# 

# EE4023 Distributed Systems

**Tic Tac Toe Assignment**

**Group: Team Serbonne**

James Shally – 16153774

Cathal Daly – 16208102

Patrick O’Hora – 16179314

Contents

[Brief 3](#_Toc25745320)

[Roles 3](#_Toc25745321)

[PHP Interface 4](#_Toc25745322)

[Project Structure 4](#_Toc25745323)

[PHP 5](#_Toc25745324)

[The Soap Client 5](#_Toc25745325)

[Displays 6](#_Toc25745326)

[JavaScript 7](#_Toc25745327)

[Ajax 7](#_Toc25745328)

[CSS and User Interface 8](#_Toc25745329)

[Java 11](#_Toc25745330)

[Project Structure 11](#_Toc25745331)

[Restful Web Client 12](#_Toc25745332)

[JFrame & ActionListener 12](#_Toc25745333)

[Threading 14](#_Toc25745334)

[Screens 15](#_Toc25745335)

[Testing and Conclusion 16](#_Toc25745336)

# Brief

Our brief was to create versions of Tic Tac Toe, one PHP and one Java, and both would communicate over the same soap client. This would demonstrate how two implementations can communicate effectively without knowing the database schema and only requiring one webservice that can be accessed by both.

# Roles

Due to Patrick's strong Java skills, we decided to split the work accordingly. James and Cathal both have experience using PHP and JS over placement, so the split made sense. Once we each completed our work we went through both implementations and discussed how and why it all worked. We had team meetings in the early stages of the project in order to set goals for the coming weeks. By having a plan, it made it easier to manage the workload and ensure that we implemented all of the functionality before the deadline.

|  |  |  |
| --- | --- | --- |
|  | PHP | Java |
| Interfaces | James and Cathal | Patrick |
| UI | James and Cathal | Patrick |
| Report | James and Cathal | Patrick |

# PHP Interface

## Project Structure

When developing our PHP Interface for Tic Tac Toe the team decided very early on in development, we would aim to have many components that worked as a cohesive unit rather than a few large components. This would help us extend and understand our code throughout the development process. We organised our structure in the following way:

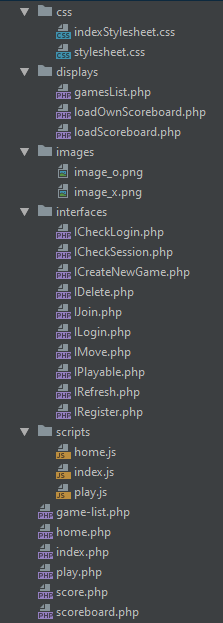


Figure 1: PHP Interface structure

As shown, we kept our Interfaces, scripts, displays, CSS, and images in their own directories. This allowed us to immediately jump to the correct file location during development and made understanding our code easier.

## PHP

### The Soap Client

As shown in Figure 1, we kept our PHP files in three mains areas. The PHP in the root of the directory contained mostly HTML and JS, so we will go into little detail about those for now. Let’s look at the ICheckSession.php interface:

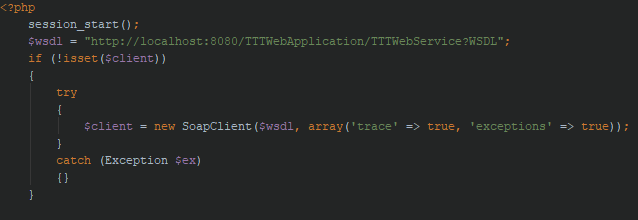


Figure 2: ICheckSession.php

ICheckSession.php starts a session for the user, followed by connecting to the soap client with a variable called $client. This variable will be used in almost all of the interfaces when required to get data from the soap client.

Now let’s look at an interface that uses $client, IRegister.php:

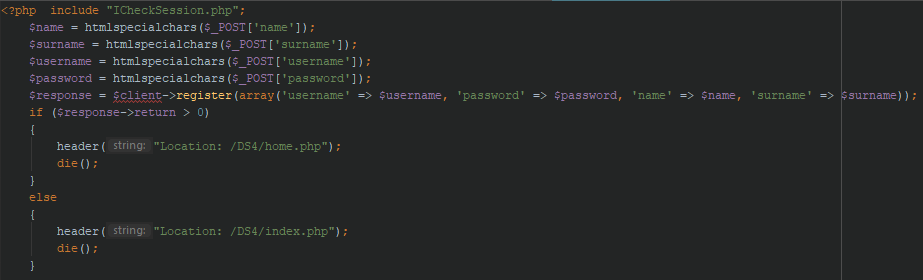


Figure 3 IRegister.php

As you can see the interface takes in the details from a form:

* First name
* Surname
* Username
* Password

Then it utilizes $client to send data to the soap clients function register, which registers the user in the database. Upon completion, if successful it sends the user to home.php otherwise it keeps the user on index.php to try again.

### Displays

Let’s move onto the “displays” section of our interface. This contains PHP code that echo’s HTML code, we did this to keep our HTML code easy to understand and distance PHP from HTML. For example, here is loadOwnScoreboard.php which loads the users own scores:

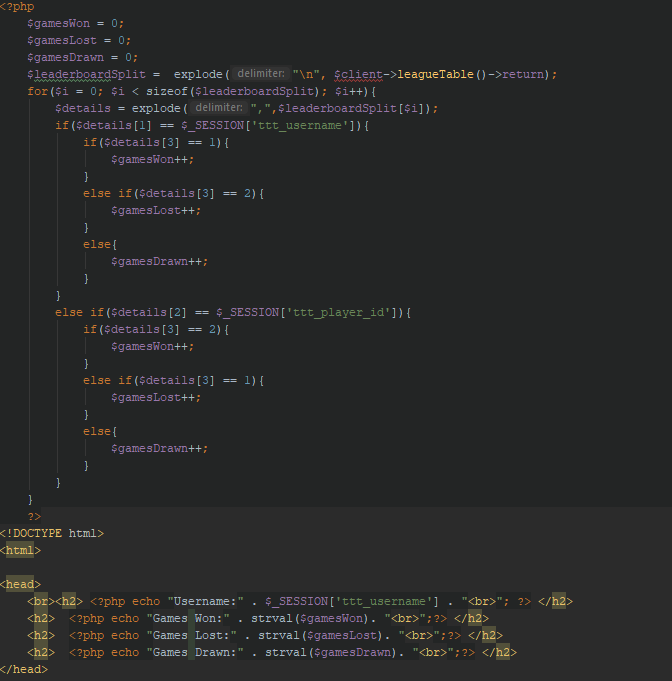


Figure 4 loadOwnScoreBoard.php

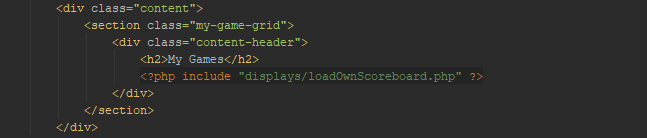


Figure 5 score.php

The images above show how we used $client to retrieve league data, which is then processed to show how many times a user has won, lost, and drawn matches by echo-ing the data back to the HTML file. Then Figure 5 shows how we then projected that data onto the correct screen.

## JavaScript

### Ajax

To have our code working as a cohesive unit we were required to learn how to send data from our JavaScript files to our PHP files. We also didn’t want to refresh the entire page every time we did so, therefore we decided to use AJAX. AJAX took some practicing before we had it working as smoothly as we liked. Let’s examine a short AJAX example we used:

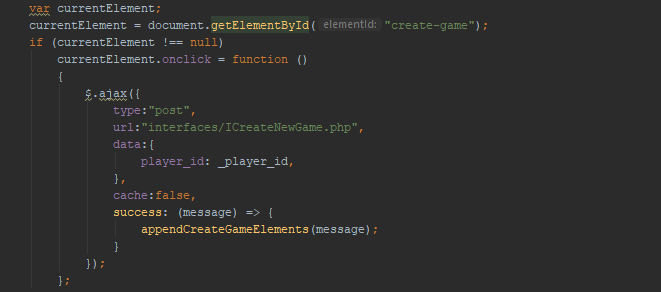


Figure 6 Home.js AJAX example

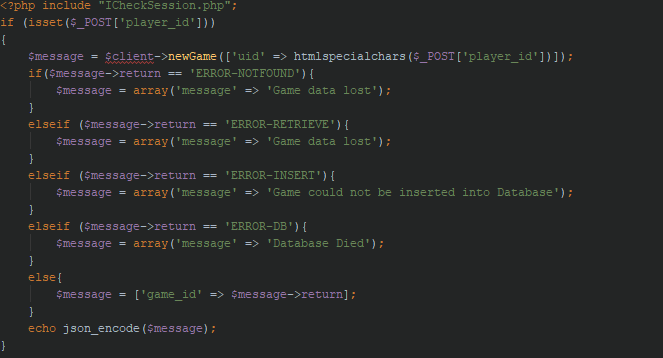


Figure 7 ICreateNewGame.php

In figure 6 we see our $.ajax{()} script being used. We see the type we specified was POST, the URL was the PHP file we wanted to send our data too, and finally out data, in this case, it’s a player id. Then, we see a function call in the success area, which processed the data that was returned in the PHP file in ICreateNewGame as seen in Figure 7.

## CSS and User Interface

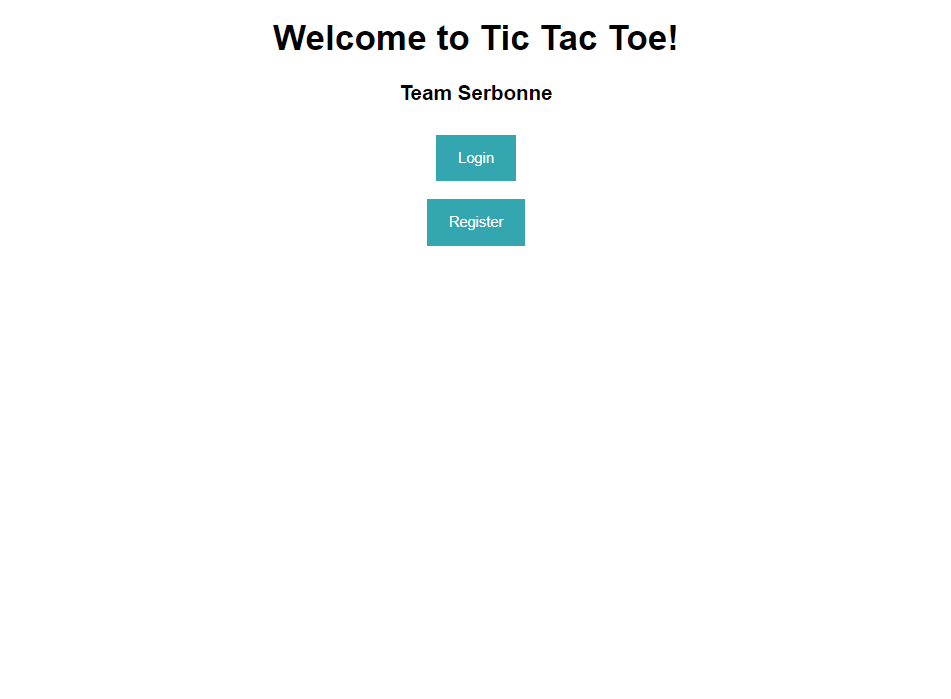
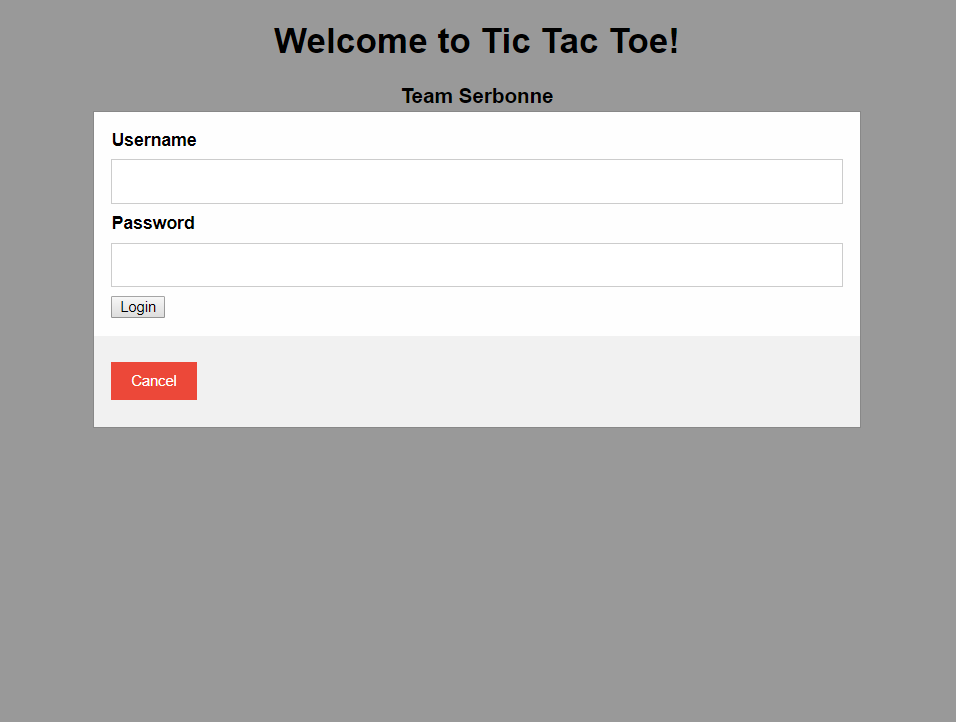
We used CSS to style our pages and create an easy to use UI for users. For our index page, we used some simple CSS and JS we found online to save time working on core functionality. We also found a template for a grid for a checkerboard game which gave us inspiration on how to complete our Tic Tac Toe board. Here are some of the pages:

Figure 8 Login at index.php

Figure 9 Index.php

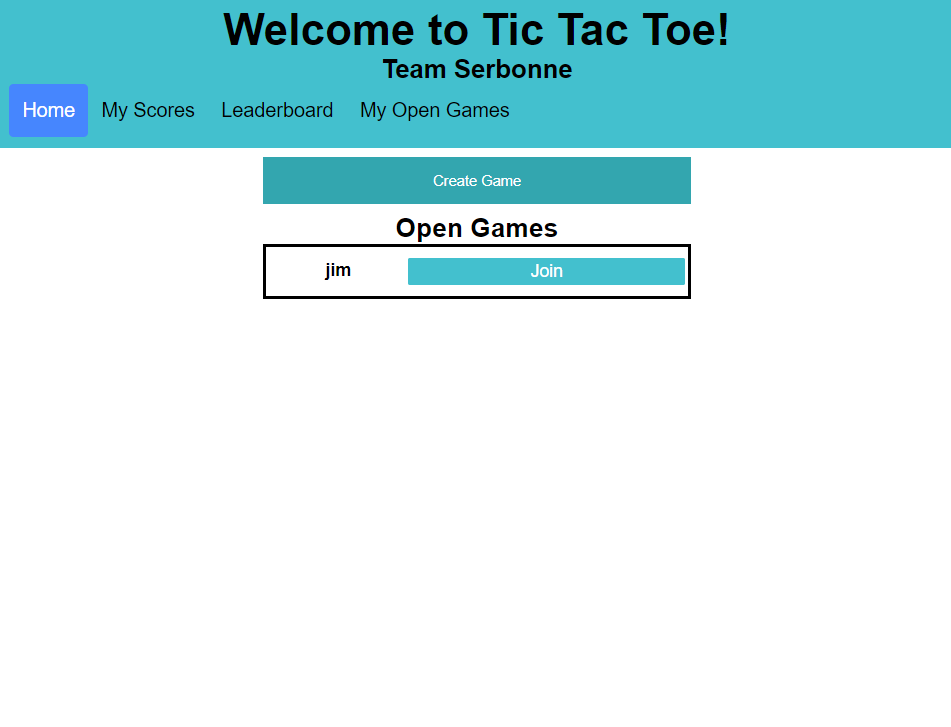
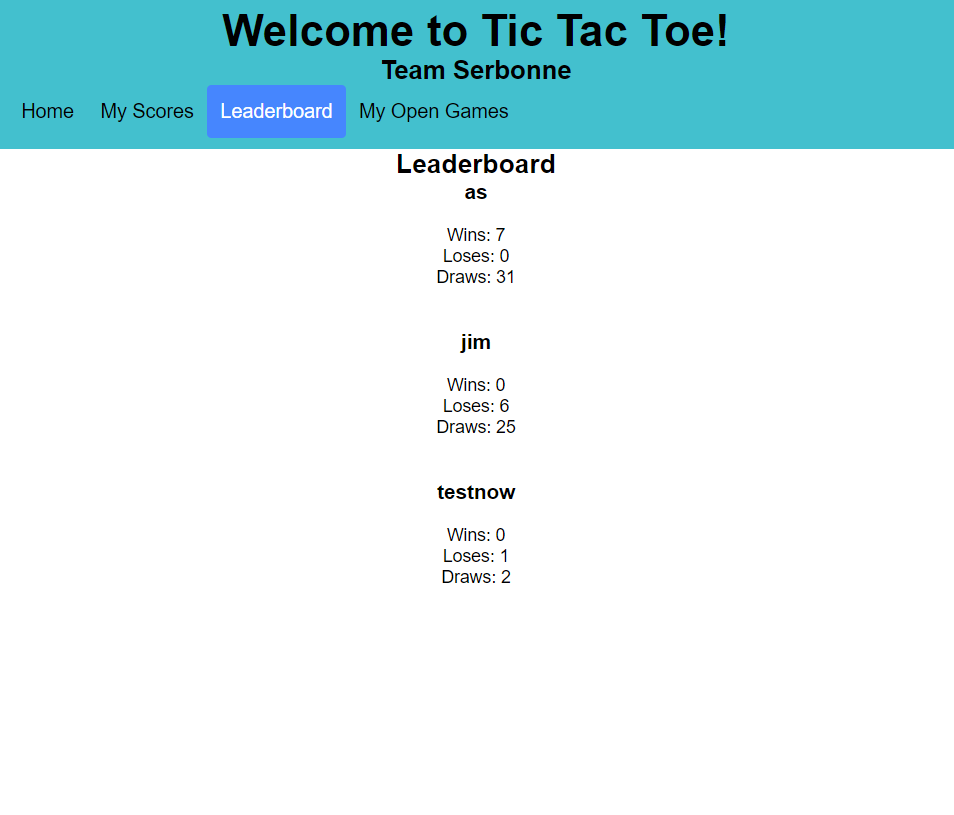


Figure 10 Scoreboard

Figure 11 Jim has an open game, also option to create game

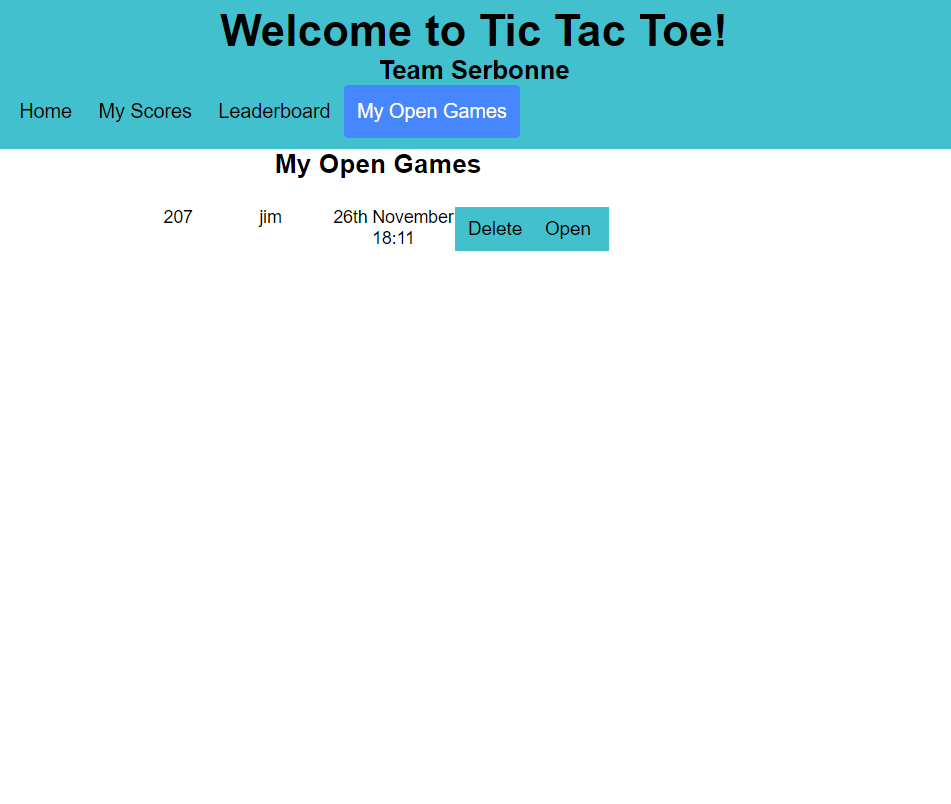
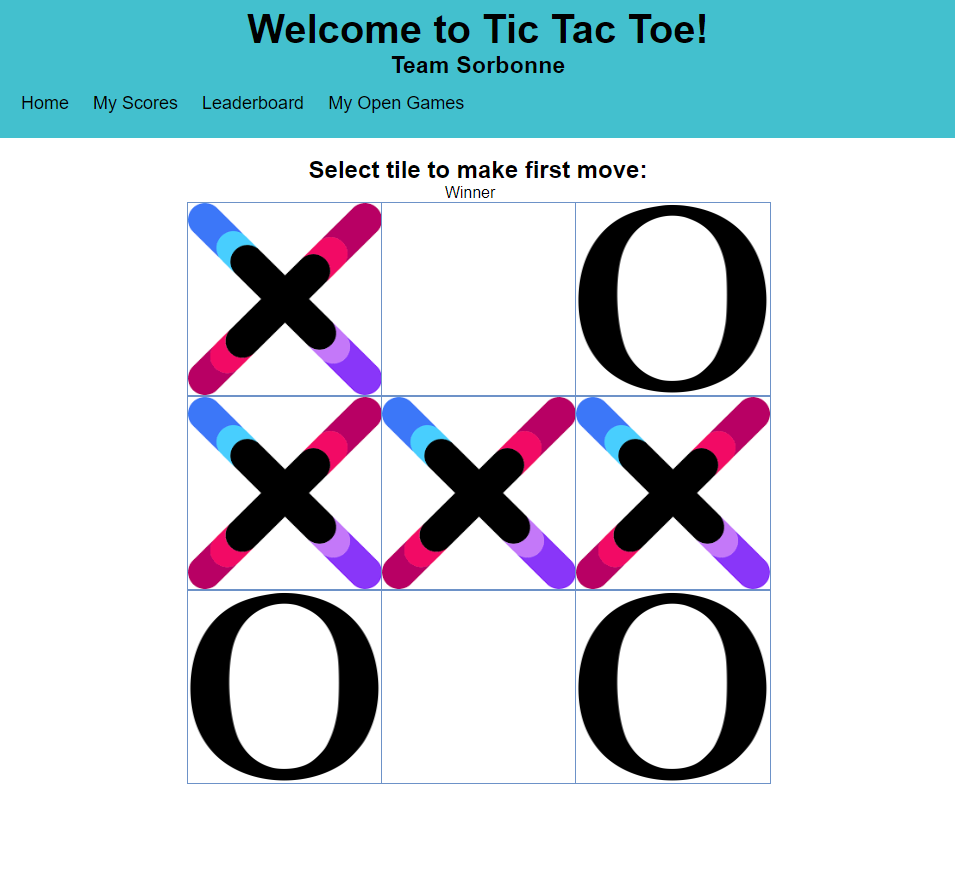


Figure 12 Play Tic Tac Toe

Figure 13 Open or Delete a game you created

# Java

## Project Structure

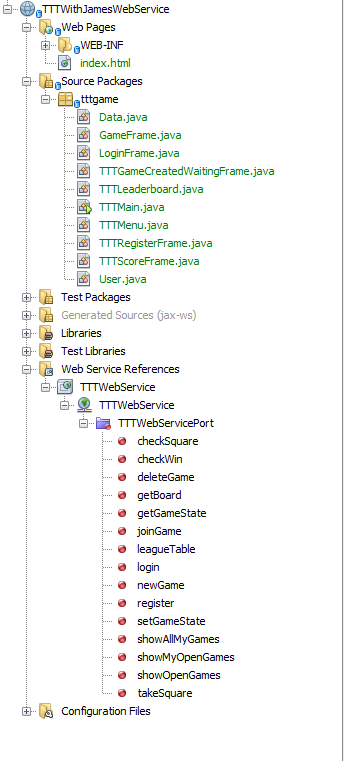
In terms of project structure, we have a fairly simplified version of the java side. Mainly a java class representing each individual screen. For this, we used javax’s native swing library so each class extended from JFrame. Besides these classes, we also created two entity classes called User and Data. User was used to track core user details and pass them from screen to screen. While the Data object was used to track the return codes of the webservice and save them as strings to help when trying to interpret the code later.

Figure Java Interface File Structure

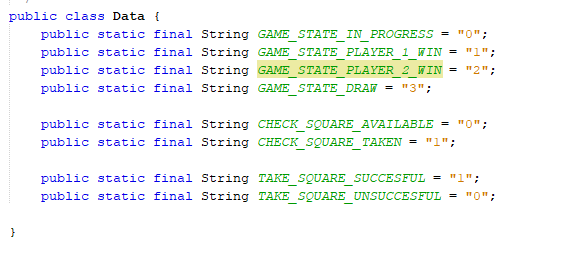


Figure Java Data Class



## Restful Web Client

Through NetBeans, we added the WSDL of the webservice and this allowed us to act as a client to the webservice. James got to host the webservice off his machine at home and serve his address so it was great to test it consistently with other people's code through this.

Let's look at a simple example of using this web client.



Figure Soap WebService Connection



Figure Example of Webservice use

Simply instantiating it and calling the corresponding method with the required parameters is all that is needed after referencing the correct WSDL.

## JFrame & ActionListener

Using JFrame and Swing was fairly straightforward also. When creating a new screen, it was required to extend from JFrame, this gave us all the functionality we needed to setup buttons and display all the data necessary. Seen below in our menu we added buttons to a grid and associated an action listener to each button. Then added the grid to our JFrame itself.





Figure Main Menu Java

In order to interact with events, we used the ActionListener interface, having been introduced to this during the module, it seemed to be the easiest way to write event-driven code in java and it was a simple integration between that and swing. Seen below upon pressing a button on the grid, an event is then fired. We can check the source of the event with our corresponding buttons, isolating the button which fired the event and then acting accordingly. In this case, creating a new screen depending on which button was pressed.

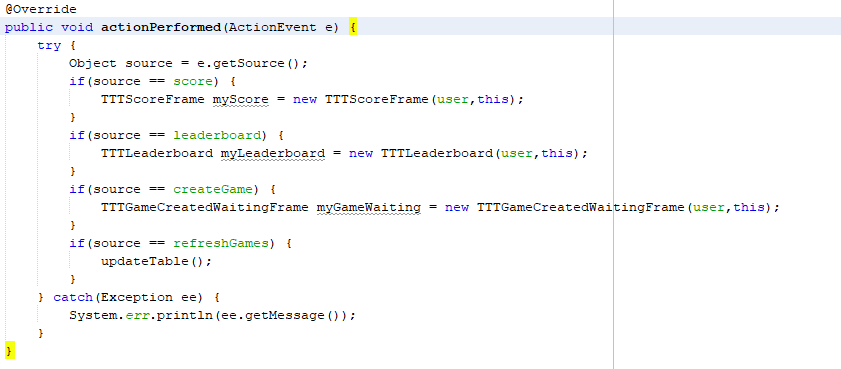


Figure actionPerformed method buttons on the main menu

## Threading

We utilised threading in the main game frame in order to track whether the board has been updated, and if it has:

1. Update the UI
2. Check if somebody won
3. Decide who’s turn it is

While the game is in progress, we do this every second. We did not want there to be any delay in how long it takes to update the board after an opponent's move so that is why we decided to do execute the thread so often.

In terms of implementation we used the Runnable interface to create our thread. Specifying the run() method. To facilitate executing once every second, we just told it to sleep for 1000s while it was in the loop.



# Screens

Figure A Tic Tac Toe game being played on Java

Figure Java Main Menu displaying open games

# Testing and Conclusion

To test the interfaces against each other, a team member hosted the glassfish server publicly. This allowed us to test our implementations from another separate network from each other and the server. Our interfaces were capable of playing a game against one another quickly and without error. By doing so this ensured our interfaces both worked correctly and together on any webservice that provides the same data.

After both interfaces were completed, as a team we better understood how communication between different interfaces using different languages can communicate effectively through webservices. Our PHP interface can effectively and quickly communicate with our Java interface without either knowing about one another or the implementation used.