Distributed Systems
Chat Manual

Zhong-Xi Lu & Thomas Van Bogaert

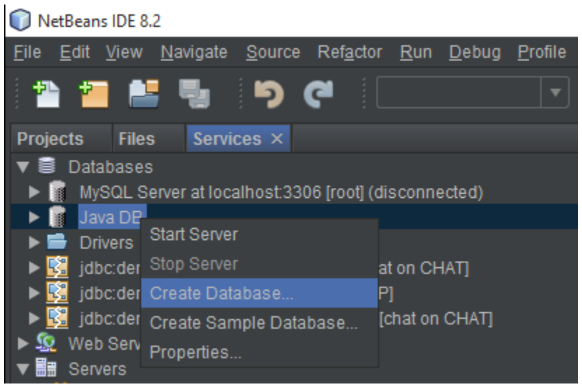
1 Used Technologies

- Java EE
- HTML, CSS (Bootstrap), JS, AJAX
- Servlets, Beans, Entity Classes
- GlassFish Server
- JavaDB
- socat
- Netbeans

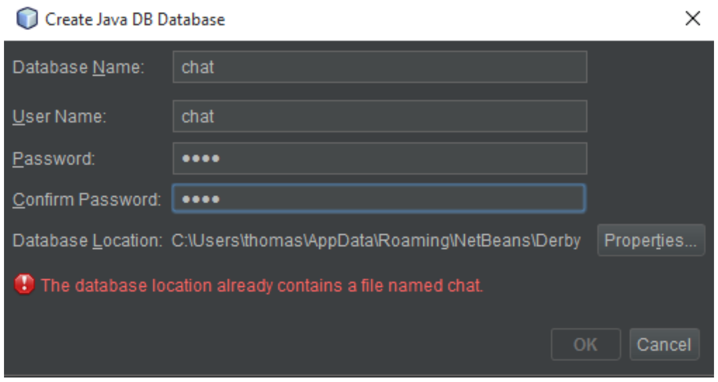
2 Set up

2.1 Set up database server

1. Open Netbeans
2. Open the Services tab
3. Expand 'Databases'
4. Right click 'Java DB' and select 'Create Database...'


NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile
Projects Files Services
Databases
MySQL Server at localhost:3306 [root] (disconnected)
Java DB
Drivers
Web Serv
Servers
Start Server
Stop Server
Create Database...
Create Sample Database...
Properties...

5. Enter the following:



Database Name: chat

User Name: chat

Password: ••••

Confirm Password: ••••

Database Location: C:\Users\thomas\AppData\Roaming\NetBeans\Derby

The database location already contains a file named chat.

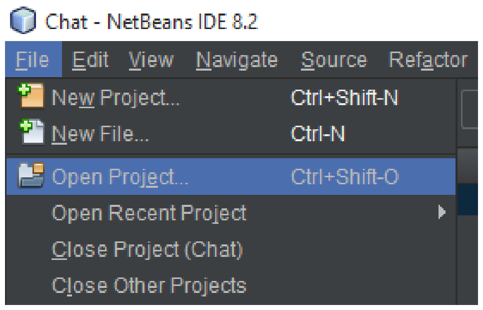
With the password being "chat" and click 'OK'

6. Open a terminal and cd to `'groep_8/socat'`
7. Set execute permissions on `socat/build_socat` (`chmod +x build_socat.sh`)
8. Run `./build_socat.sh` (socat will run after it is compiled. socat needs to be running to access the database from a remote server.)

2.2 Set up client servers

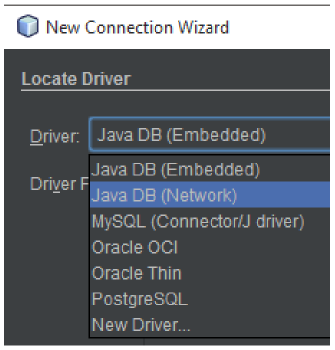
2.2.1 For each server

1. Open Netbeans
2. Open the Chat project by selecting 'File' → 'Open Project...'

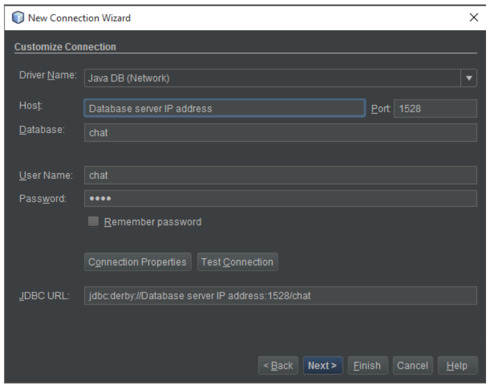


then find the location of the Chat project (`groep_8/Chat`)

3. Open the Services tab
4. Right click 'Databases' and select 'New Connection...'
5. Select the 'Java DB (Network)' driver and click 'Next'



6. Enter the following:

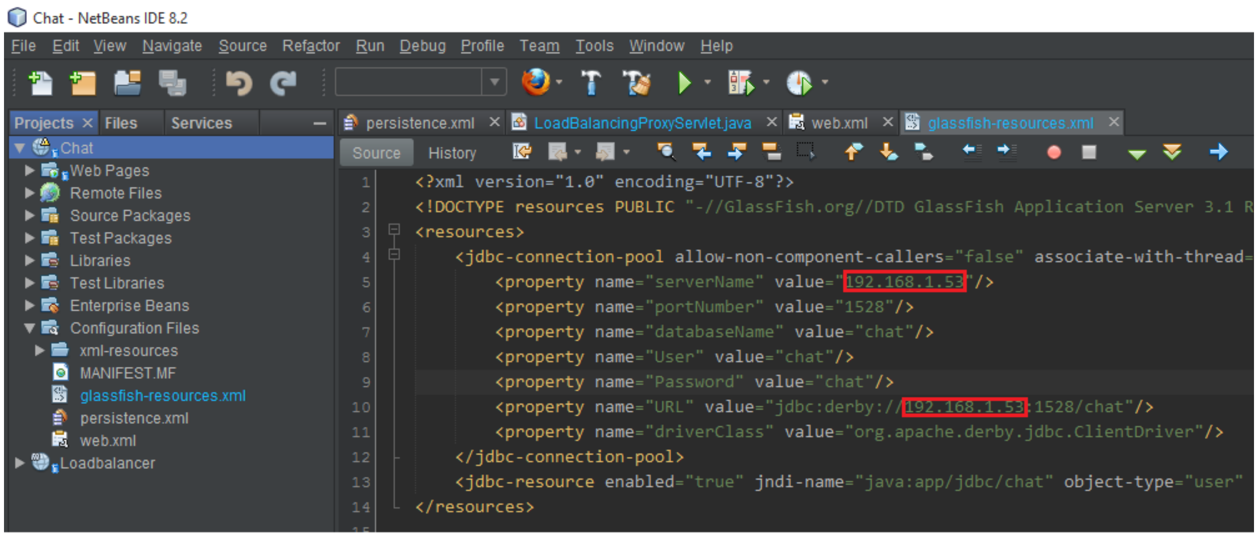


Driver Name: Java DB (Network)
Host: Database server IP address Port: 1528
Database: chat
User Name: chat
Password: ••••
Remember password

JDBC URL: jdbc:derby://Database server IP address:1528/chat

With 'Database server IP address' the IP address of the server on which the database is running and as Password 'chat'. Then click 'Finish'

7. Open the Projects tab
8. Expand 'Chat'
9. Expand 'Configuration Files'
10. Open `glassfish-resources.xml`
11. Enter the IP address of the database server in the following locations in `glassfish-resources.xml`:

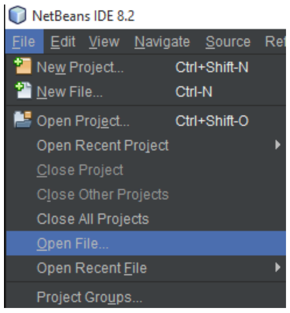


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 R
<resources>
    <jdbc-connection-pool allow-non-component-callers="false" associate-with-thread=
        <property name="serverName" value="192.168.1.53"/>
        <property name="portNumber" value="1528"/>
        <property name="databaseName" value="chat"/>
        <property name="User" value="chat"/>
        <property name="Password" value="chat"/>
        <property name="URL" value="jdbc:derby://192.168.1.53:1528/chat"/>
        <property name="driverClass" value="org.apache.derby.jdbc.ClientDriver"/>
    </jdbc-connection-pool>
    <jdbc-resource enabled="true" jndi-name="java:app/jdbc/chat" object-type="user"
</resources>
```

12. Right click on the 'Chat' project and click 'Deploy'

2.2.2 On only one client server

1. Open the `create_tables.sql` file by selecting 'File' → 'Open Project...'



   then open the location of the `create_tables.sql` file (`groep_8/db/create_tables.sql`)
2. Right click in the file and select 'Run File'.
3. Select the newly created database connection and click 'OK'.

2.3 Set up load balancer

1. Copy the maven repository to the right location:

   (a) ∼: `mkdir .m2`

   (b) ∼: `cp -a groep_8/maven_repo/. ∼/.m2`

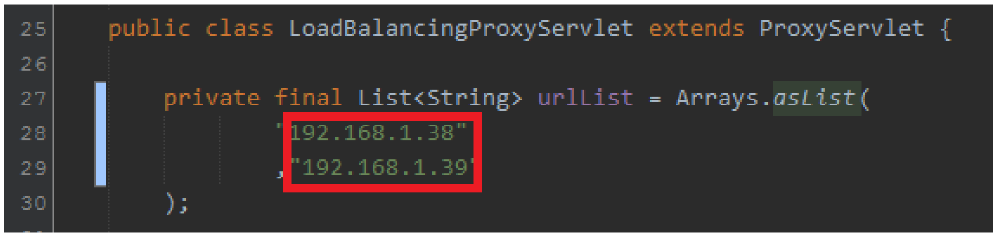
2. Open the Loadbalancer project in Netbeans

3. Open the Projects tab

4. Expand 'Loadbalancer' → 'Source Packages' → 'Servlets'

5. Open the 'LoadBalancingProxyServlet.java' file

6. Change the IP addresses in the 'LoadBalancingProxyServlet.java' file to the IP addresses of the servers running the Chat application



```
public class LoadBalancingProxyServlet extends ProxyServlet {

    private final List<String> urlList = Arrays.asList(
            "192.168.1.38"
            ,"192.168.1.39"
    );
```

7. Deploy the Loadbalancer the same way as we did the Chat applications.

If everything went right, the load balanced Chat application is accessible at `{IP address of load balancer server}:8080/Loadbalancer`

3 Architecture

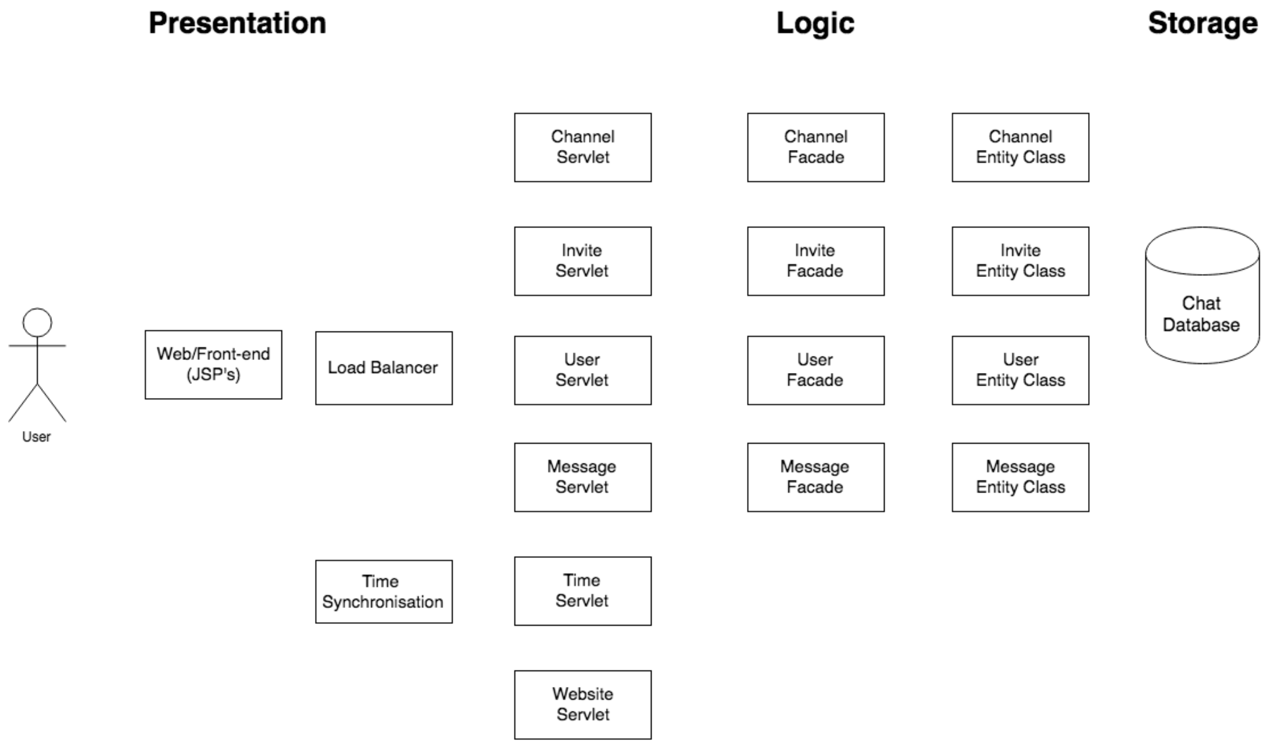


Figure 1: Chat Login Page

This is our service oriented architecture. In the front-end, we have jsp's that makes requests to the server through the load balancer. The load balancer forwards this request to the appropiate server/servlet. Each servlet then makes calls to the facades that is responsible for all the business logic. If a change to the database needs to be done, a facade can rely on an interface of the database (entity classes).

4 Design Choices

As previously mentioned, we tried to make a layered architecture (service-oriented). This way, each layer has one responsibility; servlets are responsible for handling requests, facades for doing all the logic, entity classes for the communication with the database. On the front-end, we have jsp's that can easily process all the data received from the server. Furthermore, we make ajax calls to the server, because it's reasonably fast and you can easily manipulate the responses and integrate it in your jsp's. Above that, those call are asynchronous, which makes it a lot easier to send different requests at roughly the same time.

5 Chat Tutorial

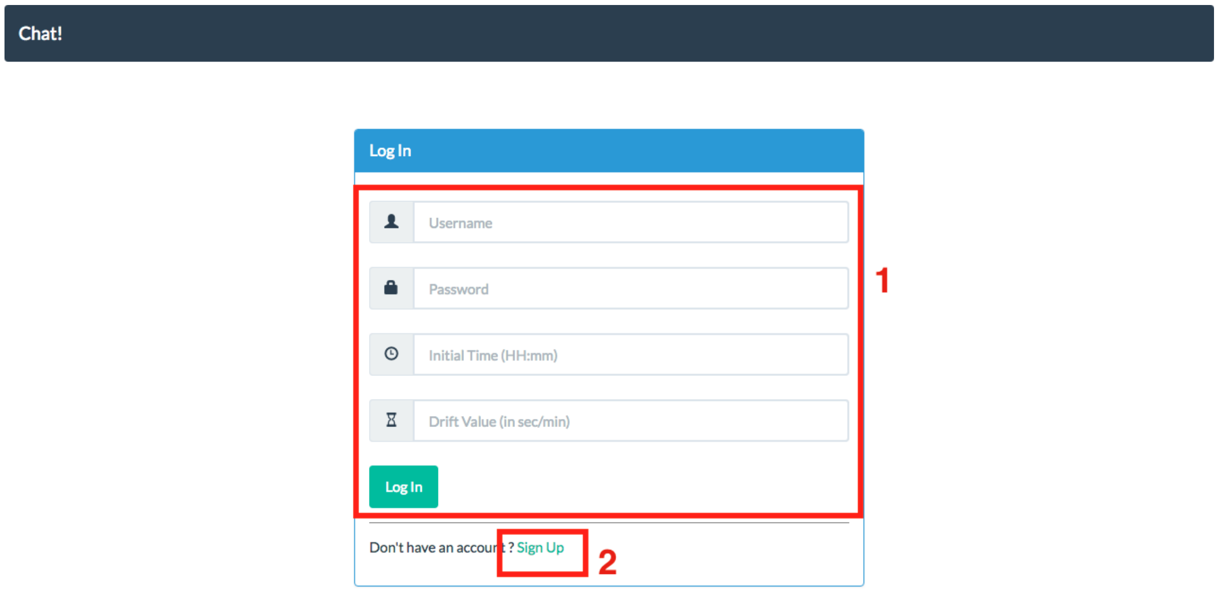
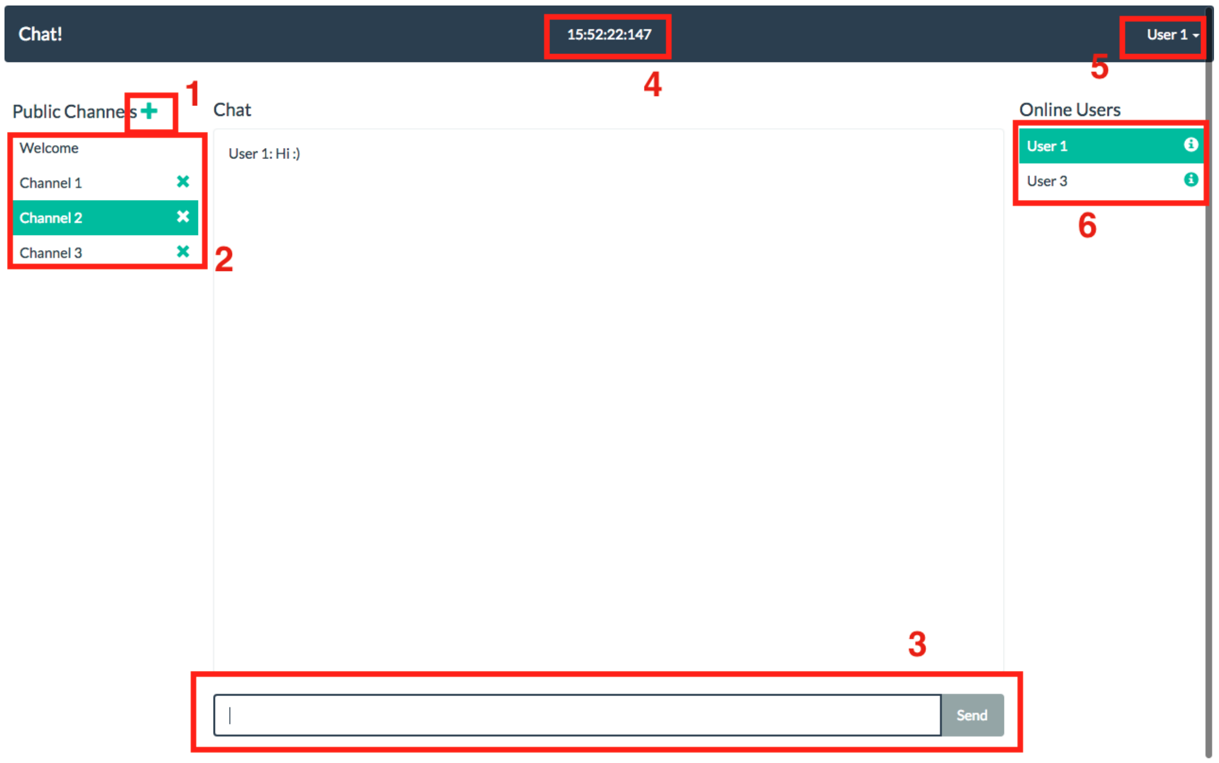


Figure 2: Chat Login Page

If a user browses to our chat application the page above will be shown.

1. A user can login by filling in the necessary information. Note that an initial time (HH:mm) and drift value must be specified.
2. Or, if the user doesn't have an account yet, he/she can create one here.



User 1: Hi :)

Figure 3: Chat Homepage

The figure above shows the main screen when the user has logged in or registered. The user can do the following:

1. If the user is a moderator, a new public channel can be added by clicking the plus-sign. After this, a window will pop up where a name for the channel must be specified.
2. Here, a list of all the active public channel are shown (green indicates the channel of the user); the user can simply join a channel by clicking on it. If the user is a moderator, the channel can be removed as well by clicking the X-sign next to the channel name.
3. This is the input field to send a message to the current channel of the user. The message will then appear in the chat box displayed above.
4. The current time for this specific user is shown here and is synchronised every 10 seconds with the server.
5. A user can log out by clicking on his username and selecting the "Log out" option.
6. All the current online users are shown here (green indicates the name of the user and red indicates a moderator). By clicking on a user, you can invite them to a private

channel (a channel name must be given as well), the other user can then accept or decline the invite. In addition, a moderator can also view all the messages a user sent (in all public/private channels) by clicking the info-sign next to the name. This will one a new page with the history of that user.