Authentication Service Design Document

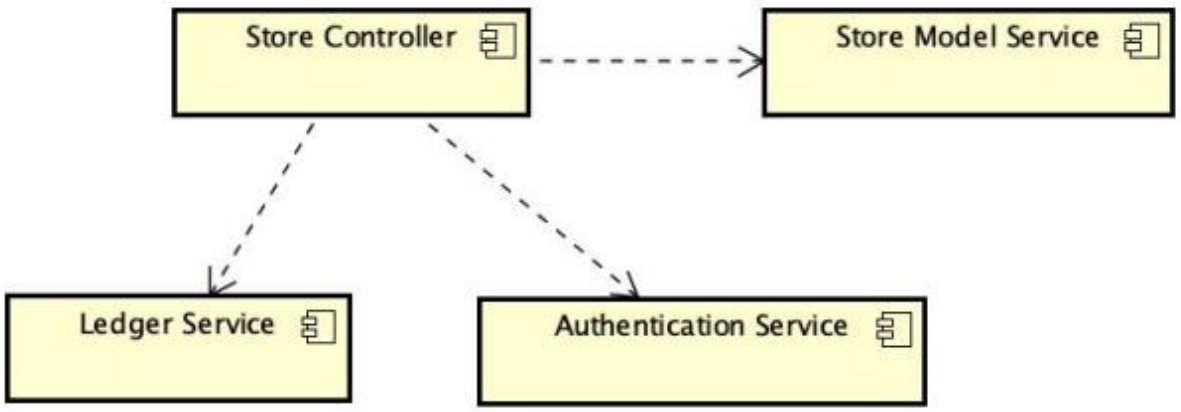
Date: 11/20/2019
Author: Tofik Mussa
Reviewer(s): Trisha Singh, Peter Chen

Introduction

This design document defines the Authentication as a standalone, generic service that is multipurpose and reusable service. However, it leans towards integrating previously defined services such as the Store Model Service, Store Controller Service and Ledger Service for a secure and satisfactory customer experience. The Authentication Service determines if the user is who he/she claims is and if he/she has permission to perform certain tasks. Several workflows will be examined in the lens of the Store 24X7 system.

Overview

Security is crucial to a self-regulated system like the Store 24X7 system. A poorly managed ecosystem may result in an economic loss, bad reputation and safety issues. A robust system that identifies users with their corresponding permissions is an essential component of the Authentication Service. It prevents intruders from abusing the Store 24X7 system and it reassures valuable customers that the system is reliable and trustworthy. The overall cost of developing a robust authentication service by far outweighs the amount of development hours spent on it. Since it is a maintainable and well-designed system, it anticipates future changes pertaining to the Open Closed Principle. The system is modular with clearly defined boundaries and loose dependencies making it integrate easily with other subsystems. For a high-level overview, see the image below



Requirements

In the context of the Store 24X7 system, the authentication service maintains an inventory of users, role, permissions, resources and tokens which enables to guard the store. The Store Model Service and the Store Controller Service delegate authentication/authorization to this service.

The Systems Architecture document emphasizes the fully automated nature of the store and having a robust record of users provides an ease of access. Security is also one of the nonfunctional requirements and even though it is advisable to use a commercial of the shelf software for it, building a custom solution accomplishes flexible development tuned to the requirements.

The requirements document mandates the use of design patterns to communicate with stakeholders using ubiquitous language and to avoid reinventing the wheel. An effective use of design patterns specifically, the visitor pattern, the composite pattern and the singleton pattern enabled the developer to conceptualize a rather complex system with several moving pieces. The composite pattern builds the entitlement hierarchy constituting of roles, permissions and resource roles. Those entities mimicked a file/directory system and the composite pattern came out as a natural choice. As a side note, the blocks and transactions in the ledger system could have also been modeled by the composite pattern having blocks as composite objects and transactions as leaves even though there were some restrictions as to how many transactions can be contained in a block. The visitor pattern is used to keep track of and collect all the entities that make up the authentication service. The visitor pattern also enabled a quick, recursive navigation to find permissions granting access of a resource to a user. The singleton pattern was also a good fit to the AuthenticationService class since in a centralized service like this one, it is essential to lump together the state in one instance of an object.

On a high level the Authentication Service can undertake the following functions

1) **Register users and update their credentials**
2) **Guard the Store Model Service from malicious attackers**
3) **Give least privilege access to users. Grant global or restricted access of resources**
4) **Revoke access to unauthenticated users**
5) **Maintain sessions for users**
6) **Provide log in/log out functionality**
7) **Maintain a record of users and their roles/permissions**

Use Cases

The Authentication Service supports the following use cases.

1) **Maintain the userbase**
    - The Authentication service stores the users with their corresponding roles and permissions.
    - New users can be added anytime
    - All customers that enter the store by default are registered as guest users
    - Users can deregister themselves and will be converted back to guest users
2) **Update credentials**
    - Users can update their credentials and are recommended to do so frequently for security reasons.
    - Users who forget their username/password can also retrieve their account using voice/face prints
3) **Perform authentication and authorization**
    - Password protected APIs like the Store Model Service can utilize the Authentication Service to valid the tokens passed into their APIs. The Authentication Service can check if the token is associated with a valid user, if it is unexpired and if the token gives the required permissions to a user.
4) **Log in/log out functionality and maintaining sessions**
    - Users can log in and obtain a session that is valid for 15 minutes
    - Users are recommended to log out once finishing tasks. However, the system will wait for 15 minutes before it logs users out due to inactivity.
    - The Store Controller Service generates a token for every action it executes on behalf a customer. Therefore, there might be multiple tokens associated with a customer
5) **Encrypt credentials**
    - Passwords are hashed before they are stored to make it difficult for intruders to gain access
6) **Give least privilege access to users**
    - Users are given access to the minimum resources that they need. For actions that require a higher privilege or for tasks that don't involve an individual customer, the Store Controller Service performs authentication on behalf of customers.

The use case diagrams are categorized by actors. Please refer to the actor's section to find out about use case diagrams.

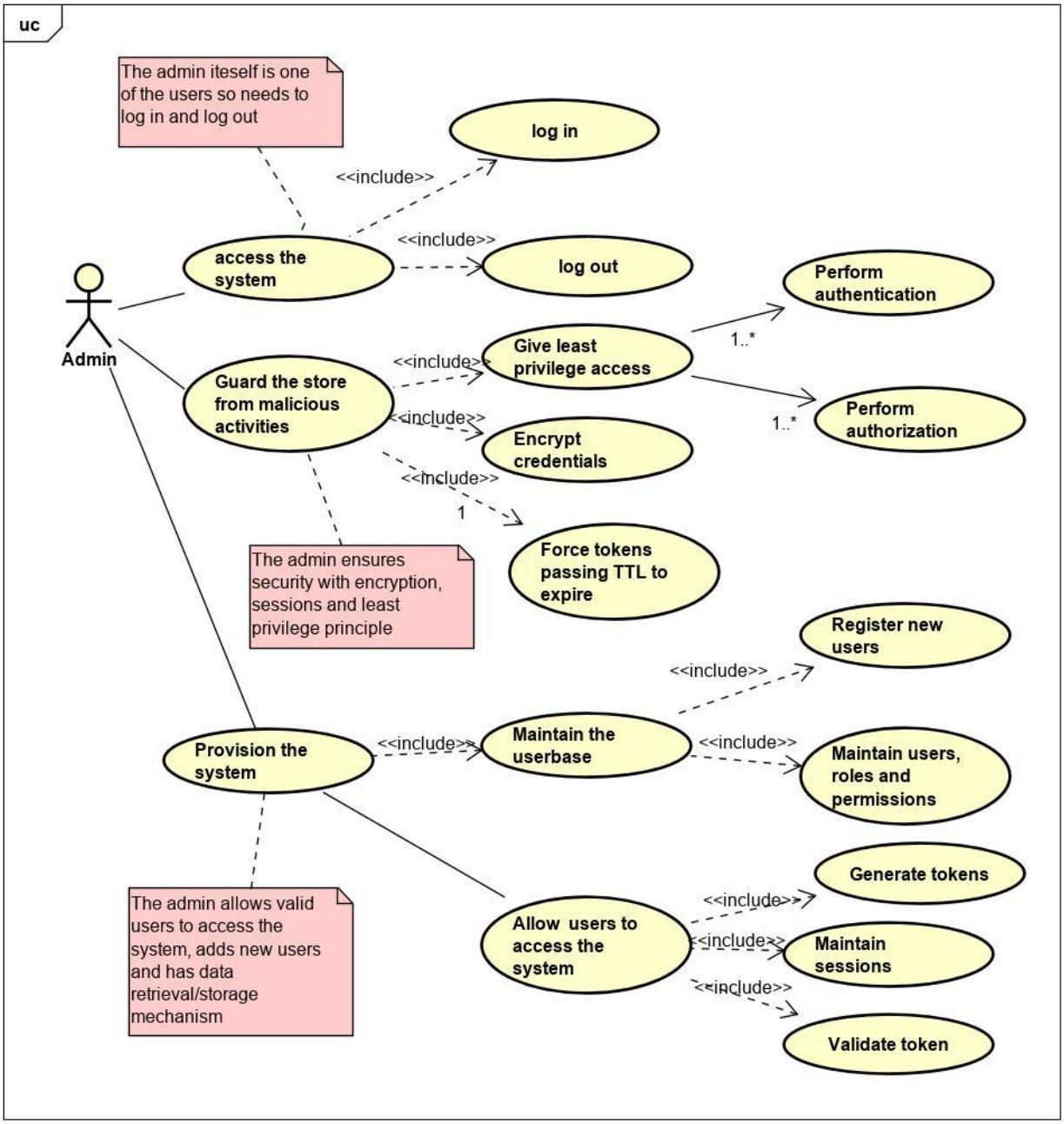
Actors

Admin

The administrator as shown in the picture below has the most privilege. It keeps the Authentication Service in check and some of the tasks it can perform are

1) **Access the system**
    - Log in as a user
    - Log out as a user
2) **Guard the store from malicious activities** – this is the most important role of the admin
    - Give least privilege access
        (a) Perform authentication
        (b) Perform authorization
    - Encrypt credentials
    - Force tokens exceeding time to live to expire
3) **Maintain the userbase**
    - Register new users
    - Maintain users, roles and permissions
4) **Allow users to access the system**
    - Generate tokens
    - Maintain sessions
    - Validate token

Please refer the diagram below



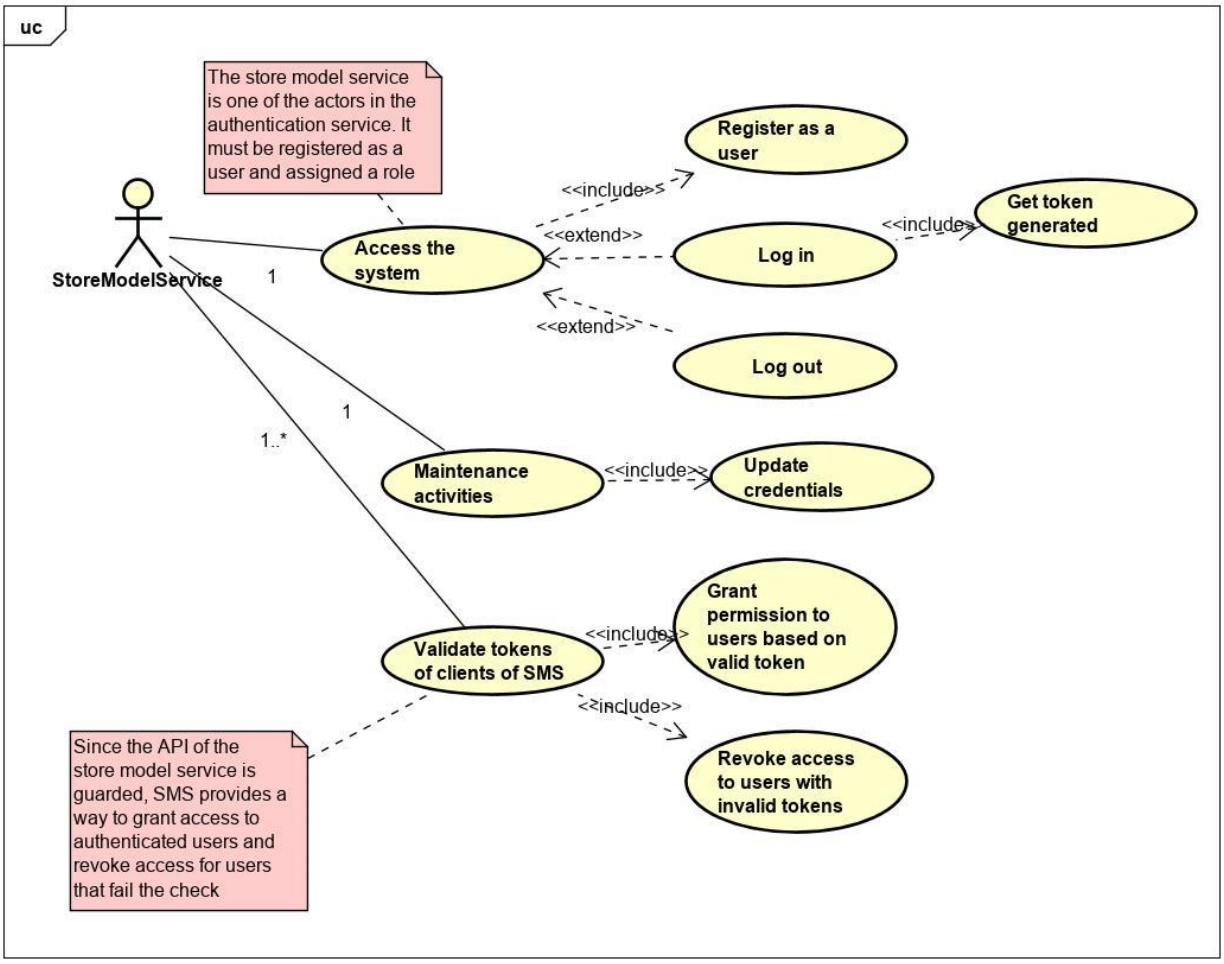
Store Model Service

The Store Model Service is one of the actors in the Authentication Service. It relies on the Authentication Service to perform authentication/authorization. The Store Model Service is fully guarded and inaccessible without a valid authentication token. The first step that the Store Model Service does before undertaking any actions confirm with the Authentication Service if the token passed in is associated with a valid user, not expired and has the correct permissions. The Store Model Service can

also register and obtain a token just like any other user even though there is no valid use case for that in the scope of this implementation.

The Store Model Service can perform the following actions on the Authentication Service.

1) **Validate tokens passed into the Store Model Service API**
    - Grant access to users with a valid, unexpired token with the required permissions
    - Revoke access to users with invalid token and throw an InvalidTokenException
2) **Access the Authentication Service**
    - Register as a user
    - Log in/log out
    - Update credentials

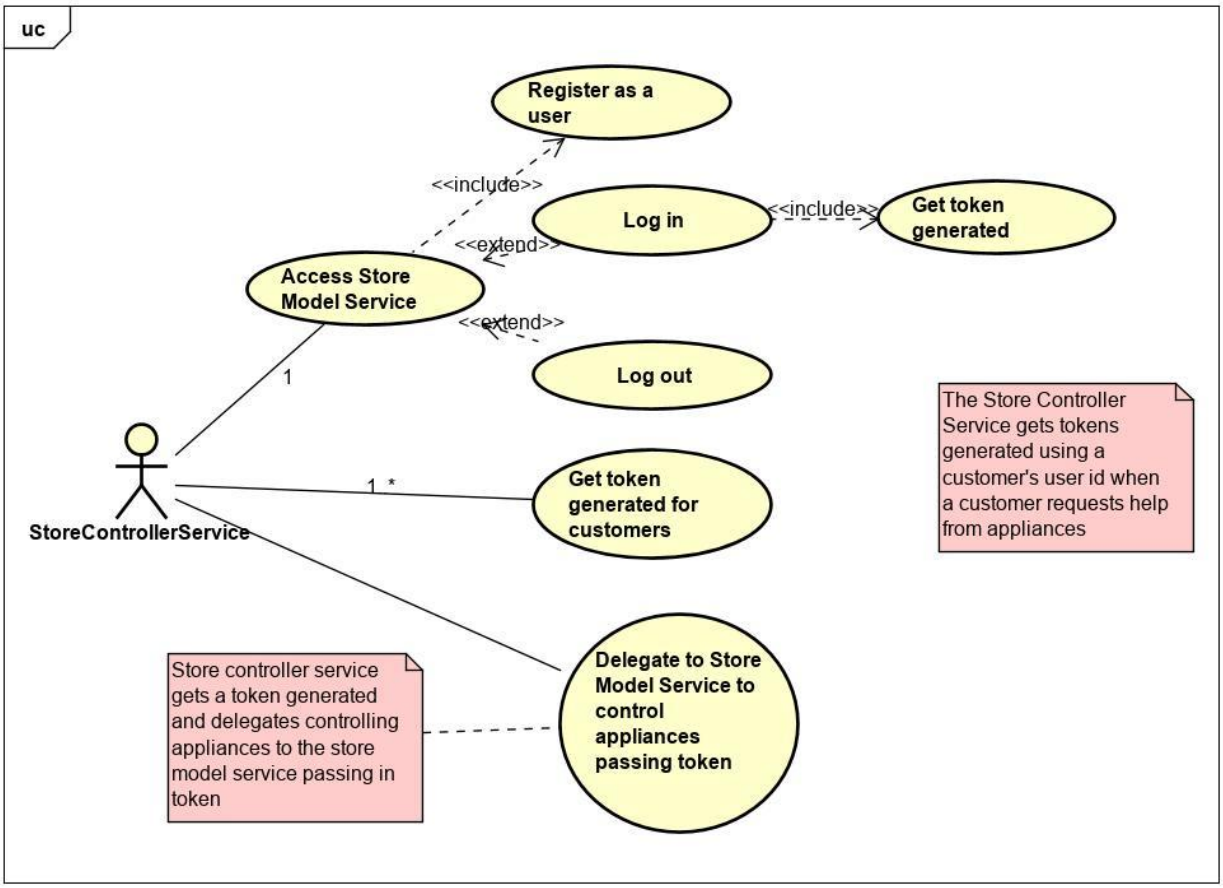


Store Controller Service

The Store Controller Service gets an authentication token generated by the Authentication Service on behalf of customers and on behalf of itself. The Store Controller Service has the highest privileges as a user next to the Administrator. If an action requires a higher privilege like commanding Appliances during emergency or removing association of a basket with a customer, the action will be performed by a token generated for the Store Controller Service. If the action entails simple tasks like telling a robot to get a product from a shelf to a customer, the action will be performed by a token generated for a customer.

The Store Controller Service can

1) **Register as a user like any other user and obtain the highest privilege next to the Administrator**
2) **Log in and get token generated**
3) **Log out or have a session timeout due to inactivity for 15 minutes**
4) **Get tokens generated for customers for each action that doesn't require a high privilege**
5) **Pass a token to the Store Model Service API and delegate controlling appliances to it**



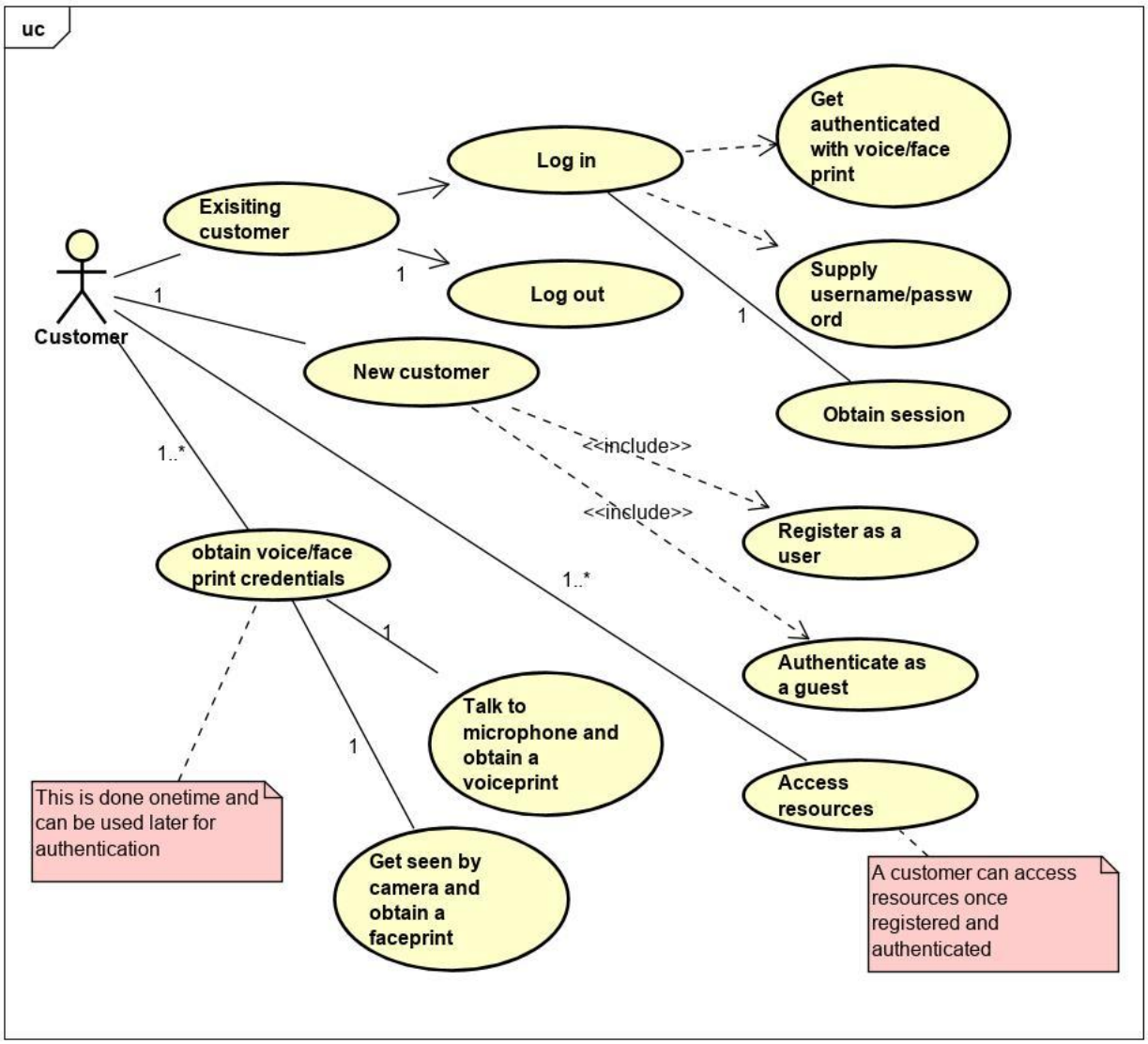
Customer

Customers are the main users in the Authentication Service. When customers are defined, they are also added as a user in the Authentication Service. Customers may later provide a username and password. When a camera detects their appearance a faceprint will be created. A voiceprint will also be created when they speak to a microphone all of which can be used as a method of authentication in lieu of username/password. All customers are by default considered guest users until they register.

Customers can

1) **Can get registered as a user in the Authentication Service when they sign up to use the store**
2) **Can authenticate as a guest without being formally registered**
3) **Can log in**
    - Using a username and password
    - Using voice/face recognition
    - Obtain a session – the Store Controller service generates a token to the customer for every action he wants to perform involving appliances. The short-lived ness of a token is not visible to the end user.
4) **The customer may log out or choose to stay logged in. The system will automatically log him out due to inactivity for 15 minutes**
5) **Access password protected resources in the store**
6) **Obtain voice/print credentials. This can happen in two ways**
    - Talking to a microphone for the first time will give the customer a voiceprint that will be stored in the authentication service. This voiceprint can be used later for authentication. The user must authenticate with username/password before this happens
    - When a customer walks to a store for the first time, his faceprint will be recorded and used later for authentication. The user must authenticate with username/password before this happens.

Please refer the diagram below


uc
Get authenticated with voice/face print
Log in
Exisiting customer
Log out
Supply username/password
Customer
New customer
Obtain session
<<include>>
<<include>>
Register as a user
obtain voice/face print credentials
Authenticate as a guest
Talk to microphone and obtain a voiceprint
Access resources
This is done onetime and can be used later for authentication
Get seen by camera and obtain a faceprint
A customer can access resources once registered and authenticated
1
1
1
1
1..*
1..*

Implementation

This section is mostly about the Authentication service. The common syntax of the test scripts and commands to compile/run are also shown.

The Authentication Service has the following **interfaces**

1) **IAuthenticationService** with implementations
    - AuthenticationService

This is the interface exposed to the outside world. It provides add/update/get functionalities.

2) **IVisitor** with implementations
    - CheckAccessVisitor
    - InventoryVisitor

It leverages the visitor pattern pertaining to the requirements to traverse through inventories to check a required permission to a resource.

3) Visitable with implementations
    - AuthenticationService
    - User
    - Role
    - ResourceRole
    - Permission
    - Resource
    - AuthenticationToken

These entities implement so that the visitor can perform additional actions

4) Credential
    - UsernamePassword
    - VoicePrint
    - FacePrint

***T**his is a convenience marker interface to use different types of credentials interchangeably.*

The Authentication Service has the following **Abstract class**

1) **Entitlement** with concrete classes
    - Role
    - ResourceRole
    - Permission

Using the dependency inversion principle per the requirements, the implementation heavily relies on interfaces preventing tight coupling. For a full understanding of the all the classes involved, refer the Class Dictionary section.

Command syntax

The Authentication Service supports a command line interface. A TestDriver main class in the test module accepts scripts, parses them and orchestrates tasks.

```
#Creates a command processor
register-cmd userid cmd

#Creates a user
register-user userid cont

#Add credentials
add-credentials userid cmd username cmd-processor password cmd123

#Create a permission object
create-permission permid CREATE permname perm_create permdesc creating_perm

#Create a role object
create-role roleid admin rolename administrator roledesc admin_of_store

#Adds permission to a role
add-permission-to-role roleid admin permid CREATE

#Adds a child role to a parent role
add-child-role-to-parent-role roleid admin childroleid internal_system

#Gets role by id
get-role-by-id roleid admin

#Gets permission by id
get-permission-by-id CREATE

#Gets user by id
get-user-by-id userid cmd

#Adds entitlement to user
add-entitlement-to-user userid cmd roleid admin

#Logs in a user which generates a token
log-in userid cmd username cmd-processor password cmd123

#Creates a resource
create-resource resourceid store_123 resourcename Kmart

#Prints the details of inventory
get-authentication-inventory-print
```

#Gets resource by id
get-resource-by-id store_123

#Creates a resource role
create-resource-role resroleid commander resrolename commander_of_resource resroledesc commander_of_resource resourceid store_123

#Adds entitlement to resource role
add-entitlement-to-resource-role resroleid commander roleid internal_system

#Adds resource to resource role
add-resource-to-resource-role resroleid commander resourceid store_123

#Adds resource role to user
add-resource-role-to-user userid cmd resroleid commander

#Gets resource role by id
get-resource-role-by-id commander

#Gets entitlement by id
get-entitlement-by-id registered_user

#Adds child resource role to parent resource role
add-child-resource-role-to-resource-role roleid admin childroleid commander

#Creates a customer and adds him as a user
define-customer cust_AB first_name JSON last_name WALLACE type guest email_address json.wallace@ymail.com account json userid cust_AB

#Adds username/password credentials to a customer
add-credentials userid cust_AB username cust_AB password 123cust_AB

#Adds voiceprint/faceprint credentials to a customer
add-voiceprint userid cust_AB --voice:cust_AB--
add-faceprint userid cust_AB --face:cust_AB--

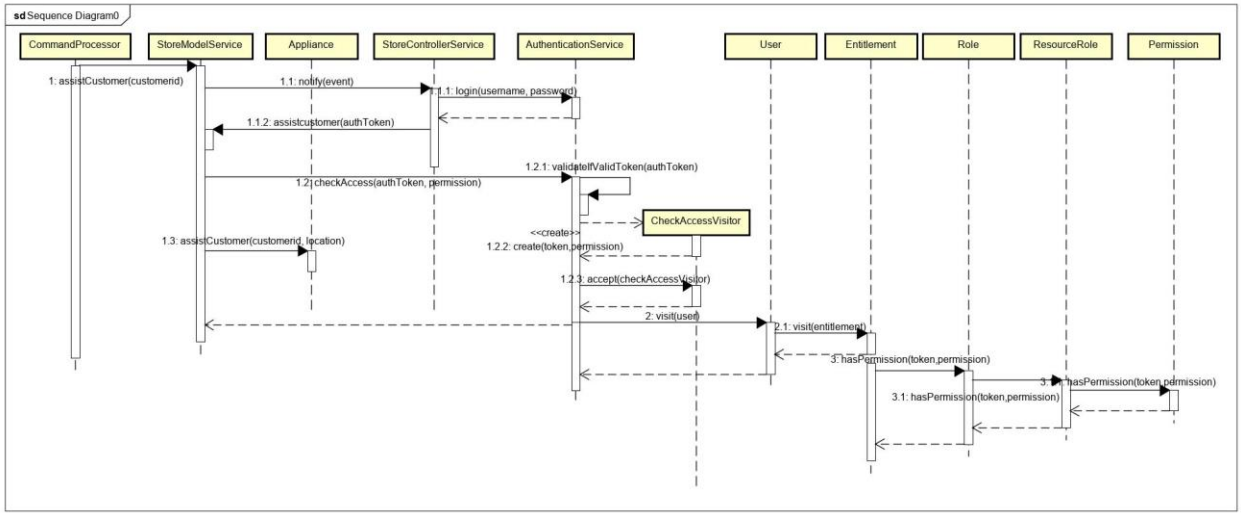
#Logs in a customer with voice/face print
log-in-face userid cust_22 --face:cust_22--
log-in-voice userid cust_E2 --voice:cust_E2--

#Logs out a customer
log-out userid cust_AB

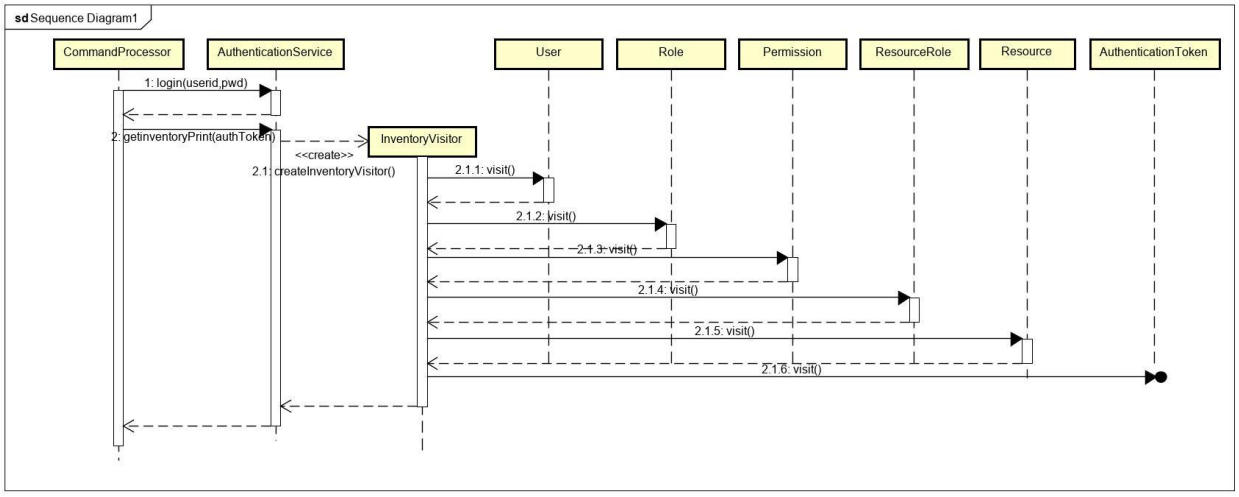
Commands to run the program

```
#Compile
javac com/cscie97/store/ledger/*.java  com/cscie97/store/model/*.java
com/cscie97/store/controller/*.java com/cscie97/store/authentication/*.java
com/cscie97/store/test/*.java

#Run a happy path test script
java -cp . com.cscie97.store.test.TestDriver store.script

#Validate how exceptions are handled
java -cp . com.cscie97.store.test.TestDriver store-exception.script
```

Sample sequence diagram for assisting customer

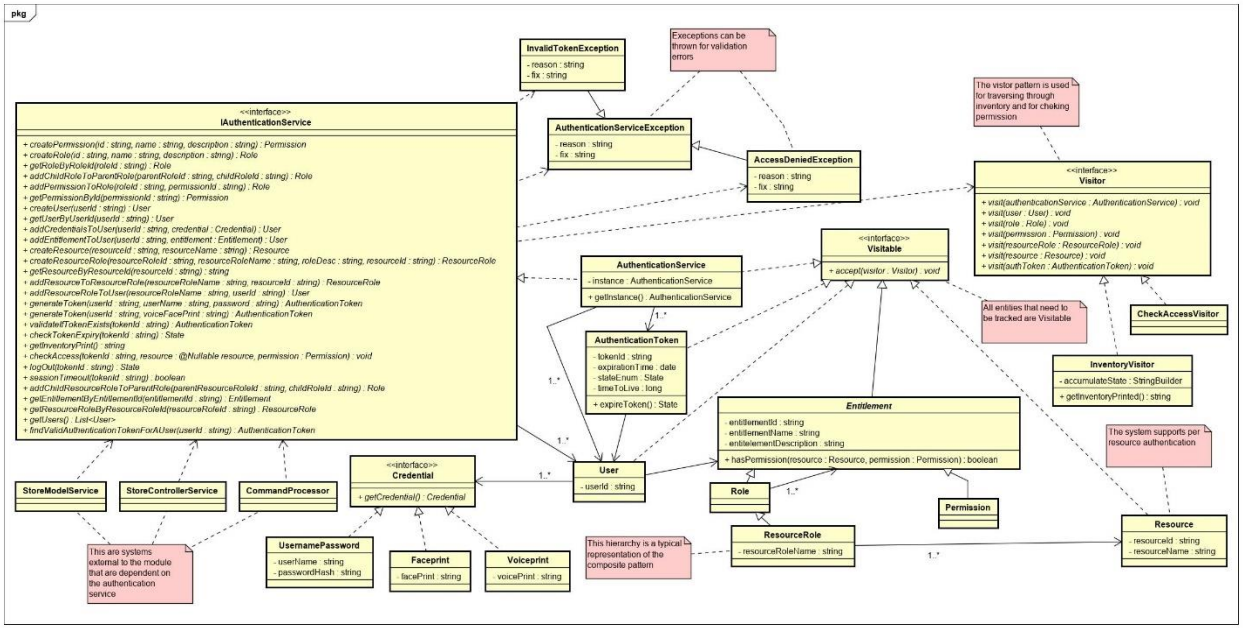


Sample sequence diagram for gathering inventory details and printing



Class Diagram

The following diagram mainly shows classes under the package 'com.cscie97.store.authentication'. A snapshot of the services dependent on the authentication service are also shown. There were some modifications to the existing systems to allow security controls.



Class Dictionary

This section describes the classes under the package 'com.cscie97.store.authentication'. Finer implementation details are also discussed when necessary. Modifications to the existing Store Model Service are also discussed.

IAuthenticationService

A top-level interface exposing methods accessible to the outside world. This interface is tasked with defining entities and providing access to them once they are defined. This interface is an orchestration engine and an entry point to the service.

Methods

Method Name	Signature	Description
createPermission	(String permissionId, String permissionName, String permissionDescription): Permission	Creates a new permission

createRole	(String roleId, String roleName, String roleDescription): Role	Creates a new role
addPermissionToRole	(String roleId, String permissionId): Role	Adds permission to a role
addChildRoleToParentRole	(String parentRoleId, String childRoleId): Role	Adds a child role to a parent role. The parent role is returned
getRoleById	(String roleId): Role	Finds a role by id
getPermissionById	(String permissionId): Permission	Finds a permission by id
createUser	(String userId): User	Creates a user
getUserByUserId	(String userId): User	Gets a user by id
addCredentialsToUser	(String userId, Credential credential): User	Adds a credential to a user
addEntitlementToUser	(String userId, Entitlement entitlement): User	Adds an entitlement to a user
createResource	(String resourceId, String resourceName): Resource	Creates a new resource
getResourceByResourceId	(String resouceId): Resource	Gets a resource by id
createResourceRole	(String resourceRoleId, String resourceRoleName, String resourceRoleDesc, String resourceId): ResourceRole	Creates a new resource role
addEntitlementsToResourceRole	(String resourceRoleId, String entitlementId): ResourceRole	Adds a child entitlement to a resource role. The parent resource role is returned
addResourcesToResourceRole	(String resourceId, String resourceRoleId): ResourceRole	Adds resources to a resource role. The parent resource role is returned
getResourceRoleByResourceRoleId	(String resourceRoleId): ResourceRole	Gets a resource role by id
getEntitlementByEntitlementId	(String entitlementId): Entitlement	Gets an entitlement by id
addResourceRoleToUser	(String userId, String resourceRoleId): User	Adds a resource role to a user. The user is returned
addChildResourceRoleToParentRole	(String parentRoleId, String childResourceRoleId): Role	Adds a child resource role to a parent resource role. The parent resource role is returned

generateToken	(String userId, String 17ogout1717, String password): AuthenticationToken	Generates an authentication token if the userid, username and the password hashed then stored match with a user. There will be a unique id assigned to the token and is valid for 15 minutes. This is triggered by logging in.
generateToken	(String userId, String voiceFacePrint): AuthenticationToken	Generates an authentication token if the userid and voice/face print match with a user. There will be a unique id assigned to the token and is valid for 15 minutes. This is triggered by logging in.
validateIfTokenExistsAndIsValid	(String tokenId): AuthenticationToken	An initial check if a token exists whether it is valid and has the required permissions or not.
checkTokenExpiry	(String tokenId): State	Looks up a token by token id and returns whether it is expired or not
sessionTimedOut	(String tokenId): boolean	This is triggered when a token lasts for more than 15 minutes
logOut	(String userId): State	A user can call this method at any point and the token will be forced to expire
getInventoryPrint	(): String	Used to get a detailed report of the entities
getUsers	(): List<User>	Returns a list of registered users
getTokens	(): List<AuthenticationToken>	Gets list of tokens managed. Only used just internally by objects in the same package
findValidAuthenticationTokenForAUser	(String userId): AuthenticationToken	Returns a list of tokens valid or not. This method is used just internally by the Store Controller Service to find out which token is associated with a user. Once the Store Model Service receives the token, further validation will be done by the Authentication Service to check for the correct permissions
checkAccess	(String tokenId, Resource resource, Permission permission): AccessDeniedException	This method checks if any of the entities provisioned by the Store Model Service grant access to the user associated with the token id. A visitor pattern simplifies the way of traversing the entitlement tree using the CheckAccessVisitor class which returns whether a permission is found.

AuthenticationService

Provides implementation for the authentication service. It stores users, entitlements, resources and tokens

Methods

Method Name	Signature	Description
createPermission	(String permissionId, String permissionName, String permissionDescription): Permission	Creates a new permission
createRole	(String roleId, String roleName, String roleDescription): Role	Creates a new role
addPermissionToRole	(String roleId, String permissionId): Role	Adds permission to a role
addChildRoleToParentRole	(String parentRoleId, String childRoleId): Role	Adds a child role to a parent role. The parent role is returned
getRoleById	(String roleId): Role	Finds a role by id
getPermissionById	(String permissionId): Permission	Finds a permission by id
createUser	(String userId): User	Creates a user
getUserByUserId	(String userId): User	Gets a user by id
addCredentialsToUser	(String userId, Credential credential): User	Adds a credential to a user
addEntitlementToUser	(String userId, Entitlement entitlement): User	Adds an entitlement to a user
createResource	(String resourceId, String resourceName): Resource	Creates a new resource
getResourceByResourceId	(String resouceId): Resource	Gets a resource by id
createResourceRole	(String resourceRoleId, String resourceRoleName, String resourceRoleDesc, String resourceId): ResourceRole	Creates a new resource role
addEntitlementsToResourceRole	(String resourceRoleId, String entitlementId): ResourceRole	Adds a child entitlement to a resource role. The parent resource role is returned

addResourcesToResourceRole	(String resourceId, String resourceRoleId): ResourceRole	Adds resources to a resource role. The parent resource role is returned
getResourceRoleByResourceRoleId	(String resourceRoleId): ResourceRole	Gets a resource role by id
getEntitlementByEntitlementId	(String entitlementId): Entitlement	Gets an entitlement by id
addResourceRoleToUser	(String userId, String resourceRoleId): User	Adds a resource role to a user. The user is returned
addChildResourceRoleToParentRole	(String parentRoleId, String childResourceRoleId): Role	Adds a child resource role to a parent resource role. The parent resource role is returned
generateToken	(String userId, String 19ogout1919, String password): AuthenticationToken	Generates an authentication token if the userid, username and the password hashed then stored match with a user. There will be a unique id assigned to the token and is valid for 15 minutes. This is triggered by logging in.
generateToken	(String userId, String voiceFacePrint): AuthenticationToken	Generates an authentication token if the userid and voice/face print match with a user. There will be a unique id assigned to the token and is valid for 15 minutes. This is triggered by logging in.
validateIfTokenExistsAndIsValid	(String tokenId): AuthenticationToken	An initial check if a token exists whether it is valid and has the required permissions or not.
checkTokenExpiry	(String tokenId): State	Looks up a token by token id and returns whether it is expired or not
sessionTimedOut	(String tokenId): boolean	This is triggered when a token lasts for more than 15 minutes
logOut	(String userId): State	A user can call this method at any point and the token will be forced to expire
getInventoryPrint	(): String	Used to get a detailed report of the entities
getUsers	(): List<User>	Returns a list of registered users
getTokens	(): List<AuthenticationToken>	Gets list of tokens managed. Only used just internally by objects in the same package
findValidAuthenticationTokenForAUser	(String userId): AuthenticationToken	Returns a list of tokens valid or not. This method is used just internally by the Store Controller Service to find out which token is associated

		with a user. Once the Store Model Service receives the token, further validation will be done by the Authentication Service to check for the correct permissions
checkAccess	(String tokenId, Resource resource, Permission permission): AccessDeniedException	This method checks if any of the entities provisioned by the Store Model Service grant access to the user associated with the token id. A visitor pattern simplifies the way of traversing the entitlement tree using the CheckAccessVisitor class which returns whether a permission is found.

Properties

Property Name	Type	Description
instance	IAuthenticationService	Used to provide a singleton instance to clients

Associations

Association Name	Type	Description
users	List<User>	Users stored here
tokens	List<AuthenticationToken>	Tokens stored here. Easy for lookup
entitlements	List<Entitlement>	Entitlements stored here. Easy for lookup
resources	List<Resource>	Resources stored here. Easy for lookup

Ivisitor

This interface sets the contract for the type of visitors that are defined in this service. Implementations are free to provide logic that leverages the visitor design pattern and abides by this contract.

Methods

Method Name	Signature	Description
visit	(AuthenticationService authenticationService): void	Visits AuthenticationService. Implementations can vary
visit	(User user): void	Visits User. Implementations can vary

visit	(Role role): void	Visits Role. Implementations can vary
visit	(Permission permission): void	Visits Permission. Implementations can vary
visit	(ResourceRole resourceRole): void	Visits ResourceRole. Implementations can vary
visit	(Resource resource): void	Visits Resource. Implementations can vary
visit	(AuthenticationToken authenticationToken): void	Visits AuthenticationToken. Implementations can vary

Visitable

All entities that need to be visited implement this interface

Methods

Method Name	Signature	Description
accept	(Ivisitor visitor): void	Any object of type Ivisitor can be passed into this method. Provisioned entities need to implement it.

CheckAccessvisitor

This is a special type of visitor that traverses through the entitlement tree that was structured using the composite pattern and checks for a specific permission along its way. Each of the entitlement entities delegate to their children to find permissions. The permission flag values are retained when traversing the entitlements tree and if anywhere in the tree, the hasPermission flag is true, this visitor returns a positive response

Methods

Method Name	Signature	Description
visit	(AuthenticationService authenticationService): void	Delegate to users to find permission
visit	(User user): void	Delegate to entitlements to find permission
visit	(Role role): void	Tries to find if the permission is found in the role object and delegates to its children if it is

		not found
visit	(Permission permission): void	Tries to find if the permission matches and backtracks if it doesn't
visit	(ResourceRole resourceRole): void	Tries to find if a global permission is found in the resource role object or tries to find a restricted permission tied to a resource that it contains. Delegates to its children if permission is not found
visit	(Resource resource): void	Not applicable
visit	(AuthenticationToken authenticationToken): void	Not applicable
hasPermission	(): boolean	The permission flag values are retained when traversing the entitlements tree and if anywhere in the tree, hasPermission == true, this method will return a positive response

Properties

Property Name	**Type**	**Description**
hasPermission	List<Boolean>	Stores the result of traversing the tree looking for permission

Associations

Association Name	**Type**	**Description**
token	AuthenticationToken	A token the user is associated with the user must be injected
resource	Resource	A resource the user is asking permission to use maybe passed in. Global access to all resources is checked for if this is not passed.
Permission	Permission	The permission the user is asking to be granted

InventoryVisitor

This visitor traverses through all the entities and collects their details to be used later for printing. The navigation logic is inside the visitor itself in this case.

Methods

Method Name	Signature	Description
getInventoryPrint	(): String	Returns the accumulated details by the visitor
visit	(AuthenticationService authenticationService): void	Accumulate details of authentication service and delegate to its children
visit	(User user): void	Accumulate details of user object and delegate to its children
visit	(Role role): void	Accumulate details of role object and delegate to its children
visit	(Permission permission): void	Accumulate details of a permission object. Since this is a leaf object, it doesn't delegate to its children unlike the other objects
visit	(ResourceRole resourceRole): void	Accumulate details of resource role object and delegate to its children
visit	(Resource resource): void	Accumulate details of a resource object
visit	(AuthenticationToken authenticationToken): void	Accumulate details of an authentication token object

Properties

Property Name	Type	Description
accumulateState	StringBuilder	Used to accumulate the details of entities

User

A user object contains credentials and entitlements. Credentials can be voice prints, face prints or username/password combo. It also implements the Visitable interface. Registered users have higher privileges than guest users. The admin is responsible to assign users the appropriate roles during registration. Until a user is registered, he will only be treated as a guest user.

Methods

Method Name	Signature	Description
addCredentials	(Credential credential)	Adds credential to a user.

	: void	
addEntitlements	(Entitlement entitlement): void	Adds entitlement to a user
checkCredentials	(String 24ogout2424, String 24ogout2424): 24ogout24	Username/password combo checked if they match with this user's credentials
checkCredentials	(String voiceFacePrint): boolean	Faceprint/voiceprint checked if they match with the user
computePasswordHash	(String password): String	Computes the hash of a password passed in
accept	(Ivisitor ivisitor): void	Used by the visitor to access this object

Properties

Property Name	**Type**	**Description**
userId	String	Uniquely identifies a user

Associations

Association Name	**Type**	**Description**
credentials	List<Credential>	A user can have multiple credentials, but they must be different types of credentials like voice print, face print or username/password combo.
Entitlements	List<Entitlement>	A user has one to many entitlements. Entitlements can be composed of each other

Credential

A marker interface for credentials of type VoicePrint, FacePrint and UsernamePassword.

Methods

Method Name	**Signature**	**Description**
getCredential	(): Credential	Returns the object itself

UsernamePassword

Contains username and password

Methods

Method Name	Signature	Description
getCredential	(): Credential	Returns the object itself
setPasswordHash	(String password): void	Hashes the password and stores it

Properties

Property Name	Type	Description
userName	String	Username of a user
passwordHash	String	Contains the hash of password

VoicePrint

Contains voice print of a user

Methods

Method Name	Signature	Description
getCredential	(): Credential	Returns the object itself

Properties

Property Name	Type	Description
voicePrint	String	Voice print of a user

FacePrint

Contains face print of a user

Methods

Method Name	Signature	Description
getCredential	(): Credential	Returns the object itself

Properties

Property Name	Type	Description
facePrint	String	Face print of a user

Entitlement

A non-instantiable class that is visitable and used to define the component of the composite pattern. Entities to be described later extend this class to make use of a parental hierarchy. It defines some abstract methods to simplify checking for permission.

Methods

Method Name	Signature	Description
hasPermission	(AuthenticationToken authToken, Resource resource, Permission permission): boolean	Abstract method concrete objects should provide different implementations for based on whether they are composite or leaf objects

Role

A Role leverages the composite pattern to self-contain other roles and permissions. Therefore, a role is a composite object and implements the Entitlement interface.

Methods

Method Name	Signature	Description
addEntitlement	(Entitlement entitlement): void	Adds a child entitlement to a role

accept	(Ivisitor ivisitor): void	Attaches a visitor to a role
hasPermission	(AuthenticationToken authToken, Resource resource, Permission permission): boolean	Checks if this role has the required permission and delegates to its children if it doesn't

Properties

Property Name	Type	Description
roleId	String	Id of a role that uniquely identifies a role
roleName	String	Name of a role
roleDescription	String	Description of a role

Associations

Association Name	Type	Description
entitlements	List<Entitlement>	Contains children entitlement objects which themselves can be composed of other entitlement objects

ResourceRole

A ResourceRole is a special type of a role. A ResourceRole like a Role leverages the composite pattern to self-contain other entitlements. Therefore, a resource role is a composite object and implements the Entitlement interface.

Methods

Method Name	Signature	Description
addEntitlement	(Entitlement entitlement): void	Adds a child entitlement to a resource role
addResources	(Resource resource): void	Adds a resource to a resource role
accept	(Ivisitor ivisitor): void	Attaches a visitor to a resource role

hasPermission	(AuthenticationToken authToken, Resource resource, Permission permission): boolean	Checks if this resource role has the required permission and has global access to a resource or if the resource matches what is contained here. Delegates to its children if it doesn't
checkResource	(Resource resource): 28ogout28	Used internally to check if a resource match

Properties

Property Name	**Type**	**Description**
resourceRoleId	String	Id of a resource role that uniquely identifies a resource role
resourceRoleNam e	String	Name of a resource role
resourceRoleDesc ription	String	Description of a resource role

Associations

Association Name	**Type**	**Description**
entitlements	List<Entitlement>	Contains children entitlement objects which themselves can be composed of other entitlement objects
resources	List<Resource>	Contains resources that can be given

Permission

A leaf object that contains whether access is granted or not.

Methods

Method Name	**Signature**	**Description**
hasPermission	(AuthenticationToken authToken, Resource resource, Permission permission): boolean	Checks if the permissions match using a permission id
accept	(Ivisitor ivisitor): void	Makes a visitor access this object

Properties

Property Name	Type	Description
permissionId	String	Id of permission that uniquely identifies it
permissionName	String	Name of permission
permissionDescription	String	Description of permission

AuthenticationToken

An authenticated user will be assigned a token that is valid for 15 minutes from the time of authentication.

Methods

Method Name	Signature	Description
29ogout	(): State	Called when a user is logged out and expires the token. For a security precaution, it is recommended to log out after finishing performing tasks even though the session automatically logs out users after the expiration time.
expireToken	(): State	Called when a user's session exceeds 15 minutes
accept	(Ivisitor ivisitor): void	An authentication token is visitable by visitors like InventoryVisitor

Properties

Property Name	Type	Description
tokenId	UUID	Automatically assigned unique during token generation
timeToLive	long	15L is the default time to live for a token
expirationTime	long	Expiration is calculated by current time plus time to live
expirationState	State	Expiration state is an enum that can be either ACTIVE or EXPIRED

Associations

Association Name	Type	Description
user	User	A token is associated with just one user, but a user can have many authentication tokens

State

The only two valid states for a token are active and expired

Properties

Property Name	Type	Description
ACTIVE	String	Enum value for a valid token
EXPIRED	String	Enum value for a token when a user logs out a session times out

AuthenticationServiceException

The parent of all the exceptions thrown in the Authentication Service.

Properties

Property Name	Type	Description
reason	String	Reason of exception
fix	String	A hint how to fix it

AccessDeniedException

Extends AuthenticationServiceException and thrown when a user is not granted permission to a resource

Properties

Property Name	Type	Description
reason	String	Reason of exception
fix	String	A hint how to fix it

InvalidTokenException

Extends AuthenticationServiceException and thrown when a user is trying to authenticate with an invalid token

Properties

Property Name	Type	Description
reason	String	Reason of exception
fix	String	A hint how to fix it

MODIFICATION TO THE STORE MODEL SERVICE

A PermissionType enumeration was added to the Store Model Service. The permissions were categorized into CREATE, READ, UPDATE and DELETE. The Authentication Service administrator assigns these permissions based on roles.

Properties

Property Name	Type	Description
permission	String	Specifies the type of permission that can be used to add more metrices for permission
CREATE	String	An enum value for creating a resource
READ	String	An enum value for reading a resource
UPDATE	String	An enum value for updating a resource
DELETE	String	An enum value for deleting a resource

Implementation Details

Design choices

- **Privileges**

I found it natural to categorize permissions into four categories: create, read, update and delete. These privileges are defined in the context of the Store Model Service and are confined to it. However, the Authentication service can handle other types of privileges. The increasing order of privileges is

1) **READ** – guest users have the least privilege and can perform simple tasks that don't involve altering the entities

2) **UPDATE –** this is reserved for privileges starting from registered users and above. Registered users however are not equipped to perform administrative tasks under this category like updating inventory count. Updating a basket is an example for what a registered user can perform.

3) **DELETE –** this is reserved for privileges starting from internal systems and above. Internal systems like the Store Controller Service have the internal systems privilege and can perform tasks like removing a customer's association with a basket through the Store Model Service API.

4) **CREATE –** this is reserved for the administrator which is the highest privilege. Provisioning the store such as defining aisles and products is solely restricted to the administrator.

- **Having the Store Model Service be open accept any token and delegate to the CheckAccessVisitor to look for permissions/validity**

The authentication logic in the Store Model Service is the topmost logic resulting in an AccessDeniedException if authentication fails. The Store Controller Service obtains a token on behalf of the customer or on behalf of itself without worrying about whether it has the required permission to perform the actions in the command. Once the token gets to the Store Model Service, the Authentication Service will look for the required permission using the CheckAccessVisitor. This saves a trip to the Authentication Service.

- **Checking for global access then access restricted to a resource**

  By default, a global access is looked for when checking access and if that is not found, it drills down a specific access to the resource that is passed in.

- The requirement says, “When new Consumers are created, a corresponding user will be created within the Authentication Service with a default voice print and face print”. This design diverges slightly from this requirement since first time users need to authenticate with username and password until their first shopping experience where their voice/face print will be identified.

- The command processor has been singled out as a self-contained module per the requirements and is an orchestration engine per the requirements.

- The use of singleton, visitor and composite patterns have been discussed extensively in the class dictionary per requirements.

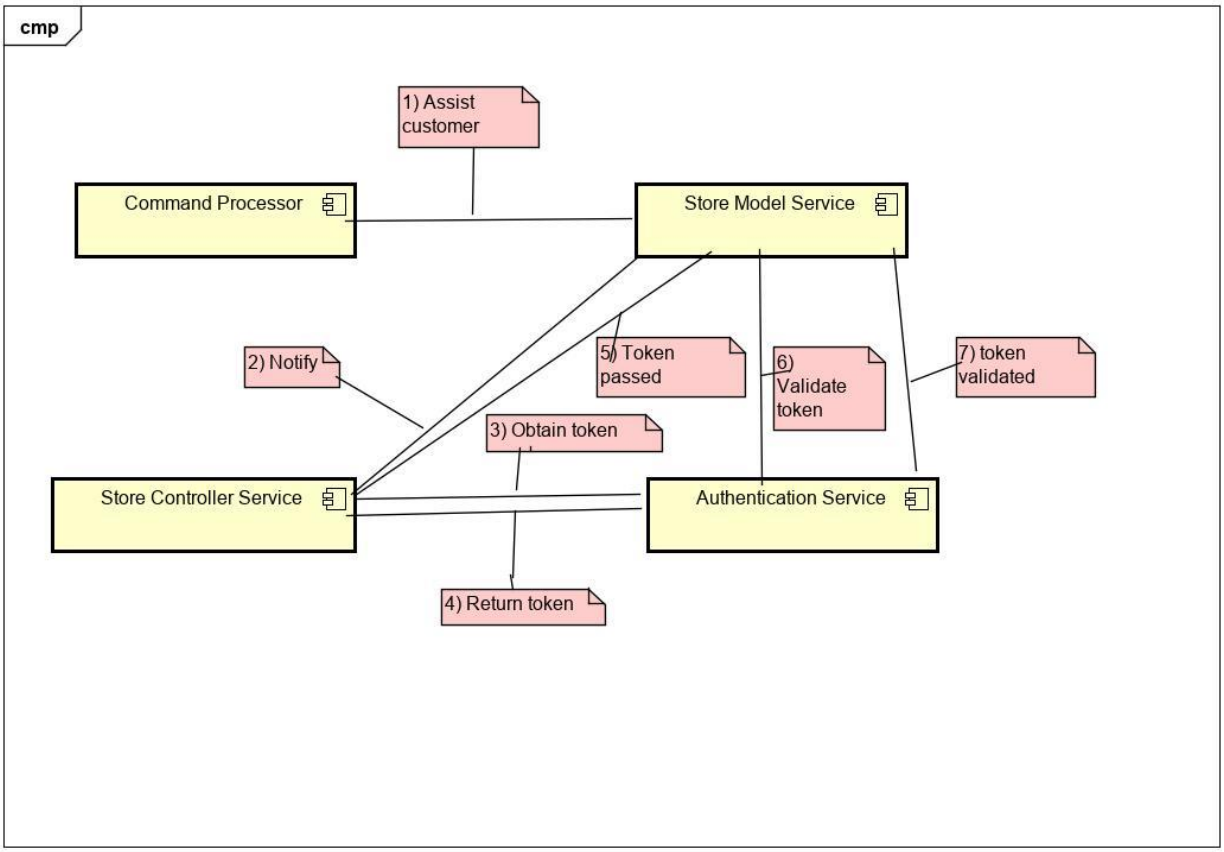
  “The visitor pattern is used to keep track of and collect all the entities that make up the authentication service” – This is handled by the InventoryVisitor class.

  “The visitor pattern also enabled a quick, recursive navigation to find permissions granting access of a resource to a user” – This is handled by the CheckAccessVisitor class.

  “The composite pattern builds the entitlement hierarchy constituting of roles, permissions and resource roles” – This represents the Entitlement, Role, ResourceRole and permission classes that are implemented.

  “The singleton pattern was also a good fit to the AuthenticationService class since in a centralized service like this one, it is essential to lump together the state in one instance of an object” – There is only one instance of AuthenticationService class.

The modules integrate seamlessly, and the full picture of the system has been realized. The image below shows how the modules interact



Exception Handling

Authentication Service defines three exceptions with one parent and two children exceptions.

- **AuthenticationServiceException** – a generic exception that is thrown mostly for housekeeping purposes. An example is when trying to add permission to a role, but the role was not initially defined. Generally, trying to access an inventory that is not defined throws this exception.
- **AccessDeniedException –** a child exception to AuthenticationServiceException. This exception is thrown when a user doesn't have the permission to access a resource. The user has a valid token in this case but doesn't have the required permission.
- **InvalidTokenException –** a child exception to AuthenticationServiceException. This exception is thrown when the token is not associated a user or is expired.

All the exceptions have two properties

- **reason** – why the exception is thrown
- **fix** – a hint how to fix the issue

The design is fault tolerant, handles exceptions gracefully and continues to operate. The error message is logged to the console as a warning. Some exceptions like AccessDeniedException are handled by the CommandProcessor itself so that an authenticated client is revoked access and is informed about it.

Testing

- **Test Driven Development**

  In addition to the test scripts, unit tests with the Junit framework may have added value by avoiding cutting corners. It also makes the system adhere to the requirements and nothing more. The Single Responsibility Principle will also be satisfied using this approach by defining boundaries around what the classes and methods handle.

- **Functional**

  Having the test scripts ready in the design face guided conceptualizing the system and defining the API. However, there is a room for more rigorous functional testing by considering many scenarios. It was not possible to test every edge case that is in a robust authentication service in the scope of this design and the implementation will benefit from adding more test scenarios.

- **Performance**

  Appropriate data structures were chosen in the design to drive the implementation towards readability and performance at the same time. Sending loads of requests and gauging how the

system handles it maybe a good approach to test performance and scalability.

- **Regression**

  The system will benefit from a continuous integration pipeline to avoid surprises at the end of development. Building the system in iterations while making sure that existing flows are not broken gives a feedback to the developer. The CI/CD plugin runs the test scripts in every build and keeps the system in check.

- **Exception Handling**

  A script is provided how to test scenarios like how the Store Model Service treats unauthenticated users or what happens when an inventory that doesn't exist is requested. The error messages and a potential fix are logged to the console. The testing can be extended to handle cross module scenarios instead of testing the modules in isolation.

Risks

- It may be advisable to use commercial of the shelf software for cross cutting concerns like security. Active Directory Federation Services is one which I have used myself in the past. Building a custom solution of my own, however gives me an opportunity to learn what goes under the hood and to debug when something goes wrong. It was also a good exercise on utilizing design patterns in a complex enough module.

- The session timeout mechanism is not robust even though users are reminded to log out. There must be some pinging functionality or a batch job that runs periodically to forces tokens that exceeded the time to live to expire.

- A vault system that spits out passwords when requested maybe a good option to store passwords. A database that stores encrypted credentials might also be an alternative.