Authentication Service Design Document

Date: 11/21/23
Author: Eric Margolis
Reviewer(s): Sara Daqiq , So Hyeong Lee

Table of Contents

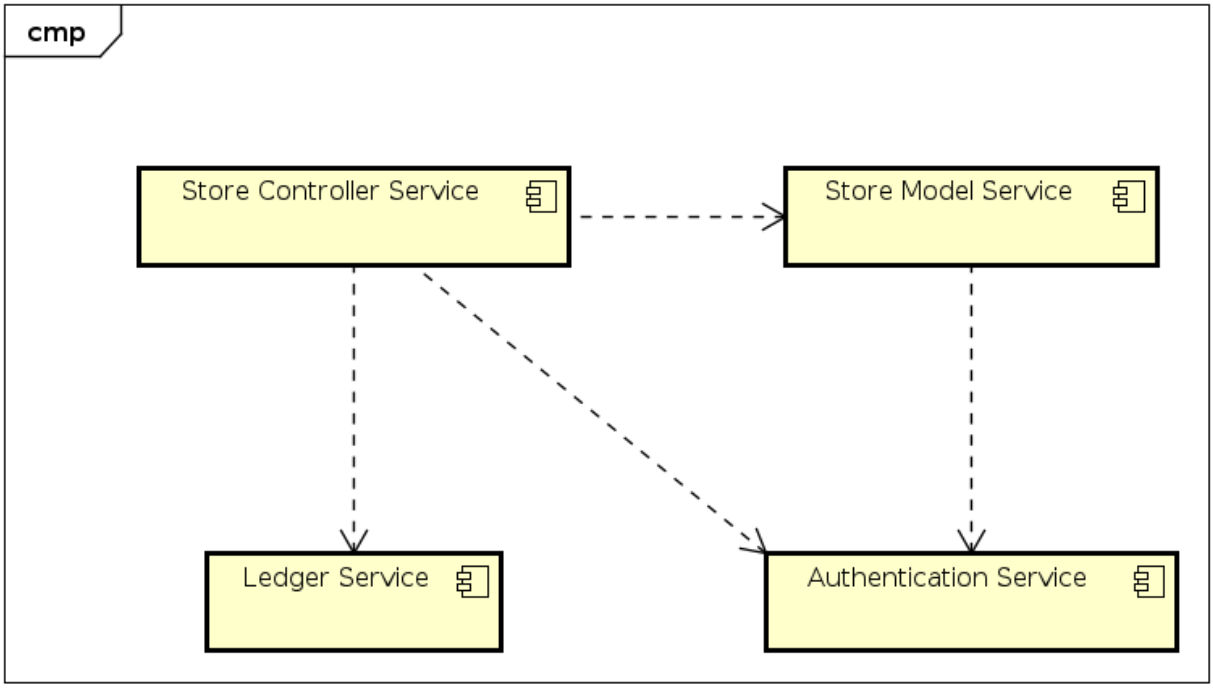
Introduction

This document describes the design of the Authentication Service module of the Store 24X7 System.

Overview

The Authentication Service is a key component of the overall Store24 System because it provides the framework for the authentication and authorization of users to resources within the system. In the case of the Store24 system, users are customers, administrators, and the Controller Service, while resources include appliances, sensors, and Model Service methods.

It must be noted that the design of the Authentication Service will be modular and applicable to any system, not limited to the Store24 system.



Description: A UML component diagram displaying the relationships between the modules of the Store24X7 System

Requirements

- Use the Composite Pattern to create roles and permissions that can be applied to users and provide authorization to use service features.
- Authenticate users using password, face ID, or voice ID
    - Passwords must be stored as a hash
- Logical and physical entities must be modeled as resources for linking resources and roles
- Associate users with zero or more roles, permissions, or sub-roles
- Create user objects that represent registered users of the store
- Create authentication tokens used to access StoreModelService resources
- Provide exception classes and handling for authentication failure and attempted unauthorized access of resources
- Login feature that authenticates users and provides appropriate access tokens
- Logout feature that invalidates the user's access token
- Modify StoreModelService methods to accept auth tokens
    - Methods check the token is non-empty and non-null
    - Methods pass the token to the authentication service along with required permission information
- Validate auth tokens
- Verify that users requesting a restricted resource have the appropriate privileges
- Utilize the Visitor Pattern to support traversing the Authentication Service objects graph to inventory Users, Resources, Accesses, Roles, and Permissions.

Use Cases

Create Resource Access:
Administrators can define access requirements to resources using permissions and roles.

Create Permissions:
Administrators can create permissions which may be granted to users to allow them to access resources.

Create Roles:
Administrators can create and add users to roles which are groupings of permissions.

Create Users:
Administrators can create users which correspond to consumers of other services in the system.

Login:
Users can log in to get an access token using password, voice ID, or face ID.

Authenticate Users:
The service authenticates users by verifying the provided credentials are correct.

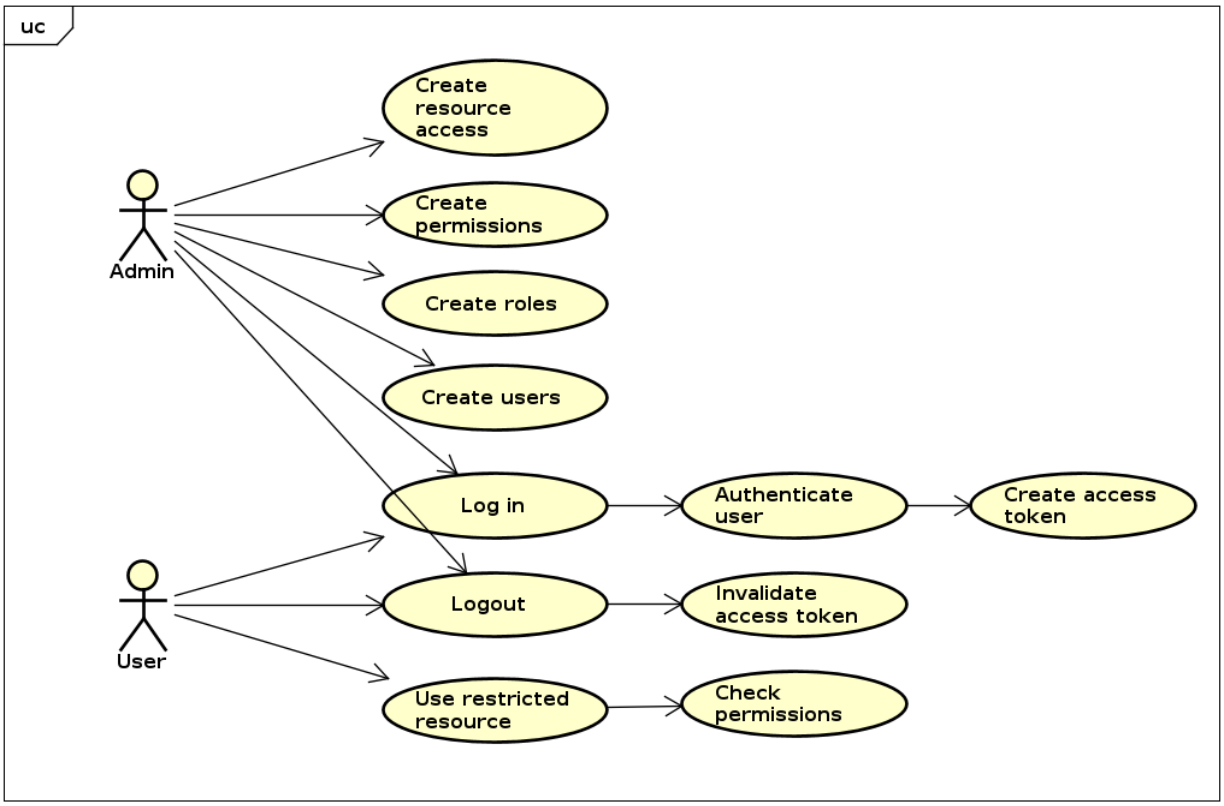
Create Access Token:
The service creates access tokens that can be assigned to authenticated users. Access tokens contain a set of privileges (roles and permissions) to indicate what the user has the privilege to access. Access tokens expire after a period of time.

Logout:
Users can log out, which invalidates their access token.

Invalidate Access Token:
Sets the access token's state to invalid, indicating that it should not be used for accessing resources any further,

Use Restricted Resource:
Authenticated users are allowed to use restricted resources if they have the appropriate privileges.

Check Permissions:
The Authentication Service checks whether a requesting user has the privilege to access a restricted resource.



Description: A UML use case diagram displaying the actors and their use cases.

Actors

Admin:

An extra-privileged user type that is able to perform executive functions such as creating permissions and resource accesses.

User:

A registered consumer of the service who may request to use restricted resources.

Main Success Scenario

1. User makes a login request using password, face ID, or voice ID
2. Authentication service validates the request
3. Authentication service generates an access token and assigns it to the user
4. User requests access to a restricted resource and provides access token
5. Resource checks the token is non-empty and non-null
6. Resource passes token and required permissions to the Authentication Service
7. Authentication Service validates the token
8. Authentication Service verifies to the resource that the token grants the user permission to the resource
9. Resource fulfills user request

Extensions:

3a. The user's credentials do not match the Authentication Service database
    .1 Authentication Service throws an AuthenticationException
    .2 Login request is denied

5a. No access token is provided
    .1 The resource throws an exception and denies the request

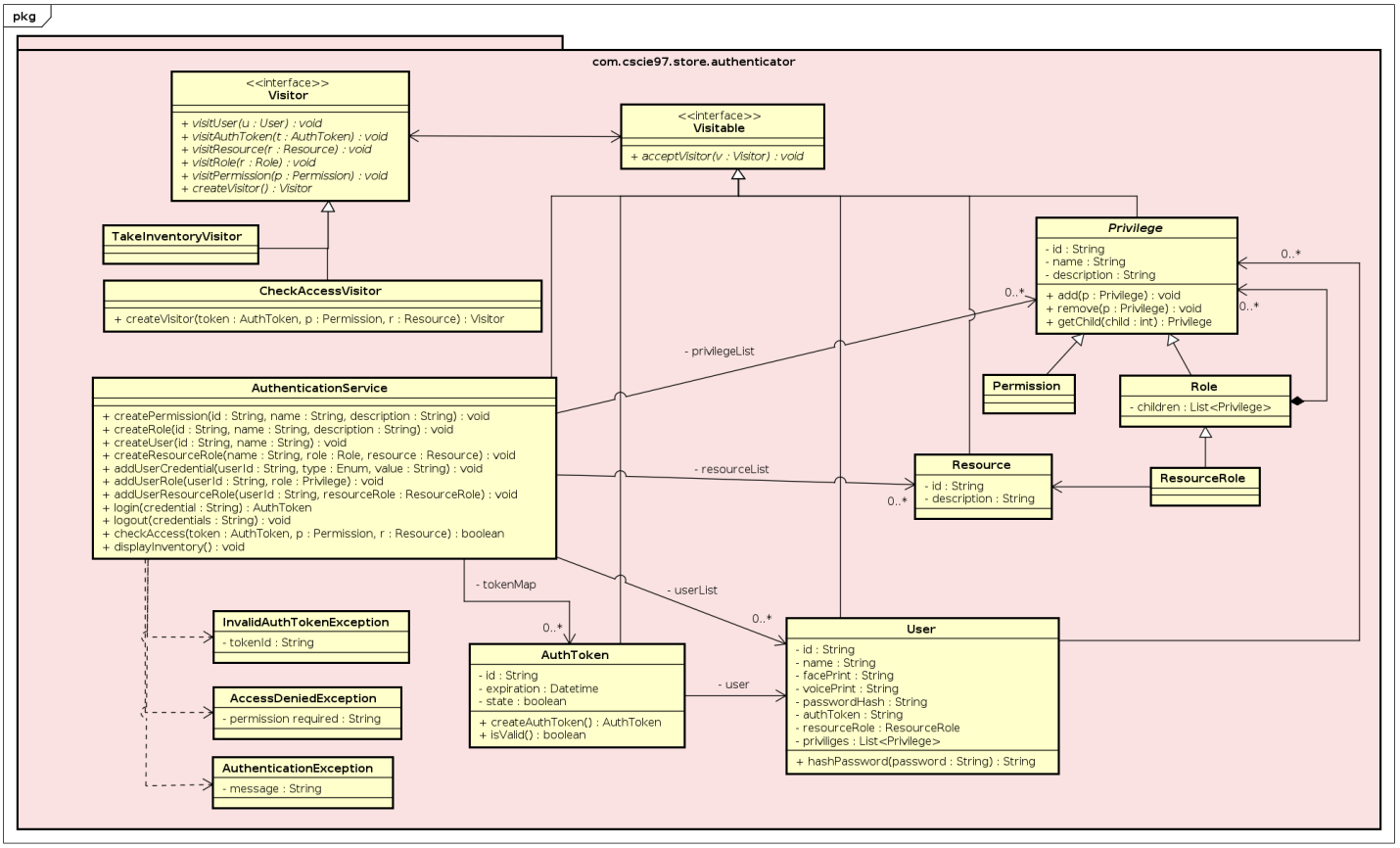
7a. The token is invalid
    .1 An InvalidAuthTokenException is thrown
    .2 The resource is notified and denies the request

8a. The user does not have privileged access to the resource
    .1 An AccessDeniedException is thrown
    .2 The resource is notified and denies the request

Design

Class Diagram

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



Description: A UML Class Diagram displaying the relationships between the Authentication

Service and supporting classes, including the Composite Pattern relationship of the abstract Privilege class and the Role and Permission classes that inherit from it.

Class Dictionary

The classes outlined in the above Class Diagram are detailed below.

Visitor (interface)

In accordance with the requirements document, this design utilizes the Visitor Pattern to check that Uses have the appropriate privileges to access any restricted resources they try to access and to take an inventory of the Authentication Service object graph.

Methods

Method Name	Signature	Description
visitXYZ	(XYZ xyz) void	Visits the object given and implements some behavior.
createVisitor	() : Visitor	This method instantiates a concrete Visitor.

Visitable (interface)

Pairs with the Visitor interface. Implements a method to accept the Visitor.

Methods

Method Name	Signature	Description
acceptVisitor	(Visitor v) : void	When called, the implemented class calls the given concrete Visitor and passes it a pointer to itself to facilitate the visitation.

TakeInventoryVisitor

This concrete Visitor visits every object in the Authentication Service object graph and gathers information about them.

CheckAccessVisitor

This concrete Visitor finds the User associated with the given AuthToken and checks whether they have privileges to match the ones required by the given resource.

Methods

Method Name	Signature	Description
createVisitor	(AuthToken token, Permission p, Resource r) : Visitor	Creates a CheckAccessVisitor that finds the User associated with the given AuthToken and checks whether they have permission to use the given resource.

AuthenticationService

This class provides a facade for user authentication and authorization checking. It has a number of methods for creating privileges, roles, users, and other objects required for authenticating users. Utilizes concrete Visitors to crawl the objects graph.
Per the requirements, this is a Singleton class.

Methods

Method Name	Signature	Description
createPermission	(String id, String name, String description) : void	Defines a new Permission
createRole	(String id, String name, String description) : void	Defines a new Role, which is a composite of type Privilege. Roles may have zero or more sub-roles and zero or more Permissions.
createUser	(String id, String name) : void	Creates a new User.
createResourceRole	(String name, Role role, Resource resource) : void	Creates a ResourceRole that binds Roles to Resources
addUserCredential	(String userId, Enum type, String value) : void	Adds the given credential to the User object.
addUserRole	(String userId, Privilege role) : void	Applies the given role to the given user.
addUserResourceRole	(String userId, ResourceRole resourceRole) : void	Applies the ResourceRole to the user. This binds the user to the resource; in the context of the Store24 User roles are bound to Stores.
login	(String credential) : AuthToken	Logs in the user with the given credential, if one exists. Throws an AuthenticationException if there is no matching user.
logout	(String credentials) : void	Logs out the user with the given credentials, if one exists. Logout invalidates the user's

		AuthToken. Throws an AuthenticationException if there is no matching user.
checkAccess	(AuthToken token, Permission p, Resource r) : bool	Uses the CheckAccessVisitor to check whether or not the user associated with the given token has matching permissions required to access the given resource.
displayInventory	() : void	Uses the TakeInventoryVisitor to get an inventory of the Authentication Service objects and displays the inventory to stdout.

Associations

Association Name	**Type**	**Description**
tokenMap	Map<String, AuthToken>	A map of all AuthTokens with IDs as keys.
userList	List<User>	A list of all Users in the system.
resourceList	List<Resource>	A list of all Resources in the system.
privilegeList	List<Privilege>	A list of all Privileges in the system.

AuthToken

The AuthToken class represents tokens with which users can access restricted resources. Each token has an ID, an expiration time, and a state: valid/invalid. Each token is associated with one user. Every public method in the StoreModelService requires an AuthToken to run. Expired tokens are rendered invalid.

Methods

Method Name	**Signature**	**Description**
createAuthToken	() : AuthToken	Generates a new AuthToken
isValid	() : Boolean	Returns True if valid, else false

Properties

Property Name	**Type**	**Description**
id	String	The unique token ID
expiration	Datetime	Sets the expiration time of the token. When

		this time is reached, the token should be rendered invalid.
state	Boolean	Valid/invalid

Associations

Association Name	Type	Description
user	User	Every AuthToken must be associated with exactly one User.

User

Users are actors in the system. Customers and administrators are both types of Users

Methods

Method Name	Signature	Description
hashPassword	(String password) : String	Returns the hash of the given password.

Properties

Property name	Type	Description
id	String	The unique ID of the user.
name	String	The user's full name
facePrint	String	The string representing the face ID of the user
voicePrint	String	The voice ID of the user
passwordHash	String	The user's password, hashed for security
authToken	String	The ID of the Auth Token the user is logged in with.
resourceRole	ResourceRole	The resourceRole contextualizes the user's role within a resource, in this case a Store.
privileges	List<Privilege>	The user's roles and permissions.

Privilege

An abstract class that represents roles and permissions associated with accessing restricted resources within the system. Part of utilizing the Composite Pattern, per the requirements.

Methods

Method Name	Signature	Description
add	(Privilege p) : void	Adds a new child component from the structure..
remove	(Privilege p) : void	Removes a child component.
getChild	(int child) : Privilege	Gets the child component corresponding to the given index.

Properties

Property Name	Type	Description
id	String	The ID of the prfivilege.
name	String	The human-friendly name of the privilege
description	String	The description of the privilege

Role

The composite component of the Privilege structure. Roles are parent nodes and made have zero or more children of type Permission or Role. Inherits from *Privilege*

Associations

Association Name	Type	Description
children	List<Privilege>	Any roles or permissions that this Role is the parent of.

Permission

The Permission is the leaf component of the structure. Permissions inherit from *Privilege*, but since they are leaf nodes in the structure, the methods do nothing.

Methods

Method Name	Signature	Description
add	(Privilege p) : void	Does nothing.
remove	(Privilege p) : void	Does nothing
getChild	(int child) : Privilege	Does nothing

Resource Role

This class ties Roles to Resources. In the Store24 System, roles are tied to Stores. Inherits from Role and *Privilege*.

Resource

Resources can be any physical or logical entities in the system. Restricted resources require AuthTokens to use.

Properties

Property Name	Type	Description
id	String	The ID of the resource.
description	String	The description of the resource.

InvalidAuthTokenException

Exception class thrown when the provided token has been invalidated by logout or expiration.

Properties

Property Name	Type	Description
tokenId	String	The ID of the token provided.

AccessDeniedException

This exception is thrown when the User associated with the provided AuthToken does not have the Privileges required to access the requested Resource.

Properties

Property Name	Type	Description

permissionRequired	String	The permission required to access the resource

AuthenticationException

Exception thrown when the AuthenticationService cannot find a User associated with the authentication credential given at the time of login.

Properties

Property Name	Type	Description
message	String	An error message tersely describing the problem.

Design Details

The Singleton Pattern is applied to the AuthenticationService, which acts similarly to the StoreObserver and StoreModelService singletons before it as providing a facade for the respective module's behavior.

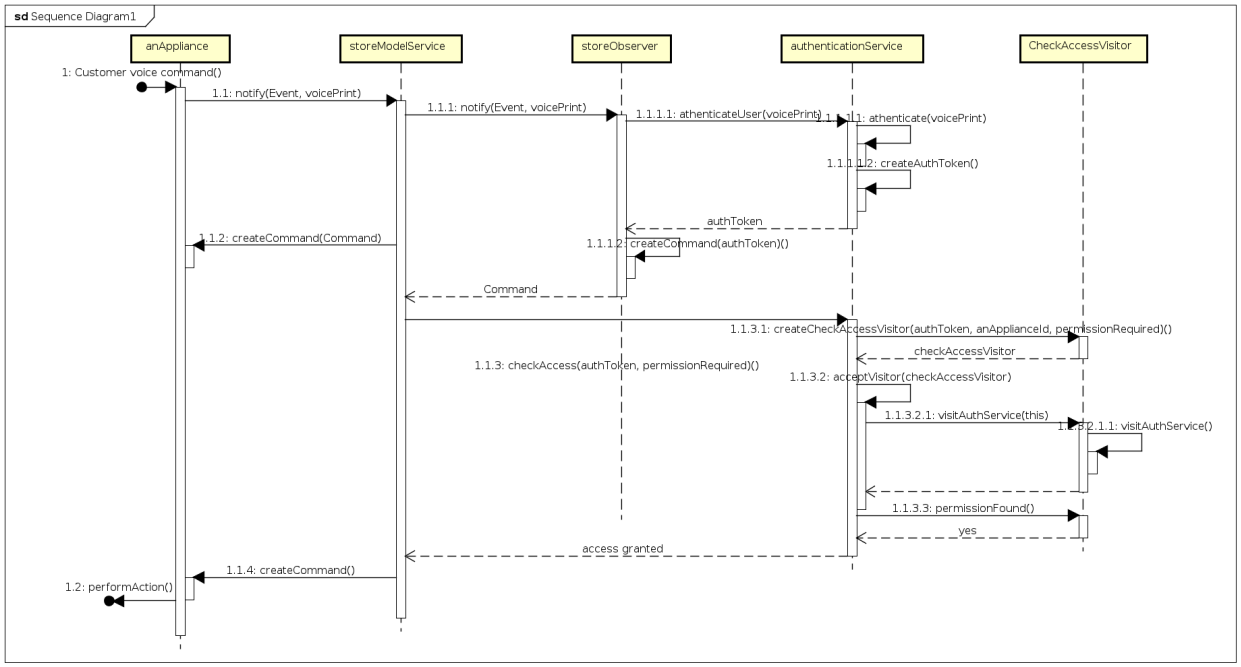
This design follows the security principles of least privilege and zero trust.
The former is achieved by applying the Composite Pattern to model roles and privileges as concrete instances of composite elements that can be applied as needed, and only as needed, to User objects, which fulfills the requirements of providing users access to service features via roles and permissions. The latter principle is followed by the Store Model Service in this case, which requires authorization tokens for all its public methods. The user objects are created by the Authentication Service, which fulfills the requirement of creating user objects to represent registered store customers.

In this design, non-administrative users are logged in automatically when they enter a store. The Store Controller retrieves a voice or face print from the appliances and sensors near the customer and uses these as credentials to authenticate the customer with the Authentication Service, which fulfills the requirements of user authentication and a login feature.

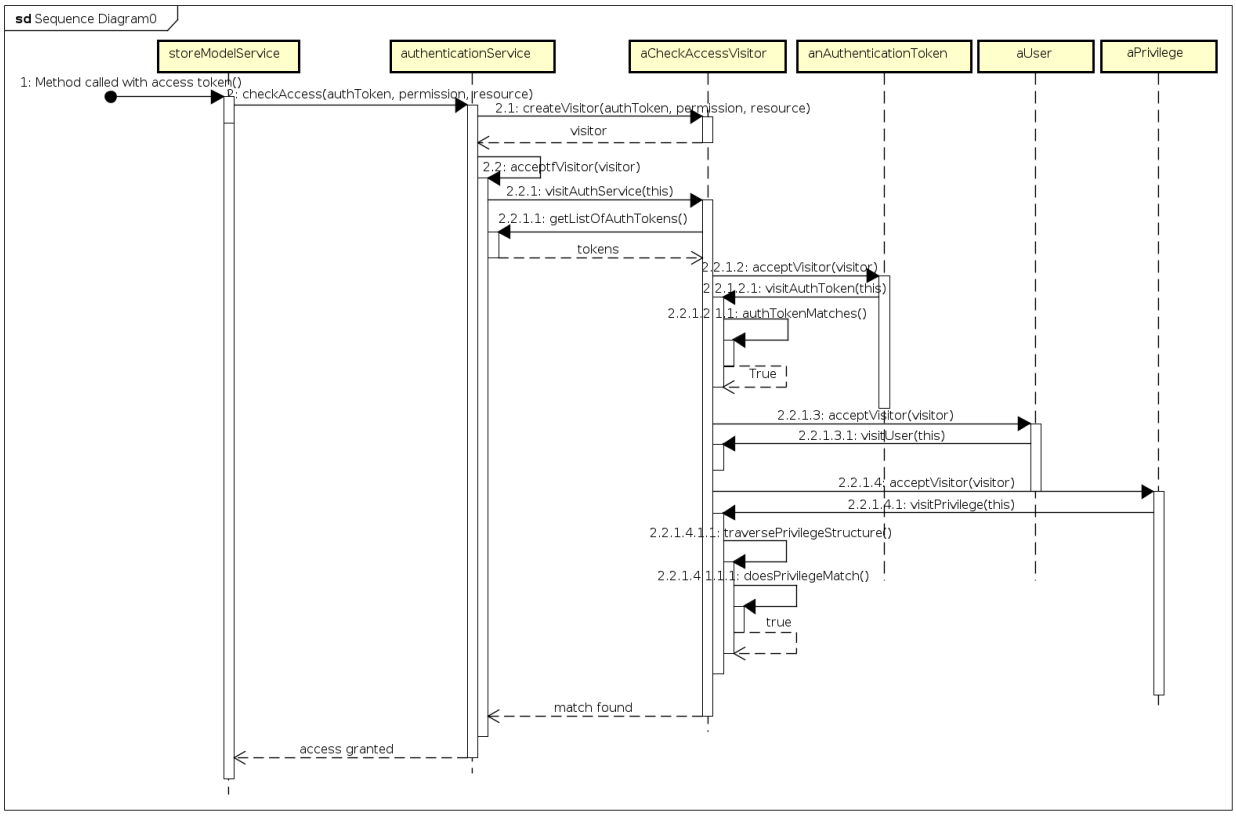
From that point, the Store Controller represents the customer when they make a request by passing the customer's auth token in a command object to the Store Model Service. The Model Service checks that the token is non-empty and non-null, then communicates with the Authentication Service to verify that the token is valid and that the customer has the privilege to access the resource. If the answer is yes, the request is granted. This fulfills the requirements of validating auth tokens, checking user privilege, and checking on the Store Model side that a token is provided.

Similar to the login flow, the Store Controller automates the logout flow through the checkout process. The Controller uses sensors to get face and voice prints from customers as they leave the store and tells the Authentication Service to invalidate the token of the customer with the matching credentials. The Authentication Service achieves this by using the Visitor pattern to find the correct customer. This logout flow thusly meets the requirements of a logout feature that invalidates the user's access token.

The following sequence diagrams illustrate key Authentication Service flows.



Caption: A UML sequence diagram displaying the authentication flow that occurs when a customer issues a voice command to a store appliance. Visitor flow is abstracted out for clarity.



Caption: This sequence diagram details the actions of the CheckAccessVisitor that were left out of the above diagram. The path displayed is the “happy path” in which the user’s token is valid and the user has the correct permissions to use the resource.

Exception Handling

Three exception classes will be implemented to meet the requirements of throwing the InvalidAuthTokenException, AuthenticationException, and AccessDeniedException in response to certain events.

- The InvalidAuthTokenException will be thrown by the AuthenticationService when the token passed from the Store Controller is expired or otherwise invalid

- An AccessDeniedException will be thrown by the AuthenticationService when a user attempts to a resource that they do not have the privileges to use

- An Authentication Exception will be thrown when a login is attempted with credentials that do not match the user credentials in the Authentication Service database

Testing

Testing will be conducted using test scripts and the event simulation functionality of the Store Model Service API. A script will be created that will model a new store instance and all the entities within it (customers, products, appliances, and so on). Customer creation will be accompanied by User object creation by the Authentication Service. An administrative account

will be required to provide login credentials at the start of the script.

The test script will cover User object creation, access token creation, login events, requests for restricted resources, auth token validation, privilege checking, logout events, and exception handling.

Functional testing will be performed by redirecting system printouts to a text file and examining the file for errors, alerts, or unexpected behavior.

Regression testing will be performed by re-running testing scripts that are known to pass.

Risks

The design does not address thread safety, which should be a consideration whenever the Singleton Pattern is being applied.

The state of the program is all being stored in local memory, and in the event of a crash it will all be lost. Databasing is recommended for storing critical information and redundant systems are recommended for ensuring that the Controller Service is able to continue running.