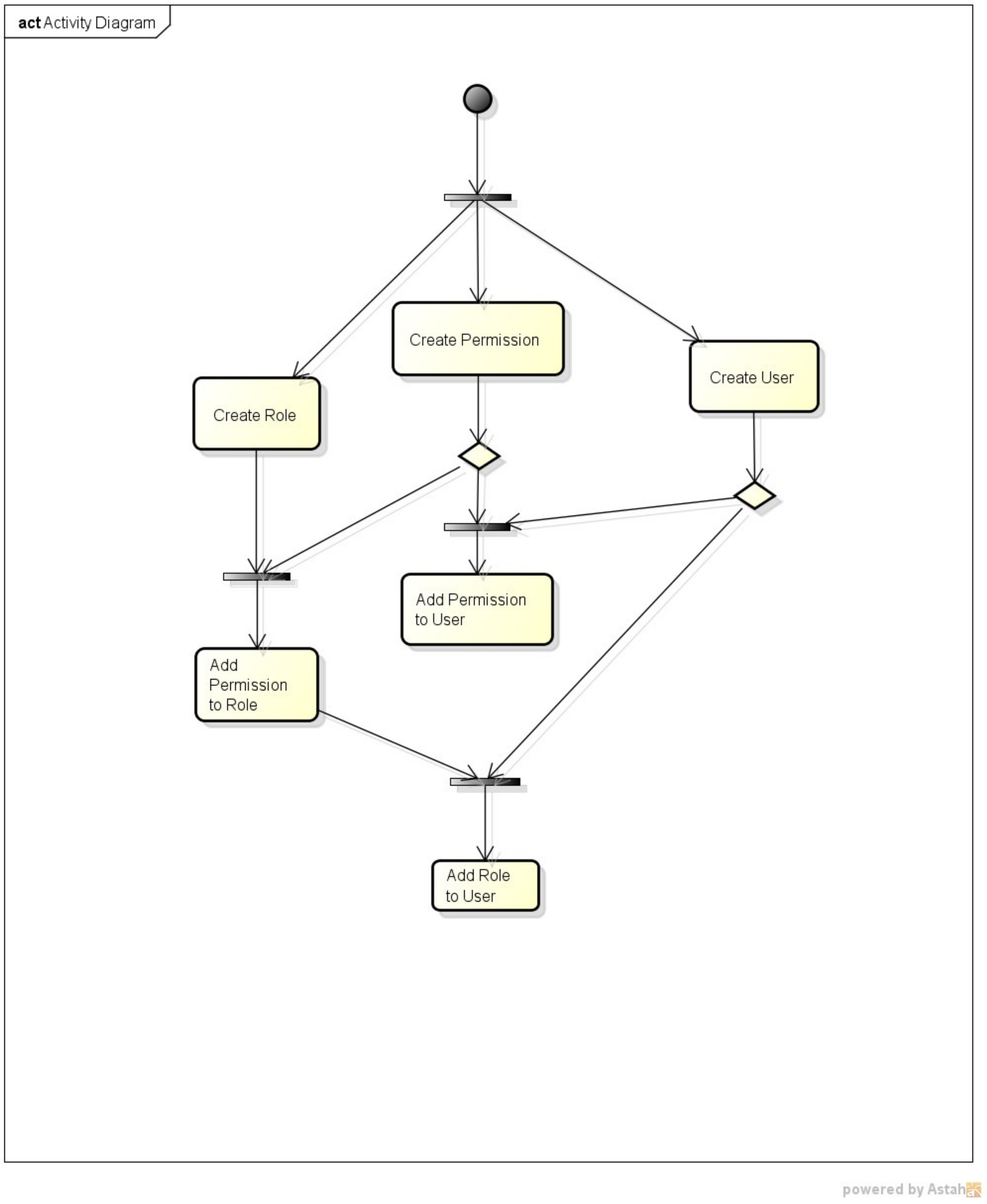
The PrintStringVisitor class is a subclass of AuthVisitor. Its various visitXXX() methods work together to print an inventory of all users, services, roles, and permissions.

Sequence Diagram



[Figure: a sequence diagram showing the process of checking a user's Access Token to verify that thy have a certain permission.

Activity Diagram



[Figure: Activity Diagram showing the process of creating users, permissions, and roles.]

Note 1: At every step of the process, the program checks to ensure that the user has appropriate permissions.

Note 2: The Authentication Service does not actually prevent Administrators from assigning empty Roles to users. However, this likely increases the chances of permissions mistakes down the line, and so is not recommended.

Requirements Compliance

All restricted access methods within the Authentication Service require an Access Token.

Allow administrators to do the following:
- Create representation of Services: the AuthenticationService method newService()
- Create Permissions specific to a service: the AuthenticationService method newPermission()
- Create Roles, which are groups of permissions: the AuthenticationService method newRole()
- Create Users: the AuthenticationSerice method newUser()
- Create sets of credentials for users: the AuthenticationService method addCredentialsToUser() or the User method addCredentials()
- Assign roles or permissions to users: the AuthenticationService method addEntitlementToUser() or the User method addEntitlement()

Allow users to do the following:
- Log in to the Authentication Service: the AuthenticationService method login()
- Request an AccessToken with their credentials: provided automatically with the the AuthenticationService method login()
- Log out, deactivating the AccessToken: the AuthenticationService method logout()

Allow client services to do the following:
- Determine whether an AccessToken gives a certain permission: the AuthenticationService method checkToken()

Additionally, the design should do the following:
- Use the Visitor pattern to provide an inventory of Users, Services, Roles, and Permissions: the AuthenticationService method printInventory(), the AuthVisitor and PrintStringVisitor classes, and the acceptVisitor() methods on several classes.

Testing

TBA: Instructions for compiling.

Usage: java TestDriver authentication.csv countries.csv devices.csv products.csv collections.csv

Note that the authentication.csv used here is different than the one provided for testing. The authentication_service service and all associated permissions are pro-loaded, and attempting to load them again will result in errors.

The TestDriver loads all relevant authentication data, logs in to the appropriate users, and imports all product and collection data. It then tests a few functions of the Authentication Service, printing errors as appropriate. Tests and their result should be self-explanatory.

Finally, the TestDriver creates a PrintStringVisitor and uses it to print an inventory of all objects in the Authentication Service.

Risks

The current password processing and storage strategies may not meet current security standards, especially in a distributed environment. The password hashing algorithm in particular is NOT secure. Care must be given to ensure secure transmission and storage of passwords and Access Tokens.

A robust system would also require more functionality, including the ability to remove permissions from Users and Roles, and the ability to the delete Credentials and Users.