

**Name:** Sergey Frolov  
**Project:** OTR-Messenger

Requirement	<user>	<task>	<accomplish my goal>	priority
UR-010	admin	change server status (launch/reset/terminate)	to provide, temporarily stop or terminate service	Critical
UR-011	admin	view list of all users	to monitor the system	High
UR-012	admin	view logged in users	to manage the system	Medium
UR-013	admin	view public keys of all users	to perform debug and security analysis on keys	Medium

Table 1. User requirements covered in this document

### Use Case Tables:

Use Case ID:	UR-010		
Use Case Name:	change server status (launch/reset/terminate)		
Description:	Allow Admin to launch, terminate and reset the settings of the server		
Actors:	admin		
Pre-Conditions:	Admin is successfully logged in to system		
Post-Conditions:	Server status changed		
Frequency of Use:	Low		
Flow of Events:		Actor Action	System Response
	1	Admin navigates to Admin Dashboard	Dashboard pops up
	2	Admin clicks "launch/reset/terminate"	Server status changes, this change is reflected(e.g.) printed on the dashboard
Variations:			
Notes and Issues:	Server binary, when ran, comes up launched. Explicit launch via GUI is only needed after explicit stop(for maintenance needs)		
Developer Notes:			

Table 2. Use Case UR-010 change server status (launch/reset/terminate)

Use Case ID:	UR-011		
Use Case Name:	view list of all users		
Description:	Allow admin to see the list of all registered users		
Actors:	admin		
Pre-Conditions:	Admin is successfully logged in to system		
Post-Conditions:	List of users is displayed in window, logged to file		
Frequency of Use:	Arbitrary, could be high during maintenance/debug		
Flow of Events:		Actor Action	System Response
	1	Admin navigates to Admin Dashboard	Dashboard pops up
	2	Admin clicks "View/log list of users"	Additional window with list of users pops up.
	3	Admin analyzes list of users and closes the window with it	
Variations:			
Notes and Issues:			
Developer Notes:			

Table 3. Use Case UR-011 view list of all users

Use Case ID:	UR-012		
Use Case Name:	view logged in users		
Description:	allow admin to see amount of currently logged in users out of total number of users, and see the usernames of logged in users		
Actors:	admin		
Pre-Conditions:	Admin is successfully logged in to system		
Post-Conditions:	Number of logged-in users out of total, and the usernames of logged-in users is printed in additional window		
Frequency of Use:	Arbitrary, could be high during maintenance/debug		
Flow of Events:		Actor Action	System Response
	1	Admin navigates to	Dashboard pops up

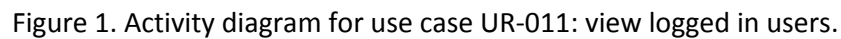
		Admin Dashboard	
	2	Admin clicks "View Logged-in Users" button	Additional window with logged-in users pops up
	3	Admin analyzes statistics and closes the window with it	
Variations:			
Notes and Issues:	May start collecting other statistics about users. Generally, we'd want to collect as few as possible in order not to compromise privacy of users.		
Developer Notes:			

Table 4. Use Case UR-012 view logged in users

Use Case ID:	UR-013		
Use Case Name:	view public keys of all users		
Description:	allow admin to see all keys for debug and security analysis purposes		
Actors:	admin		
Pre-Conditions:	Admin is successfully logged in to system		
Post-Conditions:	Window with user names and their public keys pops up		
Frequency of Use:	Arbitrary, could be high during maintenance/debug		
Flow of Events:		Actor Action	System Response
	1	Admin navigates to Admin Dashboard	Dashboard pops up
	2	Admin clicks "View user keys"	Additional window with user names and their public keys pops up
	3	Admin analyzes the keys and closes the window	
Variations:			
Notes and Issues:			
Developer Notes:			

Table 5. Use Case UR-013 view keys (have access the database)

The following diagram is for use case UR-011: view logged in users.



```

sequenceDiagram
    actor Admin
    participant AdminDashboard as :AdminDashboard
    participant OTRMessenger as :OTRMessenger
    participant OTRServer as :OTRServer
    participant AssetHandler as :AssetHandler
    participant SQLiteDBHandler as :SQLiteDBHandler

    AdminDashboard->>AdminDashboard: displayAdminDashboard()
    AdminDashboard->>AdminDashboard: clickGetLoggedInUsers()
    AdminDashboard->>OTRMessenger: SendRequest()
    OTRMessenger->>OTRServer: << TLS Connection >>
    OTRServer->>OTRServer: HandleGetUserList()
    OTRServer->>AssetHandler: getUsers()
    AssetHandler->>SQLiteDBHandler: getUsers()
    SQLiteDBHandler-->>AssetHandler: ArrayList<String> Users
    AssetHandler-->>OTRServer: ArrayList<String> Users
    OTRServer-->>OTRMessenger: << TLS Connection >>
    OTRMessenger-->>AdminDashboard: Response
  
```

**View Logged in Users**

**Participants:** Admin, :AdminDashboard, :OTRMessenger, :OTRServer, :AssetHandler, :SQLiteDBHandler

**Sequence of Interactions:**

- AdminDashboard** performs a self-call: `displayAdminDashboard()`.
- AdminDashboard** performs a self-call: `clickGetLoggedInUsers()`.
- AdminDashboard** sends `SendRequest()` to **OTRMessenger**.
- OTRMessenger** establishes a `<< TLS Connection >>` with **OTRServer**.
- OTRServer** performs a self-call: `HandleGetUserList()`.
- OTRServer** sends `getUsers()` to **AssetHandler**.
- AssetHandler** sends `getUsers()` to **SQLiteDBHandler**.
- SQLiteDBHandler** returns `ArrayList<String> Users` to **AssetHandler**.
- AssetHandler** returns `ArrayList<String> Users` to **OTRServer**.
- OTRServer** establishes a `<< TLS Connection >>` with **OTRMessenger**.
- OTRMessenger** returns `Response` to **AdminDashboard**.

**Notes:**

- AdminDashboard** popped up when admin logged in, `displayAdminDashboard()` just puts it in focus and redraws if needed.
- OTRServer** Connection was established at login, **OTRServer** is already listening for Admin's requests: after getting and parsing said request **OTRServer** will execute `HandleGetUserList()`.
- AssetHandler** is a facade for various asset handlers, including db.

Figure 2. Sequence diagram for use case UR-011: view logged in users