Authentication Service Design Document

Date: 11-16-2020

Author: Samer Akash

Reviewer(s):

# Introduction

This document defines the design of the Smart City Authentication Service.

# Overview

The Authentication Service will be used to control access to the Model Service. The Model

Service will be updated to use the Auth Token passed into each of the service methods and

give this to the Authentication Service to validate that the associated user has the permission

required to invoke the method.

The Authentication Service will assist in controlling access to the Model Service interface. The

Authentication Service provides a central point for managing Users, Resources, Permissions,

Roles, Resource Roles, and Auth Tokens.

The following diagram shows the Smart City components: The Controller Service, Model Service,

Ledger Service, and Authentication Service (highlighted in green) with their interdependencies.

Diagram

Description automatically generated

# Requirements

1. Authentication Service supports creating Resources where Resource represents a physical and logical entity, for example, an IoT Device.
2. A Resource has a unique identifier and a description.
3. Authentication Service supports creating Permissions that represent an authorization required to access a resource or function of the Smart City system.
4. Permissions have a unique id, name, and description.
5. A User may be associated with zero or more permissions.
6. Authentication Service supports creating Roles.
7. Roles are composites of Permissions.
8. Roles provide a way to group Permissions and other Roles.
9. Roles have a unique id, name, and description.
10. Users may be associated with Roles, where the user has all permissions included in the Role or sub Roles. Roles help simplify the administration of Users by providing reusable and logical groupings of Permissions and Roles.
11. Authentication Service supports creating Users
12. Users represent persons of the Smart City system.
13. Users have an id, a name, and a set of Credentials. Credentials may include a username/password, voiceprint, faceprint, and other biometric identities.
14. Hash the credentials to secure their use.
15. Users are associated with 0 or more entitlements (i.e., Roles or Permissions).
16. The Authentication process provides users AuthTokens that can then be used to access restricted Service Methods.
17. If authentication fails, an Authentication Exception should be thrown.
18. If authentication succeeds, an AuthToken is created and returned to the caller.
19. The AuthToken binds the User to a set of permissions that can be used to grant or deny access to restricted methods.
20. AuthTokens can timeout with inactivity.
21. AuthTokens have a unique id, an expiration time, and a state (active or expired).
22. Auth tokens are associated with a User and a set of Permissions.
23. Authentication Service supports user login and user logout.
24. Login accepts a User’s credentials (username, password).
25. Validate that the username exists, and then that the hash of the password matches the known hashed password.
26. Authentication Service supports user Voiceprint authentication through voice recognition
27. The voice print signature is sufficient for identifying and authenticating a user.
28. Authentication Service supports user Faceprint authentication through face recognition
29. Logout marks the given Auth Token as invalid.
30. Subsequent attempts to use the AuthToken should result in an InvalidAuthTokenException.
31. All methods defined within the Model Service should accept an AuthToken
32. Each method should validate that the AuthToken is non-null and non-empty.
33. The method should pass the AuthToken to the Authentication Service with the required permission.
34. The Authentication Service should check to make sure that the AuthToken is active and within the expiration period. Then, check that the user associated with the AuthToken has the permission required by the method.
35. The Authentication Service should throw an AccessDeniedException or InvalidAccessTokenException if any of the checks fail.
36. Exceptions should include useful information to help users understand the nature of the Exception.

# Use Cases

# Use case Diagram:

Diagram

Description automatically generated

# **Actors:**

* **Admin**: Administrator is responsible for managing the Resources, Permissions, Roles, and Users maintained by the Authentication Service. The Administrator also provisions Cities and other entities within the Smart City system and has full access to all Smart City System resources.
* **Registered User**: A city inhabitants that uses credentials to interact with the Smart City System. Inhabitants use voice commands to interact with the Smart City System. The Authentication Service supports identifying and authenticating users using the user’s voice or face print.

# **Use Cases:**

* **Create Resource Access:** Creating A Resource represents a physical and logical entity in the smart city.
* **Create** **Permission**: Creating Permissions represent an authorization required to access a resource or function of the Smart City system.
* **Create Role**: Creating Roles which are composites of Permissions.
* **Create User**: Creating Users that represent persons of the Smart City system.
* **Login**: Login accepts a user credentials to authenticate user in order create access token.
* **Identify and Authenticate User**: Find the user associated with the login credentials in order to create the access token.
* **Create Access Token:** Creation for an access token (AuthToken) that users can use to access the system.
* **Logout:** Logout marks the given Auth Token to a certain user as invalid.
* **Invalidate Access Token**: Changing the status of an AuthToken to invalid.
* **Invoke Restricted Methods:** Registered user invoking a restricted method using their AuthToken and the type of required access.
* **Check Access**: Checking the if the user has the required permissions to invoke restricted methods.

# Implementation

The following component diagram shows the Authentication Service as part of the high-level system components that make up the Smart City System:

![Diagram

Description automatically generated]()

# Class Diagram

here

# Class Dictionary

## Subject Interface

This class is a java interface class that will declare the required public methods that will implement the Observer Design pattern. The implementation class for this interface is called SubjectImpl. The interface class will have all the public methods that will be implemented in the implementation class.

This interface will be extended by the ModelService Interface.

### Associations

|  |  |  |
| --- | --- | --- |
| **Association**  **Name** | **Type** | **Description** |

### Properties

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |

### Methods

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |

# Implementation Details

Sequence Diagram:

Diagram

Description automatically generated

Some sequence description here:

# Exception Handling

## ControllerException

The Controller Exception is returned from the Controller Service API methods in response to an error condition. The Exception captures the action that was attempted and the reason for the failure.

### Properties

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| action | String | Action that was performed  (e.g., “update device”) |
| reason | String | Reason for the exception  (e.g. “Device type is not supported”) |

# Testing

Implement a test driver class called TestDriver that implements a static main() method. The

main() method should accept a single parameter, which is a command file.

The main method will call the CommandProcessor.processCommandFile(file:string) method, passing in the name of the provided command file.

The TestDriver class should be defined within the package “com.cscie97.model.test”. A test command file will be provided for testing.

# Risks

* Risks will be added here