Concordia University

Engineering and Computer Science

COMP 6231

Distributed Systems Design

Project Report on

Software Failure Tolerant/Highly Available Distributed Player Status System (DPSS)

Submitted To:

Prof. Mohamed. Taleb

Yash Jayeshkumar Pandya (40119272)

Description:

Distributed player status system (DPSS) is implemented as a distributed system to register players across multiple servers and signin and signout from those servers by the players. The transfer player account feature from the already registered server to another server is added for the user type player. Additionally, each server has an admin user that can check how many players are online and offline in all the servers. Suspend player account from the server feature is added for the user type admin. The system is built using Java CORBA and the players/admin can see a single system handling requests providing location, access transparency, Fault tolerant and Highly available Distributed system. It also manages simultaneous requests with adequate synchronization with multiple threading architecture approaches.

Design Architecture:

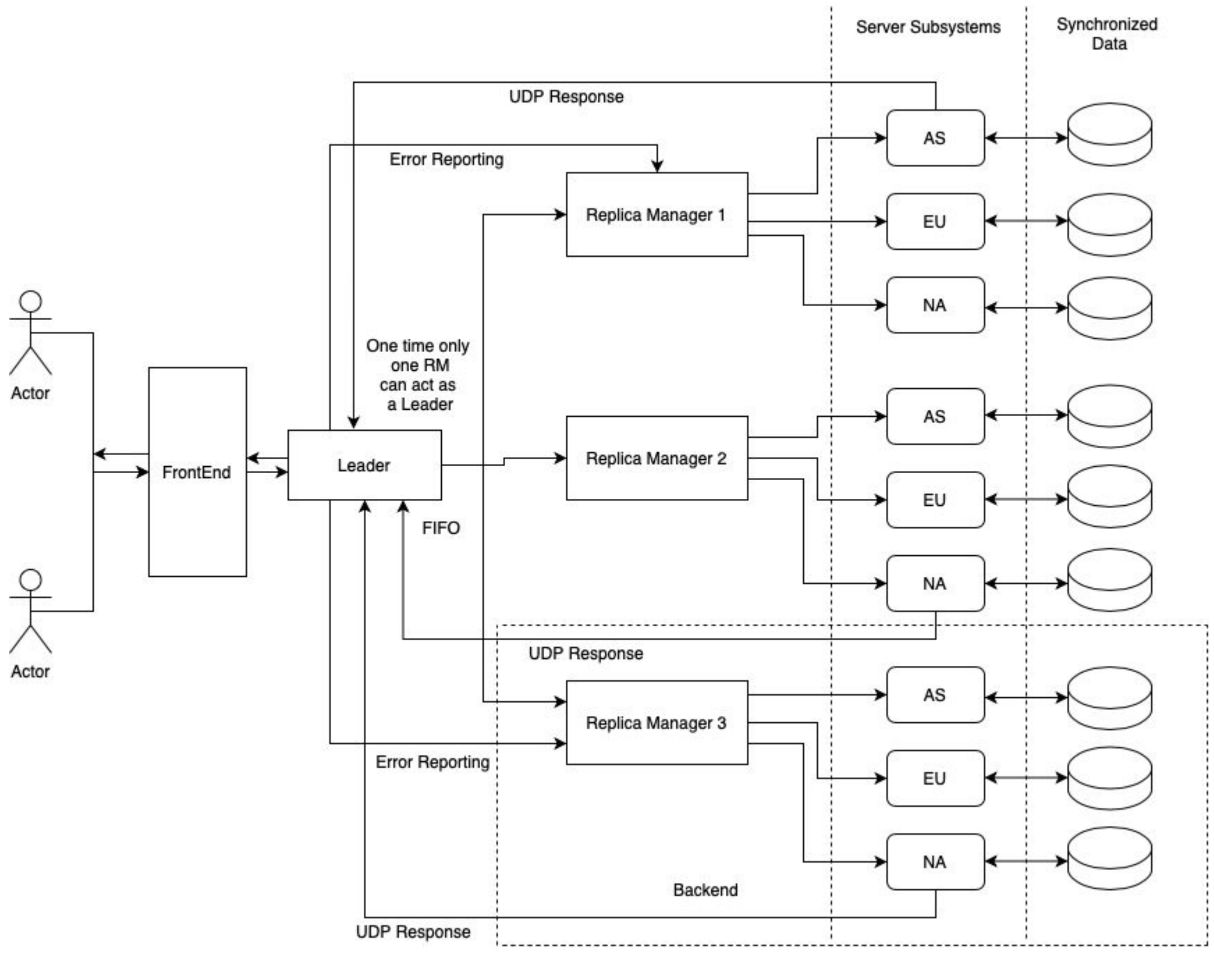


Figure 1: Design Architecture

The above diagram shows the design architecture of the distributed player status system (DPSS). There are mainly 3 Replica(s). Each having different 3 servers for each ip named as Europe (EU), Asian (AS) and North-America (NA). Each server of Replica-1 is interconnected with each other using a UDP connection on ports 8880, 8881 and 8882. Servers of Replica-2 are interconnected with each other using a UDP connection on ports 8883, 8884 and 8885. And, ports 8886, 8887 and 8888 are used for UDP communication for the servers of Replica-3.

There are 2 different classes each for different types of user that are going to access the system. AdministratorClient for admin type user and PlayerClient for user type player.

There are 3 server classes named as ASServer. EUServer and NAServer for each replicas which implements methods of IDL Interface and contains data structure.

The Player class contains the attributes and getter setter methods for the Player type of user. Similarly Administrator class for user type Admin. Please note that for admin type user default credential is always “Admin” for username and password.

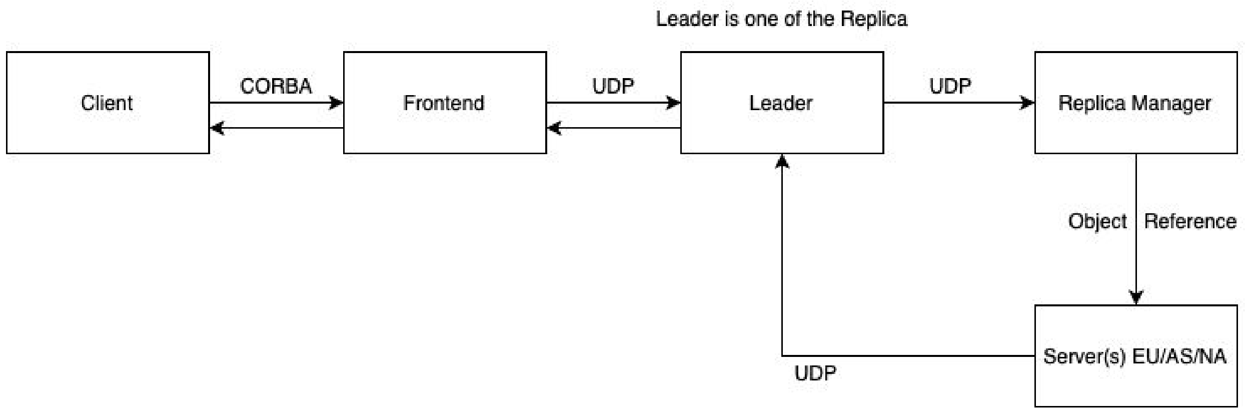


Figure 2: Heterogeneous component communication

AdministratorClient | PlayerClient

- These classes are client classes which interact with the console to get input from respective users and send it to the Frontend.

Frontend | FrontendImplementation

- The frontend class directly communicates with client classes to get input, sends requests to Leader and once response is received, based on user type it moves forward response to the clients.

ASServer | NAServer | EUServer

- These classes directly communicate with the Leader to get input. Also these classes are actual model classes which contain server wise data structure to store and retrieve information. It also contains the method implementation of the interface. It gets data from the client and after processing sends response data back to the client.

- These classes also contain UDP server interaction with respective servers. UDP servers are used in below cases.
  - Getting player status for admin user type
  - Transferring player account from one server to another

- Replica-1 servers named as RM1ASServer, RM1EUServer, RM1NAServer
- Replica-2 servers named as RM2ASServer, RM2EUServer, RM2NAServer
- Replica-3 servers named as RM3ASServer, RM3EUServer, RM3NAServer

GameServer

- This class is used to be a Leader and Manager based on the selection of appropriate Leader from the 3 replicas. It takes requests from Frontend and sends them over to the servers after receiving the response and comparing it, the correct response is sent back to Frontend. It also manages sequence order of the requests as well as fault correction of the failed replica.

- Replica-1 Leader/manager named as RM1GameServer
- Replica-2 Leader/manager named as RM2GameServer
- Replica-3 Leader/manager named as RM3GameServer

Note - A single Leader is active among the three.

RunServer

- This class is used to select the Leader and starting point of the program, which initiates the objects of the Leader and starts the all servers.

CORBA Naming Service (Frontend):

The CORBA Name Service provides an implementation of the Object Management Group (OMG) Interoperable Name Service (INS) specification.

Examples of Controllers are bound to the CORBA Naming Service with three unique strings to open the items to the admin and player client.

- ncRefFE.rebind(pathFE, hrefFE);

CORBA IDL Interface (FECorba.idl) :

Java Interface Definition Language, or Java IDL, is a usage of the CORBA particular and empowers interoperability and network connectivity with heterogeneous instances.

- string getPlayerStatus(in string userName, in string password, in string ipAddress)
- string suspendAccount(in string AdminUsername, in string AdminPassword, in string AdminIP, in string UsernameToSuspend)
- string createPlayerAccount(in string firstName, in string lastName, in string age, in string userName, in string password, in string ipAddress)
- string playerSignIn(in string userName, in string password, in string ipAddress)
- string playerSignOut(in string userName, in string ipAddress)
- string transferAccount(in string userName, in string password, in string OldIPAddress, in string NewIPAddress)

CORBA Servant (FrontendImplementation):

A servant is the invocation target containing methods for handling the remote method invocations.

- IDL interface is implemented by this class.

Data Models:

ConcurrentHashMap is used as a data model. ConcurrentHashMap contains an alphabet as a key to store all data that begins with the same username character. Value of the ConcurrentHashMap is another ConcurrentHashMap which contains the username as a key and player object as a value.

Ie. ConcurrentHashMap<String, ConcurrentHashMap<String, Player>>

Ex. username is **testuser**, it will get/create ConcurrentHashMap with key **t** (first character of username). The value of this HasConcurrentHashMapMap is another ConcurrentHashMap which will again get/create a new ConcurrentHashMap with the key **testuser** (actual username) and it will store the **Player class object/instance** in the value of Inner ConcurrentHashMap.

Logs:

To perform logging for investigating, I have used the java.util.logging logger.

Log Format:

Each log data contains below details:

- Timestamp of the request
- Type of the feature. Ie. createplayer/signin/signout/getplayerstatus
- Parameter. Ie. username
- Message. Ie. success/failure

IP Server:

Each replica’s servers, client, Frontend and Leader logs will be saved in their respective file

- logs/RM1_AS.txt
- logs/RM2_NA.txt
- logs/RM3_EU.txt
- logs/FrontEnd.txt
- logs/Leader.txt

Player/Admin Client:

For every action performed by the player or admin, a log file with username is getting created

Program Flow:

- The client sends a request to Front End using CORBA implementation.
- Front End register with CORBA Naming Service.
- After the FE forwards the request to the Leader, a unique sequence number is appended to the request and multicast to all the replicas by the Leader.
- Thus a Leader keeps a track of order of all incoming requests.
- The Leader will store the request in a buffer and make sure that request is executed using total ordering. Replicas will buffer last processed results along with request id to avoid any duplicate request processing.
- The individual servers will process the request and send the result to the Leader.
- When the Leader receives the response from the replicas, it perform the following:
    - The Leader gets information about the error to the corresponding replica manager with the request id if any inconsistent result is found.
    - The Leader will report an error to a replica manager if there is invalid response from the replica.
    - The Leader provides the response to the FE after comparing the results of all the replicas.
- If a Leader receives a different response from any Replica Manager for three consecutive times, it declares the corresponding RM to be faulty. On detection of software bugs, the respective faulty replica will be replaced.
- After receiving a successful response from the Leader, Frontend will revert it back to the client.

Intermediate Failure:

- The kind of failure executed in DPSS is: **Non-Malicious Byzantine failure**.
- In case of an outsider assault, we have endeavored to exhibit the carriage conduct or defilement of messages in the framework until three successive blunder reports from Front End.
- After the attacked replica is identified, failure recovery mechanism is initiated.
- The rest of the imitation will send their individual information to the influenced copy to take it back to the consistent state and make the data synchronized.

Server Architecture within single Replica:

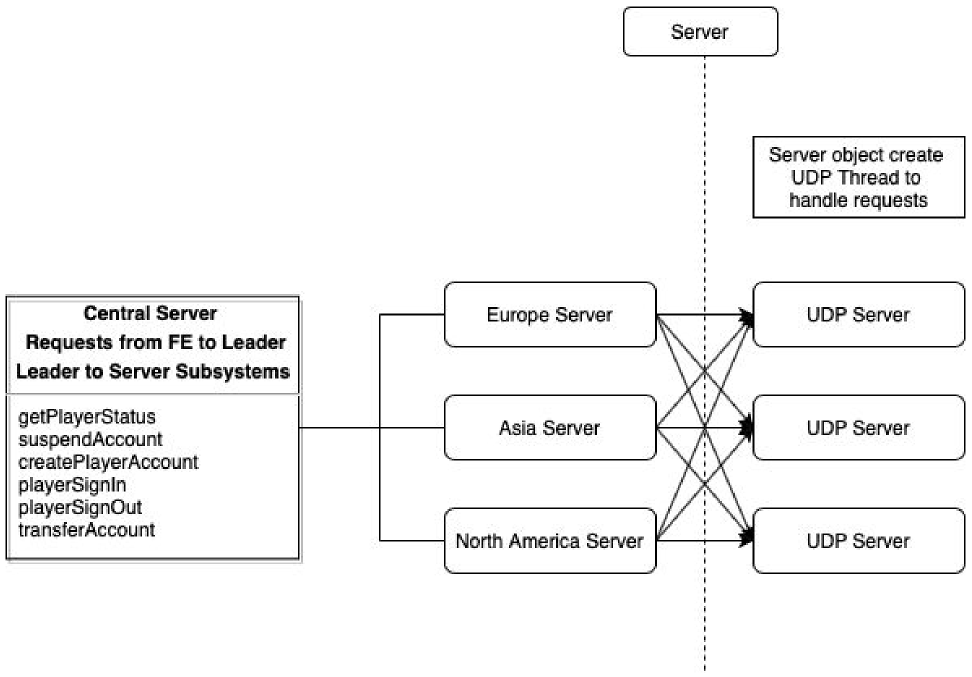


Figure 3: Server Architecture within Replica

UDP Server Design:

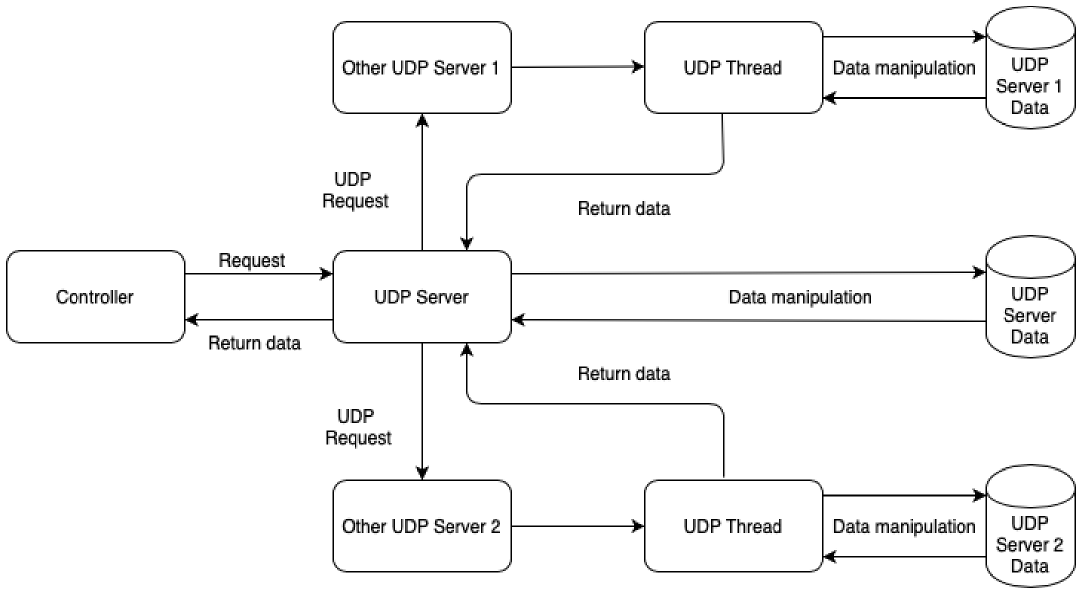


Figure 4: UDP Server Design Diagram

Concurrency Control:

The controller creates new thread to communicate to each of servers to handle requests for the same or different function at the same time. Addition to that all the operational methods are synchronized in nature so that it can handle the concurrency with the type of data structure used in DPSS to avoid inconsistency of data.

Challenges:

- Implementation of synchronization while managing multiple requests at the same time has been challenging.
- To implement this I have used synchronized method as this type of method automatically acquires the lock on the shared object such as ConcurrentHashMap in the DPSS and releases it when the method completes its execution. These synchronized methods are capable of handling multiple invocations from the numerous threads. I have used ConcurrentHashMap as ConcurrentHashMap is more synchronized in nature as well as Thread safe compared to HashMap.
- Creation of unique request id by Leader for identifying each request uniquely. Implementing algorithms for total ordering of incoming requests was a time-consuming task. For that i have used priorityQueue with read write locks and wait and notify methods on state changes. It was a tough task to maintain consistent states of all the replicas after recovering from intermediate failures and software bugs. Additional code had to be written to replicate data from all the non-faulty replicas.

Replication Algorithm:

In this project, the Leader will multicast the client’s request to all the Resource Managers (RM).

1. **Request**: The Leader after receiving request from the FE (Front End), will multicast to each of the RMs using a totally ordered (FIFO), reliable multicast primitive. Next request will only be sent after the response for the first request has been received.

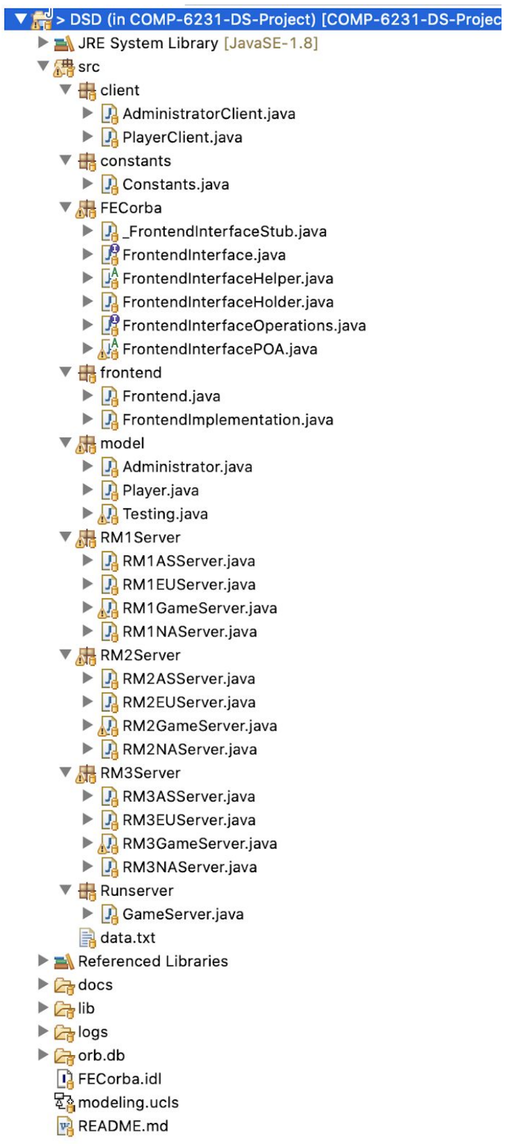
2. **Coordination**: The communication ensures that the request is received by each of the RMs in the same (total) order.

3. **Execution**: Each RM would process the request in the same total order as all the RMs and in an ideal scenario should respond identically. Each response would contain a unique request identifier.

4. **Agreement**: As multicast approach is utilized, this stage isn't required.

5. **Response**: Every RM sends its response to the Leader. Depending on the type of software bug, the system handles, for e.g. three consecutive incorrect responses received by Leader, It would send the correct replica's response to the Frontend and inform the RMs about the same so that RM can correct the data of it's subsystem servers. All the responses are delivered to the client via Frontend.

Code Structure:



```
> DSD (in COMP-6231-DS-Project) [COMP-6231-DS-Projec
  JRE System Library [JavaSE-1.8]
  src
    client
      AdministratorClient.java
      PlayerClient.java
    constants
      Constants.java
    FECorba
      _FrontendInterfaceStub.java
      FrontendInterface.java
      FrontendInterfaceHelper.java
      FrontendInterfaceHolder.java
      FrontendInterfaceOperations.java
      FrontendInterfacePOA.java
    frontend
      Frontend.java
      FrontendImplementation.java
    model
      Administrator.java
      Player.java
      Testing.java
    RM1Server
      RM1ASServer.java
      RM1EUServer.java
      RM1GameServer.java
      RM1NAServer.java
    RM2Server
      RM2ASServer.java
      RM2EUServer.java
      RM2GameServer.java
      RM2NAServer.java
    RM3Server
      RM3ASServer.java
      RM3EUServer.java
      RM3GameServer.java
      RM3NAServer.java
    Runserver
      GameServer.java
    data.txt
  Referenced Libraries
  docs
  lib
  logs
  orb.db
  FECorba.idl
  modeling.ucls
  README.md
```

Figure 5: Code Structure

Run Program:

- To **start the CORBA**, please run the following commands in terminal/cmd.
    - Windows
        - start orbd -ORBInitialPort 1050 -serverPollingTime 200&
    - MacOs/Linux
        - orbd -ORBInitialPort 1050 -serverPollingTime 200&

- Run **GameServer.java** to start the Leaders and all server subsystems with the below arguments in run configuration.
    - –ORBInitialHost localhost -ORBInitialPort 1050

- Run **Frontend.java** to start the Frontend with the below arguments in run configuration.
    - –ORBInitialHost localhost -ORBInitialPort 1050

- Run **PlayerClient.java** to start the player client functionalities with the below arguments in run configuration.
    - –ORBInitialHost localhost -ORBInitialPort 1050

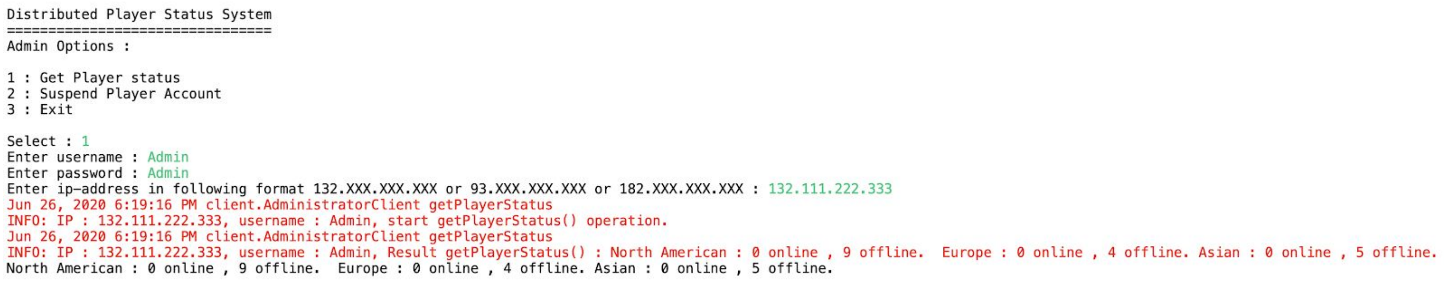
- Run **AdministratorClient.java** to start the admin client functionalities with the below arguments in run configuration.
    - –ORBInitialHost localhost -ORBInitialPort 1050

- Run **Testing.java** to verify the test cases with the below arguments in run configuration.
    - –ORBInitialHost localhost -ORBInitialPort 1050

Console Test Scenarios:

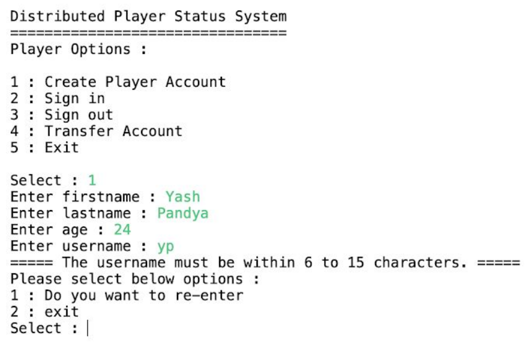
- **Before all the UI/Console tests, a few players are added using data.txt file in each server. Ie. Northamerica has 9 players loaded, Europe has 4 players loaded and Asia has 5 players loaded. Below tests are the UI/Console Interaction Tests. Synchronization Test is added in the advanced tests category.**

- **Admin Get Player status to validate online/offline counts**
    - This test checks the number of player accounts with all servers with their status.



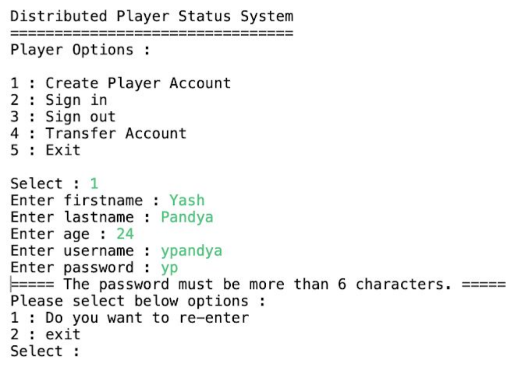
```
Distributed Player Status System
================================
Admin Options :

1 : Get Player status
2 : Suspend Player Account
3 : Exit

Select : 1
Enter username : Admin
Enter password : Admin
Enter ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 132.111.222.333
Jun 26, 2020 6:19:16 PM client.AdministratorClient getPlayerStatus
INFO: IP : 132.111.222.333, username : Admin, start getPlayerStatus() operation.
Jun 26, 2020 6:19:16 PM client.AdministratorClient getPlayerStatus
INFO: IP : 132.111.222.333, username : Admin, Result getPlayerStatus() : North American : 0 online , 9 offline.  Europe : 0 online , 4 offline. Asian : 0 online , 5 offline.
North American : 0 online , 9 offline.  Europe : 0 online , 4 offline. Asian : 0 online , 5 offline.
```

- **Create Player to check username validation error**
    - o This test checks validation on username while player account creation.



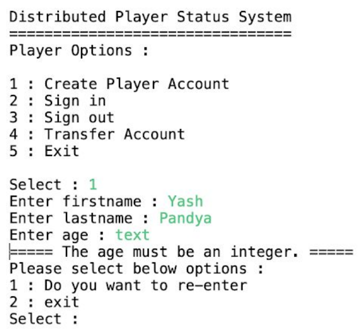
```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 1
Enter firstname : Yash
Enter lastname : Pandya
Enter age : 24
Enter username : yp
===== The username must be within 6 to 15 characters. =====
Please select below options :
1 : Do you want to re-enter
2 : exit
Select :
```

- **Create Player to check password validation error**
    - o This test checks validation on password while player account creation.



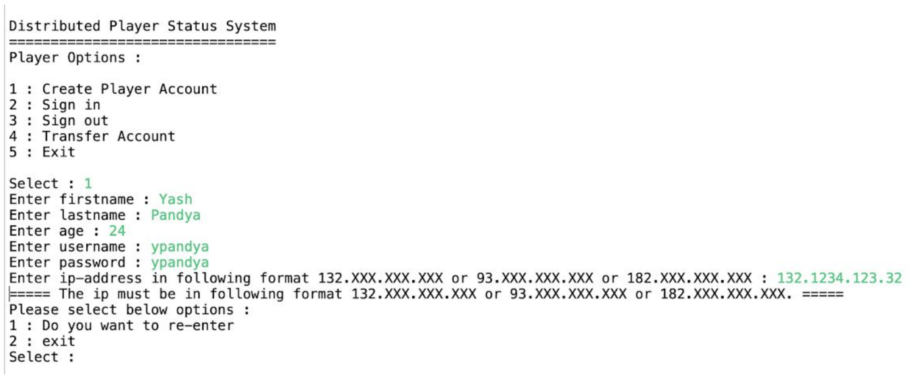
```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 1
Enter firstname : Yash
Enter lastname : Pandya
Enter age : 24
Enter username : ypandya
Enter password : yp
===== The password must be more than 6 characters. =====
Please select below options :
1 : Do you want to re-enter
2 : exit
Select :
```

- **Create Player to check age validation error**
    - o This test checks validation on age while player account creation.



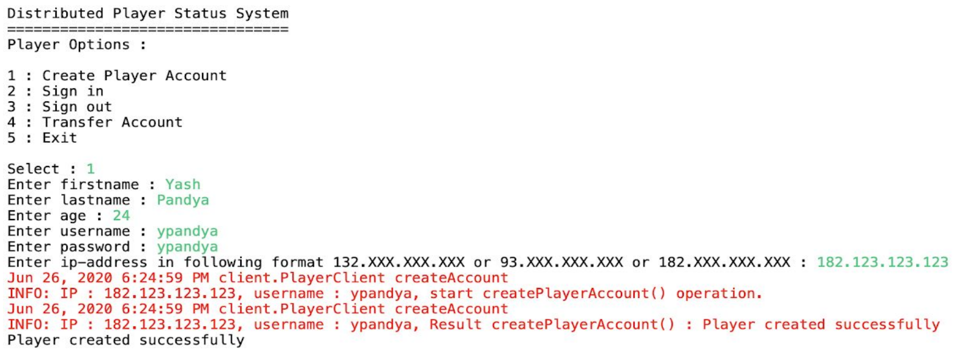
```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 1
Enter firstname : Yash
Enter lastname : Pandya
Enter age : text
===== The age must be an integer. =====
Please select below options :
1 : Do you want to re-enter
2 : exit
Select :
```

- **Create Player to check IP validation error**
    - This test checks validation on IP while player account creation.



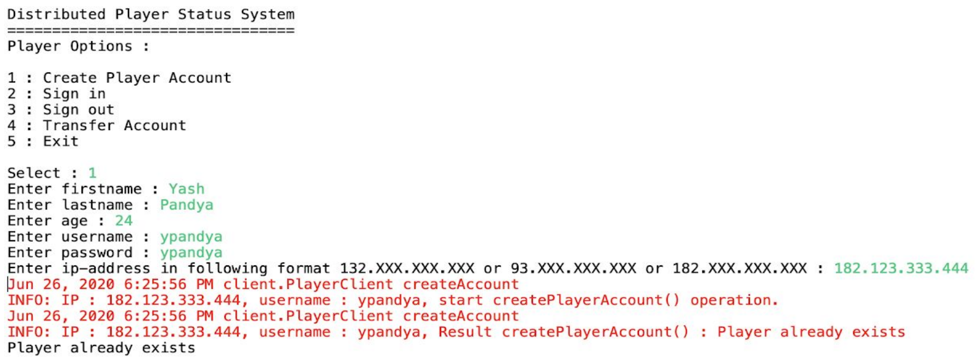
```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 1
Enter firstname : Yash
Enter lastname : Pandya
Enter age : 24
Enter username : ypandya
Enter password : ypandya
Enter ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 132.1234.123.32
===== The ip must be in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX. =====
Please select below options :
1 : Do you want to re-enter
2 : exit
Select :
```

- **Create New Player**
    - This test performs the creation of a player account.



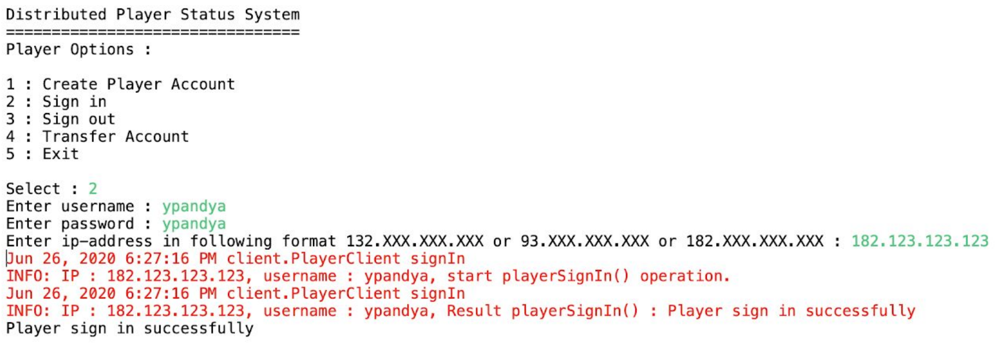
```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 1
Enter firstname : Yash
Enter lastname : Pandya
Enter age : 24
Enter username : ypandya
Enter password : ypandya
Enter ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 182.123.123.123
Jun 26, 2020 6:24:59 PM client.PlayerClient createAccount
INFO: IP : 182.123.123.123, username : ypandya, start createPlayerAccount() operation.
Jun 26, 2020 6:24:59 PM client.PlayerClient createAccount
INFO: IP : 182.123.123.123, username : ypandya, Result createPlayerAccount() : Player created successfully
Player created successfully
```

- **Create Player to validate existing user error**
    - This test checks validation on player existence while creating the player account.



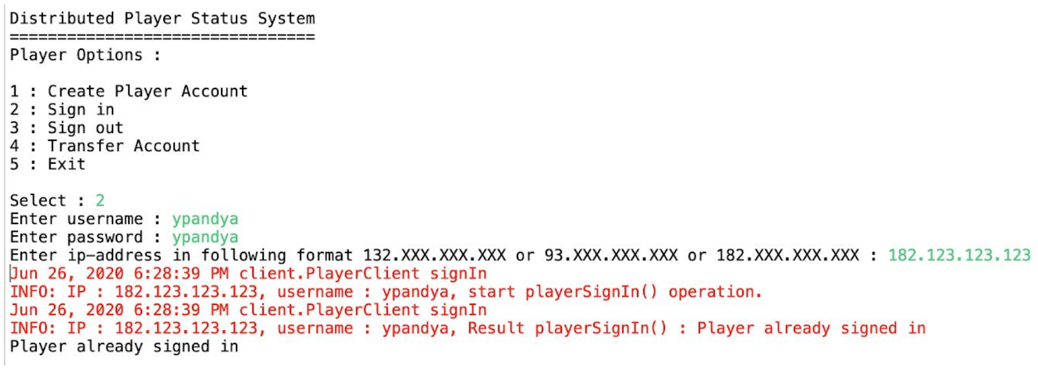
```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 1
Enter firstname : Yash
Enter lastname : Pandya
Enter age : 24
Enter username : ypandya
Enter password : ypandya
Enter ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 182.123.333.444
Jun 26, 2020 6:25:56 PM client.PlayerClient createAccount
INFO: IP : 182.123.333.444, username : ypandya, start createPlayerAccount() operation.
Jun 26, 2020 6:25:56 PM client.PlayerClient createAccount
INFO: IP : 182.123.333.444, username : ypandya, Result createPlayerAccount() : Player already exists
Player already exists
```

- **Player Sign in**
  - This test performs the sign in of a player account.



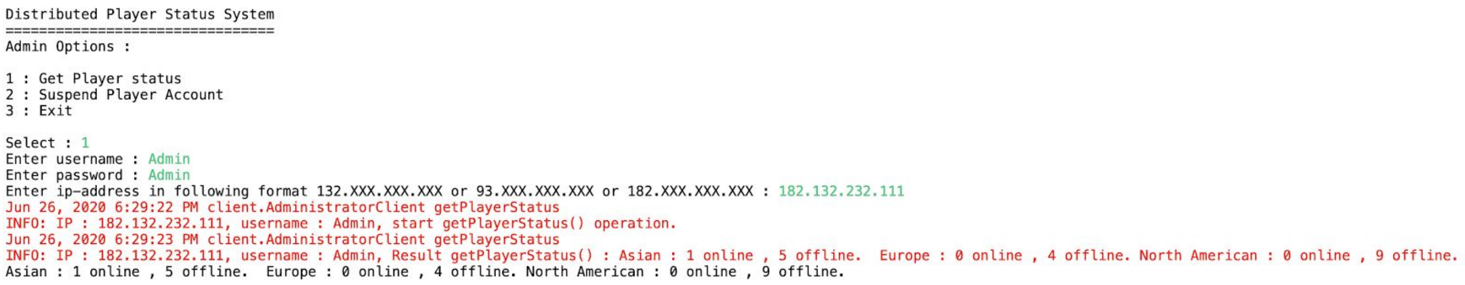
```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 2
Enter username : ypandya
Enter password : ypandya
Enter ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 182.123.123.123
Jun 26, 2020 6:27:16 PM client.PlayerClient signIn
INFO: IP : 182.123.123.123, username : ypandya, start playerSignIn() operation.
Jun 26, 2020 6:27:16 PM client.PlayerClient signIn
INFO: IP : 182.123.123.123, username : ypandya, Result playerSignIn() : Player sign in successfully
Player sign in successfully
```

- **Player Sign in to validate already signin error**
  - This checks sign in validation when a player trying to sign in again after first sign in is successful.



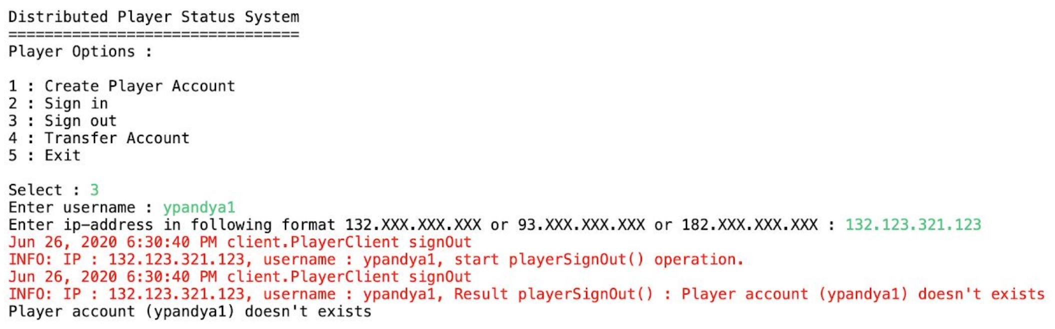
```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 2
Enter username : ypandya
Enter password : ypandya
Enter ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 182.123.123.123
Jun 26, 2020 6:28:39 PM client.PlayerClient signIn
INFO: IP : 182.123.123.123, username : ypandya, start playerSignIn() operation.
Jun 26, 2020 6:28:39 PM client.PlayerClient signIn
INFO: IP : 182.123.123.123, username : ypandya, Result playerSignIn() : Player already signed in
Player already signed in
```

- **Admin Get Player status to validate online/offline counts**
  - This test checks the number of player accounts with all servers with their status.



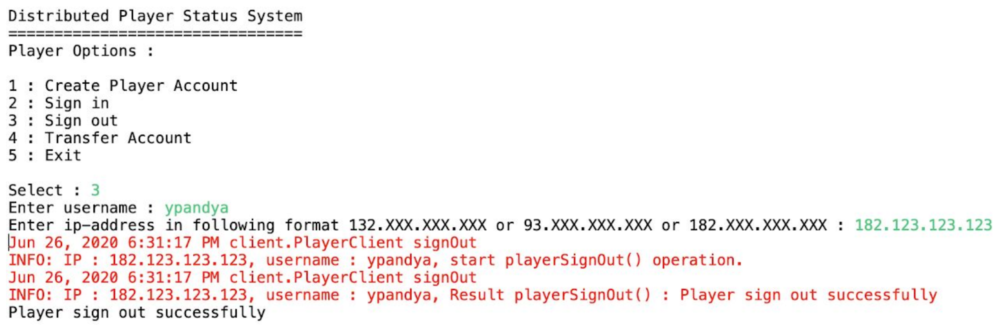
```
Distributed Player Status System
================================
Admin Options :

1 : Get Player status
2 : Suspend Player Account
3 : Exit

Select : 1
Enter username : Admin
Enter password : Admin
Enter ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 182.132.232.111
Jun 26, 2020 6:29:22 PM client.AdministratorClient getPlayerStatus
INFO: IP : 182.132.232.111, username : Admin, start getPlayerStatus() operation.
Jun 26, 2020 6:29:23 PM client.AdministratorClient getPlayerStatus
INFO: IP : 182.132.232.111, username : Admin, Result getPlayerStatus() : Asian : 1 online , 5 offline.  Europe : 0 online , 4 offline. North American : 0 online , 9 offline.
Asian : 1 online , 5 offline.  Europe : 0 online , 4 offline. North American : 0 online , 9 offline.
```

- **Player Sign out to validate username/ip error**
    - This test checks validation on username/ip error while sign out from the player.



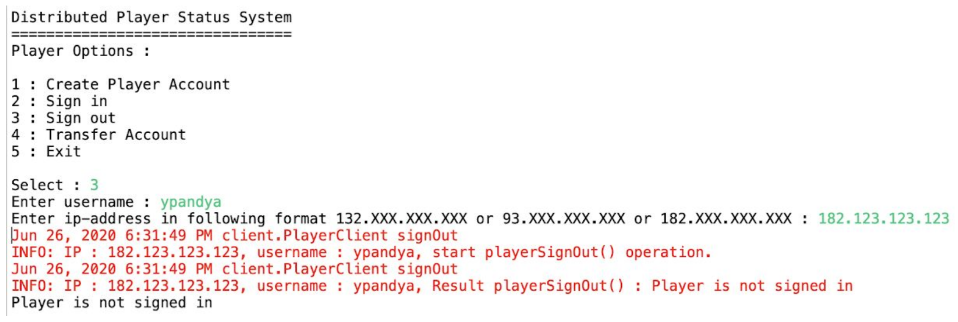
```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 3
Enter username : ypandya1
Enter ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 132.123.321.123
Jun 26, 2020 6:30:40 PM client.PlayerClient signOut
INFO: IP : 132.123.321.123, username : ypandya1, start playerSignOut() operation.
Jun 26, 2020 6:30:40 PM client.PlayerClient signOut
INFO: IP : 132.123.321.123, username : ypandya1, Result playerSignOut() : Player account (ypandya1) doesn't exists
Player account (ypandya1) doesn't exists
```

- **Player Sign out**
    - This test performs the sign out of a player account.



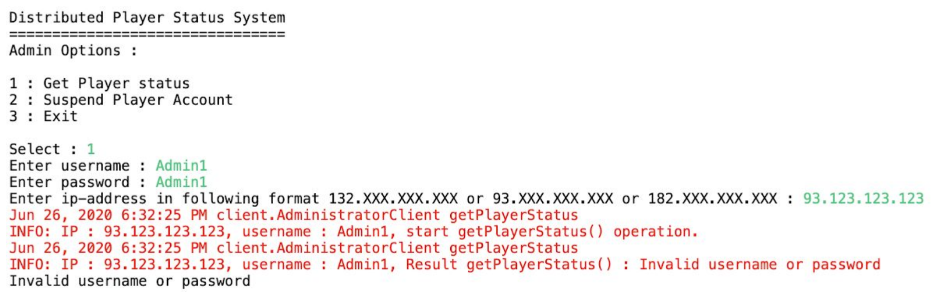
```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 3
Enter username : ypandya
Enter ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 182.123.123.123
Jun 26, 2020 6:31:17 PM client.PlayerClient signOut
INFO: IP : 182.123.123.123, username : ypandya, start playerSignOut() operation.
Jun 26, 2020 6:31:17 PM client.PlayerClient signOut
INFO: IP : 182.123.123.123, username : ypandya, Result playerSignOut() : Player sign out successfully
Player sign out successfully
```

- **Player Sign out to validate not yet signed in error**
    - This test checks validation on not signed in while sign out player account.



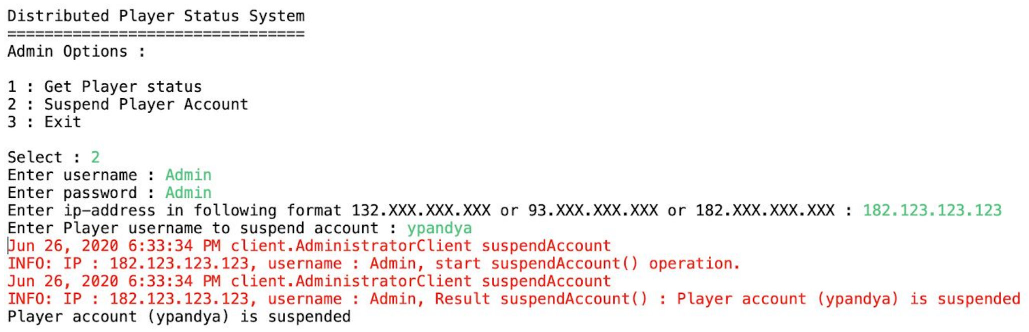
```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 3
Enter username : ypandya
Enter ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 182.123.123.123
Jun 26, 2020 6:31:49 PM client.PlayerClient signOut
INFO: IP : 182.123.123.123, username : ypandya, start playerSignOut() operation.
Jun 26, 2020 6:31:49 PM client.PlayerClient signOut
INFO: IP : 182.123.123.123, username : ypandya, Result playerSignOut() : Player is not signed in
Player is not signed in
```

- **Admin Get Player Status to validate username and password**
    - This test checks validation on wrong admin username and password.



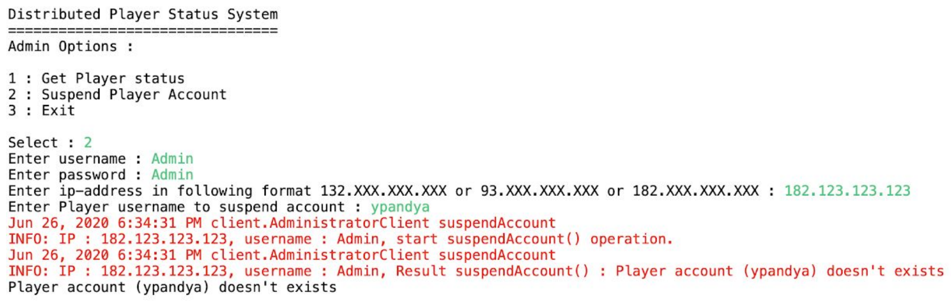
```
Distributed Player Status System
================================
Admin Options :

1 : Get Player status
2 : Suspend Player Account
3 : Exit

Select : 1
Enter username : Admin1
Enter password : Admin1
Enter ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 93.123.123.123
Jun 26, 2020 6:32:25 PM client.AdministratorClient getPlayerStatus
INFO: IP : 93.123.123.123, username : Admin1, start getPlayerStatus() operation.
Jun 26, 2020 6:32:25 PM client.AdministratorClient getPlayerStatus
INFO: IP : 93.123.123.123, username : Admin1, Result getPlayerStatus() : Invalid username or password
Invalid username or password
```

- **Admin Suspend Player Account**
    - This test performs suspend player account operation of admin user.



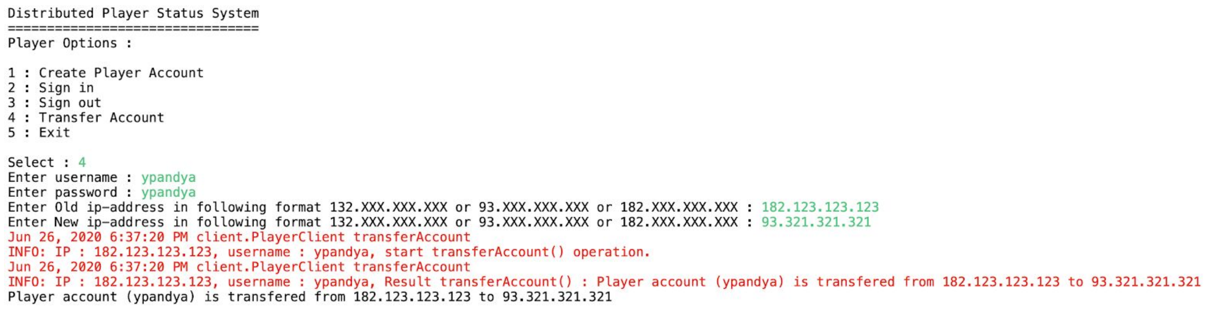
```
Distributed Player Status System
================================
Admin Options :

1 : Get Player status
2 : Suspend Player Account
3 : Exit

Select : 2
Enter username : Admin
Enter password : Admin
Enter ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 182.123.123.123
Enter Player username to suspend account : ypandya
Jun 26, 2020 6:33:34 PM client.AdministratorClient suspendAccount
INFO: IP : 182.123.123.123, username : Admin, start suspendAccount() operation.
Jun 26, 2020 6:33:34 PM client.AdministratorClient suspendAccount
INFO: IP : 182.123.123.123, username : Admin, Result suspendAccount() : Player account (ypandya) is suspended
Player account (ypandya) is suspended
```

- **Admin Suspend Player Account with error**
    - This test checks validation on player existence before suspending account.



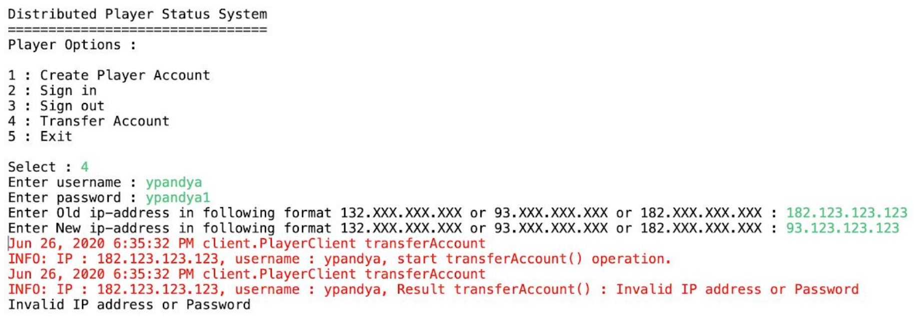
```
Distributed Player Status System
================================
Admin Options :

1 : Get Player status
2 : Suspend Player Account
3 : Exit

Select : 2
Enter username : Admin
Enter password : Admin
Enter ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 182.123.123.123
Enter Player username to suspend account : ypandya
Jun 26, 2020 6:34:31 PM client.AdministratorClient suspendAccount
INFO: IP : 182.123.123.123, username : Admin, start suspendAccount() operation.
Jun 26, 2020 6:34:31 PM client.AdministratorClient suspendAccount
INFO: IP : 182.123.123.123, username : Admin, Result suspendAccount() : Player account (ypandya) doesn't exists
Player account (ypandya) doesn't exists
```

- **Player Transfer Account**
    - This test performs transfer player accounts from one server to another.



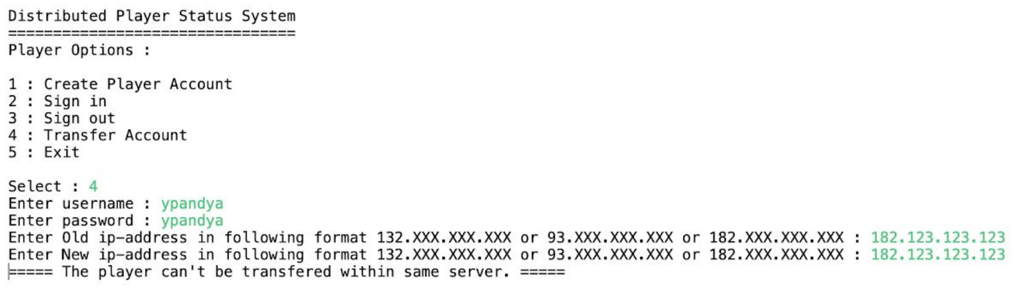
```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 4
Enter username : ypandya
Enter password : ypandya
Enter Old ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 182.123.123.123
Enter New ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 93.321.321.321
Jun 26, 2020 6:37:20 PM client.PlayerClient transferAccount
INFO: IP : 182.123.123.123, username : ypandya, start transferAccount() operation.
Jun 26, 2020 6:37:20 PM client.PlayerClient transferAccount
INFO: IP : 182.123.123.123, username : ypandya, Result transferAccount() : Player account (ypandya) is transfered from 182.123.123.123 to 93.321.321.321
Player account (ypandya) is transfered from 182.123.123.123 to 93.321.321.321
```

- **Player Transfer Account with error**
    - This test checks transfer player validation of username and password.



```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 4
Enter username : ypandya
Enter password : ypandya1
Enter Old ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 182.123.123.123
Enter New ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 93.123.123.123
Jun 26, 2020 6:35:32 PM client.PlayerClient transferAccount
INFO: IP : 182.123.123.123, username : ypandya, start transferAccount() operation.
Jun 26, 2020 6:35:32 PM client.PlayerClient transferAccount
INFO: IP : 182.123.123.123, username : ypandya, Result transferAccount() : Invalid IP address or Password
Invalid IP address or Password
```

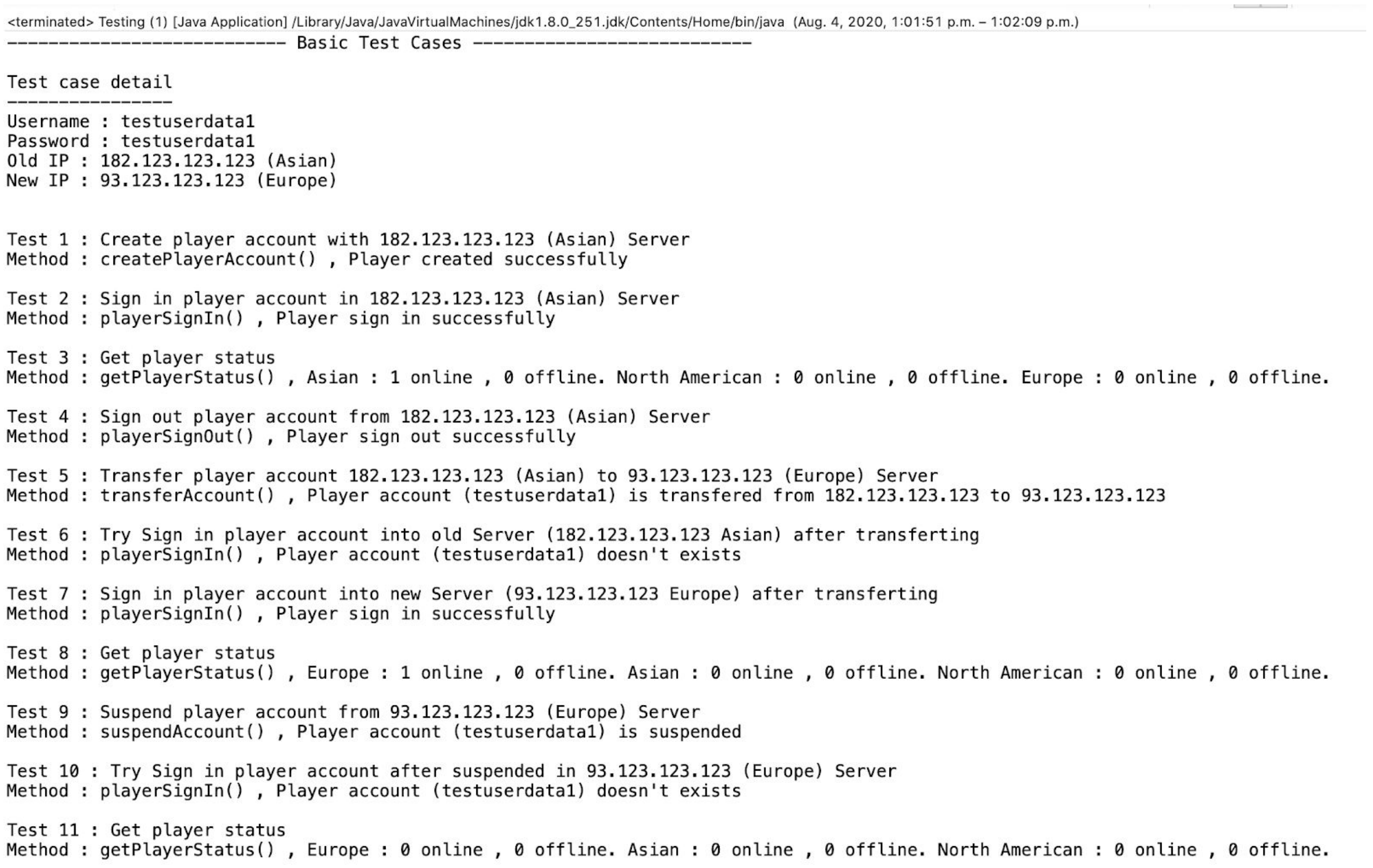
- **Player Transfer Account in same server/IP**
    - This test checks transfer player validation if the transfer server ip is the same as the current server.



```
Distributed Player Status System
================================
Player Options :

1 : Create Player Account
2 : Sign in
3 : Sign out
4 : Transfer Account
5 : Exit

Select : 4
Enter username : ypandya
Enter password : ypandya
Enter Old ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 182.123.123.123
Enter New ip-address in following format 132.XXX.XXX.XXX or 93.XXX.XXX.XXX or 182.XXX.XXX.XXX : 182.123.123.123
===== The player can't be transfered within same server. =====
```

Basic Test Scenarios:

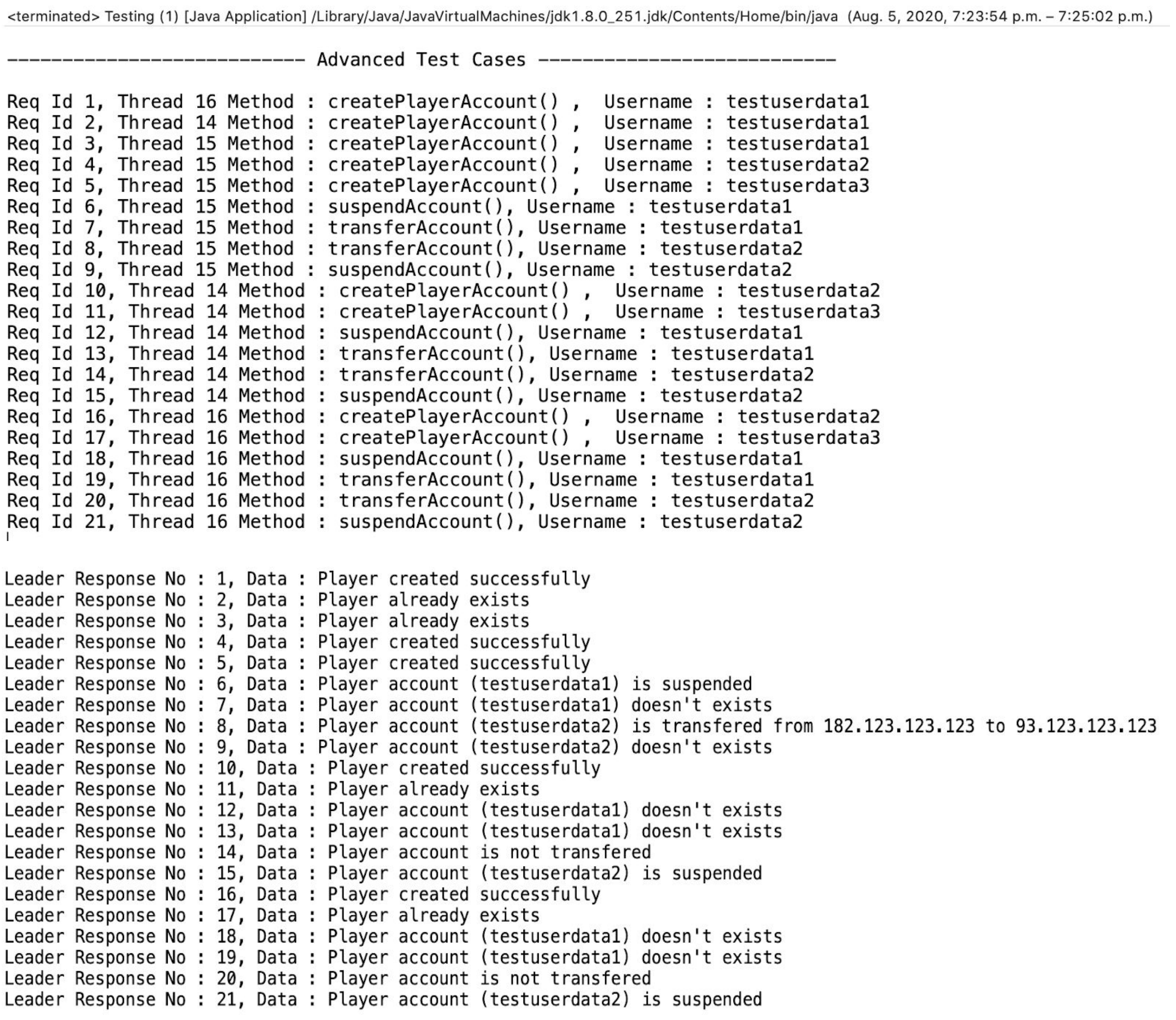
- **Below test cases are run directly on the server no interaction with the client side is involved in it to verify the correctness of the methods without console integration.**
- **No initial data was loaded while running these test cases.**



```
<terminated> Testing (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_251.jdk/Contents/Home/bin/java  (Aug. 4, 2020, 1:01:51 p.m. – 1:02:09 p.m.)
--------------------------- Basic Test Cases ---------------------------

Test case detail
----------------
Username : testuserdata1
Password : testuserdata1
Old IP : 182.123.123.123 (Asian)
New IP : 93.123.123.123 (Europe)


Test 1 : Create player account with 182.123.123.123 (Asian) Server
Method : createPlayerAccount() , Player created successfully

Test 2 : Sign in player account in 182.123.123.123 (Asian) Server
Method : playerSignIn() , Player sign in successfully

Test 3 : Get player status
Method : getPlayerStatus() , Asian : 1 online , 0 offline. North American : 0 online , 0 offline. Europe : 0 online , 0 offline.

Test 4 : Sign out player account from 182.123.123.123 (Asian) Server
Method : playerSignOut() , Player sign out successfully

Test 5 : Transfer player account 182.123.123.123 (Asian) to 93.123.123.123 (Europe) Server
Method : transferAccount() , Player account (testuserdata1) is transfered from 182.123.123.123 to 93.123.123.123

Test 6 : Try Sign in player account into old Server (182.123.123.123 Asian) after transferting
Method : playerSignIn() , Player account (testuserdata1) doesn't exists

Test 7 : Sign in player account into new Server (93.123.123.123 Europe) after transferting
Method : playerSignIn() , Player sign in successfully

Test 8 : Get player status
Method : getPlayerStatus() , Europe : 1 online , 0 offline. Asian : 0 online , 0 offline. North American : 0 online , 0 offline.

Test 9 : Suspend player account from 93.123.123.123 (Europe) Server
Method : suspendAccount() , Player account (testuserdata1) is suspended

Test 10 : Try Sign in player account after suspended in 93.123.123.123 (Europe) Server
Method : playerSignIn() , Player account (testuserdata1) doesn't exists

Test 11 : Get player status
Method : getPlayerStatus() , Europe : 0 online , 0 offline. Asian : 0 online , 0 offline. North American : 0 online , 0 offline.
```

Advanced/Concurrency Test Scenarios:

- **In Advanced/Concurrency tests, no data added into the server.**
- **Below tests cases are run directly on the server no interaction with the client side is involved in it. Synchronization test is added with CreatePlayer, SuspendPlayer and TransferPlayer account.**
- I have made 3 threads that run on Frontend and perform **FIFO** sequencing by the Leader. All the invocations take place according to order of received requests.
- Everytime output will be different based on the order of received requests. Requests are having sequence id and based on that conclusion of successful synchronized concurrent requests invocation are stated.



```
<terminated> Testing (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_251.jdk/Contents/Home/bin/java  (Aug. 5, 2020, 7:23:54 p.m. – 7:25:02 p.m.)

--------------------------- Advanced Test Cases ---------------------------

Req Id 1, Thread 16 Method : createPlayerAccount() ,  Username : testuserdata1
Req Id 2, Thread 14 Method : createPlayerAccount() ,  Username : testuserdata1
Req Id 3, Thread 15 Method : createPlayerAccount() ,  Username : testuserdata1
Req Id 4, Thread 15 Method : createPlayerAccount() ,  Username : testuserdata2
Req Id 5, Thread 15 Method : createPlayerAccount() ,  Username : testuserdata3
Req Id 6, Thread 15 Method : suspendAccount(), Username : testuserdata1
Req Id 7, Thread 15 Method : transferAccount(), Username : testuserdata1
Req Id 8, Thread 15 Method : transferAccount(), Username : testuserdata2
Req Id 9, Thread 15 Method : suspendAccount(), Username : testuserdata2
Req Id 10, Thread 14 Method : createPlayerAccount() ,  Username : testuserdata2
Req Id 11, Thread 14 Method : createPlayerAccount() ,  Username : testuserdata3
Req Id 12, Thread 14 Method : suspendAccount(), Username : testuserdata1
Req Id 13, Thread 14 Method : transferAccount(), Username : testuserdata1
Req Id 14, Thread 14 Method : transferAccount(), Username : testuserdata2
Req Id 15, Thread 14 Method : suspendAccount(), Username : testuserdata2
Req Id 16, Thread 16 Method : createPlayerAccount() ,  Username : testuserdata2
Req Id 17, Thread 16 Method : createPlayerAccount() ,  Username : testuserdata3
Req Id 18, Thread 16 Method : suspendAccount(), Username : testuserdata1
Req Id 19, Thread 16 Method : transferAccount(), Username : testuserdata1
Req Id 20, Thread 16 Method : transferAccount(), Username : testuserdata2
Req Id 21, Thread 16 Method : suspendAccount(), Username : testuserdata2

Leader Response No : 1, Data : Player created successfully
Leader Response No : 2, Data : Player already exists
Leader Response No : 3, Data : Player already exists
Leader Response No : 4, Data : Player created successfully
Leader Response No : 5, Data : Player created successfully
Leader Response No : 6, Data : Player account (testuserdata1) is suspended
Leader Response No : 7, Data : Player account (testuserdata1) doesn't exists
Leader Response No : 8, Data : Player account (testuserdata2) is transfered from 182.123.123.123 to 93.123.123.123
Leader Response No : 9, Data : Player account (testuserdata2) doesn't exists
Leader Response No : 10, Data : Player created successfully
Leader Response No : 11, Data : Player already exists
Leader Response No : 12, Data : Player account (testuserdata1) doesn't exists
Leader Response No : 13, Data : Player account (testuserdata1) doesn't exists
Leader Response No : 14, Data : Player account is not transfered
Leader Response No : 15, Data : Player account (testuserdata2) is suspended
Leader Response No : 16, Data : Player created successfully
Leader Response No : 17, Data : Player already exists
Leader Response No : 18, Data : Player account (testuserdata1) doesn't exists
Leader Response No : 19, Data : Player account (testuserdata1) doesn't exists
Leader Response No : 20, Data : Player account is not transfered
Leader Response No : 21, Data : Player account (testuserdata2) is suspended
```

Total Order:

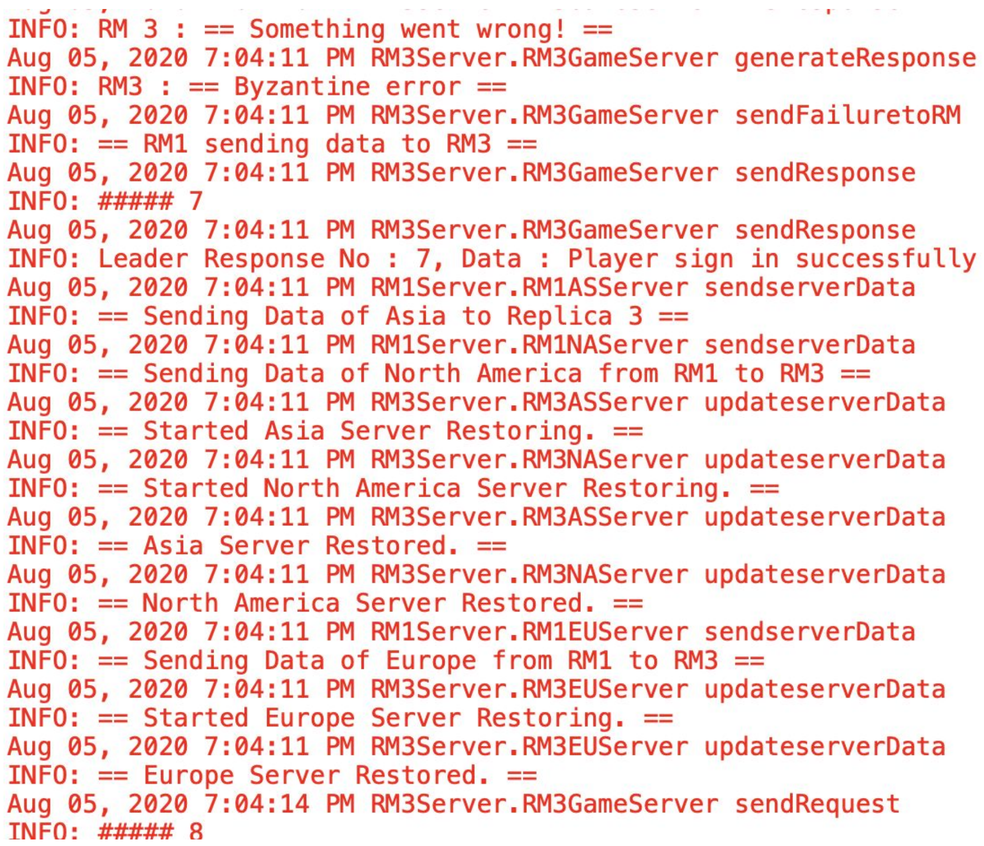
- The general approach is to attach totally ordered identifiers to messages.
- Each receiving process makes ordering decisions based on the identifiers like the FIFO algorithm, but processes keep group specific sequence numbers.
- Operations TO-sending and TO-deliver.

Software Bug Test Case / non-malicious Byzantine :

- The Leader can replace the replica if it gets three times successively incorrect response with the new replica. The new replica still preserves the state of the old replica. This is achieved using a data restoring mechanism in which data from any other replica and corrects the data of the respective server subsystem.

- In the Distributed player status system, this software bug is performed as three successively incorrect responses corresponding with requests id 5,6 and 7. After, the 7th request data restore mechanism automatically takes place between replica 1 or replica 2 to the replica 3. This error is implemented into replica 3.

- Also, meanwhile servers are getting restored, requests may come but not proceed till restoring is completed.



```
INFO: RM 3 : == Something went wrong! ==
Aug 05, 2020 7:04:11 PM RM3Server.RM3GameServer generateResponse
INFO: RM3 : == Byzantine error ==
Aug 05, 2020 7:04:11 PM RM3Server.RM3GameServer sendFailuretoRM
INFO: == RM1 sending data to RM3 ==
Aug 05, 2020 7:04:11 PM RM3Server.RM3GameServer sendResponse
INFO: ##### 7
Aug 05, 2020 7:04:11 PM RM3Server.RM3GameServer sendResponse
INFO: Leader Response No : 7, Data : Player sign in successfully
Aug 05, 2020 7:04:11 PM RM1Server.RM1ASServer sendserverData
INFO: == Sending Data of Asia to Replica 3 ==
Aug 05, 2020 7:04:11 PM RM1Server.RM1NAServer sendserverData
INFO: == Sending Data of North America from RM1 to RM3 ==
Aug 05, 2020 7:04:11 PM RM3Server.RM3ASServer updateserverData
INFO: == Started Asia Server Restoring. ==
Aug 05, 2020 7:04:11 PM RM3Server.RM3NAServer updateserverData
INFO: == Started North America Server Restoring. ==
Aug 05, 2020 7:04:11 PM RM3Server.RM3ASServer updateserverData
INFO: == Asia Server Restored. ==
Aug 05, 2020 7:04:11 PM RM3Server.RM3NAServer updateserverData
INFO: == North America Server Restored. ==
Aug 05, 2020 7:04:11 PM RM1Server.RM1EUServer sendserverData
INFO: == Sending Data of Europe from RM1 to RM3 ==
Aug 05, 2020 7:04:11 PM RM3Server.RM3EUServer updateserverData
INFO: == Started Europe Server Restoring. ==
Aug 05, 2020 7:04:11 PM RM3Server.RM3EUServer updateserverData
INFO: == Europe Server Restored. ==
Aug 05, 2020 7:04:14 PM RM3Server.RM3GameServer sendRequest
INFO: ##### 8
```

References:

- https://systembash.com/a-simple-java-udp-server-and-udp-client/
- https://www.geeksforgeeks.org/multithreading-in-java/
- https://www.geeksforgeeks.org/synchronized-in-java/
- https://objectcomputing.com/resources/publications/sett/january-2002-corba-and-java-by-don-busch-principal-software-engineer
- http://www.ejbtutorial.com/corba/tutorial-for-corba-hello-world-using-java
- https://www.geeksforgeeks.org/reentrant-lock-java/
- https://docs.oracle.com/javase/7/docs/technotes/guides/idl/POA.html