House Mate Entitlement Design Document

Date: 11/6/2015
Author: Gerald Trotman
Reviewer(s): Professor Gieseke, Takayuki Lida

Introduction

This document defines the design for the House Mate Entitlement Service API, which is a core component of the House Mate Service. This document breaks the organization down into the following:

Overview

The House Mate Service requires ongoing entitlement in order to regulate the commands and actions of various Resources (Sensors and Appliances) being called upon by different Users (Occupants). There is also this notion of an Administrator as well. And in order to enable the ability to carry out certain interfaces that have restrictions, a system for authenticating users and authorizing users to conduct corresponding actions is required. The EntitlementServiceAPI provides methods to create User accounts, to create Resources, Permissions, and Roles, define the Permissions on Resources and Roles, and further grant Roles and Permissions to those User accounts.

“Permissions” define a restricted interface action that a User may perform; let’s say, “create_resource_role” is a permission that allows Users that are granted it to call the HouseMateModelService method that imports resources. Permissions are what you can do.

“Roles” are used to collect sets of Permissions into a logical group. These groups of related Permissions collectively define who you are when a Role (or multiple Roles) are granted to a User.

“Resources” define the major functional areas of the House Mate Service. Resources are Sensor and Appliances in our HouseMateModelServiceAPI that are controlled by our HouseMateControllerAPI.

Requirements

This section defines the requirements for the EntitlementServiceAPI system. In prior sprints, the HouseMateModelServiceAPI as well as the HouseMateControllerAPI hand waved the entitlement buy allowing for a GUID token to be passed as an authorization measure before allowing user to call restricted interface methods. The EntitlementServiceAPI will allow for Users to log into the system, be given Permissions and Roles, allow admins to define these Roles and Permissions as well as Resources, allowing for a robust way of ensuring that only allowed users can carry out certain actions in the House Mate system. Only Admins should be allowed to:

- Create Resources
- Create Permissions
- Create Roles
- Create Users

Admins and RegisteredUsers, implementing Authentication, should both be allowed to:

- Login
- Logout
- Invoke Restricted Access Methods

Users can have multiple sets of credentials; meaning, they can have more than one set of username/password combos to use the service. Users are given Entitlement, which is how they are given Permission to access restricted interface methods on House Mate APIs. However, despite having different Credentials, they only have 1 set of Entitlements via the AccessToken.

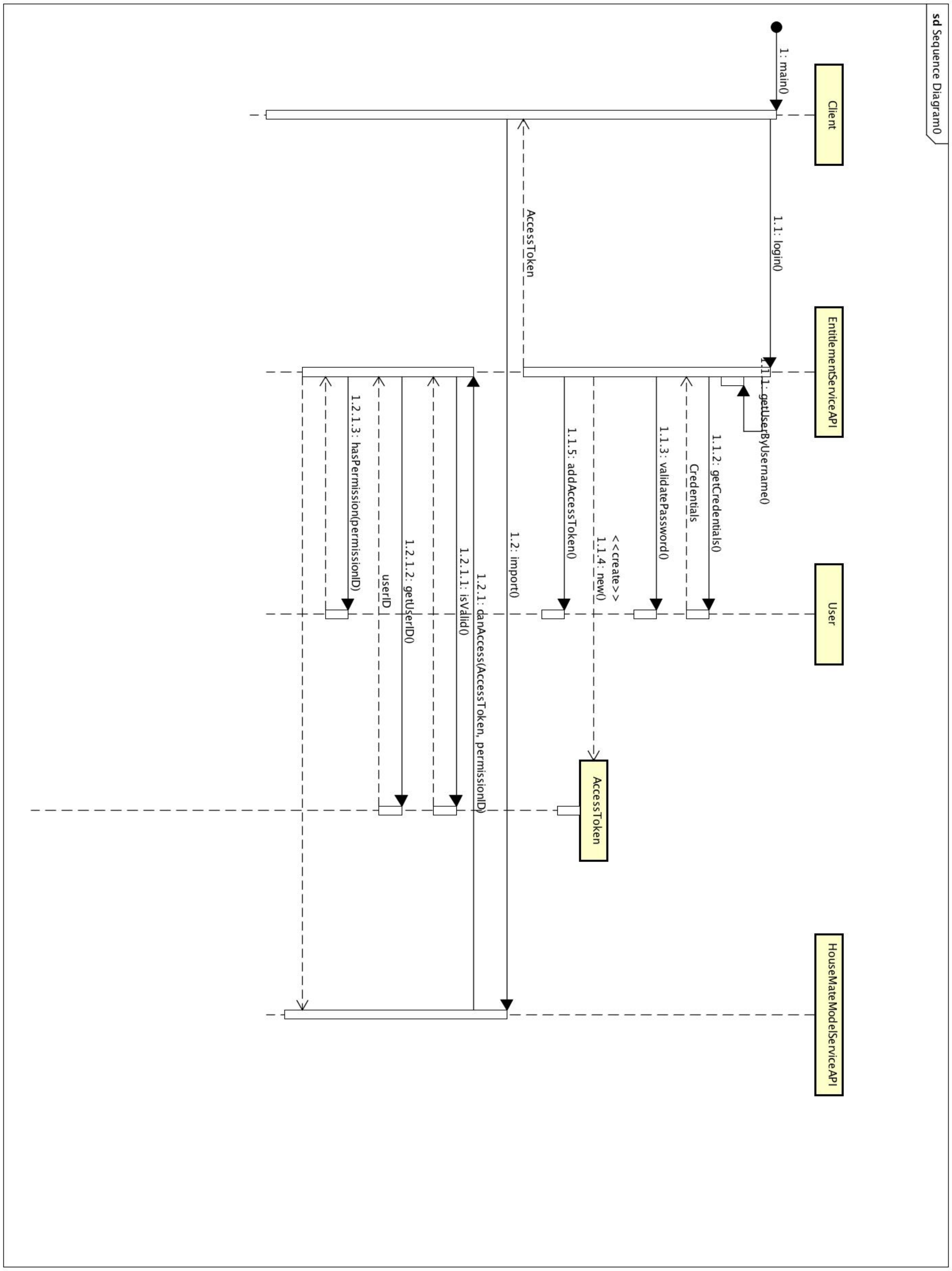
When a User calls a method that is a restricted interface on either the HouseMateModelServiceAPI, HouseMateControllerServiceAPI, or EntitlementServiceAPI, an AccessToken should be passed to the called method and validated against the EntitlementServiceAPI to ensure that only authorized users are allowed to call restricted interface methods. For more specifics see: *HouseMateEntitlementServicesRequirements.pdf*

Implementation

Sequence Diagram

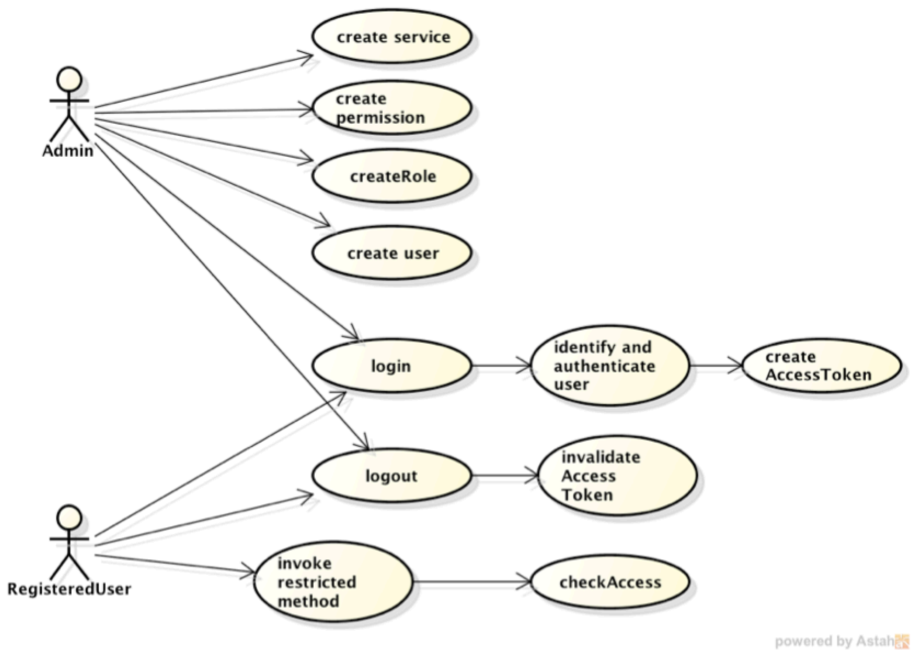
This sequence diagram shows how a client (in this case an occupant) might interact with the EntitlementServiceAPI to log into the system and then call a restricted method on the HouseMateModelServiceAPI.

House Mate Entitlement Service Design Diagram
CSCI E-97 Assignment #4



Use Cases

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



Create Resource: As you can see, only the Admin can create and define new Resources. Resources define the major areas of the House Mate Service that users may interact with (such as the HouseMateModelAPI, HouseMateControllerAPI, and EntitlementServiceAPI). This is a restricted interface and requires a valid Admin to be logged in before new Resources may be defined.

Create Permission: Only an Admin can create and define new Permissions. Permissions define the restricted actions that only Users with correct entitlement can perform in the system. This is a restricted interface and requires a valid Admin to be logged in before new Permissions can be defined.

Create Role: Only Admins can create and define new Roles. Roles may contain Permissions, and/or other Roles. Roles collect Permissions (and other Roles) to create definitions of what Users may be when they are given a "Role".

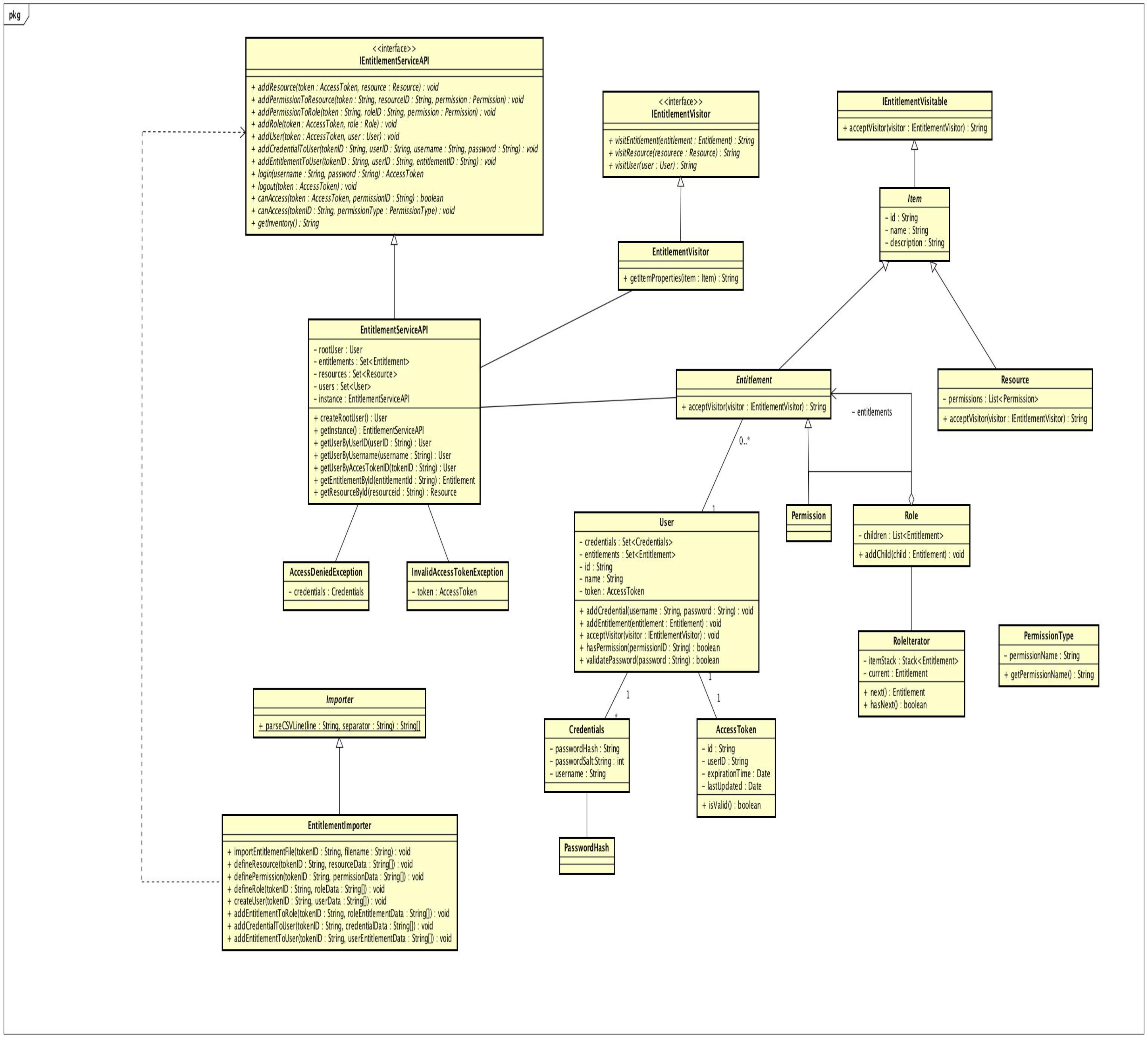
Create User: Only Admins can create and define new User accounts. Users may be granted specific Roles and Permissions. This is also a restricted interface and requires a valid Admin to be logged in before new Users may be defined.

Login: Both Admins and RegisteredUsers may login, which then will allow them to call the restricted interface methods on the House Mate Service that they have access to. Logging in entails identifying and authenticating the username and password, and obtaining an AccessToken for the User. When said User later makes calls to restricted interface methods on any of the HouseMateModelServiceAPI, HouseMateControllerServiceAPI, or EntitlementServiceAPI, they must pass the ID of their AccessToken, where their entitlements are checked to ensure they have the corresponding access. Once logged in, the user's AccessToken is only good for a period of time; if the user doesn't call any restricted interface methods before that time, then the AccessToken expires.

Logout: Both Admins and RegisteredUsers may logout of the system, which invalidates any of their current AccesTokens, and then would now allow them to call any restricted interface methods until successfully logging in again.

Class Diagram

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


pkg
<<interface>>
IEntitlementServiceAPI
+ addResource(token : AccessToken, resource : Resource) : void
+ addPermissionToResource(token : String, resourceID : String, permission : Permission) : void
+ addPermissionToRole(token : String, roleID : String, permission : Permission) : void
+ addRole(token : AccessToken, role : Role) : void
+ addUser(token : AccessToken, user : User) : 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
+ canAccess(token : AccessToken, permissionID : String) : boolean
+ canAccess(tokenID : String, permissionType : PermissionType) : void
+ getInventory() : String
<<interface>>
IEntitlementVisitor
+ visitEntitlement(entitlement : Entitlement) : String
+ visitResource(resource : Resource) : String
+ visitUser(user : User) : String
IEntitlementVisitable
+ acceptVisitor(visitor : IEntitlementVisitor) : String
Item
- id : String
- name : String
- description : String
EntitlementVisitor
+ getItemProperties(item : Item) : String
EntitlementServiceAPI
- rootUser : User
- entitlements : Set<Entitlement>
- resources : Set<Resource>
- users : Set<User>
- instance : EntitlementServiceAPI
+ createRootUser() : User
+ getInstance() : EntitlementServiceAPI
+ getUserByUserID(userID : String) : User
+ getUserByUsername(username : String) : User
+ getUserByAccesTokenID(tokenID : String) : User
+ getEntitlementById(entitlementId : String) : Entitlement
+ getResourceById(resourceid : String) : Resource
Entitlement
+ acceptVisitor(visitor : IEntitlementVisitor) : String
- entitlements
0..*
1
Resource
- permissions : List<Permission>
+ acceptVisitor(visitor : IEntitlementVisitor) : String
Permission
Role
- children : List<Entitlement>
+ addChild(child : Entitlement) : void
User
- credentials : Set<Credentials>
- entitlements : Set<Entitlement>
- id : String
- name : String
- token : AccessToken
+ addCredential(username : String, password : String) : void
+ addEntitlement(entitlement : Entitlement) : void
+ acceptVisitor(visitor : IEntitlementVisitor) : void
+ hasPermission(permissionID : String) : boolean
+ validatePassword(password : String) : boolean
AccessDeniedException
- credentials : Credentials
InvalidAccessTokenException
- token : AccessToken
RoleIterator
- itemStack : Stack<Entitlement>
- current : Entitlement
+ next() : Entitlement
+ hasNext() : boolean
PermissionType
- permissionName : String
+ getPermissionName() : String
Importer
+ parseCSVLine(line : String, separator : String) : String[]
1
*
1
1
1
Credentials
- passwordHash : String
- passwordSalt:String : int
- username : String
AccessToken
- id : String
- userID : String
- expirationTime : Date
- lastUpdated : Date
+ isValid() : boolean
PasswordHash
EntitlementImporter
+ importEntitlementFile(tokenID : String, filename : String) : void
+ defineResource(tokenID : String, resourceData : 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

Class Dictionary

This section specifies the class dictionary for the EntitlementServiceAPI. The primary classes should be defined within the package "cscie97.asn4.housemate.entitlement". 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.housemate.entitlement**

IEntitlementServiceAPI <<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 EntitlementServiceAPI. Administrators may create Resources, 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 (HousemateModelServiceAPI, HouseMateControllerServiceAPI) and in the EntitlementServiceAPI itself. Also provides a method for getting a string representation of the unique entitlement service inventory of Resources, Roles, Permissions, and Users.

Methods

Method Name	**Signature**	**Description**
addResource	(String : tokenID, Resource : resource) : void	Adds a new Resource definition to the catalog. Resources 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.
addRole	(String : tokenID, Role : role) : void	Adds a new Role. Roles may contain other Roles or Permissions to define logical groupings that correspond to types of Users that may use the EntitlementServiceAPI.
addPermissionToResource	(String : tokenID,	Adds a new Permission as a child of the

	String : ResourceID, Permission : permission) : void	Resource.
addPermissionToRole	(String : tokenID, String : roleID, String : permissionID) : void	Adds a new Permission 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, String : userID, String : entitlementID) : void	Adds a new Entitlement (can be a Role or Permission) to the User.
login : throws AccessDeniedException	(String : username, String : password) : AccessToken	Logs a user into the EntitlementService. 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.
canAccess	(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.
canAccess	(String : tokenID, PermissionType : permissionType) : boolean	Convenience method; intended to call alternate implementation of canAccess that accepts a permissionID as the second parameter.
getInventory	() : String	Returns a string representation of the entire Entitlement services, including Resources, Users, Roles, and Permissions. Uses the Visitor pattern to visit each Resource, User, Role, and Permission to inquire about the object's properties.

EntitlementServiceAPI, implements IEntitlementServiceAPI

Implements the IEntitlementServiceAPI 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 an initial account for usage in loading all of the items (Users, Roles, Permissions, Services); this is done in the private createUser() method, which is called automatically when obtaining an instance of the EntitlementServiceAPI class.

Methods

Method Name	**Signature**	**Description**
createRootUser : <<private>>	() : void	Instantiates the EntitlementServiceAPI with a "root user" that can be used to initially load the entitlement via the EntitlementImporter} This user will have all the Permissions of the EntitlementServiceAPI, and so will be able to call all the methods defined in IEntitlementServiceAPI.
getInstance : <<static, synchronized>>	() : IEntitlementServiceAPI	Returns a reference to the single static instance of the EntitlementServiceAPI. Will also create a "root user" account that can be used for loading all the Entitlement 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.
getResourceById : <<private>>	(resourceId : String) : Resource	Helper method to retrieve a Resource by its ID.

Properties

Property Name	**Type**	**Description**
ROOT_ADMINISTRATOR_USERNAME : <<private, static, final>>	String	This is the private, hardcoded username that is to be used for instantiating the API with an administrative root-user that is used by the TestDriver for loading all the Entitlement items
ROOT_ADMINISTRATOR_PASSWORD : <<private, static, final>>	String	This is the private, hardcoded password that is to be used for instantiating the API with an administrative root-user that is used by the TestDriver for loading all the Entitlement items

rootUser : <<private, static, final>>	User	When getInstance() is called, it also calls createRootUser(), which will store the rootUser here
entitlements : <<private>>	Set<Entitlement>	The unique top-level Entitlements contained in the Entitlement services; each Entitlement may only be declared at the top-level once, but may be nested arbitrarily deeply in other Entitlements.
resources : <<private>>	Set<Resource>	The unique Resources contained in the Entitlement services
users : <<private>>	Set<User>	The unique registered Users contained in the Entitlement services
instance : <<private, static>>	IEntitlementServiceAPI	Singleton instance of the EntitlementServiceAPI; returned by calling getInstance()

EntitlementImporter, extends Importer

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

- User
- Role
- Permission
- Resource

Requires a .text 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 Resources, and adding Credentials to Users and granting Roles to Users.

Methods

Method Name	**Signature**	**Description**
importEntitlementFile : <<static>>, throws ImportException, ParseException, AccessDeniedException	(tokenID : String, filename : String) : void	Public method for importing Entitlement items into the Entitlement Services, including Resources, Roles, Permissions, and Users, and setting all appropriate attributes on those objects.
defineResource : <<private, static>>, throws ParseException	(tokenID : String, entitlementData : String[]) : void	Creates services and adds them to the EntitlementServiceAPI. The format of each element in entitlementData should be: • ***define_resource***

		• resource ID • resource name • resource description
definePermission : <<private, static>>, throws ParseException	(tokenID : String, entitlementData : String[]) : void	Creates permissions and adds them to the EntitlementServiceAPI. The format of each element in entitlementData should be: • ***define_permission*** • resource ID • permission id • permission name • service description
defineRole : <<private, static>>, throws ParseException	(tokenID : String, entitlementData : String[]) : void	Creates roles and adds them to the EntitlementServiceAPI. The format of each element in entitlementData should be: • ***define_role*** • role id • role name • role description
createUser : <<private, static>>, throws ParseException	(tokenID : String, entitlementData : String[]) : void	Creates registered users and adds them to the EntitlementServiceAPI. The format of each element in entitlementData should be: • ***create_user*** • user id • user name
addEntitlementToRole : <<private, static>>, throws ParseException	(tokenID : String, entitlementData : String[]) : void	Adds Entitlements (which may be actually either Roles or Permissions) to Roles. The format of each element in entitlementData should be: • ***add_entitlement_to_role*** • role id • entitlement id
addCredentialToUser : <<private, static>>, throws ParseException	(tokenID : String, entitlementData : 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 EntitlementServiceAPI with different usernames and password. The format of each element in entitlementData should be: • ***add_credential*** • user id • username • password
addEntitlementToUser : <<private, static>>, throws ParseException	(tokenID : String, entitlementData : String[]) : void	Adds Entitlements to users, which can be either a Permission or Role. The format of each element in entitlementData should be: • ***add_entitlement_to_user*** • user id • entitlement id

IEntitlementVisitor <<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 a 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.
visitResource	(resource : Resource) : String	Visits a Resource and prints out the 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.

EntitlementVisitor, implements IEntitlementVisitor

Concrete implementation of IEntitlementVisitor. Used to aid in building up a printable inventory of the IEntitlementServiceAPI to list out all the Resources, 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 Entitlement 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.

IEntitlementVisitable <<interface>>

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

Methods

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

Item <<abstract>>, implements EntitlementServiceAPI

Abstract class representing an Entitlement item (could be a Role, Permission, or Resource) that is included in the House Mate Entitlement Services. Attributes here are common to all items in the Entitlement services, regardless of type. All Entitlement 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 Entitlement items may have additional required attributes. Each Item that lives in the Entitlement services is a unique item. Entitlement items may be added to the entitlement services and made returnable by calling IEntitlementServiceAPI.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 entitlement item values are valid.

Properties

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

Entitlement <<abstract>>, implements Item

Abstract parent class for Role and Permission. Parent class of Permission and Role.

Methods

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

Resource, extends Item, implements IEntitlementVisitable

Resources represent the major functional areas of the House Mate, which currently includes the EntitlementServiceAPI, HouseMateControllerServicesAPI, and HouseMateModelServiceAPI.

Methods

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

Properties

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

User, extends Item, implements IEntitlemetVisitable

Registered Users may call restricted interface methods on each of the IEntitlementServiceAPI, HouseMateControllerServicesAPI, and HouseMateModelServiceAPI 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	Creates a new Credentials object and adds it to

	: String) : void	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 IEntitlementServiceAPI, 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 IEntitlementServiceAPI
passwordHash	String	The hashed password; used for entitlemnt
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. Since I have no prior experience implementing this, I found it on 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 disclosed 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 HouseMateControllerServicesAPI, and HouseMateModelServiceAPI 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 entitlement 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 IEntitlementVisitable

Roles represent composition of Permissions and other Roles. A Role defines who a user is. For example, a role called "admin_role" would contain all the Permissions that collectively define all the restricted actions that only an EntitlementServiceAPI 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.

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 IEntitlementVisitable

Permissions represent actions that registered Users may carry out on the IEntitlementServiceAPI, HouseMateControllerServicesAPI, and HouseMateModelServiceAPI. Permissions ***need*** be added to Resources, 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 IEntitlementVisitable.

PermissionType

Enumeration of all Permission types that the IEntitlementServiceAPI uses, as well as the Permission types that are used by the HouseMateControllerServicesAPI, and HouseMateModelServiceAPI. These types are used in the housemate file to load the entitlement services, and also define the permissions that are required for carrying out restricted interface methods on the HouseMateControllerServicesAPI, HouseMateModelServiceAPI, and EntitlementServiceAPI.

Methods

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

Properties

Property Name	**Type**	**Description**
DEFINE_PERMISSION	String	used by the IEntitlementServiceAPI; permissionName is: "*define_permission*"
DEFINE_ROLE	String	used by the IEntitlementServiceAPI; permissionName is: "*define_role*"
ADD_ENTITLEMENT_TO_ROLE	String	used by the IEntitlementServiceAPI; permissionName is: "*add_entitlement_to_role*"
CREATE_USER	String	used by the IEntitlementServiceAPI; permissionName is: "*create_user*"
ADD_USER_CREDENTIAL	String	used by the IEntitlementServiceAPI; permissionName is: "*add_user_credential*"
ADD_ENTITLEMENT_TO_USER	String	used by the IEntitlementServiceAPI; permissionName is: "*add_entitlement_to_user*"
CREATE_RESOURCE_ROLE	String	used by the IHouseMateModelServiceAPI;

		permissionName is: “create_resource_role”
ADD_RESOURCE_ROLE_TO_USER	String	used by the IHouseMateModelServiceAPI; permissionName is: “*add_resource_role_to_usert*”

Implementation Details

The EntitlementServiceAPI is implemented as a Singleton, as returned by the static getInstance() method. Following good basic design principles, the EntitlementServiceAPI’s public methods are enumerated in the interface that it inherits from, IEntitlementServiceAPI. 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 Abstract EntitlementFactory for object creation to ensure unique ID values for each. The id’s of each of the objects in the Entitlement services should be unique GUIDs.

The EntitlementVistor 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, Resources) in the entitlement services for display.

.

Testing

Provide a testing strategy for testing the component.

- *Functional*
- *Performance*
- *Regression*
- *Exception Handling*

The premise of the TestDriver class is to exercise importing users, roles, permissions, and resources into the Entitlement via public methods on the EntitlementServiceAPI, defining the Permissions and Roles that collectively comprise Roles, as well as generate an inventory of all the Users, Permissions, Resources, and Roles in the Entitlement. My intention was to create a static main method that prompts for the Entitlement and HouseMate setup .txt files. Admittedly, I thought about this way too late in the game and was unsuccessful in implementation. See *Risks* and *Design Review Feedback*.

Risks

Document any risks identified during the design process.
Are there parts of the design that may not work or need to be implemented with special care or additional testing?

This is what I have managed to put together so far. I deliberately left out the functionality of voice pattern authentication temporarily until I fully implement the login first. I managed to implement the log in but I ran out of time trying to study the importer from the HouseMateModelServiceAPI. I created an abstract Importer class to make it generic and then try to extend that importer for both the Model and the Entitlement but wasn't successful.

I also need to get back to "create_resource_role", and "add_resource_role_to_user" as these two require that I go back into my HouseMateModelService implementation which in this case I went with a provided solution. I did not leave myself enough time to add this functionality after tackling the rest of the requirements. BUT, I hope that what I did manage to implement leaves evidence to show how easy it would be to incorporate.

I relied a little to heavily on third party implementations (such as the third party HashCode software which was fun to implement but was time consuming). I tried to implement a StringUtil.join for my parse exception but then it meant you would have to install Apache Common in order to run it and didn't feel it was worth it if I didn't have an output file to at least prove that it ran. I'd rather you verify that it compiles.