Coronet assignment by Pavel Karas

Documentation

Requirements:

The assignment combines server side (Java) and client side (Android)

- The keys are from type string.
- The values are a list of strings.
- The server should accept connections via a custom (non http) protocol over TCP.
- The server should persist the data into the filesystem.
- The server should be able to cache the key-value data in memory, so not every request will go all the way into the disk (for performance reasons).

The API includes:

- getAllKeys(String pattern) – Returns all keys matching pattern.
- rightAdd(String K, String V) – adds a value to key K, from the right.
- leftAdd(String K, String V) – adds a value to key K, from the left.
- set(String K, List<String> V) - set list of values V, to key K.
- get(String K) – gets a list by its key.

Overall

Project implemented on:

Platform:	IntelliJ IDEA Community Edition (15.0.3)
Server side:	Java, jdk 1.7u_40
Client side:	Android 5.1.1, minSdkVersion 19
Persist Data:	disk files storage

Libraries required:

Android-support-v4.jar :	Android library for implementation PagerView
Gson-2.2.4.jar:	JSON serialization and desialization
Commons-collections4-4.1:	LRUMap collection for cache mechanism.

Server side

Data Storage:

StorageHandler.java:	Storage Manager that save for each key it's own file, filename is a key and each value is stored in file as new line.
MemoryCache.java:	Cache Manager that use LRUMap to remove less used value list. Have inner Thread for cleaning work for less used values.

Handlers:

HandlerDispatcher.java:	Include method forward logic for MethodHandler.java implementations.
MethodHandler.java:	Interface that new handler must to implement to be able to handle requests.
SetMethodHandler.java:	Set new value list to given key.
RightAddMethodHandler.java:	Adds a value to key to the right.
LeftAddMethodHandler.java:	Adds a value to key to the left.
GetMethodHandler.java:	Gets a values list by its key.
GetAllKeysMethodHandler.java:	Gets a keys list by its pattern.

Web Service:

RequestHandler.java:	Worker Thread that serve client request.
WebServer.java:	Server service that create worker thread to serve client.

Protocol Model:

Request.java:	Request object from client.
Response.java:	Response sends to client.
ResponseCode.java:	Available response codes (0 – UNDEF,1 – OK, 2 – ERROR, 3 -UNSUPPORTED_METHOD).

Client side

Views:

AbstractFragment.java:	Add ProgressBar and AlertDialog usage in each fragment page that inherit.
Add_Fragment.java:	View for add right or left
Get_Fragment.java:	View for get values by key.
GetAllKeys_Fragment.java:	View for get all keys by Pattern.
Set_Fragment.java:	View for Set new values to key list.

Network:

AsyncMethodTransaction.java:	Implementation of asynchronous requests send to server with callbacks(TransactionListener).

Global:

Globals.java:	Global object for application configuration.
NetworkSettings.java:	Network Settings object from (PROJECT_DIR/assets/NetworkSettings.json).

Protocol Model:

Request.java:	Request sends to server.
Response.java:	Response Object from server.
ResponseCode.java:	Available response codes.

TCP JSON based protocol

Request Json example:

```
      {
            "mMethod" = 'get', /* "get", "set", "getAllKeys",
                                    "addLeft", "addRight" */
            "mParams" = {
                        "key" = "Fruits",
                        "value" = ["apple", "orange"]
            }
      }
```

mMethod: Method that match to implemented MethodHandler on server.

mParams: Can be value or list of values.

Response Json example:

```
      {
            "mStatus" = ResponseCode {/*0 – UNDEF
                                        1 – OK,
                  "Value"=1             2 – ERROR,
                                        3 -UNSUPPORTED_METHOD */
                  },

            "mErrorMsg" = '',

            "result" = [ "apple", "orange"]
      }
```

mStatus: Status code from server.

mErrorMsg: Error massage from server.

result: Result of MethodHandler from server.

Configuration Guide

Client:

- In KeyStorageClient directory -> Assets directory -> need to edit **NetworkSettings.json** file and sets there host and port of server.

Server:

- In KeyStorageServer directory -> need to edit **ServerConfig.json** file and sets server port.
- In KeyStorageServer project **KeyStorage** directory is used as Persist Data Storage.