Authentication Service Design Document

Date: 10/26/2020
Author: Christopher Jones
Reviewer(s):

- Allyson Bieryla
- Jaya Johnson

Design Document

The design document (in PDF format) has a clear overview of the problem and presents a short description of the organization of the rest of the document. The requirements section provides a brief and accurate overview of the Authentication Service requirements.

Includes a **use case diagram**, with notes describing each actor and use case.

- This can be found on Page 3 of the design document.

The **class diagram** provides a clear model of the Authentication Service classes. The diagram should show all dependencies and associations as well as inheritance relationships. Define the features and methods of each class.

- This can be found on Page 4 of the design document.

The **class dictionary** provides details on classes, properties, associations, and methods, including type information and descriptions. It provides sufficient detail about how to use public methods.

- This can be found on Pages 5-16 of the design document.

The **sequence diagram** provides a clear description of the interaction between the Authentication Service, Model Service, and Controller Service.

- This can be found on Page 18 of the design document.

The use of the following patterns: **Visitor, Singleton, and Composite** are included and highlighted in the design.

- The **Visitor Pattern** was applied to the CheckAccess class and the CheckInventory class that utilize the Visitor interface to allow the Authentication service to access its methods to assure, both, that the objects it creates exist, and that they have access to invoke other methods provided that they were actually created by the Authentication service.
- The **Singleton** Pattern was applied to the Authentication class such that all classes and packages will reference a singular instance of it, such that multiple Cities within the Model service and Controller city are all able get the exact same information.
- The **Composite** pattern was applied to the Role and Resource classes, such that they can embody other Roles, Resource Roles, and Permissions.

The design document makes it clear how the design meets the requirements. (**requirements should be referenced from the implementation section to help clarify how the design addresses requirements**). The general quality of the document (clarity of design, completeness, readability).

Includes sections on Testing and Risk.

- This can be found on Page 19 of the design document.

Implementation

Did the application of the design patterns help or hinder your design and implementation? Please explain how.

- While initially some of the patterns were difficult to implement (namely the Visitor pattern), once done, the value of it become more visible with the less number of edits I had to make to particular classes. The Singleton pattern and the Composite pattern, however, seemed to naturally fit into this project, and I didn't run into any issues implementing them.

How could the design have been better, clearer, or made the implementation easier?

- There were some classes that ultimately weren't used very often with the exception of providing state or being utilized as a reference point, namely the Resource class. I was hoping we'd do more with these. I also think the design could have touched more on the changes needed in the other packages as well. Upon implementation, I had to make several updates to my Model and Controller class.

Any implementation changes that you made to your design and how they continue to support the requirements?

- I ultimately used my Login method to create and establish connections to an AuthToken, as opposed to creating an AuthToken to be passed that may or may not fail upon trying to sign into it. I don't think this change alters the requirements, but it was easier to manage.

Is the design process getting easier?

- As we discover new patterns, it is getting harder, but as we get more adjusted to them, it gets easier. Otherwise, the overall process is getting easier over time.

Did the design review help improve your design?

- Yes, I reworked my design quite a few times as I was coming to understand the Visitor patterned more and more. My teammates helped direct me to a variety of example implementations that I was able to use to gain a better understanding of the process so that I could better establish my own solution.

Your comments for your review partners?

- I don't think the Resource class should be represented as a Singleton class if your Model service supports multiple cities. Each city shouldn't need to keep track of what is going on in a neighboring city.
- Not every class needs to be implemented with the Singleton pattern.
- It may not be necessary to create a class for every type of Resource and Resource Role. You could create a generic one with details describing it within, and just provide roles and permissions to that particular instance.

Comments from peer design review and optionally the functional review?

- Revisit how I implement the Visitor and Composite patterns. There shouldn't be any need to connect the Resources, Roles, etc. directly to the Authentication class.
- Consider not allowing the Permissions to be directly exposed in the Authentication service.
- Don't forget to hash the values stored within the Authentication service. Also, does the value of hash return an int?
- The checkAccess method should be accessible from the Authentication service and potentially the AuthToken.

Code compiles with the command

-
  ```
  javac cscie97/smartcity/controller/*.java
  cscie97/smartcity/model/*.java
  ```

```
cscie97/smartcity/authentication/*.java com/cscie97/ledger/*.java
cscie97/smartcity/test/*.java
```

The program should run with the command

- `java -cp . cscie97.smartcity.test.TestDriver smart_city.script`
  Test Driver processes the input file and generates the expected results.

Additional test input files and exception handling.

- This additional file is called **smart_city.script** now.