Smart city authentication service design document

Date: 11/10/20

Author: Yuri Machkasov

Reviewer: Jorge Cotillo-Herrera

Contents

Introduction:

The Smart City Authentication Service is responsible for maintaining the security of public interfaces of the Model service. All interactions with the Smart City objects, through their abstractions in these services, require the caller to be properly authenticated; different levels of access correspond to different roles and permissions assigned to actors. The authentication mechanism is managed through creation and exchange of token objects tied to each actor. For every method call a token will be supplied and checked for access to a specified entitlement or resource; if there is insufficient access, an exception will be thrown and the operation will not be completed.

Requirements:

The service will provide the framework for verifying the permissions to access and modify all Smart City resources (that is, cities themselves and all IoT devices) by a variety of actors, falling into 2 categories – administrators and registered users. At the same time the access to the maintenance of the framework (managing the permissions themselves and their assignment to users) will also be subject to verification, accessible only to the administrators. To ensure that only a single instance of the service is available, the design will use the Singleton design pattern.

The entitlements will be able to be organized in entitlement trees through the use of the Composite design pattern; for example, a role (that is provided through Model Service API) will be capable of holding a list of permissions as well as other roles. Another variety of permission will be a resource role, providing access to a resource (either a collection of devices or entire cities).

To verify identity, two mechanisms will be provided: authentication through "something you are" (biometric characteristics: voiceprints or faceprints) or through "something you know" (a password).

The administrator account, upon authenticating, will be able to:

- Provision resources
- Provision and modify users
- Create and modify entitlements and assign them to users

All users will be able to:

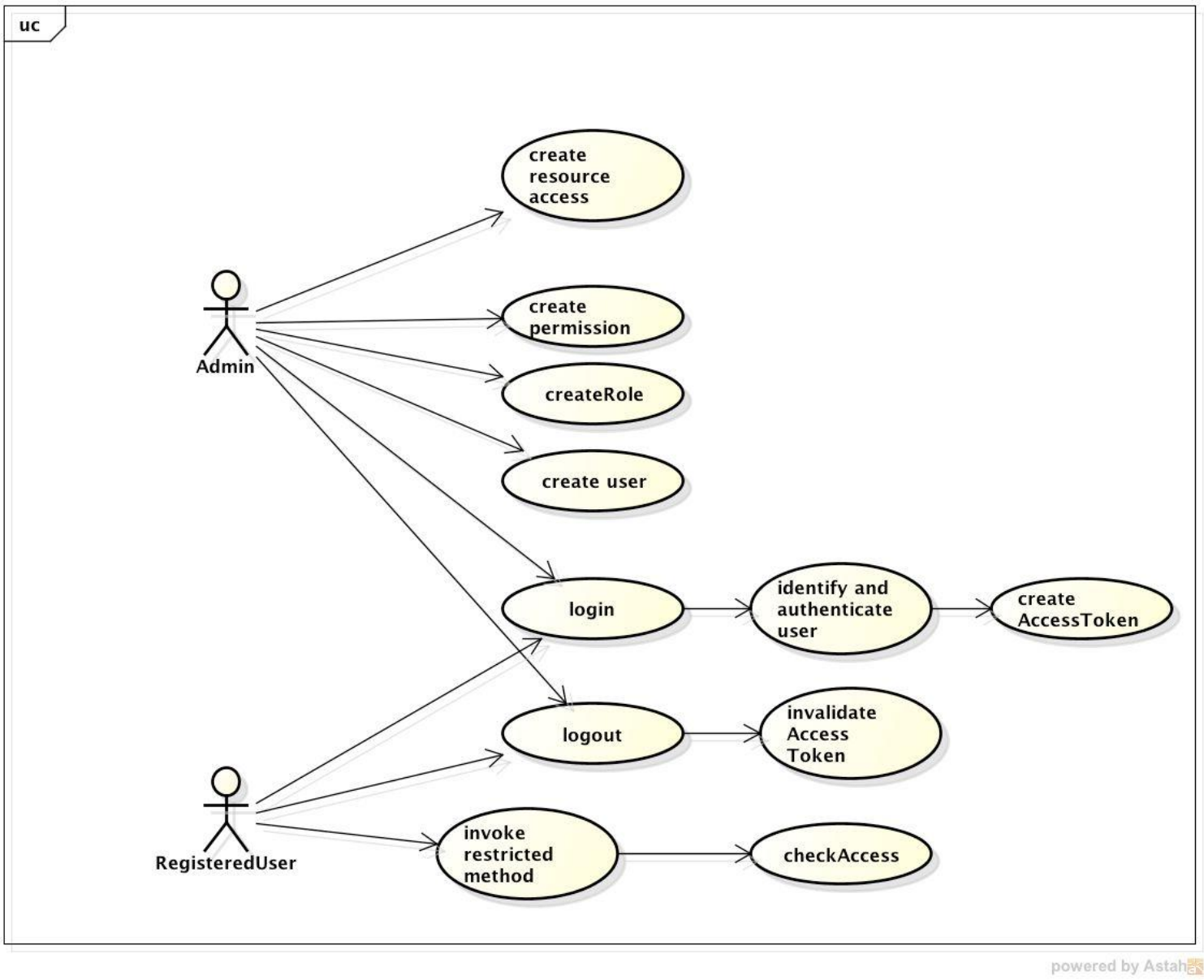
- Log into the system to obtain authentication tokens representing the level of access granted to them (through permissions or roles)
- Use the token to obtain access to those public interfaces of the Model Service that are allowed to be used by a user with these roles and permissions
- Log out of the system (and have the token received upon login invalidated)

Further details are available in the Service Requirements document.

The service will utilize the Visitor design pattern to traverse the tree of permissions for a user to determine the level of access to a certain resource or operation. Also, by utilizing the same pattern, the service will provide a way to inventory all hierarchies of its subordinate objects.

Use cases:

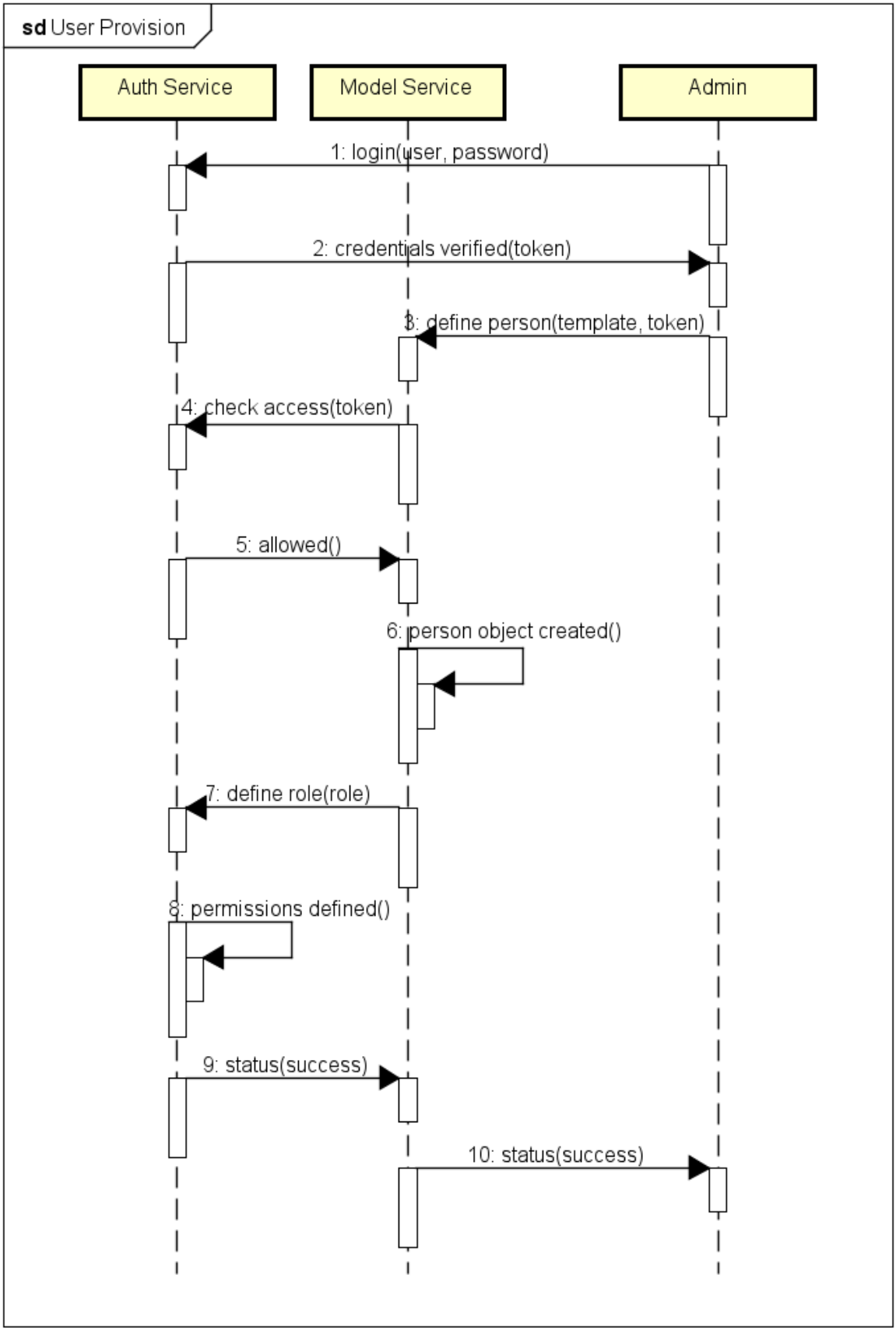
The use case diagram is reproduced here from the requirements document. The requirements that are represented by those use cases are outlined in the previous section.



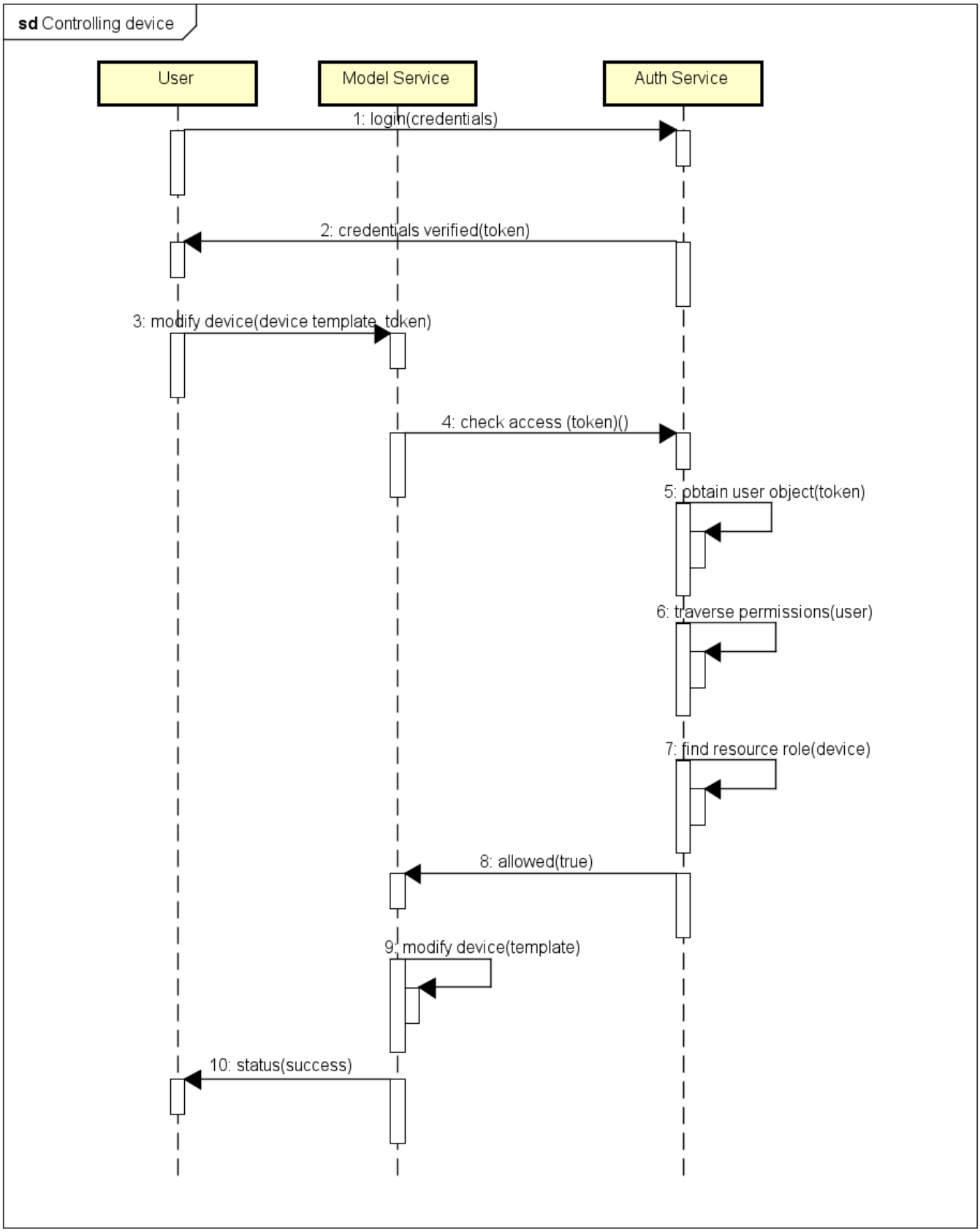
Sequence diagram:

To illustrate the control flow, consider the following actions.

1. The administrator is provisioning a person object in the model service



A user (a resident) is requesting a modification of the status of a device (driving a car)



Both diagrams illustrate the successful authentication of the caller and subsequent execution of the requested API method; the first has an additional step of defining the user object in the authentication service, while the second, in the assumption that the user does not have an explicit permission (through a role), includes the entitlement tree traversal.

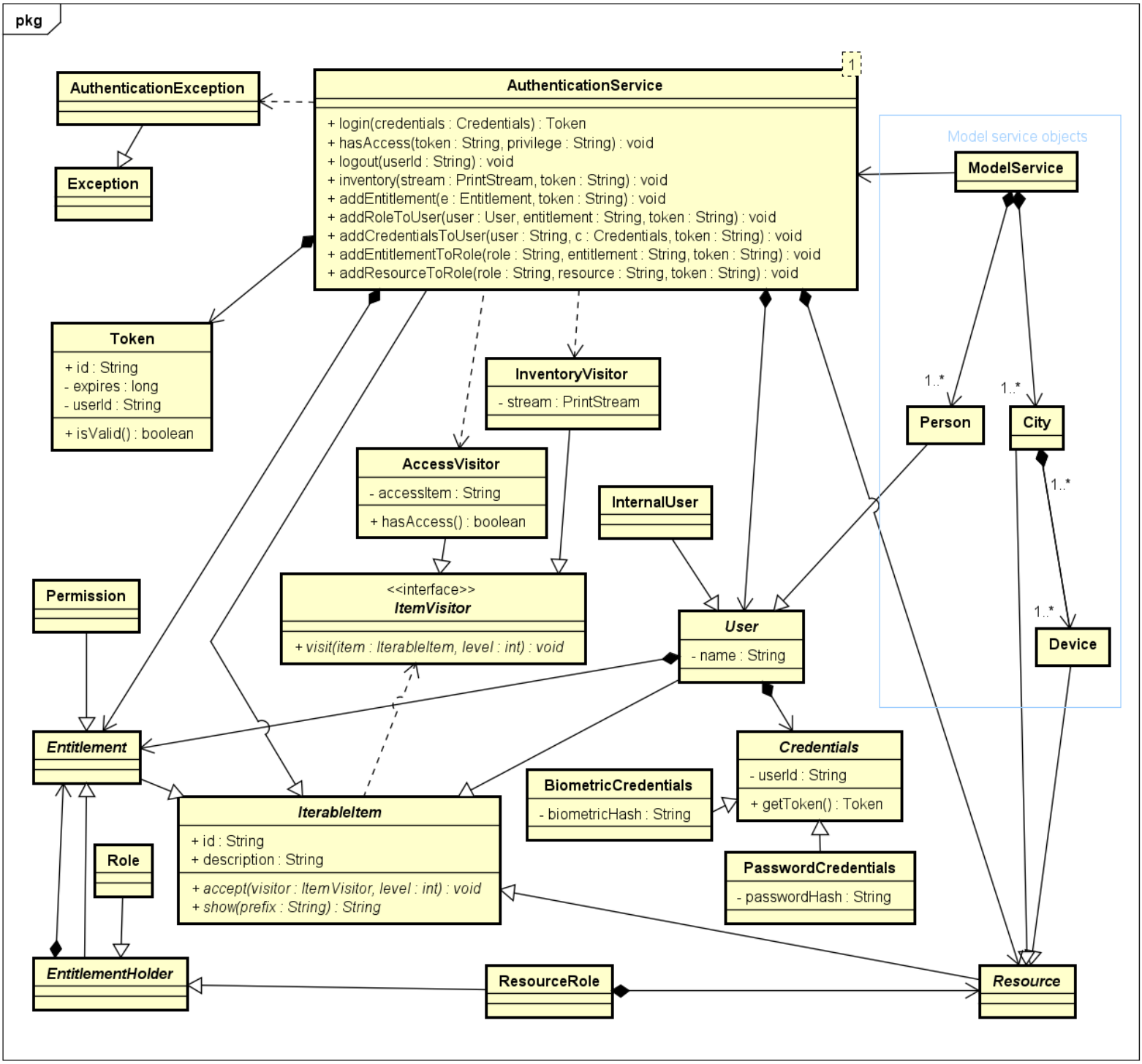
Implementation

The impact of the requirements on the implementation is twofold: they define the operations that need to be supported on the authentication objects, and also dictate the interaction between the model service and its objects and the authentication service. The requirements list the creation and modification of the Authentication users as a necessary capability of this service; however, if a design decision is made to derive the Person object from the Authentication service user, and not simply to include the latter in the former, this requirement is redundant – it will be automatically fulfilled by using the Define Person interface of the Model Service (subject to access restrictions). The same holds true concerning authentication service Resources – their provisioning will be included in the definition of the derived City and Device objects by the Model service.

Building the class hierarchies in this way that also captures the design intent by assigning the resource nature to model service devices and cities and user nature to model service persons. At the same time it leverages the existing functionality by making the provisioning methods of the model service also handle the manipulations of the corresponding objects in the authentication service.

Class diagram

This diagram represents the structure of the Authentication Service and the relations of its objects to those of the Model Service:



Class description

The design of this service utilizes certain classes defined and implemented as part of the Model Service. These are the Model Service itself, Person, City and Device. The internal structure and functionality of them is not relevant to this component, and they are included in the diagram as placeholders, except they are now reflecting their usage of (for the Model Service) or inheritance (for the other three) from the Authentication service objects.

AuthenticationService

This is the main class, providing the interface for authenticating users and providing the facility to verify access and perform inventory

Attributes

Attribute name	Attribute Type	Description
tokenStore	Collection<Token>	a collection of all active tokens
userStore	Collection<User>	a collection of all known service users; is managed by the model service person management interface
entitlementStore	Collection<Entitlement>	a collection of all defined entitlements
resourceStore	Collection<Resource>	a collection of all known resources; is managed by the model service city and device management interface

Methods

Method name	Signature	Description
login	credentials: Credentials	Is invoked before the user can access the public methods of the Model Service (or of the Auth service itself). If the credentials are valid, returns an access token object which is also added to the store; otherwise, an instance of AuthenticationException is thrown
logout	token: String	Searches the token store for the supplied token, terminates its validity and removes from the store.
hasAccess	token: String, entitlement: String	The main method that determines if the supplied token allows the user to have access to the specified privilege, which can be either an entitlement or a resource. Does nothing if access is confirmed; otherwise throws an instance of AuthenticationException
inventory	stream: PrintStream, token: String	Verifies that the supplied token corresponds to the admin access level. If so, outputs a formatted text representing the hierarchical structure of all objects stored by the service (Users, Resources and Entitlements); otherwise throws an instance of AuthenticationException
addEntitlement	e: Entitlement, token: String	Verifies that the supplied token allows access to entitlement store. If so, adds the specified

		entitlement to the store; otherwise throws an instance of AuthenticationException
addRoleToUser	user: User, entitlement: String, token: String	Verifies that the supplied token allows access to user store. If so, adds the specified entitlement to the user; otherwise (or if no such entitlement has been defined) throws an instance of AuthenticationException
addCredentialsToUser	user: User, c: Credentials, token: String	Verifies that the supplied token allows access to user store. If so, adds the specified credentials to the user; otherwise throws an instance of AuthenticationException
addEntitlementToRole	role: String, entitlement: String, token: String	Verifies that the supplied token allows access to entitlement store. If so, adds the specified entitlement to the role; otherwise (or if no such entitlement or role has been defined) throws an instance of AuthenticationException
addResourceToRole	role: String, resource: String, token: String	Verifies that the supplied token allows access to entitlement store. If so, adds the specified resource to the resource role; otherwise (or if no such resource or role has been defined) throws an instance of AuthenticationException

IterableItem

An abstract class that is capable of accepting an instance of a visitor; each implementing class will follow its internal structure and invoke the visitor on the subordinate objects

Attributes

Attribute name	Attribute Type	Description
id	String	a unique identifier
description	String	optional description string

Methods

Method name	Signature	Description
accept	visitor: ItemVisitor, level: int	An abstract method; implementers should apply the visitor to the object itself and all subordinate objects. The level argument represents the depth of the call within the hierarchy; implementers should take care to increment it when descending hierarchies
show	prefix: String	returns a multiline text representation of the object, with each line prepended by the supplied prefix

User

An abstract class that represents an actor of the authentication service. Inherits from the IterableItem; is inherited by the Model Service Person.

Attributes

Attribute name	Attribute Type	Description
name	String	an optional name string
credentials	Collection<Credentials>	a collection of all credentials identifying this user
entitlements	Collection<Entitlement>	a collection of all roles and permissions for this user

InternalUser

An instance of the service user that does not correspond to a physical person. An administrator of the service may be an instance of this class.

Credentials

An abstract class that is the base of all implementations of user credentials.

Attributes

Attribute name	Attribute Type	Description
userId	String	the user to which these credentials belong

Methods

Method name	Signature	Description
getToken	AuthenticationService	If the underlying credentials matches at least one of the known credentials for this user, generates an access token; otherwise, throws an instance of AuthenticationException

PasswordCredentials

The credentials implementation that represents a password-based authentication. The password is MD5-hashed before being stored

Attributes

Attribute name	Attribute Type	Description
passwordHash	String	the hash of the supplied password

BiometricCredentials

The credentials implementation that represents a biometric-based authentication. The biometric information (in text form) is MD5-hashed before being stored

Attributes

Attribute name	Attribute Type	Description
biometricHash	String	the hash of the supplied password

Entitlement

The base abstract class for all forms of user entitlements. Inherits from IterableItem.

Permission

The simple Entitlement implementation. Does not have any internal structure.

EntitlementHolder

The abstract class representing an entitlement that, in addition to its own identifier, can contain other entitlement instances.

Attributes:

entitlements	Collection<Entitlement>	a collection of all subordinate entitlements

Role

The simple EntitlementHolder implementation. Does not have any additional functionality.

ResourceRole

The EntitlementHolder implementation which has a reference to one or more resources. The access to this entitlement is determined through either matching the requirement with its own permission, or with the name of the resource (if the resource represents a city, then the permission is extended to all devices of this city)

Attributes:

resources	Collection<String>	identifiers (in a combined <city>:<device> form) of all resources this role allows access to

Token

Represents an access token tied to a specific user. Contains information about its expiration.

Attributes

Attribute name	Attribute Type	Description
userId	String	the user to which this token has been issued
id	String	the identifier of the token itself
expires	long	the epoch expiration time

Methods

Method name	Signature	Description
isValid	void	inquires of the current validity of the token

ItemVisitor

An interface that follows the Visitor pattern; delegates the traversal of IterableItem hierarchies to the items themselves.

Methods

Method name	Signature	Description
visit	item: IterableItem, level: int	performs the visiting action of the current item in the traversal sequence; the level argument is an indication of the depth of the item tree

InventoryVisitor

Implementation of the visitor interface which collects text-based representation of a hierarchy of iterable items into the supplied PrintStream, formatting it with indentation according to the level of the hierarchy. If called on the service object itself, will inventory the user, resource and entitlement stores.

Attributes

Attribute name	Attribute Type	Description
stream	PrintStream	the stream to which the information is appended

AccessVisitor

Implementation of the visitor interface which provides the main functionality of the service: the answer to the question whether a user is authorized to access an entitlement or resource.

Attributes

Attribute name	Attribute Type	Description
accessItem	String	the identifier of an item (entitlement or resource) to check access to

Method name	Signature	Description
hasAccess	void	after the traversal is complete, contains a boolean flag indicating if access has been granted

Implementation details

As already mentioned, the design makes some of the objects of the Model Service inherit from the Authentication Service classes. This has been done for reasons both philosophical (because, for example, the Person object has the nature of a User; if we were to try to imagine a Person that is not a User, such Person would have no way of interacting with the city and its resources) and practical (to eliminate the bureaucracy involved in separate provisioning of Model and Authentication Service objects that would be related and the maintenance of their relationships). The downside of this decision may be that the identifiers of Authentication Service objects are not independent of their counterparts; the security impact of this can be mitigated by hashing them in the Auth Service implementation (not done within the scope of this project).

The AuthenticationService itself is implemented as a singleton, to ensure that only one instance of it is ever acted upon by the clients. Also, the service object inherits from the IterableItem, to enable the instances of the ItemVisitor to operate on it directly, further encapsulating its internal stores.

Each Permission is also a singleton; the goal is to prevent redefining permissions with the same identifier.

The entire functionality of Role has been lifted up to a common parent of it and the ResourceRole, to prevent an instantiatable class inheriting from another.

Exception handling

The authentication service generates instances of AuthenticationException; it is used to communicate the failure of any of the public methods of the service, since none of them (with the exception of login) returns a value (and even login can throw an exception if the supplied credentials are invalid. The exception contains a string representation of the error. When the authentication service methods are called by the Model Service or the Controller, this exception is caught and its message is wrapped in the exception appropriate to either client

Testing

The testing builds on the CLI client developed for the model service. All provisioning commands (cities, devices and persons) of the model service testing are being reused to now provision the corresponding authentication service objects (resources and users). In addition, the placeholders for user roles which have been implemented in the model service are now meaningful and trigger changes in the service state.

Several new commands have been added to the command processor, corresponding to the requirements and use cases outlined in the requirements document. The parser framework remains in place, and the same TestDriver class is reused.

A test-specific command which pauses the execution of the suite has been added to validate the correct handling of token expiration.

The “A4-scripts.txt” file contains the functional test cases. It creates and modifies several objects from the model service domain, performing integration and regression testing, and then proceeds to exercise the functionality that is specific to the authentication service. Each command is preceded by a comment that explains the scenario being tested and the expected result. Validation is manual, by reading the resulting output stream and verifying the command outcomes.

According to the testing-first design principle, the same test script was being used throughout the development process, with regression testing conducted after all significant changes to the design and implementation.

The results of running the suite are included in the file “A4-out.txt”.

Risks

The concerns outlined for the testing previous modules carry over – namely the testing harness deficiencies in that it requires human interaction to verify the results.

Similarly, the implementation still relies on holding the application state in memory, which means that the crash of the application server would mean unrecoverable data loss.

The hashing of the credentials utilizes MD5, which is known to be easily compromised; the productized version should be switched to a more secure algorithm.