Authentication Service Design Document

Date: 11/14/2021
Author: Burak Ufuktepe
Reviewers: Fatma Ekim, Haley Huang

Contents

Introduction 3
Overview 3
Requirements 4
Resources 4
Permissions 4
Roles 4
Users 4
Authentication 4
Store Model Service 4
Use Cases 5
Actors 6
Use Cases 7
Class Diagram 8
Class Dictionary 9
AuthenticationService 9
User 14
Role 16
ResourceRole 17
Resource 17
Permission 18
AuthToken 18
Credential 19
CredentialType (enum) 20
InventoryVisitor 20
AuthenticationVisitor 21
Visitor (Interface) 22
Visitable (Interface) 22
Entitlement (Abstract) 23

AuthenticationException 23
AccessDeniedException 24
InvalidAccessTokenException 24
Sequence Diagram 25
Design Details 27
Requirements 27
Exception Handling 28
Testing 28
Risks 29

Introduction

This document defines the design for the Authentication Service which is responsible for controlling access to the Store Model Service.

Overview

The Authentication Service manages the authentication of users and utilizes face and voice recognition. Both the Store Controller Service and Store Model Service are clients of the Authentication Service. The Store Controller Service uses the Authentication Service to obtain authorization tokens which are used for calling Store Model Service methods. When the Store Model Service receives a request with an authorization token, it communicates with the Authentication Service to determine whether the request should be permitted or denied. The UML component diagram in Figure 1 shows how the Authentication Service fits into the Store 24x7 System.

Users of the Store 24X7 System are identified by their biometric voice and face prints. When a voice command is received from a microphone, the Store Controller Service uses voice prints to obtain authorization tokens. On the other hand, face prints are used for events that are initiated by cameras such as enter store and customer seen events.

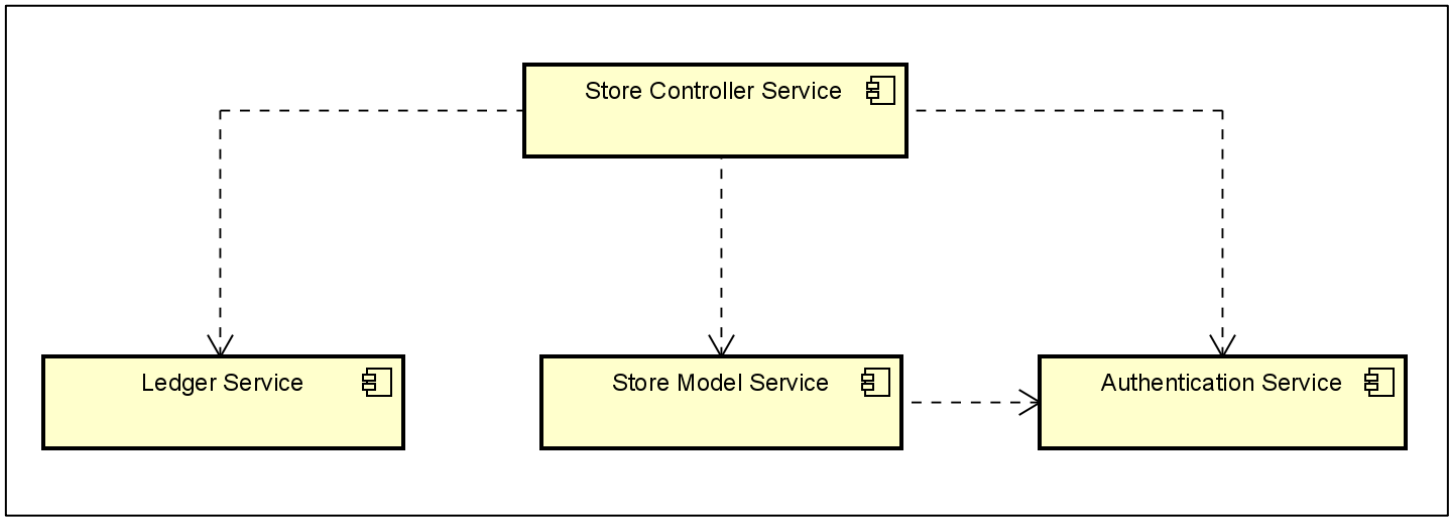

Figure 1: UML Component Diagram for the Store 24x7 System

Requirements

This section provides a summary of the requirements for the Authentication Service. The following functions should be supported by the Authentication Service:

Resources

1. Create resources such as sensors or appliances.
2. Each resource should have a unique ID and a description.

Permissions

3. Crate permissions. Permissions are required to access resources or functions of the Store system.
4. Each permission should have a unique ID, name, and a description.

Roles

5. Create roles which are groups of permissions and other roles.
6. Each role should have a unique ID, name, and a description.

Users

7. Create users which represent registered users of the Store system.
8. Each user should have a unique ID, name, and a set of Credentials such as username/password, voice print, and face print.
9. The password should be hashed.
10. Each user may be associated with one or more roles.

Authentication

11. The Authentication Service provides AuthTokens to users.
12. AuthTokens are used to access Store Model Service methods and they bind users to a set of permisisons.
13. Each AuthToken should have a unique ID, a state (active/expired) and an expiration time.
14. An Authentication Exception should be thrown if authentication fails.
15. A user may login with a username and password. The Authentication Service checks if the user exists and if the hash of the given password matches the hash of the password that is corresponding to the username.
16. Voice print and face print signatures are used for identifying and authenticating users.
17. A voice print is simulated using a string in the form of “--voice:<username>--”.
18. A face print is simulated using a string in the form of “--face:<username>--”.
19. Logging out invalidates an AuthToken.
20. An attempt to use an invalid AuthToken should result in a InvalidAuthTokenException.

Store Model Service

21. All Store Model Service methods should accept an AuthToken and check that the provided AuthToken is non-null and non-empty. Then the method should pass the AuthToken along with the required permission to the Authentication Service.
22. When the Authentication Service receives an AuthToken, it should first check that the AuthToken is active and then check that the user associated with the AuthToken has the required permission. If any of the checks fail, the Authentication Service should throw an AccessDeniedException or an InvalidAccessTokenException.

Use Cases

The following Use Case diagram describes the use cases supported by the Authentication System.

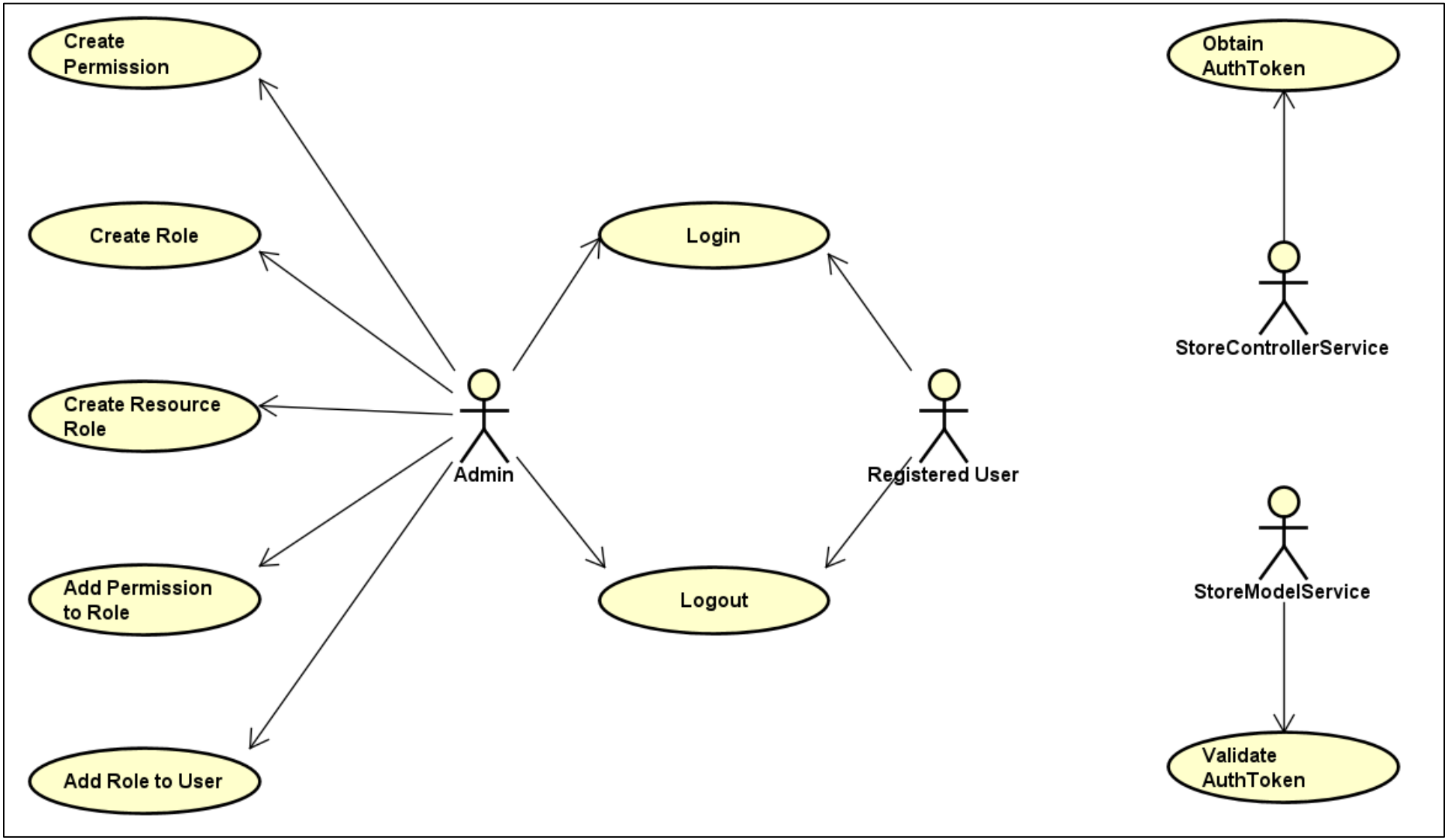

Figure 2: Authentication Service UML Use Case Diagram

Actors

The actors of the Authentication Service include Admin, Registered Users, Store Controller Service, and the Store Model Service.

Admin

Admin is the user with administrative privileges. The admin can define permissions, roles, resources, and resource roles. The admin can also add permissions to roles and add roles to users. Just like a registered user, the admin can login and logout of the system.

Registered User

Registered users are customers of the Store 24X7 system. Registered users can login and logout of the system. Each registered user is associated with a voice print, a face print and a set of permissions.

Store Controller Service

The Store Controller Service is the service responsible for controlling the devices based on status updates from the sensors and appliances within the store. The Store Controller Service uses a voice print, a face print or a username/password combination to obtain an Auth Token from the Authentication Service. The resulting Auth Token is used for calling the Store Model Service methods.

Store Model Service

All methods of the Store Model Service accept an Auth Token. In order to validate a given Auth Token, the Store Model Service passes the AuthToken to the Authentication Service along with the permission required for the method.

Use Cases

Create Permission

Permissions are created by the admin. Each permission represents an authorization required to access a resource or function of the Store system. A User may be associated with zero or more permissions.

Create Role

Roles are composites of permissions and they are created by the admin. Each role has a unique ID, name, and description.

Create Resource Role

Resource roles are defined by the admin. Each resource role binds the consumer with the store and the appropriate role.

Add Permission to Role

Roles consist of permissions and other roles. The admin adds permissions to roles. This provides reusable and logical groupings of permissions and roles which simplify the administration of users.

Add Role to User

The admin is responsible for adding roles to the users. Users may be associated with one or more roles, where the user has all permissions included in the role or sub-roles.

Login

Login accepts a user's username and password. Both the admin and registered users can login to the system in order to obtain Auth Tokens.

Logout

The admin and registered users can logout of the system to invalidate their Auth Tokens.

Obtain Auth Token

The Store Controller Service obtains an Auth Token from the Authentication Service using a credential such as a voice print, a face print or a username/password combination. The Authentication Service finds the user with a matching credential and returns an Authentication Token for the user.

Validate Auth Token

All Store Model Service methods accept an Auth Token. The Store Model Service utilizes the Authentication Service to validate the auth tokens that it receives. The Authentication Service checks to make sure that the AuthToken is active, and within the expiration period, and that the user associated with the Auth Token has the permission required by the method.

Class Diagram

The following class diagram defines the classes defined in this design. Also, notes are added to show the Singleton, Visitor, and Composite Patterns.

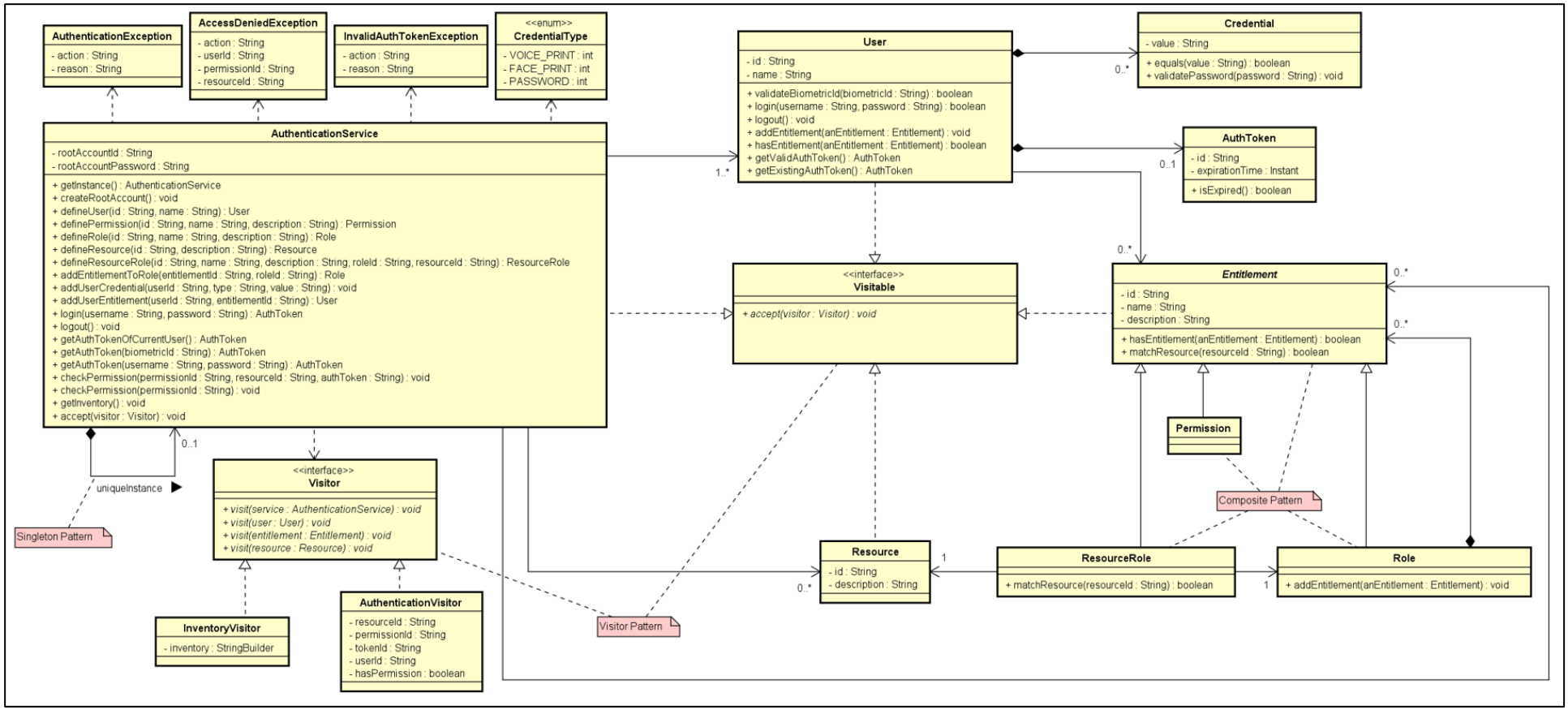

Figure 3: Authentication Service Class Diagram

Class Dictionary

This section specifies the class dictionary for the Authentication Service which is defined within the package com.cscie97.authentication.

AuthenticationService

The AuthenticationService provides the API used by clients of the Authentication Service and implements the Visitable interface. It manages users, entitlements, and resources. The Authentication Service controls access to the Store Model Service by generating and validating authentication tokens.

Methods

Method Name	Signature	Description
getInstance	():AuthenticationService	If the uniqueInstance parameter is null, generate a new AuthenticationService instance and assign it to uniqueInstance. Return the uniqueInstance. **Note:** this method is part of the **Singleton Pattern**.
createRootAccount	():void	Generate a user for the root account, add it to userMap, and set its password to rootAccountPassword.
defineUser	(id:String, name:String):User	Check if the logged-in user has create_user permission and if the given user ID already exists. Then, create a new User for the given ID and name. Add the User to userMap.. **Throw:** an AuthenticationException if the user ID already exists.
definePermission	(id:String, name:String, description:String):Permission	Check if the logged-in user has permission to create entitlements. Create a new Permission for the given ID, name, and description. Add the Permission to entitlementMap. **Throw:** an AuthenticationException if the permission ID already exists.
defineRole	(id:String, name:String, description:String):Role	Check if the logged-in user has permission to create entitlements. Create a new Role for the given ID, name, and description. Add the Role to entitlementMap. **Throw:** an AuthenticationException if the role ID already exists.

Method Name	Signature	Description
defineResource	(id:String, description:String):Resource	Check if the logged-in user has create_resource permission. Then, create a new Resource for the given ID and description. Add the Resource to resourceMap. **Throw:** an AuthenticationException if the resource ID already exists.
defineResourceRole	(id:String, name:String, description:String, roleId:String, resourceId:String): ResourceRole	Check if the logged-in user has create_entitlement permission. Using the given role ID retrieve the Entitlement from entitlementMap. Using the given resource ID retrieve the Resource from resourceMap. Create a new ResourceRole for the given ID, name, description, and retrieved Entitlement and Resource instances. Add the ResourceRole to entitlementMap. **Throw:** an AuthenticationException if: • the resource role ID already exists • the given role ID does not exist • the given role ID does not correspond to a Role instance • The given resource ID does not exist
addEntitlementToRole	(entitlementId:String, roleId:String):Role	Check if the logged-in user has update_entitlement permission. Perform the following checks: • if the logged-in user has update_entitlement permission • if the given entitlement ID exists • if the given role ID exists • if the given role ID corresponds to a Role • if the given role already contains the given entitlement Then add the entitlement to the role. **Note:** the Entitlement that is added to the Role might be of type Permission, Role or ResourceRole as all of these types inherit from Entitlement. **Throw:** an AuthenticationException if: • the given entitlement ID does not exist • the given role ID does not exist • the given role ID does not correspond to a Role instance

Method Name	Signature	Description
addUserCredential	(userId:String, type:String, value:String):void	Check if the logged-in user has update_user permission. Validate the credential type and user ID. Then, add the credential to the given user by setting the password, voice print or face print of the User based on the given type. **Throw:** an AuthenticationException if: • the given user ID does not exist • the given type does not match any CredentialType (VOICE_PRINT, FACE_PRINT, PASSWORD) • the type is PASSWORD and the given value does not meet password requirements provided in the Credential section.
addUserEntitlement	(userId:String, entitlementId:String):User	Check if the logged-in user has update_user permission, the given user ID and entitlement ID exist. Then, call the addEntitlement method on the user to add the entitlement to the user. **Throw:** an AuthenticationException if: • the given user ID does not exist • the entitlement ID does not exist
login	(username:String, password:String):AuthToken	Find the user with the given username and password. Call getValidAuthToken method on the User to get a valid Auth Token. **Note:** if there is already a user logged in to the system, that user will be logged out automatically and the new user will be logged in. **Throw:** an AuthenticationException if no match is found for the given username/password combination.
logout	():void	Check if a user is logged-in to the system. Then, logout the current user. **Throw:** an AuthenticationException if no user is logged in to the system.
getAuthTokenOfCurrentUser	():AuthToken	Check if the logged-in user has get_user_authtoken permission. Retrieve the current AuthToken of the logged-in user. **Throw:** an AuthenticationException if the current user does not have get_user_authtoken or if no user is logged-in currently.

Method Name	Signature	Description
getAuthToken	(biometricId:String):AuthToken	Check if the logged-in user has get_user_authtoken permission. Iterate over each user and look for a matching biometricId. Once a match is found, return a valid (unexpired) AuthToken for the user. **Throw:** an AuthenticationException if no match is found for the given biometric ID.
getAuthToken	(username:String, password:String):AuthToken	Check if the logged-in user has get_user_authtoken permission. Iterate over each user and look for a matching username/password combination. Once a match is found, return a valid (unexpired) AuthToken for the user. **Throw:** an AuthenticationException if no match is found for the given username/password combination.
checkPermission	(permissionId:String, resourceId:String, authToken:String):void	Create an AuthenticationVisitor instance and call its visitor method. Check if the user associated with the Auth Token has the required permission for the given permission ID and resource ID. **Notes:** • this method uses the **Visitor Pattern**. • a null value can be passed in for the resource ID if the permission is not associated with any resource. **Throw:** an InvalidAuthTokenException if the given Auth Token is invalid (not found). **Throw:** an AccessDeniedException if the user associated with the Auth Token does not have the required permission.
checkPermission	(permissionId:String):void	Check if the logged-in user has the given permission. Utilizes the overloaded checkPermission method. **Throw:** an AuthenticationException if no user is currently logged-in to the system or if the overloaded checkPermission throws an InvalidAuthTokenException or AccessDeniedException.
getInventory	():void	Check if the logged-in user has auth_readonly_role permission. Create an InventoryVisitor instance and call its visitor method. Then call the visitor's getInventory

Method Name	Signature	Description
		method to print out the inventory. **Note:** this method uses the **Visitor Pattern**.
accept	(visitor:Visitor):void	Accepts a Visitor object and calls its visitor method per the **Visitor Pattern**.

Properties

Property Name	Type	Description
rootAccountId	String	Unique ID of the root account which has full AuthenticationService access.
rootAccountPassword	String	Default password of the root account.

Associations

Association Name	Type	Description
userMap	Map<String, User>	Map of user IDs and user objects. Used for looking up users.
entitlementMap	Map<String, Entitlement>	Map of entitlement IDs and entitlement objects. Used for looking up entitlements.
resourceMap	Map<String, Resource>	Map of resource IDs and resource objects. Used for looking up resources.
uniqueInstance	AuthenticationService	Unique instance of the AuthenticationService. **Note:** this parameter ensures that one and only one AuthenticationService object is instantiated, per the **Singleton Pattern**.
loggedInUser	User	User that is currently logged in to the system.

User

The User class represents users of the Store system. Each user has an ID, name, and a set of credentials. Also, users are associated with 0 or more entitlements. Users obtain Auth Tokens from the Authentication Service using username/password combinations or biometric prints. The User class implements the Visitable interface.

Methods

Method Name	Signature	Description
validateBiometricId	(biometricId:String):boolean	Return true if the given biometricId matches the user's face print or voice print. Otherwise, returns false.
login	(username:String, password:String):boolean	Return true if the given username and password match the user's username and password. Otherwise, return false. **Note:** the given password is hashed before comparing it to the user's hashed password.
logout	():void	Set the user's AuthToken to null. **Throw:** an IllegalArgumentException if the user is already logged out (the user's authToken is null).
addEntitlement	(anEntitlement:Entitlement):void	Add the given entitlement to the user's set of entitlements. **Throw:** an IllegalArgumentException if the given entitlement is already added.
hasEntitlement	(anEntitlement:Entitlement):boolean	Iterate over the user's entitlements and call the hasEntitlement method on each entitlement. Return true if any of the hasEntitlement methods returns true. Otherwise, return false. **Note:** this method is used before adding an entitlement to the user in order to determine if the user already has the entitlement.
accept	(visitor:Visitor):void	Accepts a Visitor object and calls its visitor method per the **Visitor Pattern**.
getValidAuthToken	():AuthToken	Return the user's Auth Token if it is not null and expired. Otherwise, generate a new Auth Token for the user and return it. **Note:** this method is used when the Authentication Service requests an Auth Token from the user.
getExistingAuthToken	():AuthToken	Return user's current Auth Token.

Method Name	Signature	Description
		Note: this method is used by the AuthenticationVisitor to compare Auth Tokens.

Properties

Property Name	Type	Description
id	String	Unique identifier for the user.
name	String	Name of the user.

Associations

Association Name	Type	Description
entitlements	Set<Entitlement>	Set of entitlements associated with the user.
authToken	AuthToken	Authentication token of the user.
facePrint	Credential	Face print of the user.
voicePrint	Credential	Voice print of the user.
password	Credential	Password of the user.

Role

The Role class extends the Entitlement abstract class and provides a way to group Permissions and other Roles. Each role has a unique ID, name, and description. However, these properties are inherited from the Entitlement abstract class and are not explicitly defined in the Role class. The Role class is part of the **Composite Pattern** and each Role represents a composite object.

Methods

Method Name	Signature	Description
accept	(visitor:Visitor):void	Accepts a Visitor object and calls its visitor method per the **Visitor Pattern**. **Note:** this method is not explicitly implemented in the Role class. The implementation is inherited from the Entitlement class which implements the Visitable interface. However, for clarity, it is included in this table.
addEntitlement	(anEntitlement:Entitlement):void	Add the given entitlement to the role's set of entitlements. **Throw:** an IllegalArgumentException if the given entitlement is already added.
hasEntitlement	(anEntitlement:Entitlement):boolean	Return true if the role's ID matches the given entitlement's ID. Otherwise, iterate over the role's entitlements and call the hasEntitlement method on each entitlement. Return true if any of the hasEntitlement methods returns true. Otherwise, return false. **Note:** this method is used before adding an entitlement to the role in order to determine if the role already has the entitlement.

Associations

Association Name	Type	Description
entitlements	Set<Entitlement>	Set of entitlements associated with the role.

ResourceRole

The ResourceRole class extends the Entitlement class and binds the Consumer with the Store and the appropriate Role. Each ResourceRole is associated with a Resource and an Entitlement object. Additionally, each ResourceRole has a unique ID, name, and description. However, these properties are inherited from the Entitlement abstract class and are not explicitly defined in the ResourceRole class. The ResourceRole class is part of the **Composite Pattern** and each ResourceRole represents a composite object.

Methods

Method Name	Signature	Description
accept	(visitor:Visitor):void	Accepts a Visitor object and calls its visitor method per the **Visitor Pattern**. **Note:** this method is not explicitly implemented in the ResourceRole class. The implementation is inherited from the Entitlement class which implements the Visitable interface. However, for clarity, it is included in this table.
matchResource	(resouceId:String):boolean	Return true if the given resource ID and the ID of the resource that is associated with the ResourceRole are the same. Otherwise, return false.

Associations

Association Name	Type	Description
role	Entitlement	Entitlement associated with the ResourceRole.
resource	Resource	Resource associated with the ResourceRole.

Resource

The Resource class represents physical entities such as stores or appliances. Each resource has a unique ID and description. The Resource class implements the Visitable interface.

Methods

Method Name	Signature	Description
accept	(visitor:Visitor):void	Accepts a Visitor object and calls its visitor method per the **Visitor Pattern**. **Note:** The logic for traversing objects is located in the Visitor class.

Properties

Property Name	Type	Description
id	String	Unique identifier for the resource.
description	String	Description of the resource.

Permission

The Permission class extends the Entitlement abstract class and represents a permission required to access a resource or function of the Store system. Each Permission has a unique ID, name, and description which are inherited from the Entitlement abstract class. These properties are not explicitly defined in the Permission class. The Permission class is part of the **Composite Pattern** and each Permission represents a leaf object.

Methods

Method Name	Signature	Description
accept	(visitor:Visitor):void	Accepts a Visitor object and calls its visitor method per the **Visitor Pattern**. **Note:** this method is not explicitly implemented in the Role class. The implementation is inherited from the Entitlement class which implements the Visitable interface. However, for clarity, it is included in this table.

AuthToken

The AuthToken class represents an authentication token which is used to access restricted Store Model methods. Each AuthToken has a unique ID and expiration time. The state of an AuthToken is checked using the isExpired method.

Methods

Method Name	Signature	Description
isExpired	():boolean	Return true if the AuthToken is expired. Otherwise, return false.

Properties

Property Name	Type	Description
id	String	Unique identifier of the AuthToken.
expirationTime	Instant	Expiration time of the AuthToken.

Credential

The Credential class contains a hash string of a face print, a voice print or a password. Biometric prints and passwords are considered sensitive information. Therefore, each Credential object includes only the hashed string of a biometric print or a password.

Methods

Method Name	Signature	Description
equals	(value:String):boolean	Return true if the hashed value of the given string is equal to the value property of the Credential instance.
validatePassword	(password:String):void	Validate the given password. **Throw:** an IllegalArgumentException if the given password is null or contains: • no digits • no lowercase letters • no uppercase letters • no special characters • whitespace • less than 8 characters

Properties

Property Name	Type	Description
value	String	Hashed string of the Credential.

CredentialType (enum)

CredentialType is an enum that represents different types of credentials.

Properties

Property Name	Type	Description
VOICE_PRINT	CredentialType	Credential type representing a voice print.
FACE_PRINT	CredentialType	Credential type representing a face print.
PASSWORD	CredentialType	Credential type representing a password.

InventoryVisitor

The InventoryVisitor class implements the Visitor interface and visits each user, resource, and entitlement of the Authentication Service in order to provide an inventory of all Users (including their AuthTokens and Credentials), Resources, Roles, and Permissions. The InventoryVisitor has an inventory property that stores information about the objects of the Authentication Service.

Methods

Method Name	Signature	Description
visit	(service:AuthenticationService):void	Iterate over each user, resource, and entitlement of the Authentication Service and call accept methods on each object.
visit	(user:User):void	Append user information to the inventory property including AuhToken and Credential information associated with the user.
visit	(entitlement:Entitlement):void	Append entitlement information to the inventory property.
visit	(resource:Resource):void	Append resource information to the inventory property.

Properties

Property Name	Type	Description
inventory	StringBuilder	StringBuilder that includes information about the objects of the Authentication Service.

AuthenticationVisitor

The AuthenticationVisitor class implements the Visitor interface and is used to determine if a user has the required permission for the requested action. The implementation details for traversing the objects of the Authentication Service is contained in the visit methods.

Methods

Method Name	Signature	Description
visit	(service:AuthenticationService):void	Iterate over each user and call the accept method on each user. Check if a user with a matching AuthToken is found after each call. If a matching token is found and of the token is active then iterate over the entitlements of the user and call the accept method on each entitlement.
visit	(user:User):void	If the given user's AuthToken and the AuthenticationVisitor's AuthToken match, then assign the user's ID to the userId property. Otherwise, do not perform any action.
visit	(anEntitlement:Entitlement):void	First check if the resourceId is null or if the resourceId is applicable to the given entitlement. Then, check if the given entitlement's ID matches the permission ID. If they do, then set hasPermission to true and return. Otherwise, visit the entitlement's children and call the accept methods on them.
visit	(resource:Resource):void	Not applicable. The AuthenticationVisitor does not visit resources. However, this method is included as the AuthenticationVisitor implements the Visitor interface.

Properties

Property Name	Type	Description
resourceId	String	Unique ID of the resource associated with the permission. This can be null if the permission is not associated with any resource.
permissionId	String	Unique ID of the permission provided by the client.
tokenId	String	Auth Token provided by the client.

Property Name	Type	Description
userId	String	ID of the user whose Auth Token matches the given Auth Token.
hasPermission	boolean	True if the given Auth Token has the required permission. Otherwise, false. **Note:** this property is used by the Authentication Service to determine whether the given Auth Token has the required permission or not.

Visitor (Interface)

Provides method signatures for visiting the AuthenticationService, Users, Entitlements, and Resources.

Methods

Method Name	Signature	Description
visit	(service:AuthenticationService):void	Method signature for visiting the AuthenticationService.
visit	(user:User):void	Method signature for visiting a User.
visit	(entitlement:Entitlement):void	Method signature for visiting an Entitlement.
visit	(resource:Resource):void	Method signature for visiting a Resource.

Visitable (Interface)

The Visitable interface provides a method signature for the accept method which enables objects to be visitable.

Methods

Method Name	Signature	Description
accept	(visitor:Visitor):void	Method signature that enables objects to be visitable.

Entitlement (Abstract)

The Entitlement abstract class implements the Visitable interface and represents Permissions, Roles, and Resource Roles. Each Entitlement object has a unique ID, name, and description.

Methods

Method Name	Signature	Description
accept	(visitor:Visitor):void	Accepts a Visitor object and calls its visitor method per the **Visitor Pattern**.
hasEntitlement	(anEntitlement:Entitlement):boolean	Return true if the Entitlement's ID and the given entitlement's ID match. Otherwise, return false.
matchResource	(resourceId:String):boolean	Return true. Any child class that is associated with a resource needs to overwrite this method. If the child class is not associated with any resource, this method can be inherited as is.

Properties

Property Name	Type	Description
id	String	Unique identifier for the entitlement.
name	String	Name of the entitlement.
description	String	Description of the entitlement.

AuthenticationException

The AuthenticationException is returned from the AuthenticationService methods in response to an error condition. It captures the attempted action and the reason for failure.

Properties

Property Name	Type	Description
action	String	Action that was performed.
reason	String	Reason for the exception.

AccessDeniedException

The AccessDeniedException is returned to indicate that a user does not have access to a resource or function. It includes the action that was performed, the user ID, permission ID, and resource ID.

Properties

Property Name	Type	Description
action	String	Action that was performed.
userId	String	Unique ID of the user that does not have the required permission.
permissionId	String	Permission ID associated with the denied request.
resourceId	String	Resource ID associated with the denied request.

InvalidAccessTokenException

The InvalidAccessTokenException is returned to indicate that the given Auth Token doesn’t exist or is expired.

Properties

Property Name	Type	Description
action	String	Action that was performed.
reason	String	Reason for the exception.

Sequence Diagram

The sequence diagram in Figure 4 shows the interaction between the Authentication Service, Store Model Service, and Store Controller Service for the broken glass event.

The sequence can be summarized as follows:

1. The microphone detects a sound of breaking glass and alerts the StoreModelService (please note that for the purposes of this assignment, the events are generated by the CommandProcessor class based on the given script).
2. The StoreModelService notifies the StoreControllerService using the Observer Pattern.
3. The StoreControllerService utilizes the CommandFactory class to generate a CleanUpAisleCommand.
4. Then the StoreControllerService logins to the AuthenticationService using its username and password and receives an authToken.
5. StoreControllerService then calls the execute method of the CleanUpAisleCommand.
6. The CleanUpAisleCommand calls the cleanUpAisle method of the StoreModelService and passes the AuthToken of the StoreControllerService.
7. Before instructing a robot to clean up the aisle, the StoreModelService first calls its checkPermission method to see if the given authToken has permission to control robots.
8. The checkPermission method calls the AuthenticationService's checkPermission method which utilizes the **Visitor Pattern** to check if the given authToken has the required permission to control robots.
9. The checkPermission method throws InvalidAuthTokenException if the given authToken is not found or is expired. Also, it throws an AccessDeniedException if the given authToken does not have the required permission. In this case, the checkPermission method does not return anything as the StoreControllerService has permission to control robots.
10. The StoreModelService then instructs a robot to clean up the aisle (please note that this part is not included in the sequence diagram in order to focus on the interaction between the Authentication Service, Store Model Service, and Store Controller Service).

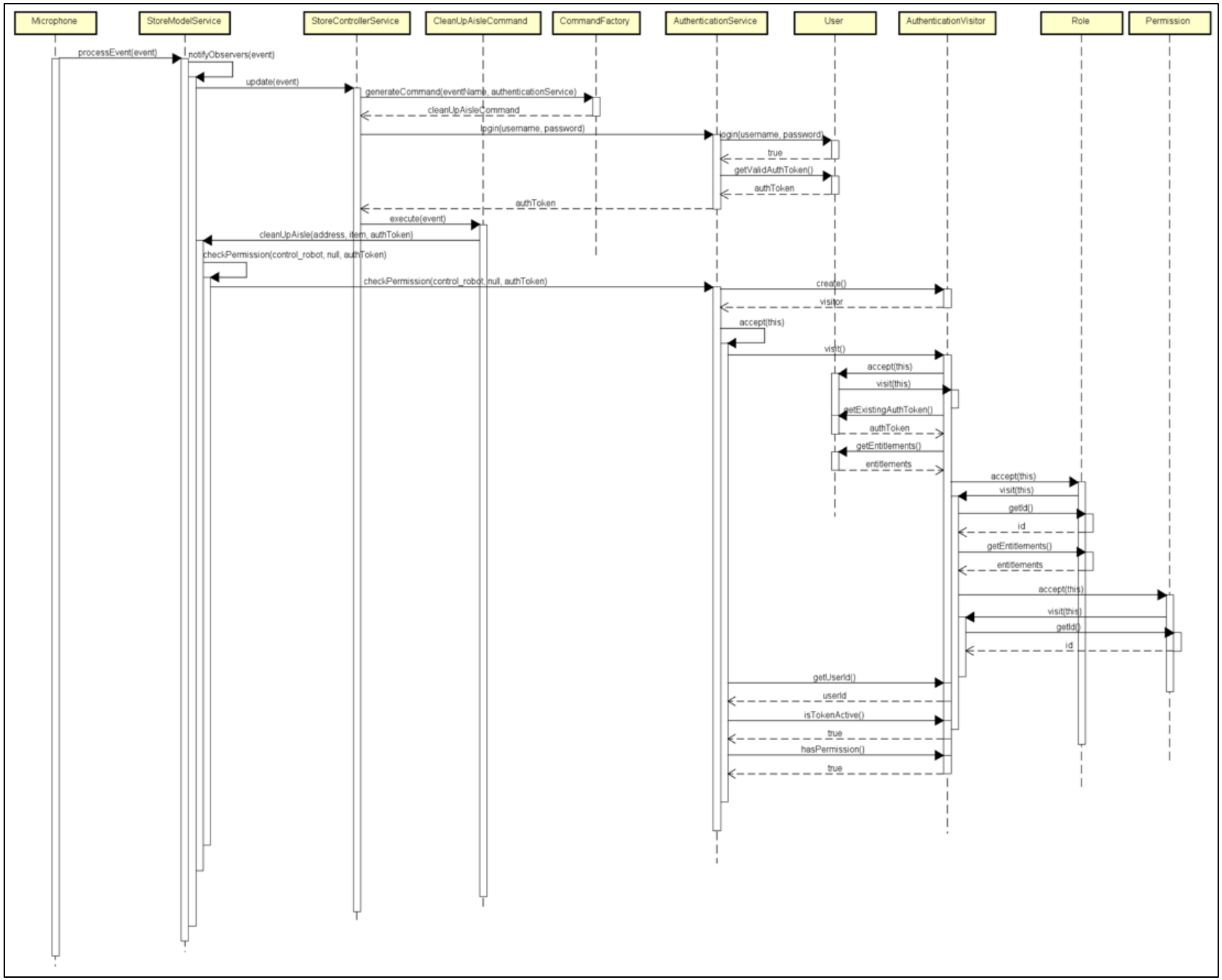

Figure 4: Sequence Diagram for the Broken Glass Event

Design Details

The Authentication Service provides an API for its clients and manages Users, Entitlements (Permissions, Roles, Resource Roles), and Resources.

The Authentication Service utilizes the **Visitor Pattern** to check if a given Auth Token has access to a specific resource or function of the Store system. This is accomplished using an AuthenticationVisitor that visits the users of the Authentication Service and finds the user with the given Auth Token. If the Auth Token is not expired, the AuthenticationVisitor then visits the entitlements of that user and checks whether any of the entitlements include the required permission. After traversing the entitlements, the AuthenticationVisitor sets its hasPermission property to true if the required permission is found. Finally, the Authentication Service calls the AuthenticationVisitor's hasPermission method to determine whether the given Auth Token has the required permission or not for the requested resource or function.

The **Visitor Pattern** is also used to traverse the objects of the Authentication Service to provide an inventory of all Users, Resources, Auth Tokens, Credentials, Roles, and Permissions. User, Resource, and Entitlement classes implement the Visitable interface which defines the accept method. An InventoryVisitor object visits each User, Resource, and Entitlement of the Authentication Service. Each User instance includes Credentials and Auth Tokens. Therefore, the InventoryVisitor does not explicitly visit Credential and AuthToken objects. The InventoryVisitor has an inventory property that stores information about the objects of the Authentication Service. After all objects are visited, the Authentication Service calls the getInventory method of the InventoryVisitor to obtain the inventory of all Users, Resources, Auth Tokens, Roles, and Permissions.

The **Composite Pattern** is used to model Permissions, Roles, and Resource Roles. These classes inherit from the Entitlement abstract class which includes a unique ID, name, and description. Roles are composites of Entitlements whereas Permissions do not include any Entitlements. A Resource role extends the Entitlement class and binds the consumer with the store and the appropriate role.

Only one instance of the AuthenticationService is needed. In order to ensure that one and only one AuthenticationService object is instantiated, the **Singleton Pattern** is utilized. This is achieved using a private AuthenticationService class constructor and a static AuthenticationService variable. The client can instantiate the AuthenticationService object by calling the static getInstance method of the AuthenticationService which ensures only one instance of the AuthenticationService object exists.

Requirements

- When a store or device (sensor or appliance) is created, a resource is automatically created in the AuthToken (satisfies requirement 1).
- Each resource has a unique ID and description (satisfies requirement 2).
- Permission objects are used by the AuthenticationVisitor to determine if an auth token has access to a resource or function of the Store system (satisfies requirement 3).
- Each Permission object has a unique ID, name, and a description (satisfies requirement 4).
- Role objects contain entitlements which can be permissions or other roles (satisfies requirement 5).
- Each Role object has a unique ID, name, and a description (satisfies requirement 6).
- User objects represent registered users of the Store system (satisfies requirement 7).
- Each User object has a unique ID and name. The ID of the user is considered a username. Additionally, a user may have a voice print, face print and password (satisfies requirement 8).
- The password is hashed before it is stored in the Credential object (satisfies requirement 9). The face prints and voice prints are also hashed as these are considered sensitive information.
- Each user has a set of Entitlements. Hence, each user may be associated with one or more roles (satisfies requirement 10).

- When a user tries to login to the Authentication Service using a username and password, the Authentication Service iterates over its userMap to find a match for the given username/password combination (the hash of the given password is compared to the hash of the user's password). If a match is found, the Authentication Service logs in the user and returns an AuthToken (satisfies requirements 11 and 15). Also, a user may get an AuthToken from the Authentication Service using a face print or a voice print (satisfies requirement 16). An Authentication Exception is thrown if the given username/password, face print, or voice print are invalid (satisfies requirement 14).
- Voice prints and face prints are simulated using strings in the form of "--voice:<username>--" and "--face:<username>--", respectively (satisfies requirements 17 and 18).
- The Credential class is used to represent voice prints, face prints, and passwords.
- An Auth token is represented by the AuthToken object. Each Auth Token object has an ID and expiration time. The expiration time is determined by the TTL_MINS static final integer defined in the AuthToken class which is 15 minutes. Instead of using a state (active/expired) parameter, the isExpired method is used to determine if the AuthToken is expired or not (satisfies requirement 13).
- Logging out sets the AuthToken of the logged-in user to null (satisfies requirement 19).
- When the Store Model Service API receives a request, it first calls its checkPermission method to see if the given authToken is non-null and non-empty. Then it calls the checkPermission method of the Authentication Service and passes an AuthToken, a permission ID, and optionally a resource ID to determine if the AuthToken has the required permission. Each user may be associated with an AuthToken and a set of permissions (satisfies requirements 12 and 21).
- The AuthenticationVisitor determines if a user has the required permission for the requested action using the **Visitor Pattern**. If the given AuthToken does not exist in the Authentication Service or if it is expired, an InvalidAuthTokenException is thrown. Also, if the given authToken does not have the required permission an AccessDeniedException is thrown (satisfies requirements 20 and 22).

Exception Handling

The error conditions in the CommandProcessor methods result in a CommandProcessorException. The CommandProcessorException contains the failed command and the reason for the failure. Also, the line number of the command is included if the commands are read from a file.

For validating method arguments, IllegalArgumentExceptions are used and these exceptions are caught by the AuthenticationService class. The AuthenticationService class gets the message from these exceptions and throws AuthenticationExceptions.

AuthenticationExceptions can also result from error conditions in the Authentication Service methods. It captures the attempted action and the reason for failure. Furthermore, the Authentication Service might throw an InvalidAuthTokenException if the given Auth Token is invalid or an AccessDeniedException might be thrown in case the user associated with the given Auth Token does not have the required permission. The AccessDeniedException includes information about the error condition such as the user ID, permission ID, and the resource ID.

Testing

A test driver class is implemented with a static main() method which accepts a command file. The test driver can be found at com\cscie97 \test\TestDriver.java. The main() method calls the processCommandFile(file:string) method of the CommandProcessor.

Also two test scripts are provided; test_1.script and test_2.script. Both scripts include a sample store configuration, ledger configuration, sample events, and queries. test_1.script demonstrates functionality whereas test_2.script demonstrates exception handling.

Risks

Personal information such as customer names, emails, or blockchain account addresses can easily be exposed by third parties as these are stored as strings. These types of sensitive information need to be encrypted in order to prevent data breaches.