

UniBoxDoc

alextape

Published
with GitBook



Table of Contents

1. [Introduction](#)
2. [Concept](#)
3. [Server](#)
 - i. [Installation](#)
 - ii. [Configuration](#)
4. [Client](#)
 - i. [Getting Started](#)
 - ii. [Sending Messages](#)
 - iii. [Receive Messages](#)
 - iv. [Handle Highscores](#)
5. [User Guide](#)
6. [Requests](#)
 - i. [Asynchron Requests](#)
 - ii. [Synchron Requests](#)
7. [Hints and Bugs](#)

UniBox

UniBox is an simple and easy to use messaging system implementing custom "games" for distributed gaming clients.

Introduction

The UniBox backend is connecting various gaming clients via http based network connections. Basically it consists of a Servlet Stack linked to a MySQL database (to store user names, passwords, rankings), an easy to use client API and a core library, to cover cross-package dependencies.

It is designed as a asynchron message mediator. Messages can be injected by simple http requests against the backend services. The backend will queue injected messages and the corresponding thread workers will deliver and receive the messages via asynchron long polling http responses based on a [comet-pattern](#) immediatly.

In fact each concrete game represents a screened messaging channel in principle. Besides this functionality of sending messages between a several group of connected clients, the frontend provides a simple chat box to get in bound with other player directly.

The [Wiki Page](#) of this project contains all available information about the usage and the functionality of this project.

Package Overview

Package	Description	JavaDoc	Builds
UniBoxServer	Servlet-based	JavaDoc	WAR
UniBoxCore	Shared dependencies for Server and Client API	JavaDoc	JAR
UniBoxClient	Client API	JavaDoc	JAR
UniBoxGame	Simple JavaFX Application Demo	JavaDoc	JAR

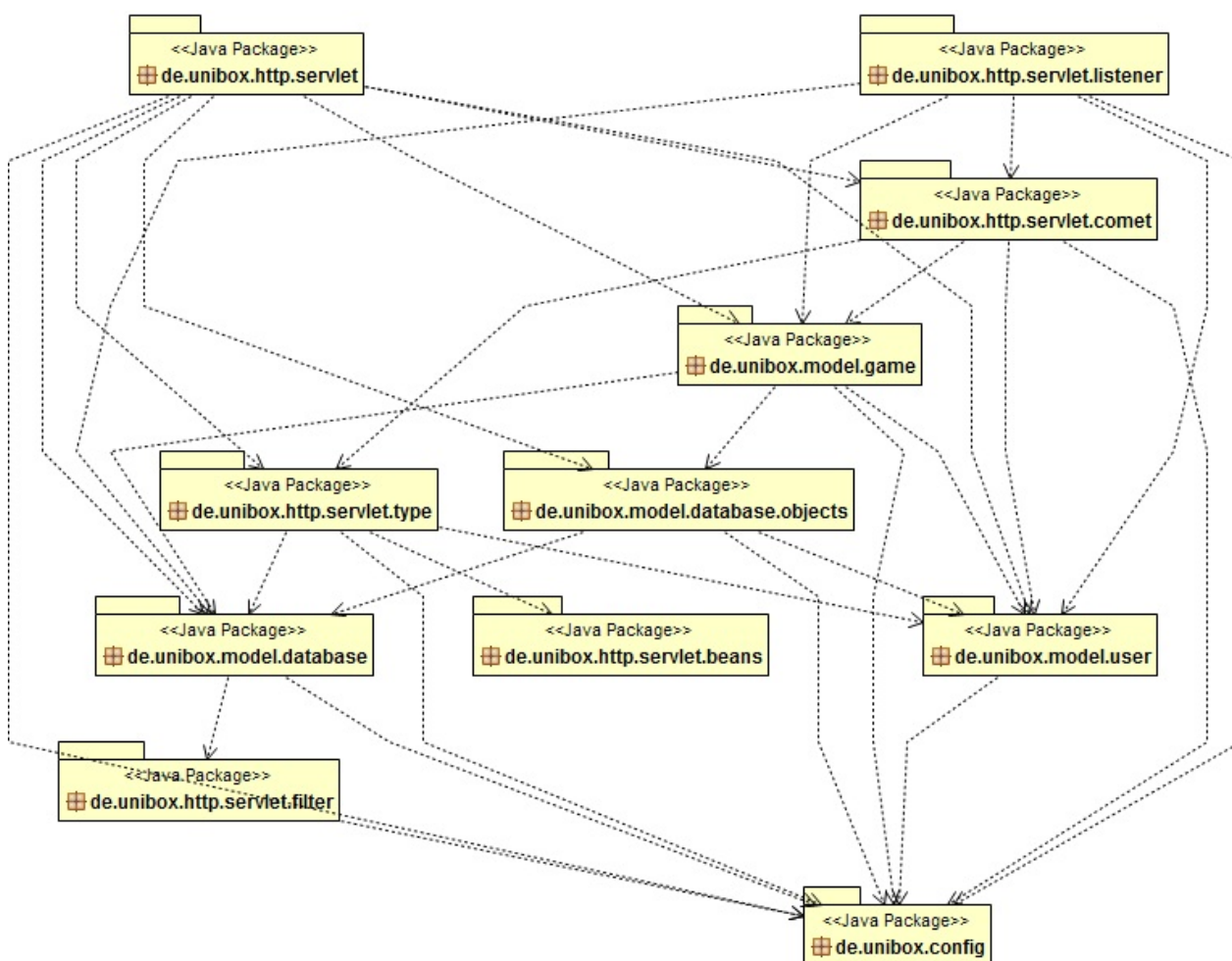
Concept

The communication architecture is basically a comet pattern. Any client, regardless of which kind, can call a certain listener route and will receive a long polling http handle on which all incoming messages transferred instantly. If a client needs to send a message, it will call a parameterized (static) url to inject a new message into the communication lifecycle of the backend. The backend is able to switch messages directly to related receivers and will deliver the messages via the long polling comet routes.

Note: To be prepared for any kind of firewall configurations within the network or the serving system the communication is completely processable over one single port. By default it would be the tomcat destination port e.g. 8080. But you can use any other port if you want to.

Server Structure

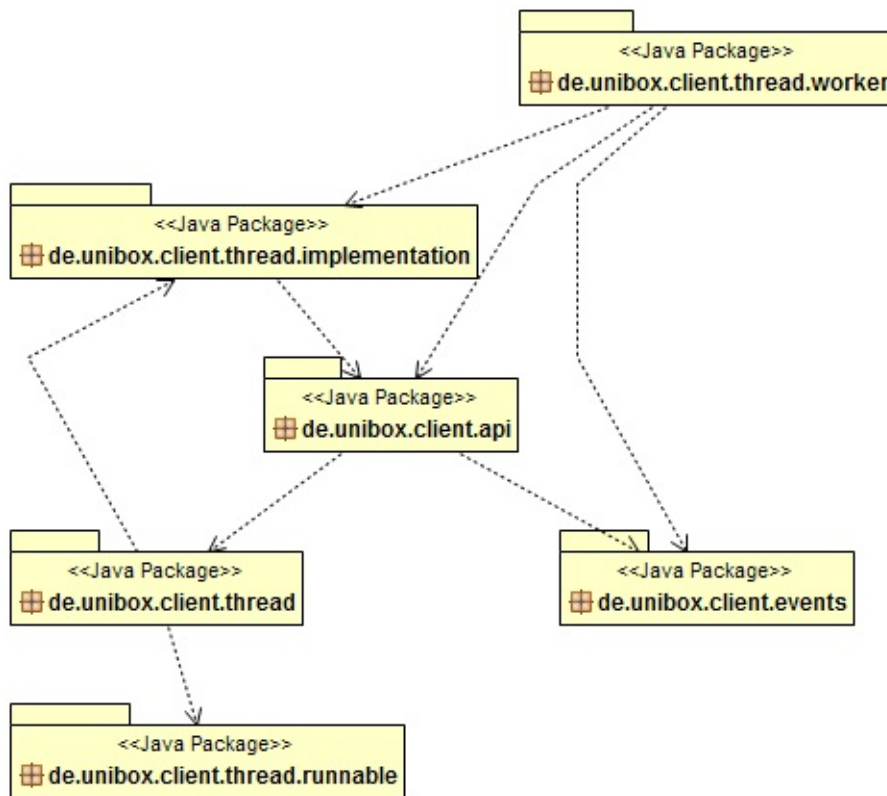
The modular and high flexible structure of UniBoxServer is splitted into several Packages.



To get a more detailed overview you can take a look at the [UML Diagram](#) of the current state.

Client API Structure

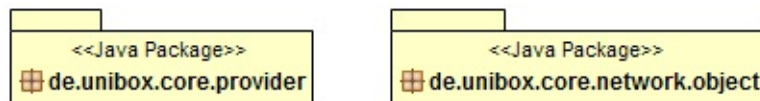
The Client API is accessible through the ClientProvider class located in the de.unibox.client.api Package. It is a static Wrapper-Class which gives access to all needed functionalities provided by the backend.



To get a more detailed overview you can take a look at the [UML Diagram](#) of the current state.

Core Structure

The UniBoxCore centralized the intersection-classes of UniBoxServer and UniBoxClient.



To get a more detailed overview you can take a look at the [UML Diagram](#) of the current state.

Server Guide

The Server is providing a web-frontend to get a quick overview of the current running game sessions. Furthermore there is a chat functionality which is mostly intended to permit a communication between online/registered players before they joined an enclosed game channel.

Installation

Follow these few steps to deploy unibox.

Deploy database

To setup the mysql database visit your phpMyAdmin (or similar tool) to create a new database and inject both sql scripts in the right order.

1. [Create Script](#)
2. [Insert Script](#)

Deploy WAR file

First of all you need the WAR File. After download you can copy the WAR file to you tomcat webapps folder.

```
%CATALINA_HOME%\webapps
```

IMPORTANT: rename "UniBoxServer.war" to "UniBox.war" !

After that, start your Tomcat e.g. run the Tomcat startup script.

```
%CATALINA_HOME%\bin\startup.bat
```

Your WAR file will automatically stripped to a folder that has the same name (without extension).

```
%CATALINA_HOME%\UniBoxServer\
```

Take a look into the tomcat configuration and take the port for the HTTP-Connector.

```
%CATALINA_HOME%\conf\server.xml
```

The default value is 8080.

```
<Connector port="8080" protocol="HTTP/1.1"
  connectionTimeout="20000"
  redirectPort="8443" />
```

Good job!

Access the following URL

```
http://localhost:port/UniBoxServer/
```

Initial Admin Credentials:

User:	Admin
Password:	user

Configuration

UniBox is providing two properties files to provide a simple way of configuration. You can find the relevant config files here:

```
%CATALINA_HOME%\webapps\UniBox\WEB-INF\config
```

Database Configuration

```
\server.properties
```

To connect the UniBoxServer with your MySQL Database, you got to configure these parameters depending on your MySQL Setup:

```
!!! Database Configuration !!!

# Server URL/IP
DB_SERVER=localhost

# Database Name
DB_NAME=unibox

# Driver and Protocol
DB_DRIVER=com.mysql.jdbc.Driver
DB_PROTOCOL=jdbc:mysql:

# Database Credentials
DB_USER=root
DB_PASSWORD=root
```

Just fill up with valid data and restart/start your tomcat instance.

Logging Configuration

There are several options to improve logging for the server application.

```
%CATALINA_HOME%\logs\UniBox.log
```

Process Logging Configuration

```
\log4j.properties
```

You got several options to configure the Log4j Listener:

```
log4j.logger.UniBoxLogger=DEBUG, C, fileappender

log4j.additivity.UniBoxLogger=false
log4j.appender.C=org.apache.log4j.ConsoleAppender
log4j.appender.C.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.C.layout.ConversionPattern=[%c] [%d{dd MMM yyyy - hh:mm:ss}] %5p - %m %n

log4j.appender.fileappender=org.apache.log4j.RollingFileAppender
log4j.appender.fileappender.File=${catalina.base}/logs/UniBox.log
log4j.appender.fileappender.MaxFileSize=500KB

log4j.appender.fileappender.MaxBackupIndex=3
log4j.appender.fileappender.layout=org.apache.log4j.PatternLayout
log4j.appender.fileappender.layout.ConversionPattern=%p %t %c - %m%n
```

UniBoxServer is using several [logging levels](#):

```
DEBUG
WARN
ERROR
INFO
```

Just edit the **%LEVEL%** entry in the configuration to get filtered/non-filtered loggings.

```
log4j.logger.UniBoxLogger=%LEVEL%, C, fileappender
```

[Visit the Manual](#) to get more information

Logging Level Configuration

```
\server.properties
```

To get a process based logging for e.g. debugging issues, you can activate logging for a special kind of functionality.

Maybe you just want to get a logging for all methods/actions aiming at authorization issues. Then you can simply e.g. set `LOG_AUTHENTICATION` to true. After restarting/starting the tomcat server you will get a more consistent logging related to this topic.

```
!!! Logging Configuration !!!

# Log Handling/Process of Async Sessions
LOG_ASYNC_SESSIONS=true

# Log Authentication Process
LOG_AUTHENTICATION=true

# Log Communication Process
LOG_COMMUNICATION=true

# Log Database Events
LOG_DATABASE=true

# Log Game relevant Tasks
LOG_GAMEPOOL=true

# Log Request Headers for each Request
LOG_REQUEST_HEADER=true

# Log Requested URI's and statistic values (e.g. response time)
LOG_REQUESTED_URI=true

# Log Thread Processes
LOG_THREADS=true
```

Client Guide

The UserBoxClient is providing a User API.

To get in touch with the service and to play your JavaFX game against another remote player you first have to get valid credentials by your administrator. After that you can login into the backend dashboard to change your default password.

The Dashboard is displaying all available game channels and the current result statistics. Furthermore you can create new game channels and chat with other remote players via the chat tab.

NOTE: The backend is fully responsible, so you can even use it with your smartphone or tablet as a simple but functional chat system.

Getting started with your JavaFX Application

After you set the UniBoxClient.jar as a needed Dependency of your project you are able to setup your needs.

First of all you got to set your Username and Password together with the IP of the UniBoxServer Instance. To be flexible the API will provide different methods for that task to fit your needs.

You can pass your credentials as parameter like

```
java -jar YourGame.jar <ip> <user> <password>
```

If you prefer that way, simple add the following calls to your main():

```
public static void main(final String[] args) {  
  
    // init credentials  
    ClientProvider.setIp(args[0]);  
    ClientProvider.setUsername(args[1]);  
    ClientProvider.setPassword(args[2]);  
  
    Application.launch(args);  
  
}
```

If you like a more static way you can place these methods in your start() method as well:

```
public final void start(final Stage primaryStage) {  
  
    // init credentials  
  
    ClientProvider.setIp("192.168.0.150");  
    ClientProvider.setUsername("user");  
    ClientProvider.setPassword("password");  
  
    // ...  
  
}
```

As the credentials and the IP were set. You can call the login routine to register your Application to the backend service.

```
ClientProvider.login();
```

Now you got a valid cookie to talk on an authorized level to the backend workers. To start the main loop of the Client API you got to connect the local workers/threads of your Application with the Backend.

```
ClientProvider.connect();
```

Sending messages

Now you should be able to send different Types of Messages. Basically you just need the `sendGameMessage()` method to send your game states, changes or events.

```
ClientProvider.sendChatMessage(<STRING>);  
ClientProvider.sendCustomMessage(<STRING>);  
ClientProvider.sendErrorMessage(<STRING>);  
ClientProvider.sendGameMessage(<STRING>);  
ClientProvider.sendSystemMessage(<STRING>);
```

NOTE: You got to implement your own protocol. UniBoxClient will not parse or cast any message content.

Receive Messages

To receive messages send by the backend you got to bind a custom handler to your main stage. The API is providing an abstract class that implements a `EventHandler` for incoming messages. To get it working just copy paste this code to your derived `JavaFX Application Class`.

```
/**
 * UniBoxClient: bind event handler for incoming messages.
 */
ClientProvider.bind(primaryStage, new IncomingMessageHandler() {

    /**
     * (non-Javadoc)
     *
     * @see
     * de.unibox.client.api.IncomingMessageHandler#handle(java.lang.
     * String, java.lang.String)
     */
    @Override
    public void handle(final String user, final String msg) {

        // handle incoming messages here on <MainThread>

    }
});
```

All incoming messages will be published within the `handle()` method, while "user" contains the sender name and "msg" the concrete payload e.g. the message string.

Handle Highscores

Last but not least you are able to manipulate the highscore list.

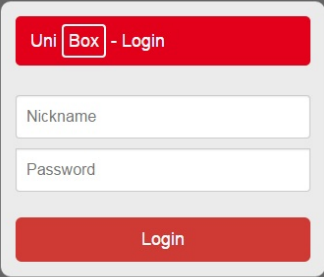
NOTE: You are not able to commit a result without being part of a game (you must join a game before).

```
ClientProvider.reportWinResult(); // +1 point  
ClientProvider.reportDrawResult(); // +/- 0 points  
ClientProvider.reportLoseResult(); // -1 point
```

User Guide

If you have installed the UniBoxServer like described before you are able to request the login mask via <http://server:port/UniBox>.

Log in using your login credentials:

A screenshot of a login interface. It features a dark gray background. In the center, there is a white rectangular box with rounded corners. Inside this box, at the top, is a red header bar with the text "Uni Box - Login" in white. Below the header, there are two white input fields: the first is labeled "Nickname" and the second is labeled "Password". At the bottom of the white box is a red button with the text "Login" in white.

If you can see the Dashboard. **Good job!**

Uni
Box

Hey, user1
Dashboard
Chat
Options
Logout

UniBox - InfoBox online..

Create new Game

Game Panel
Filter..

	ID	Game	Name	Places	Joined
Join	2	Vier Gewinnt	Susis Spiel	0/2	none
Join	1	Tik Tak To	Peters Spiel	0/2	none
Join	3	Tik Tak To	Franks Spiel	0/2	none
Join	4	Snake	Danas Spiel	0/1	none
	ID	Game	Name	Places	Joined

4 of 4
Previous
Next

Ranking Panel
Filter..

Rank	Player	Score
1	user4	4
2	user3	3
3	user2	2
4	user1	1
5	user5	1
Rank	Player	Score

5 of 5
Previous
Next

Usage

On the dashboard view you can create, join or leave games via the *Game Panel*. You can see actual rankings stats with the *Ranking Panel*. Each Panel got input and column filters to find a certain entity fast, even in pagination mode.

First of all you should change your password. There is a options pane in the right top corner:

The dashboard features a top navigation bar with the UniBox logo, user status 'Hey, user1', and links to 'Dashboard', 'Chat', 'Options', and 'Logout'. A dropdown menu under 'Options' includes 'Reconnect Chat', 'Reload Page', and 'Change password'. Below the navigation bar is a status bar 'UniBox - InfoBox online..'. The main content area contains two panels: 'Game Panel' and 'Ranking Panel'. The 'Game Panel' table lists games with columns for ID, Game, Name, Places, and Joined. The 'Ranking Panel' table lists players with columns for Rank, Player, and Score. Both panels include a filter input and pagination controls.

ID	Game	Name	Places	Joined
2	Vier Gewinnt	Susis Spiel	0/2	none
1	Tik Tak To	Peters Spiel	0/2	none
3	Tik Tak To	Franks Spiel	0/2	none
4	Snake	Danas Spiel	0/1	none

Rank	Player	Score
1	user4	4
2	user3	3
3	user2	2
4	user1	1
5	user5	1

Hit *change password* to open the formular:

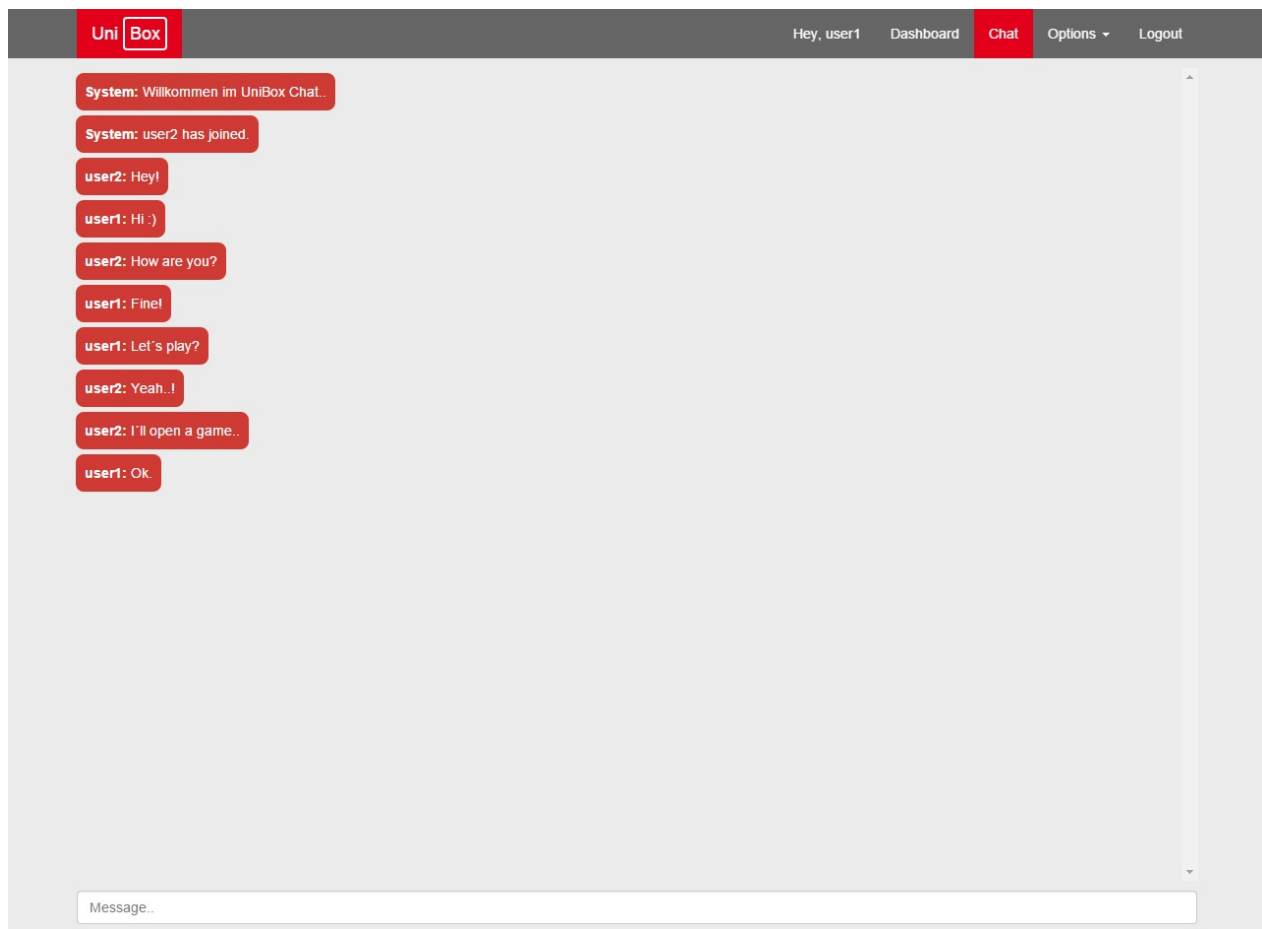
The 'Change password' modal is displayed over the dashboard. It contains three input fields: 'Current Password', 'New Password', and 'Confirm Password'. The 'New Password' field has a note 'Minimum of 6 characters'. At the bottom of the modal are 'Cancel' and 'Commit' buttons. The background dashboard content is dimmed.

ID	Game	Name	Places	Joined
2	Vier Gewinnt	Susis Spiel	0/2	none
1	Tik Tak To	Peters Spiel	0/2	none
3	Tik Tak To	Franks Spiel	0/2	none
4	Snake	Danas Spiel	0/1	none

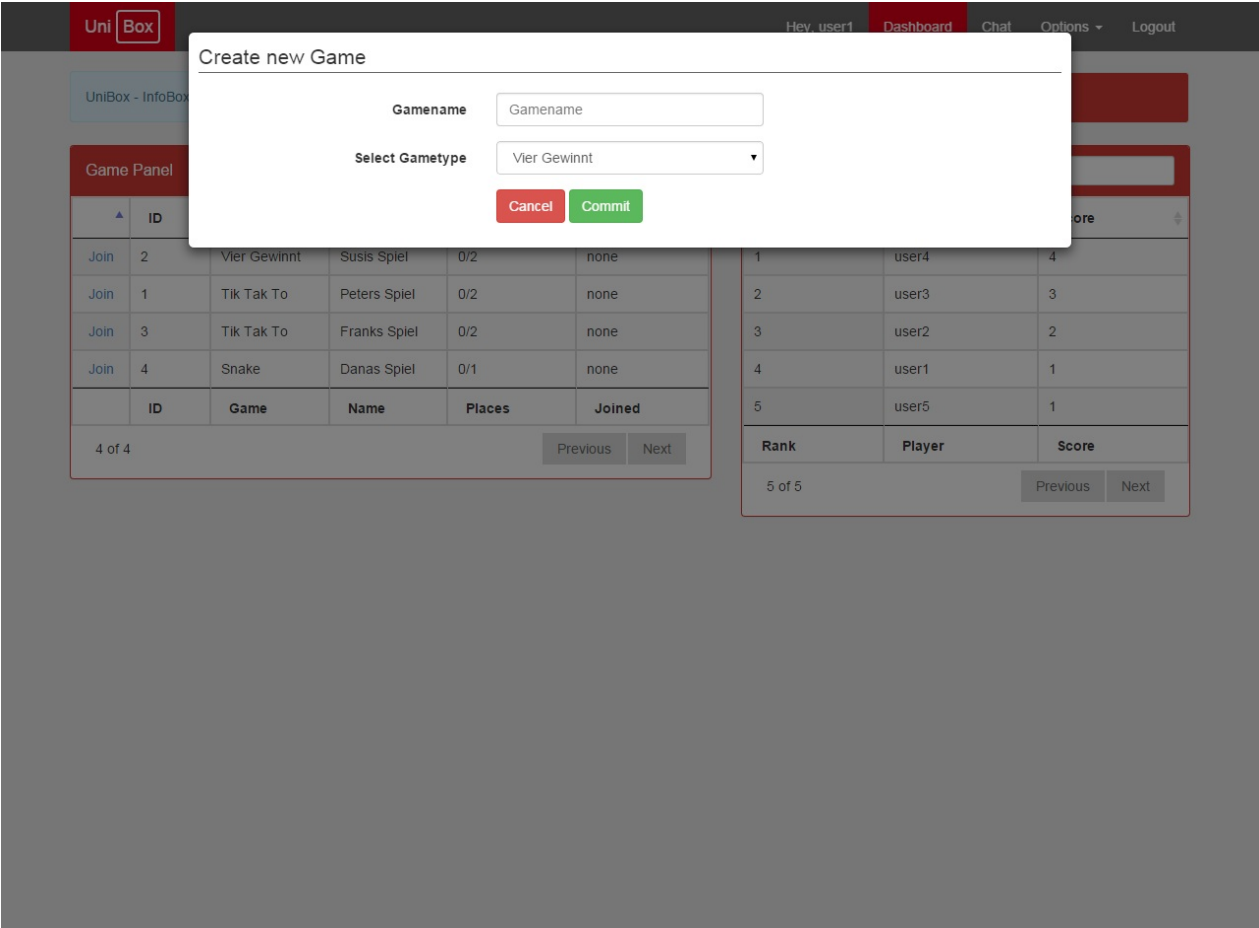
Rank	Player	Score
1	user4	4
2	user3	3
3	user2	2
4	user1	1
5	user5	1

Fill out the formular and hit the *Commit* Button to confirm your request.

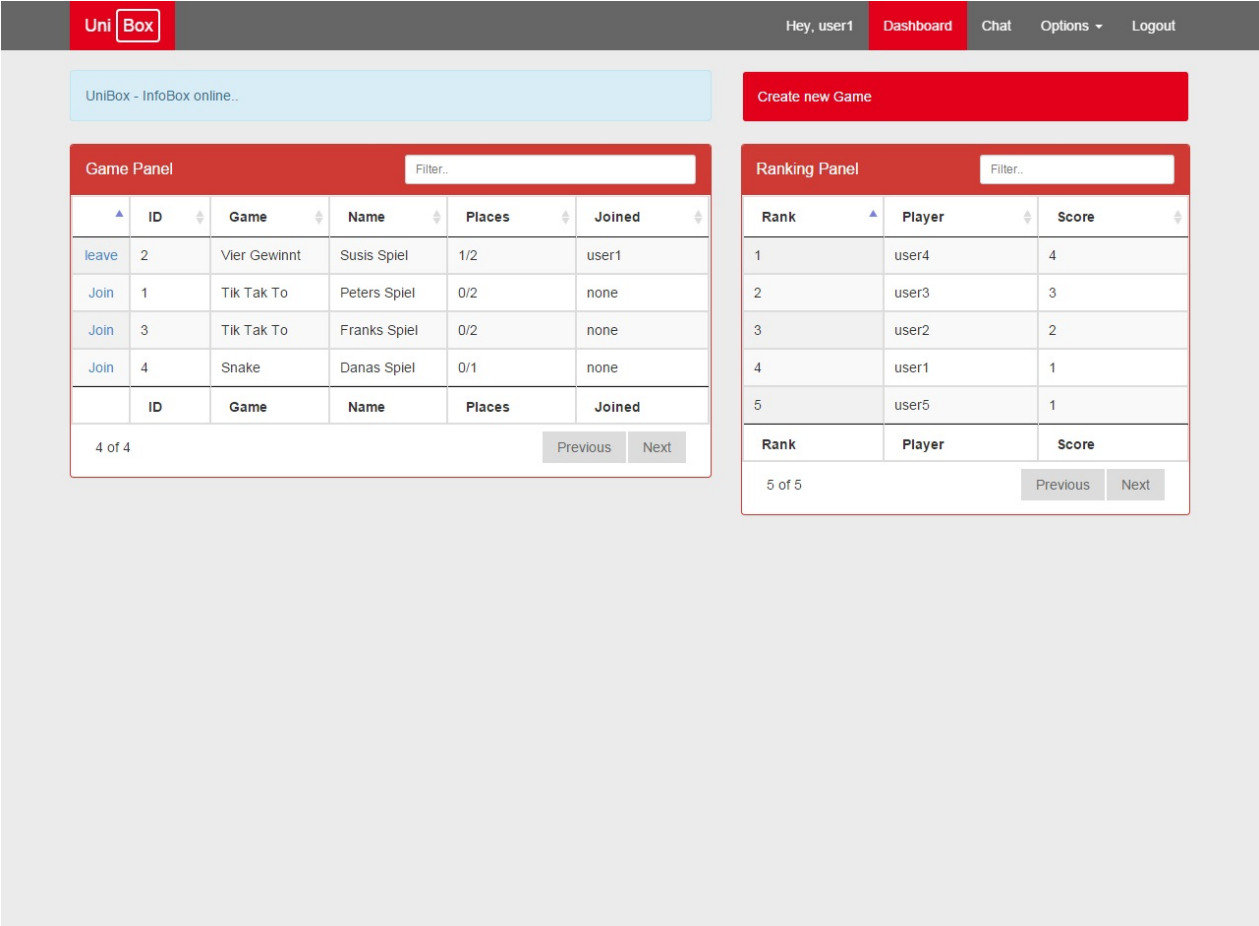
After that you can give the chat function a try:



You can create a new game:



Or join an existing one:



Administration

Default Administrator Credentials:

Username	Password
Admin	user

As you are logged in with an Admin Account you got an extra option tab in the right top corner:

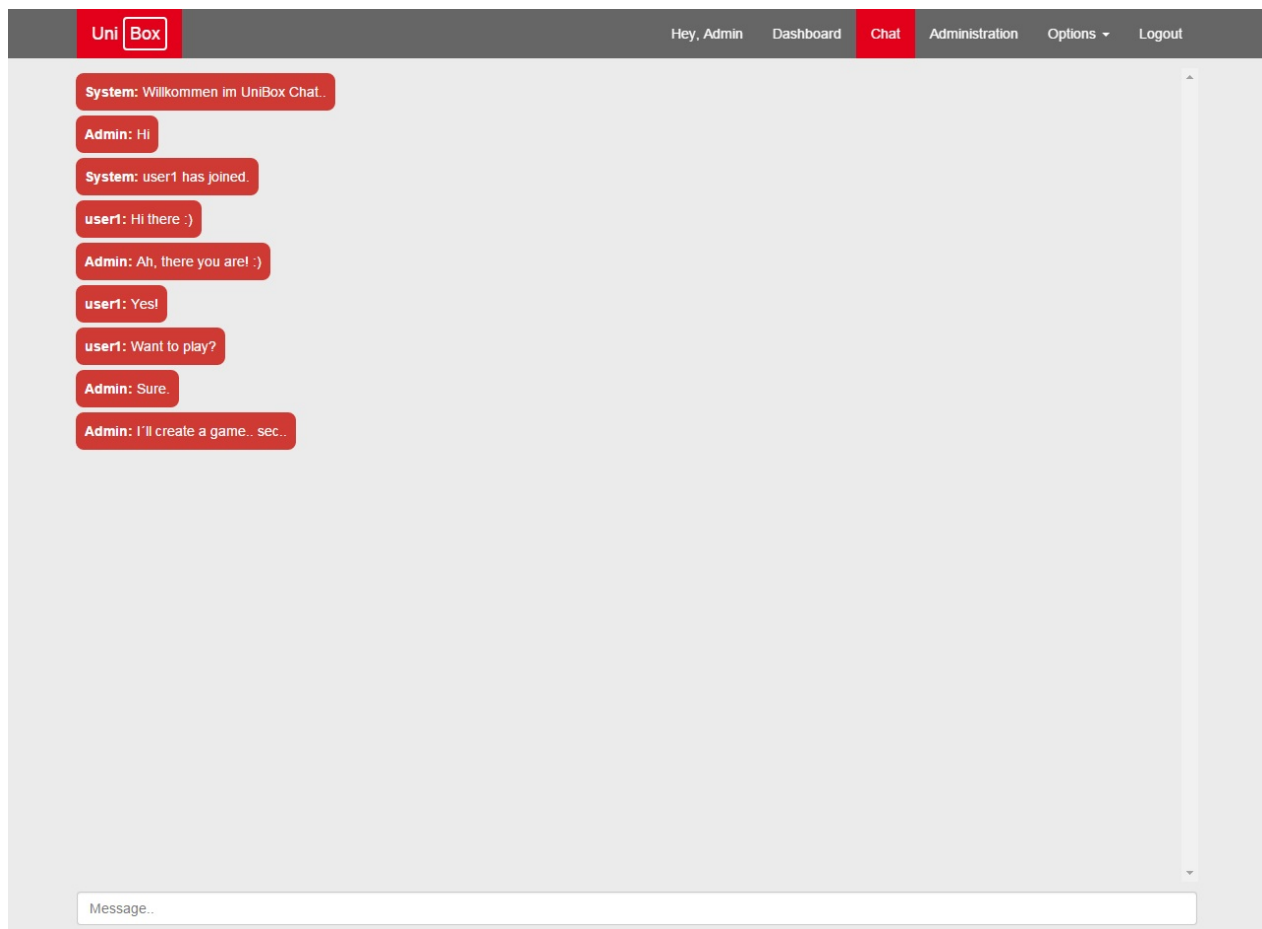
- Dashboard
- Chat
- **Administration**
- Options
- Logout

The screenshot shows the UniBox Administration Dashboard. At the top, there is a navigation bar with the UniBox logo, the user 'Hey, Admin', and tabs for 'Dashboard', 'Chat', 'Administration' (which is active), 'Options', and 'Logout'. Below the navigation bar, there is a light blue box with the text 'UniBox - InfoBox online...'. To the right of this box is a red button labeled 'Create new Game'. The main content area is divided into two panels. The left panel, titled 'Game Panel', contains a table with columns: ID, Game, Name, Places, and Joined. It lists four games: 'Vier Gewinnt', 'Tik Tak To', 'Tik Tak To', and 'Snake'. Each game has a 'Join' link. The right panel, titled 'Ranking Panel', contains a table with columns: Rank, Player, and Score. It lists five players: 'user4', 'user3', 'user2', 'user1', and 'user5'. Both panels have a 'Filter...' input field at the top and pagination controls at the bottom.

ID	Game	Name	Places	Joined
2	Vier Gewinnt	Susis Spiel	0/2	none
1	Tik Tak To	Peters Spiel	0/2	none
3	Tik Tak To	Franks Spiel	0/2	none
4	Snake	Danas Spiel	0/1	none

Rank	Player	Score
1	user4	4
2	user3	3
3	user2	2
4	user1	1
5	user5	1

Admin's can chat as well:



As you click on the *Administration* Tab in the right top corner:

Uni Box
Hey, Admin
Dashboard
Chat
Administration
Options
Logout

Admin Dashboard

Create User

Username..

Default password: user

☐ grant admin privileges

Create

Delete User

None selected

..be patient deleting Admin Users! "Reset Database" is able to restore the default Admin Account!

Delete

Delete Game

None selected

..Scores will remain.

Delete

Reset Scores

Reset

..reset result table.

Reset Database

Reset

..drop all tables and create new ones with default values defined in WEB-INF/database.

WARNING: All clients except you will be automatically disconnected as consequence of invalid credentials and gamestates.

You are able to:

- Create Users (even with admin-rights)
- Delete (multiple) Users
- Delete (multiple) Games
- Reset Highscores (global)
- Reset Database (global)

Note: Please read the instructions displayed on that page carefully. They have direct and maybe heavy impacts on the functionality of unibox.

Requests

Root: `http://server:8080/UniBox`

All requests are defined beyond the static route of your tomcat instance. Basically we differ between **public**, **registered** and **administrative** routes. Public routes can be called by any client. Registered routes require a session and a bound user object which is attached during the authentication process and accessible through a valid cookie generated after confirmed credentials.

Asynchron Requests

Plain Comet Listener

```
Type:      GET
URL:       /Communicator
Parameter: -
Output:    text/html
Realm:     public
```

This request delivers a long polling http response to receive **plain** messages.

JavaScript Comet Listener

```
Type:      GET
URL:       /Communicator/JavaScript
Parameter: -
Output:    text/html
Realm:     public
```

This request delivers a long polling http response to receive **javascript** messages. **GAME Message will not delivered via this route.**

Serial Comet Listener

```
Type:      GET
URL:       /Communicator/Serial
Parameter: -
Output:    text/html
Realm:     registered
```

This request delivers a long polling http response to receive **serialized object** messages. **CHAT Messages will not delivered via this route.**

Synchron Requests

Login Mask

```
Type:      GET|POST
URL:       /Login
Parameter: error=[STRING]
ContentType: text/html
Realm:     public
```

This Route is forward handle to /login.html which is a static page serving the login formular. If anything went really wrong while using the webservice you will be redirected to this screen and a error will be attached as a value for the error parameter.

Request Connection Cookie

```
Type:      POST
URL:       /Communicator
           /JavaScript
           /Serial
Parameter: action=connect
           nick=[STRING]
           password[BASE64_STRING]
Output:    text/html
Realm:     registered
```

This request is able to promote a new user as a SYSTEM Message (e.g. "User XY joined.."). No matter which of these three available urls you call to promote your login, the result will be the same. Basically this method can be used to retrieve a cookie without visiting the login mask.

NOTE: All registered requests are able to authorize you if the request contains nick and password parameters. But in case of the performance issue it is much better to store the received cookie after the first request you do to get instant access without claiming the authorize routine each time again and again.

Sending Messages

The UniBox Server is implementing different type of messages.

```
package de.unibox.core.network.object;

/**
 * The Enum MessageType.
 */
public enum CommunicatorMessageType {

    /** The chat. */
    CHAT,

    /** The error. */
    ERROR,

    /** The game. */
    GAME,

    /** The js command. */
```

```

    JS_COMMAND,

    /** The ping. */
    PING,

    /** The system. */
    SYSTEM,

    /** The plain. */
    PLAIN
}

```

Each message type has it's own prediction. The following routes are able to deliver different kinds of messages around the registered clients. Please read carefully if you want to deliver messages beside the default messaging functions derived by the Client API or overwrite the default functionality of the frontend javascript app.

Plain Message

```

Type:      POST
URL:       /Communicator
Parameter: action=push
           message=[STRING]
Output:    text/html
Realm:     registered

```

This route can be used to push plain messages. The concrete message content will be specified within the message parameter value. If anything went wrong you will get a 422 Http-State. Plain Messages are currently forwarded to all registered clients. Recommended usage would be a base64 encoded message string even if each client has to encode the string itself.

NOTE: Right now neither the Client API nor the Dashboard is using PLAIN Messages. It is currently just a kind of generic message which is basically not needed in the UniBox Lifecycle.

Javascript Message

```

Type:      POST
URL:       /Communicator/Javascript
Parameter: action=push
           message=[STRING]
Output:    text/html
Realm:     registered

```

This route is used to push a new javascript message. The concrete message content will be specified within the message parameter value. If anything went wrong you will get a 422 Http-State. Javascript Messages are forwarded as CHAT Messages. These Messages will be broadcasted to all Dashboard Clients immediatly.

Serial Message

```

Type:      POST
URL:       /Communicator/Serial
Parameter: action=push
           message=[STRING]
Output:    text/html
Realm:     registered

```

This route is used to push serial message. The concrete message content will be specified within the message parameter value. The whole Communication Object will get serialized and transmitted. If anything went wrong you will get a 422 Http-

State. Serial Messages can be any [type of message](#) and will be forwarded to all clients connected through the UniBox API.

NOTE: Serial Messages will NOT be forwarded to any Dashboard Client.

Request the Dashboard

```
Type:      GET|POST
URL:       /Dashboard
Parameter: -
ContentType: text/html
Realm:     registered
```

This request is basically a request dispatcher to the full responsive dashboard implementation. The dashboard will work fine with almost any browser on almost any device.

Get user list

```
Type:      GET
URL:       /Database
Parameter: action=getUsers
Output:    application/json
Realm:     registered
```

This request will retrieve the list of registered users as a JSON object.

Get game list

```
Type:      GET
URL:       /Database
Parameter: action=getGames
Output:    application/json
Realm:     registered
```

This request is serving the list of registered games. Including position wiring and Usernames of joined players. The response is a JSON object.

Get ranking list

```
Type:      GET
URL:       /Database
Parameter: action=getRanking
Output:    application/json
Realm:     registered
```

This route is serving the list of registered ranking-points. The response is a JSON object.

Get category list

```
Type:      GET
URL:       /Database
Parameter: action=getCategories
Output:    application/json
Realm:     registered
```

This route is serving the list of registered categories. The response is a JSON object.

Create game

```
Type:      POST
URL:       /Database
Parameter: action=createGame
           gameId=[INTEGER]
           catID=[INTEGER]
Output:    text/html
Realm:     registered
```

This route can be used to create a new game entry. The name of this game can be choosen and the game category e.g. catid has to be set to relate the game to a concrete game category. This call will force all remote clients to refetch the their game list.

Create score

```
Type:      POST
URL:       /Database
Parameter: action=createResult
           status=[win|draw|lose]
Output:    text/html
Realm:     registered
```

This route can be used to create a new result entry for a finished game. To get it done, you have to set the gameid, the corresponding playerid and the scoring, which yill be set in the database. This call will force all remote clients to refetch the their ranking list.

Join game

```
Type:      GET|POST
URL:       /Game
Parameter: action=joinGame
           gameId=[INTEGER]
Output:    text/html
Realm:     registered
```

This route can be used to join a existing game by the related game id. If the user is already joined to a game, the backend will automatic leave the previous game to let the user join to this one.

Leave game

```
Type:      GET|POST
URL:       /Game
Parameter: action=leaveGame
Output:    text/html
Realm:     registered
```

This route can be used to leave a game in which the user is currently joined.

Which game

```
Type:      GET|POST
```

```

URL:           /Game
Parameter:     action=whichGame
Output:        text/html
Realm:         registered

```

This route will return the gameId of the currently joined game, depending on the connected user session.

Change user password

```

Type:          POST
URL:           /Auth
Parameter:     action=changePassword
               oldPassword=[STRING]
               inputPassword=[STRING]
               inputPasswordConfirm=[STRING]
Output:        text/html
Realm:         registered

```

This request can be triggered to change the user password.

Logout user

```

Type:          GET
URL:           /Auth
Parameter:     action=logout
Output:        text/html
Realm:         registered

```

This request can be used by any client to invalidate the corresponding session. The callback will be a redirection to the public login formular.

Administrative requests

Create category

```

Type:          POST
URL:           /Admin
Parameter:     action=createCategory
               gameTitle=[STRING]
               numberOfPlayers=[INTEGER]
Output:        text/html
Realm:         administrative

```

This request can be used to create a new game category. The name of this category can be choosen and the maximal count of players able to play together in one game instance has to be set.

Create user

```

Type:          POST
URL:           /Admin
Parameter:     action=createUser
               name=[STRING]
               adminRights[0|1]
Output:        text/html
Realm:         administrative

```

This request can be used to create a new user. To grant admin privileges to this user the adminrights parameter can be set to true (1) or false (0). The default password will be 'user'.

Delete user

```
Type:      GET
URL:       /Admin
Parameter: action=deleteUser
           userId=[INTEGER]
Output:    text/html
Realm:     administrative
```

TODO

Delete game

```
Type:      GET
URL:       /Admin
Parameter: action=deleteGame
           gameId=[INTEGER]
Output:    text/html
Realm:     administrative
```

TODO

Reset scores

```
Type:      GET
URL:       /Admin
Parameter: action=resetScores
Output:    text/html
Realm:     administrative
```

TODO

Reset database

```
Type:      GET
URL:       /Admin
Parameter: action=resetDatabase
Output:    text/html
Realm:     administrative
```

TODO

Hints

The API is providing a implementation of the native object serialization routine:

```
ObjectSerializerImpl.objectToString(final Serializable object);

ObjectSerializerImpl.stringToObject(final String string, final Class<E> clazz);
```

With these methods you are able to send objects directly:

```
AnyObject any = new AnyObject();

String serializedObject = ObjectSerializerImpl.objectToString(any);

ClientProvider.sendGameMessage(serializedObject);
```

Remember that the receiver has to know which object is arriving. So you need the Class of the serialized object on both sides (sender and receiver).

If the receiver wants to pick the serialized object, just do this:

```
/**
 * UniBoxClient: bind event handler for incoming messages.
 */
ClientProvider.bind(primaryStage, new IncomingMessageHandler() {

    @Override
    public void handle(final String user, final String msg) {

        AnyObject any = ObjectSerializerImpl.stringToObject(msg, AnyClass.class);

        // any is an exact copy of the object send by the sender

    }
});
```

Just to food your thought. You can even capsule objects:

```
ArrayList<AnyObject> list = new ArrayList<AnyObject>();
list.add(obj1);
list.add(obj2);

String serializedObject = ObjectSerializerImpl.objectToString(list);

ClientProvider.sendGameMessage(serializedObject);
```

Bugs

- In *Internet Explorer 11* the Chat-Function is not working until the app is recieving the second/third incoming message.