CS628:

OBJECT-ORIENTED SOFTWARE DEVELOPMEMT LAB

TEAM MEMBERS

Sai Manoj Cheruvu

Shruti Varade

Domakonda Sesank

TOPIC: JAVA INTEROPERABILITY UNIT TESTING

TABLE OF CONTENTS

PROJECT DESCRIPTION

The Java SDK interoperability project is a comprehensive initiative focused on improving the compatibility and communication capabilities between Java and other SDKs, with Dart as an initial example. The project aims to enable seamless data sharing and collaboration between Java and Dart clients by establishing encrypted channels and facilitating secure transmission of AES keys and data.

By leveraging the interoperability features provided by this project, developers can easily integrate Java and Dart applications, enabling them to exchange encrypted AES keys and share data seamlessly. The project includes a set of well-defined test cases that cover key scenarios such as key creation, modification, and deletion. These test cases ensure the proper functioning and accuracy of data transmission, as well as the verification of received data.

With the Java SDK interoperability project, developers can harness the strengths of both Java and Dart to build robust and flexible applications. The project's focus on encryption and secure data sharing enhances the reliability and privacy of communication between Java and Dart clients. By streamlining the interoperability process and providing comprehensive test coverage, the project empowers developers to create innovative solutions that leverage the capabilities of both Java and Dart SDKs, opening up new possibilities for cross-platform development.

TASKS PERFORMED

1) **Interoperability tests:**

- getFromDart:

Defined DART and JAVA Clients. Established Connection with the server. Defined Shared key and got the corresponding value of the key. Validate if the value that is being put in the dart client is same as that of this value.

String value=atClient.get(sharedKey).get()

assertEquals(“hello”,value);

- getInJava:

Defined values for key, namespace. Call Validate method to check if Sharedwith value is null or empty. Checked if the name of atkey is empty using assertFalse. Checked if sharedwith value is null using assertNotNull. Checked if sharedwith value is empty using assertFalse. Established Connection with the server. Get value of corresponding key from DART using atClient.get(). Handled Interrupted Exception and execution exception if the call to get key fails.

- sendToDart:

Defined DART and JAVA Clients. Established Connection with the server. Defined Shared key and got the corresponding value of the key. Defined namespace for the shared key. Send the key-value pair to DART.

- sendToJava:

Defined values for key, namespace. Call Validate method to check if Sharedwith value is null or empty. Checked if the name of atkey is empty using assertFalse. Checked if sharedwith value is null using assertNotNull. Checked if sharedwith value is empty using assertFalse. Established Connection with the server. Define shared key and set the value to key. Handled Interrupted Exception and execution exception if the call to put key fails.

2) **Key Deletion:**

- PublickeyPutTest:

Established Constants, ROOT_URL and created AtSign Instances. Established Connection with the server. Created public key, assigned a value to the key and send it to server. Verified if the key exists by using atClient.get to fetch the value of the key. Compared the value fetched with the original value using assertEquals.

- PublickeyDeleteTest:

Created public key. Deleted the key on server using atClient.delete, which returns a response starting with 'data:'. Checked if the response returned starts with 'data:' using assertTrue. Implemented Exception handling to handle deletion failure. To check if the key has been deleted, called atClient.get(), which throws ATKeyNotFound Exception if the key is not found.

- PublicKeyPutTest:

Established connection to server. Created public key, assigned a value to the key and send it to server. Verified if the key exists by using atClient.get to fetch the value of the key. Compared the value fetched with the original value using assertEquals.

- TestCreateAndCheckSelfKey:

Defined Constants and established connection to server. Created a self-key, defined a value to be assigned to the key and placed it on server. Verified if the self key exists on server using atClient.get, which fetches the value corresponding to the key. Checked if the value defined is equal to the value fetched using assertEquals.

- DeleteSelfKeyTest:

Created self-key. Deleted the key on server using atClient.delete, which returns a response starting with 'data:'. Checked if the response returned starts with 'data:' using assertTrue. Implemented Exception handling to handle deletion failure. To check if the key has been deleted, called atClient.get(), which throws ATKeyNotFound Exception if the key is not found.

- TestCreateAndCheckSelfKey:

Defined Constants and established connection to server. Created a self-key, defined a value to be assigned to the key and placed it on server. Verified if the self key exists on server using atClient.get, which fetches the value corresponding to the key. Checked if the value defined is equal to the value fetched using assertEquals.

- SharedKeyDeleteTest:

Created Shared key. Deleted the key on server using atClient.delete, which returns a response starting with 'data:'. Checked if the response returned starts with 'data:' using assertTrue. Implemented Exception handling to handle deletion failure. To check if the key has been deleted, called atClient.get(), which throws ATKeyNotFound Exception if the key is not found.

- SharedPut:

Created a shared key and defined names for key, namespace. Call Validate method to check if Sharedwith value is null or empty. Checked if the name of atkey is empty using assertFalse. Checked if sharedwith value is null using assertNotNull. Checked if sharedwith value is empty using assertFalse. Established Connection with the server. defined a value to be assigned to the key and placed it on server. Verified if the self key exists on server using atClient.get, which fetches the value corresponding to the key. Checked if the value defined is equal to the value fetched using assertEquals.

3) Key Modification:

- PublicKeyPutTest:

Established Constants, ROOT_URL and created AtSign Instances. Established Connection with the server. Created public key, assigned a value to the key and send it to server. Verified if the key exists by using atClient.get to fetch the value of the key. Compared the value fetched with the original value using assertEquals.

- createandModifyTest:

Established Connection with the server. Created public key, assigned a value to the key and send it to server. Verified if the key exists by using atClient.get to fetch the value of the key, which returns a response starting with 'data:'. Checked if the response returned starts with 'data:' using assertTrue. Defined a new value and assigned it to the same key. Placed the

new key-value on server and checked if atclient.get() fetches the same value that is defined using assertEquals.

- selfKeyPutTest:

Established Constants, ROOT_URL and created atSign Instances. Established Connection with the server. Created self-key, assigned a value to the key and send it to server. Verified if the key exists by using atClient.get to fetch the value of the key. Compared the value fetched with the original value using assertEquals.

- createandModifyTest:

Established Connection with the server. Created self-key, assigned a value to the key and send it to server. Verified if the key exists by using atClient.get to fetch the value of the key, which returns a response starting with 'data:'. Checked if the response returned starts with 'data:' using assertTrue. Defined a new value and assigned it to the same key. Placed the new key-value on server and checked if atclient.get() fetches the same value that is defined using assertEquals.

- sharedKeyPutTest:

Established Constants, ROOT_URL and created atSign Instances. Established Connection with the server. Created shared key, assigned a value to the key and send it to server. Verified if the key exists by using atClient.get() to fetch the value of the key. Compared the value fetched with the original value using assertEquals.

- modifySharedKeyTest:

Established Connection with the server. Created shared key, assigned a value to the key and send it to server. Verified if the key exists by using atClient.get to fetch the value of the key, which returns a response starting with 'data:'. Checked if the response returned starts with 'data:' using assertTrue. Defined a new value and assigned it to the same key. Placed the new key-value on server and checked if atclient.get() fetches the same value that is defined using assertEquals.

4) **Key Creation:**

- privateKeyBuilderTest:

Verified whether the generated key is set private. PrivateHiddenkeybuilder._atKey.metadata.isPublic returns true if the generated key is set private. Else It returns false. Checked if the value being returned is false using assertEquals.

Verified whether the generated key is hidden. PrivateHiddenkeybuilder._atKey.metadata.isHidden returns true if the generated key is hidden. Else It returns false. Checked if the value being returned is true using assertEquals.

- privateKeyMethod:

Verified the name of the key. Created a key and assigned the name ‘privatekey’ to the key. Retrieved the key name using privateHiddenKeyBuilder._atKey.name and checked equality of both the values using assertEquals.

- privateNamespace:

Verified the namespace of the key. Created a key and assigned a namespace name ‘privateNameSpace’ to the key. Retrieved the namespace name using privateHiddenKeyBuilder._atKey.getNamespace and checked equality of both the values using assertEquals.

- privateTimeToBirth:

Verified time to birth of the key. Created a key and assigned timeToBirth value 11 to the key. Retrieved the value using privateHiddenKeyBuilder._atKey.metadata.ttb and checked equality of both the values using assertEquals.

- privateTimeToLive:

Verified time to live of the key. Created a key and assigned timeToLive value 10 to the key. Retrieved the value using privateHiddenKeyBuilder._atKey.metadata.ttl and checked equality of both the values using assertEquals.

CONCLUSION:

Overall, the project was successful in achieving compatibility between the Java and Dart SDKs, allowing for smooth communication and data sharing. Functionalities for managing keys, such as key generation, change, and deletion, have been implemented. The project's accomplishments improve the two SDKs' compatibility and interoperability, enabling developers to safely share data. The project's accomplishment of its goals demonstrates how well it works to enhance key management and interoperability between Java and Dart SDKs.