

COuntdown

Applied project and minor dissertation for Bsc (Hons) of Science in Computing in Software Development

April 2019

Authors:

Thomas Duffy

G00339581@gmit.ie

Marian Ziacik

G00340481@gmit.ie

Supervisor:

Martin Kenirons

Table of Contents

[*Acknowledgements* 2](#_Toc7109413)

[Introduction 3](#_Toc7109414)

[Limitations 4](#_Toc7109415)

[Known bugs 4](#_Toc7109416)

[Research: 5](#_Toc7109417)

[Project Structure 6](#_Toc7109418)

[How the application works and code review 8](#_Toc7109419)

[Full run with correct words and showing data save 30](#_Toc7109420)

[Styling our app : 35](#_Toc7109421)

[Deployment and running the application 37](#_Toc7109422)

[Methodology 44](#_Toc7109423)

[Technology Review 52](#_Toc7109424)

[*Conclusion* 55](#_Toc7109425)

[*Learning outcomes* 56](#_Toc7109426)

[*Future Investigation* 57](#_Toc7109427)

[Appendix 58](#_Toc7109428)

[*References* 58](#_Toc7109429)

*Acknowledgements*

TODO-Thank people who have helped

Introduction

The main aim of this project was to showcase skills we have learned over our 4-year course in GMIT.

The goal was to create a three-tier application which includes a front-end, back-end and database. The project uses JSP and Java Servlets to display the web page to a user, the user interacts with this webpage and the servlet uses Java to run the logic of the game and then return data back to the servlet which sends that data onto a JSP page to be displayed to the user. We use tomcat server to serve up the JSP web pages.

A close up of text on a white background

Description automatically generated

When the user enters a word, we first take in that word and store it as a variable, we pass this word out on an API call to the Oxford English dictionary to check the word is a valid English word before allowing the player to score points. If the word is found using the online dictionary then we match up how many of the player’s letters match the random letters they were given and we score the player’s attempt and allow the player to progress to the next stage of the game.

At the end of the game the overall score of the player is sent to a database so we can keep a record of who has done the best. There is a page in the app which will display the best players who scored the highest.

We used AWS (Amazon Web Services) to host our app. We cover this in the deployment section.

Limitations

* We only have a word game currently, if we were to continue work on this project adding in a numbers game of some sort would be where we would go next.
* Sometimes it’s better to go for a word that just has a high scoring value letter rather than trying to use as many letters as possible. To solve this we should use a multiplier determined by the amount of letters used and multiply the score by this amount e.g the player enters a 3 letter word using 3 of the random letters and the total value of the 3 letters is 5, we should then multiply the score by 3 to give 15. This would reward a player for using more letters rather than someone just using “x” which has a value of 10 alone.
* We should have added case insensitive login, allowing a user to login to the system by capitalizing or not their name e.g. tom and Tom should both resolve to the same username.
* For bootstrap we only used the Grid System for large screens, screens equal to or greater than 1200px wide. We should have support for medium or small screen sizes which would enhance the experience on smaller screen devices.
* The application should have been designed to play on a secure connection through HTTPS protocol.
* We should have added a results page at the end of each round so the player knows where they are for their overall game at the end of each round. To add this now would involve a redesign of a good few of the java and jsp pages.
* We

Known bugs

* The player can use the back button in the browser to go back in a round and re-enter a new word.
* The URL’s are displaying incorrectly on the first run of the game, they display one behind but after the player finishes one full run of the game and starts a new game they are in the correct order.
* Bootstrap was tested on a screen 2736x1824px (Microsoft surface pro 17) and it displayed the app with stretched elements. It didn’t scale properly even though bootstrap manual refers resolution up to 1440px. We tested the laptop between resolutions 1080 and 1440 and we didn’t get the desired results.
* Sometimes the audio can be slightly delayed in the first round, but this problem hasn’t shown up in recent tests, either on local machines or on the amazon server.
* If you go back in the browser using the browser back button and replay the round the sound doesn’t load in. We looked this up and we found information that it is a general problem with Chrome. We tested it in FireFox and it works so it seems to be a Chrome issue. This is also a move that the player should not be allowed to make as it is considered cheating.

Research:

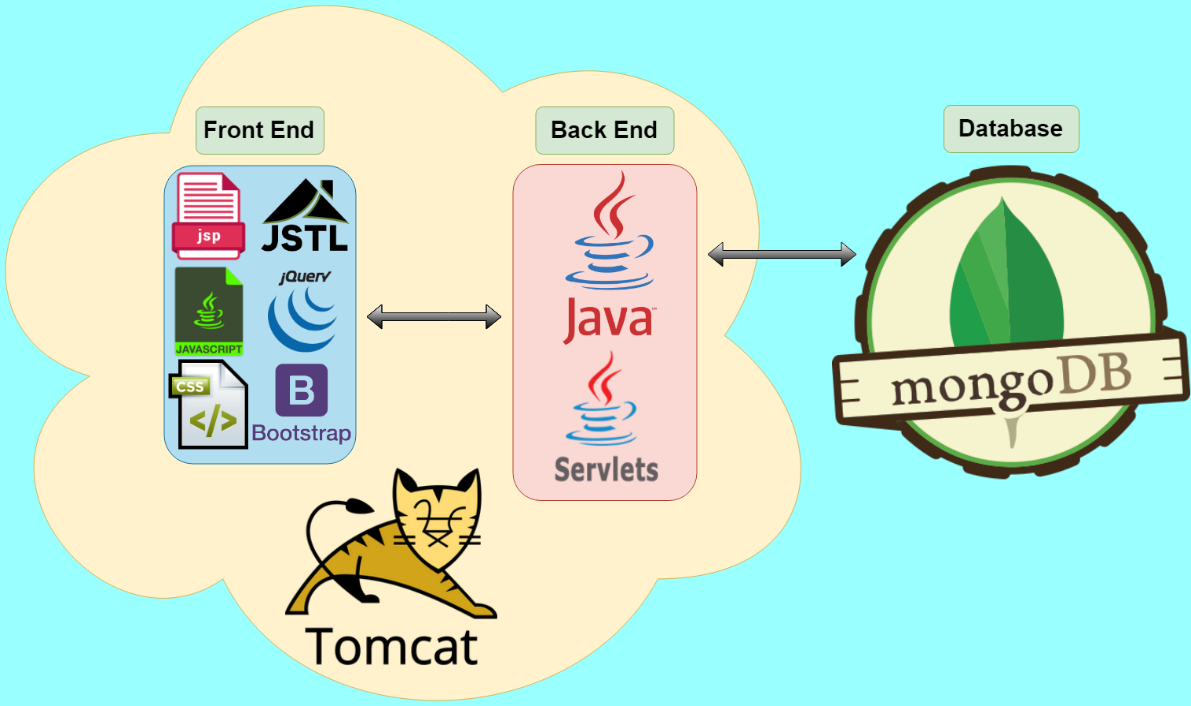
The original idea was to use spring boot java server that would connect to mongo DB, which in turn would talk to a React JS app as the root resource that connects to the HTTP API as a user. This idea was abandoned because we had our group reduced from 3 members to 2 members, and when we started coding the project we realised that we that we needed to spend a lot of time getting used to these technologies and without the third member that was proving very difficult. We have a branch in our GitHub which has a working basic spring boot and react app. Our problems were mainly getting these technologies working with our main game which was coded in Java.

So, we moved onto using JSP and Java servlets to display web pages and a Java backend that does the logic for the game. Originally, we included a Python script that makes a call to the oxford English dictionary, but we later changed this to a Java API call. The code and running app with the python script are on a separate branch in our GitHub also.

The issue with using the python script was that we were using text files to pass information between the Java and the Python. Java code would get the word and print it to a txt file and then using Java we can make a bash command which we would use to call the Python script. We then set the Java code to sleep for 5 seconds to give the Python script a chance to load in the word from the text file, look up the API dictionary and send the result back to another text file, a 200 code if the word was found and 404 if it wasn’t. This created an issue if there was more than one person playing the game the text file could potentially have incorrect data inside. Also, if internet connection speeds were slow and the script took longer than the 5 seconds to run the result from the previous round would be taken.

So, to fix these issues we removed the python script and kept our app coded in just Java. We cover how we make the API call in our code review section.

Project Structure

Project Diagram

Deployment Diagram

UML diagram of Java classes

A screen shot of a computer

Description automatically generated

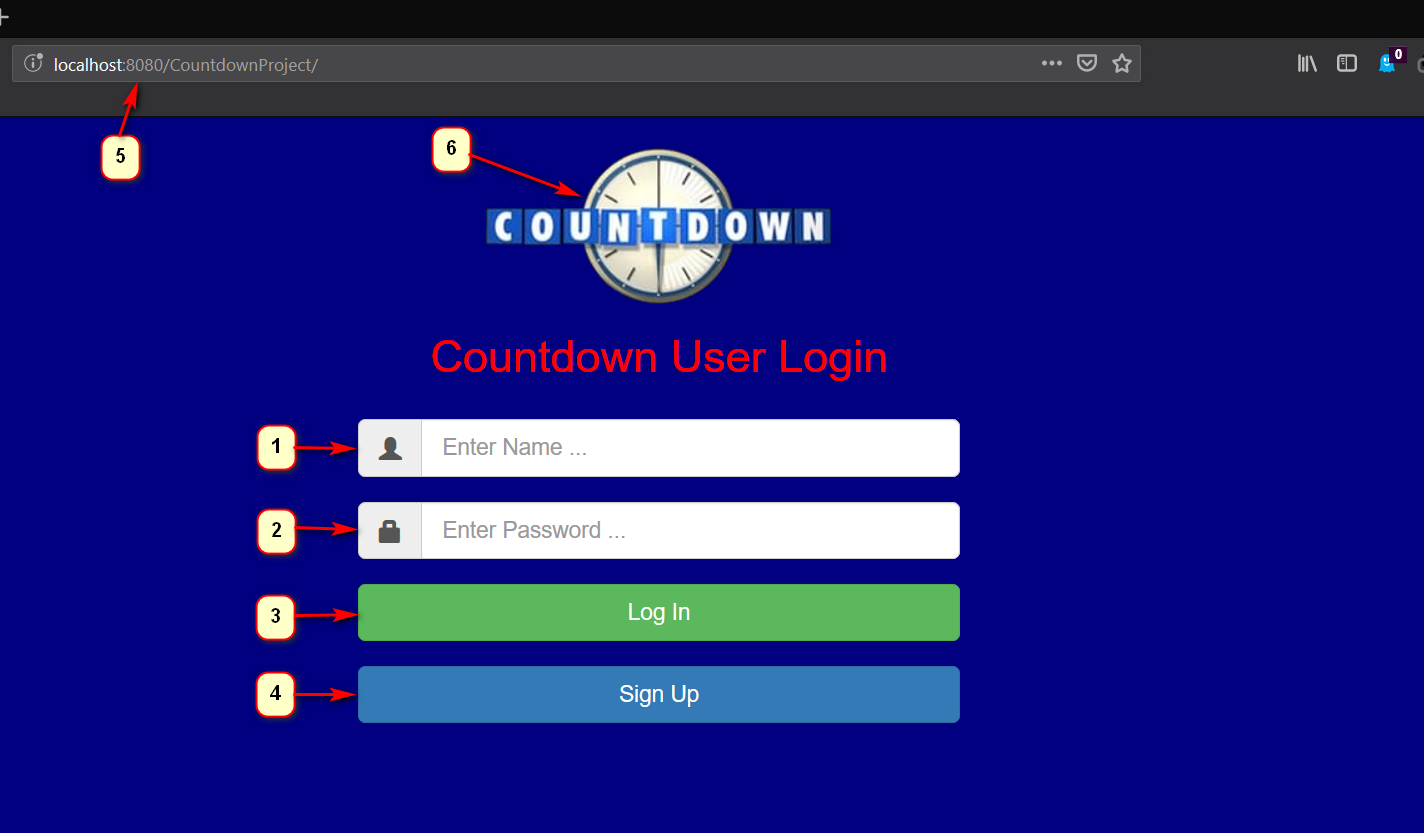
A circuit board

Description automatically generatedUML diagram of JSP pages

How the application works and code review

Countdown is a web app which uses JSP and Java Servlets to talk with a Java backend which does all the logic of the game. The servlets are used to call methods from the classes in the Java backend and use the results to pass data onto a JSP page to be displayed to the user. This data which is displayed to the user will either be used to progress them onto the next step of the game or display the end results of their game to them.

Using Tomcat to serve the JSP pages, tomcat uses a default setting where it looks for an index file to serve after staring up. We use this index file as the first page of our web app.



1. A text box for the user to enter their Username, this must match a username password combination which is present in our database.
2. A text box for the user to enter their password, this must match a username password combination which is present in our database.
3. Login button, this will redirect the user to the next page of our app. Welcome.jsp
4. Sign up button, this will redirect the user to a sign-up page which will be shown below
5. URL where the app is being run, this version is currently run on a local machine using port 8080.
6. Countdown logo present across the whole app.

We have a form in the index page for adding the name and password

placeholder="Enter Name ..." name="login\_id" maxlength="10">

placeholder="Enter Password ..." name="login\_pwd">

The form has a method POST and it is mapped to UserLogin Java servlet using the @webservlet

@WebServlet("/UserLogin")  
public class UserLogin extends HttpServlet

UserLogin Java servlet has a post method which checks the username and password by calling a method searchUserInDb located in MongoDBUtil java class.

protected void doPost(HttpServletRequest req, HttpServletResponse resp)

MongoDBUtil java class contains all methods and settings for the mongoDb database

private static final int *PORT\_NO* = 27017;  
private static final String *URL* = "localhost";  
private static final String *DB\_NAME* = "user\_records";  
  
private static MongoClient *mongoClntObj*;  
private static MongoDatabase *db*;

It will connect to our database “user\_records” and checks collection “user” against entered name and password

private static MongoClient getConnection()

private static MongoDatabase getDB()

public static boolean searchUserInDb(String loginId, String loginPwd) throws Throwable {

MongoCollection<Document> col = *getDB*().getCollection("user");

return userFound;

If the correct name and password is found, we set it to return true and keep the user on index page

boolean isUserFound = MongoDBUtil.*searchUserInDb*(login, pwd);  
if (isUserFound) {  
 req.getSession().setAttribute(AttributeKeys.*SESSION\_USER\_NAME*, login);  
 req.getRequestDispatcher("/welcome.jsp").forward(req, resp);  
 System.*out*.println("User Login Successful");

The UserLogin Java servlet logic checks for every possible combinations of user input, such as entering incorrect name or password or both, empty input, database unreachable etc. and in these cases the user is kept on the index page and not allowed to progress to the game.

Empty data :

if (login == null || pwd == null || login.trim().length() == 0 || pwd.trim().length() == 0) {  
 req.setAttribute("login\_error", AttributeKeys.*USER\_LOGIN\_EMPTY*);  
 req.getRequestDispatcher("/index.jsp").forward(req, resp);

Incorrect data :

req.setAttribute("login\_error", AttributeKeys.*USER\_INCORRECT\_LOGIN*);  
req.getRequestDispatcher("/index.jsp").forward(req, resp);  
System.*out*.println("User Incorrect Login");

Database unavailable :

req.setAttribute("connection\_error", AttributeKeys.*DATABASE\_CONNECTION\_ERROR*);  
req.getRequestDispatcher("/index.jsp").forward(req, resp);  
System.*out*.println("Database Connection Error");  
e.printStackTrace();

We have created a class called AttributeKeys, where we keep all static strings that are used across the application to keep the code clean

public class AttributeKeys {  
 public static final String *SESSION\_USER\_NAME* = "s\_userName";  
 public static final String *SESSION\_TOTAL\_SCORE* = "s\_totalScore";  
 public static final String *SESSION\_ROUND\_NUM* = "s\_roundNum";  
  
 public static final String *USER\_ROUND\_SCORE* = "userRoundScore";  
 public static final String *USER\_GUESS\_WORD* = "userGuessWord";  
  
 public static final String *USER\_LOGIN\_EMPTY* = "Name and Password cannot be empty";

public static final String *USER\_EXISTS* = "User with this name already exists";

public static final String *USER\_INCORRECT\_LOGIN* = "Please enter correct Login details";

public static final String *DATABASE\_CONNECTION\_ERROR* = "Database Connection Error";

public static final String *DATA\_SAVE\_SUCCESS* = "Data Saved Successfully";  
 public static final String *DATA\_SAVE\_FAIL* = "Data Save Failed";

This is useful for keeping consistency across our application with naming style. Instead of consistently renaming these variables when we need them in multiple places, we create them here and reuse as needed.

Next, we will look at the user sign up section of the app.

A screenshot of a cell phone

Description automatically generated

We have a similar scenario here in the sign-up page, where the user is redirected from index page clicking by the SignUp button.

<a class="btn btn-primary btn-lg btn-block" href="signUp.jsp">Sign Up</a>

SignUp jsp page again contains form for adding the name and password as we seen previously. The form has a post method and it is mapped to UserSignUp Java servlet.

@WebServlet("/UserSignUp")  
public class UserSignUp extends HttpServlet {

UserSignUpJava servlet has a post method and checks username and password by calling a method searchUserByName located in MongoDBUtil java class.

protected void doPost(HttpServletRequest req, HttpServletResponse resp)

boolean isUserFound = MongoDBUtil.*searchUserByName*(login);

public static boolean searchUserByName(String loginId)

searchUserByName method will connect to database, search the database for the username and returns the boolean value true or false based on the findings.

MongoCollection<Document> col = *getDB*().getCollection("user");

List<BasicDBObject> obj = new ArrayList<BasicDBObject>();  
obj.add(new BasicDBObject("id", loginId));

return userFound;

If the value is true, this means that the user exists, and the user will have to choose different user name and password to sign up or go back to login page to enter name and correct password

boolean isUserFound = MongoDBUtil.*searchUserByName*(login);  
if (isUserFound) {  
 req.setAttribute("login\_error", AttributeKeys.*USER\_EXISTS*);  
 req.getRequestDispatcher("/signUp.jsp").forward(req, resp);  
 System.*out*.println("User with this name already exists");

If the value is false, this means that the user does not exists, userSignedUp will be called and after that the user will be successfully redirected to welcome page and ready to play the game. The name and password details will be recorded to the database for future login.

MongoDBUtil.*userSignedUp*(login, pwd);  
req.getSession().setAttribute(AttributeKeys.*SESSION\_USER\_NAME*, login);  
req.getRequestDispatcher("/welcome.jsp").forward(req, resp);  
System.*out*.println("User Sign Up Successful");

In any other cases, i.e. user adds the name and doesn’t enter password, user leaves both fields empty, and when the database is unreachable, they will be redirected back to sign up page

if (login == null || pwd == null || login.trim().length() == 0 || pwd.trim().length() == 0) {  
 req.setAttribute("login\_error", AttributeKeys.*USER\_LOGIN\_EMPTY*);  
 req.getRequestDispatcher("/signUp.jsp").forward(req, resp);

req.setAttribute("connection\_error", AttributeKeys.*DATABASE\_CONNECTION\_ERROR*);  
req.getRequestDispatcher("/index.jsp").forward(req, resp);  
System.*out*.println("Database Connection Error");  
e.printStackTrace();

In the index page and sign up page we added error messages for all scenarios.

<div align="center">  
 <h3 id="logErr" class="p-3 mb-2 text-danger">${login\_error}</h3>  
 <h3 id="conErr" class="p-3 mb-2 text-danger">${connection\_error}</h3>  
</div>

They are connected to JavaScript located in webapps/resources/js folder and are linked through the script in the head of the page

<script type="text/javascript" src="resources/js/messages.js"></script>

The code is responsible for displaying the messages based on user’s interaction with the login and sign up page and it hides the messages by default. It is triggered when the user makes illegal action

$(document).ready(function() {  
 $('#login\_id, #login\_pwd').click(function() { //click into text fields   
 $('#logErr').hide();   
 $('#conErr').hide();  
 });   
});

The js code is linked to UserLogin and UserSignUp, the java then renders the pages and display relevant error code.

req.setAttribute("login\_error", AttributeKeys.*USER\_LOGIN\_EMPTY*);

req.setAttribute("login\_error", AttributeKeys.*USER\_INCORRECT\_LOGIN*);

req.setAttribute("connection\_error", AttributeKeys.*DATABASE\_CONNECTION\_ERROR*);

A text box for the user to enter a username, this will be saved to the database and used to identify the player. When a score is saved it will be associated with this username.

1. A text box for the user to enter a password, along with the username entered above this will provide a username/password combo for the player to log in again.
2. Submit button to commit the username and password to the database
3. Back button to go back to the previous page

When the user clicks on submit, we have a doPost method which has request, response. The request will have the parameters passed in when the user tried to sign-up. These will be set as login\_id and login\_pwd.

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {  
  
 String login = req.getParameter("login\_id");  
 String pwd = req.getParameter("login\_pwd");

If the user tries to submit an empty form, we should handle that by keeping them on the current page and not allowing them to pass to game

if (login == null || pwd == null || login.trim().length() == 0 || pwd.trim().length() == 0) {  
 req.setAttribute("login\_error", AttributeKeys.*USER\_LOGIN\_EMPTY*);  
 req.getRequestDispatcher("/signUp.jsp").forward(req, resp);

A screenshot of a cell phone

Description automatically generated

If the user does enter a name but it is a name that already exists in our database, we need to stop them from using that name to avoid confusion on scores.

A screenshot of a cell phone

Description automatically generated

The code to handle this problem looks like

try {  
 boolean isUserFound = MongoDBUtil.*searchUserByName*(login);  
 if (isUserFound) {  
 req.setAttribute("login\_error", AttributeKeys.*USER\_EXISTS*);  
 req.getRequestDispatcher("/signUp.jsp").forward(req, resp);  
 System.*out*.println("User with this name already exists");

We use a boolean and when that finds the record with the same name as the one the user just tried to sign up with it becomes true and sends out the error and keeps the user on the sign up page.

So, if we try to sign up to the game with a name that isn’t present in our database, we should be allowed to add that record and continue to the game. Here is the collection of users I have registered in my local system.

A screen shot of a computer

Description automatically generated

By entering a name that isn’t present we will be allowed to add the record

A screenshot of a cell phone

Description automatically generated

We are then progressed onto the welcome page before we start the game.

A screenshot of a cell phone

Description automatically generated

And that new record present on the database

A screenshot of a computer

Description automatically generated

If the user is signing in, then an incorrect username and password combination will result in the following error being displayed.

A screenshot of a cell phone

Description automatically generated

In our code we check if a user has entered correct login details in UserLogin.java

We have a doPost method which has request, response. The request will have the parameters passed in when the user tried to login. These will be set as login\_id and login\_pwd.

// Reading post parameters from the request  
String login = req.getParameter("login\_id");  
String pwd = req.getParameter("login\_pwd");

With these set we can check firstly that the user hasn’t submitted an empty form

if (login == null || pwd == null || login.trim().length() == 0 || pwd.trim().length() == 0)

{  
req.setAttribute("login\_error", AttributeKeys.*USER\_LOGIN\_EMPTY*);  
req.getRequestDispatcher("/index.jsp").forward(req, resp);

If the user has entered an empty form, then we keep them on the current page and don’t allow access.

However if the user enters data but it doesn’t match any Username/Password combination record we have in our database we have to keep them on the current page and let them know their login was unsuccessful.

else {  
 req.setAttribute("login\_error", AttributeKeys.*USER\_INCORRECT\_LOGIN*);  
 req.getRequestDispatcher("/index.jsp").forward(req, resp);  
 System.*out*.println("User Incorrect Login");  
}

We also must handle instances where the database may be unavailable.

catch (Throwable e) {  
 req.setAttribute("connection\_error", AttributeKeys.*DATABASE\_CONNECTION\_ERROR*);  
 req.getRequestDispatcher("/index.jsp").forward(req, resp);  
 System.*out*.println("Database Connection Error");  
 e.printStackTrace();  
}

Again, because we can’t verify the user’s login details in this case, we keep them on the current page and display an error to them.

If the database is unavailable for whatever reason the user will see the following error and be unable to progress.

A screenshot of a cell phone

Description automatically generated

When the database becomes unreachable if the player is already passed the login section and they try to check high scores for example they will be presented with the following. “Database Connection Error” will be displayed on the jsp page.

A close up of a sign

Description automatically generated

In the case of the database being unreachable (as we show in our metrology testing, where we show how to make the database unreachable).

This is the catch on DisplayHighscores java servlet that deals with the database unreachable error when the player tries to display the top scores on highScores.jsp while the database is unavailable

catch (Throwable throwable) {  
 request.setAttribute("connection\_error", AttributeKeys.*DATABASE\_CONNECTION\_ERROR*);  
 System.*out*.println("Data display failed");  
 throwable.printStackTrace();

Displaying the error message with jsp

<h3 id="conErr" class="p-3 mb-2 text-danger">${connection\_error}</h3>

Added the script link specified in jsp head

<script type="text/javascript" src="resources/js/messages.js"></script>

Javascript code for the error message is located in webapps/resources/js/messages.js

$(document).ready(function() {  
 $('#login\_id, #login\_pwd').click(function() { //click into text fields   
 $('#logErr').hide();   
 $('#conErr').hide();  
 });   
});

If the database can’t be reached so the player high score results are unavailable and the player will not be able to add a new record to the database when they finish the game.

A screenshot of a cell phone

Description automatically generated

In the case where data save is not successful due to the database being unavailable . We have a catch on roundResult java servlet finishGame method where the MongoDBUtil.saveResult is called.

} catch (Throwable e) {  
 System.*out*.println("Data Save Failed");  
 session.setAttribute("data\_save\_fail", AttributeKeys.*DATA\_SAVE\_FAIL*);  
  
 e.printStackTrace();  
}

public static final String *DATA\_SAVE\_FAIL* = "Data Save Failed";

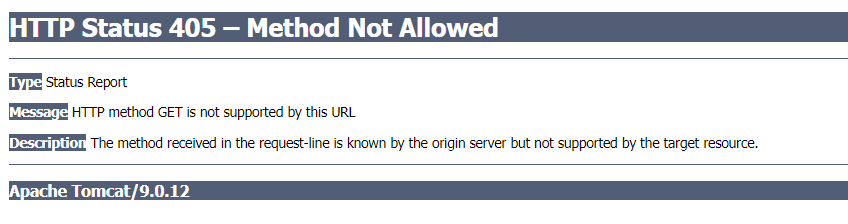
Displaying the error message on finalResult.jsp

<div align="center">  
 <h3 class="p-3 mb-2 text-danger">${data\_save\_fail}</h3>  
 <h3 class="p-3 mb-2 text-danger">${data\_save\_success}</h3>  
</div>

There is a security feature we added where even if you know the URL of the next page, the system will block you from accessing it if you have not provided the required login credentials.

The URL of the next page is <http://63.33.99.89:8080/CountdownProject_war/UserLogin>

Trying to access that URL without login gives this error



In our mongo code we have the mechanisms to not allow the user to get to a page after the index page without first being checked for valid login credentials.

If the user enters correct details then we check to see if it’s in our database and if it is we allow the user to progress to the next page of our app which is the welcome page, welcome.jsp

else {  
  
 try {  
 boolean isUserFound = MongoDBUtil.*searchUserInDb*(login, pwd);  
 if (isUserFound) {  
 req.getSession().setAttribute(AttributeKeys.*SESSION\_USER\_NAME*, login);  
 req.getRequestDispatcher("/welcome.jsp").forward(req, resp);  
 System.*out*.println("User Login Successful");



1. New Game button, this will start up a new game of countdown
2. Rules button, this will display the rules of our game of countdown to the user
3. High scores button, this will display a list of high scores stored on the database
4. The username used to login displayed

The rules button opens this page in a new tab.



Our rules page sets out the game rules. We have used the scrabble scoring system for the letters as we felt this would give players a greater chance of having different scores. High scores will open the high scores page, we order the results by their high scores, so the better scores are near the top

A screen shot of a smart phone

Description automatically generated

1. This column contains the username
2. This column contains the player’s score
3. This column contains the date and time the user got their score

Selecting New game starts the game

In the welcome.jsp we have the following code which lets us map the button press to the java servlet.

<a class="btn btn-primary btn-lg btn-block" href="NewGame">New Game</a>

Looks for NewGame mapping in Java src folder.

@WebServlet("/NewGame")

Finds it in NewGame.java and NewGame.java sets up our new game by setting the total score to zero and the round number to 1.

request.getSession().setAttribute(AttributeKeys.*SESSION\_TOTAL\_SCORE*, 0);  
request.getSession().setAttribute(AttributeKeys.*SESSION\_ROUND\_NUM*, 1);

and then we call the first method from our java

GamePlay.*nextRound*(request, response, getServletContext());

In the nextRound method we need to set up the random letters the player will need to use to get a score.

List<String> randomLetters = GenerateRandomLetters.*list*();

Goes to our java class for generating the random letters.

In the list method we create a string array with all the letters of the alphabet

String[] alphabetArr = { "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R",  
 "S", "T", "U", "V", "W", "X", "Y", "Z" };

Then we use the array to create a list of strings

List<String> alphabetList = Arrays.*asList*(alphabetArr);

And we use Collections.shuffle to randomize the order of the letters

Collections.*shuffle*(alphabetList);

Then we select the first 10 elements of the list and that will be the 10 random letters we display to the player.

List<String> randLetters = alphabetList.subList(0, 10);

Then using

String noListString = String.*join*("", randLetters);  
CompareWordToLetters.*listToString*(noListString);

We pass the string of letters to another method for use later when we must compare the user’s word against the random letters.

This is the result the player sees displayed to them.

A screenshot of a video game

Description automatically generated

1. A timer displayed inside the countdown logo. When this reaches zero the form will auto submit, and the user won’t get a score
2. The current round number
3. The player’s current overall game score
4. The random letters the user must use to score points
5. Text box for the player to enter their word
6. Submit button that will take in the player’s word and check it against the online dictionary and if it passes that it will check it against the random letters given to see how many letters were used.
7. Rules button, this will display the rules of our game of countdown to the user in a new tab
8. High scores button, this will display a list of high scores stored on the database in a new tab

When the player clicks the submit button in our java backend calls the resolveRound method that is located in RoundResult Java servlet.

private static void resolveRound(HttpServletRequest request) throws IOException {  
 HttpSession session = request.getSession();  
 int currentTotalScore = (int) session.getAttribute(AttributeKeys.*SESSION\_TOTAL\_SCORE*);  
 String uGuess = request.getParameter(AttributeKeys.*USER\_GUESS\_WORD*);  
  
 int roundScore = 0; //0 explicitly - if wrong guessing  
  
 if (Dictionary.*wordExists*(uGuess)) {  
 System.*out*.println("Word is valid.");  
 roundScore = CompareWordToLetters.*countRoundScore*(uGuess);  
 }  
 request.setAttribute(AttributeKeys.*USER\_ROUND\_SCORE*, roundScore);  
 session.setAttribute(AttributeKeys.*SESSION\_TOTAL\_SCORE*, currentTotalScore + roundScore);

The session keeps track of user’s current total score. We don’t need to keep the word for entire game session (only for the round result), that’s why it is requested as a parameter.

The guess word is then checked against the online Oxford Dictionary API, that is located in Dictionary class.

We set the roundScore explicitly to 0, so if the player enters an invalid word then no points will be allocated. If the word is valid then we call countRoundScore method in the CompareWordToLetters class which will add up the players score based on the letters they used.

This is also an example of where we used the AttributeKeys we set up earlier on, as can be seen this helps make the code more readable.

In CompareWordToLetters we have a countRoundScore method which we pass in the user’s word and create two character HashSets that we will use

Set<Character> set1 = new HashSet<Character>();  
Set<Character> set2 = new HashSet<Character>();

for (char c : *wordString*.toCharArray()) {  
 set1.add(c);  
}  
for (char c : userGuessWord.toCharArray()) {  
 set2.add(c);  
}

Set1 is the random letters that we generated before and set2 is the word the player entered. Then we want to keep all common letters between both sets.

set1.retainAll(set2);

This gives us the letters we will use to score the player’s round number.

We then use a switch statement on those letters to add up the player’s score

switch (c) {  
 case 'A':  
 roundScore += 1;  
 break;

all the way through the alphabet.

Back in the game if the clock runs down and the player has not submitted a word, they will be progressed onto the following

A screenshot of a cell phone

Description automatically generated

1. The players round score, zero because they ran out of time
2. The player’s word guess, empty because they ran out of time
3. Overall game score, unaffected because the player did not score this round

This is done in JavaScript. We have a function and when it times out it will auto submit the form.

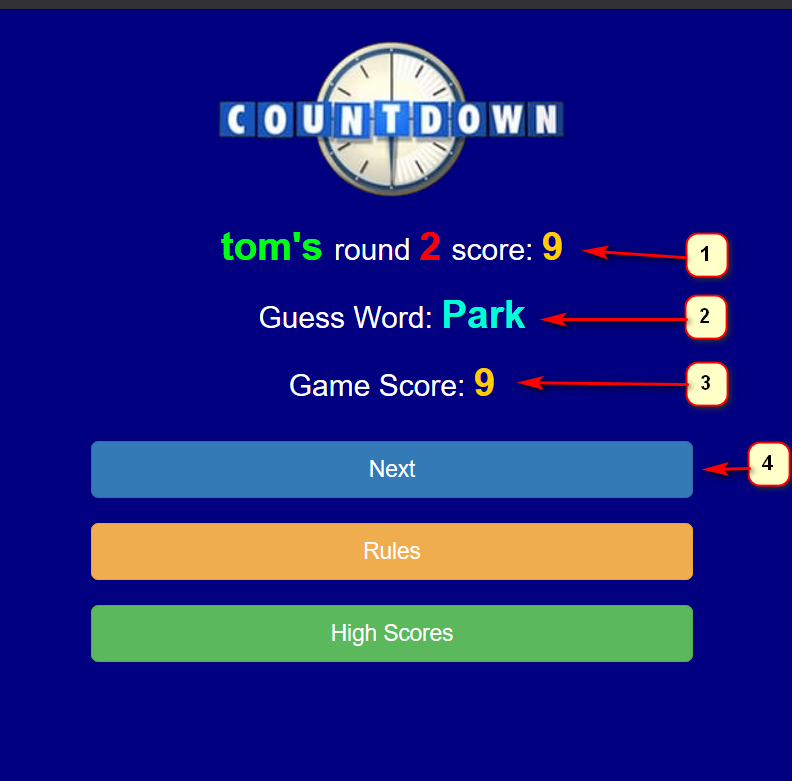
$(document).ready(function() {  
 setTimeout(function() {  
 $("#word\_form").submit();  
 }, 31000);  
});

If the user submits a word inside the time frame

A screenshot of a cell phone

Description automatically generated

Then they will get scored based on the letters used



1. The users score based on using the letters P,R,K from the random letters
2. The work the player entered
3. The player’s overall game score after getting this rounds score
4. Button to move to the next round of the game

An incorrect word will be picked up by the Oxford dictionary API and we won’t score the player for any letters used.

A screenshot of a cell phone

Description automatically generated

Result :

A screenshot of a cell phone

Description automatically generated

1. Incorrect word results in no score
2. The word entered
3. Overall game score unaffected by the incorrect word
4. Move to next round of the game

The Oxford dictionary API.

String app\_id = "ID";  
String app\_key = "KEY";  
String language = "en";  
String url = "https://od-api.oxforddictionaries.com:443/api/v1/entries/" + language + "/" + word.toLowerCase();  
  
URL urlForGetRequest = new URL(url);  
  
HttpURLConnection connection = (HttpURLConnection) urlForGetRequest.openConnection();  
connection.setRequestMethod("GET");  
connection.setRequestProperty("app\_id", app\_id);  
connection.setRequestProperty("app\_key", app\_key);  
int responseCode = connection.getResponseCode();  
  
return responseCode == HttpURLConnection.*HTTP\_OK*;

Full run with correct words and showing data save

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

If it is not the last round (round 3) then we call the method showResults and increment the round number by 1 to indicate we are moving to the next round.

} else {  
 *showResults*(request, response, getServletContext());  
 *incrementRoundNum*(request);  
}

Incrementing the round number in the following method

private static void incrementRoundNum(HttpServletRequest request) {  
 HttpSession session = request.getSession();  
 int lastRoundNum = (int) session.getAttribute(AttributeKeys.*SESSION\_ROUND\_NUM*);  
 session.setAttribute(AttributeKeys.*SESSION\_ROUND\_NUM*, lastRoundNum + 1);  
}

When we are at the end of a round but not the end of the game (round 3) we keep the player going back to the result.jsp page.

private static void showResults(HttpServletRequest request, HttpServletResponse response, ServletContext ctx)  
 throws ServletException, IOException {  
  
 RequestDispatcher dispatcher = ctx.getRequestDispatcher("/result.jsp");  
 dispatcher.forward(request, response);  
}

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Moving onto the next and final round, round 3.

A screenshot of a cell phone

Description automatically generated

Data save is done now by calling saveResult method located in MongoDBUtil Java class.

Here we want to add to the games collection by appending onto the existing collection with our new record.

public static void saveResult(String userName, int gameScore) throws Throwable {  
 MongoCollection<Document> col = *getDB*().getCollection("games");  
 String dateNow = *DATE\_FORMAT*.format(new Date());  
 // Add Document to Database/Collection  
  
 Document myNewDoc = new Document();// Create a new Mongo Document  
  
 System.*out*.println("Add Documents to Database/Collection");  
 myNewDoc.append("name", userName).append("score", gameScore).append("date", dateNow);  
  
 col.insertOne(myNewDoc);  
}

We format the date by:

private static final SimpleDateFormat *DATE\_FORMAT* = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");

This gives us a very detailed date stamp down to the seconds to allow us to tell records apart as best as possible.

The method finishGame located in RoundResult class is called when we want to wrap up the game

There is also a method isGameFinished, which compares number of rounds against actual number of rounds saved in the session.

private static boolean isGameFinished(int lastRoundNum) {  
 return lastRoundNum == *NO\_OF\_ROUNDS*; //3  
}

This is used to check that we have the correct round.

int lastRoundNum = (int) request.getSession().getAttribute(AttributeKeys.*SESSION\_ROUND\_NUM*);

As this is the final round finishGame is called and player will be redirected to finalResult.jsp and we want to save the score to our database.

private static void finishGame(HttpServletRequest request, HttpServletResponse response, ServletContext ctx)  
 throws ServletException, IOException {  
 // Saving the score to database  
 HttpSession session = request.getSession();  
 try {  
 MongoDBUtil.*saveResult*(  
 (String) session.getAttribute(AttributeKeys.*SESSION\_USER\_NAME*),  
 (int) session.getAttribute(AttributeKeys.*SESSION\_TOTAL\_SCORE*));  
 System.*out*.println("Data saved successfully");  
 session.setAttribute("data\_save\_success", AttributeKeys.*DATA\_SAVE\_SUCCESS*);

A screenshot of a cell phone

Description automatically generated

The player is shown their overall game score, given the on screen prompt that their data has been saved to the database and if they scored well enough they will be visible in the high scores page.



1. New record in the database

Code fors Displaying the high scores is done by the getTopTen method located in MongoDBUtil class.

public static List<GameToDatabase> getTopTen() throws Throwable

MongoCollection<Document> collection = *getDB*().getCollection("games");

In here we have created array list and passed the generics(struct) class GameToDatabase to it, in which we have set all variables and getters for pulling data from the database. So, now we have an array of lists, that has a particular structure(based on the data we want to retrieve). The result will be used to return a list for each record that is added to the array.

List<GameToDatabase> result = new ArrayList<>();

This is the generics :

public class GameToDatabase {  
  
 String userName;  
 int gameScore;  
 String date;  
  
 public String getUserName() {  
 return userName;  
 }  
  
 public int getGameScore() {  
 return gameScore;  
 }  
  
 public String getDate() {  
 return date;  
 }

From there we can create an instance of GameToDatabase, pass all the values which will be saved to the array and result will be returned.

while (cur.hasNext()) {  
 List<Object> values = new ArrayList(cur.next().values());  
  
 GameToDatabase gameResult = new GameToDatabase();  
 gameResult.userName = (String) values.get(1); //1 is name  
 gameResult.gameScore = (int) values.get(2); //2 is score  
 gameResult.date = (String) values.get(3); //3 is date  
  
 result.add(gameResult);  
 }  
return result;

We also have a way to display only 10 best results and they are sorted by the user score in descending order.

MongoCursor<Document> cur = collection.find().sort( new BasicDBObject( "score" , -1 ) ).limit(10).iterator();

The getTopTen method is then called in DisplayHighScores Java servlet, that is linked to the highScores.jsp page. DisplayHighScores uses a doGet method because we are not passing anything.

protected void doGet(HttpServletRequest request, HttpServletResponse response)

List<GameToDatabase> topTen = MongoDBUtil.*getTopTen*();  
request.setAttribute("topTen", topTen);

RequestDispatcher dispatcher = ctx.getRequestDispatcher("/highScores.jsp");

In the highScores we use jstl tags to loop through the getTopTen’s result array and that is how they are displayed on the screen.

tbody>  
 <**c:forEach** items="${topTen}" var="gameResult" >  
 <tr>  
 <td>${gameResult.userName}</td>  
 <td>${gameResult.gameScore}</td>  
 <td>${gameResult.date}</td>  
 </tr>  
 </**c:forEach**>  
</tbody>

We have to add the tag library to the jsp page, without this the page will not work. This is added at the top of highScore.jsp page.

<%@ **taglib** prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<%@ **taglib** uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

Styling our app :

We have used bootstrap’s grid system in all jsp pages and it allows us to to divide the screen into three different sections and the whole body of the application is all wrapped up in the container-fluid divider.

<body class="text-center">  
  
 <div class="container-fluid">  
  
 <div class="row">  
 <div class="col-lg-4"></div>  
 <div class="col-lg-4">  
  
 //code  
  
 </div>  
 <div class="col-lg-4"></div>  
 </div>  
  
 </div>  
</body>

This is the application view in full screen on a 1080p screen.

A screenshot of a computer

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedThis is the app half screen. This is the app at it’s smallest size

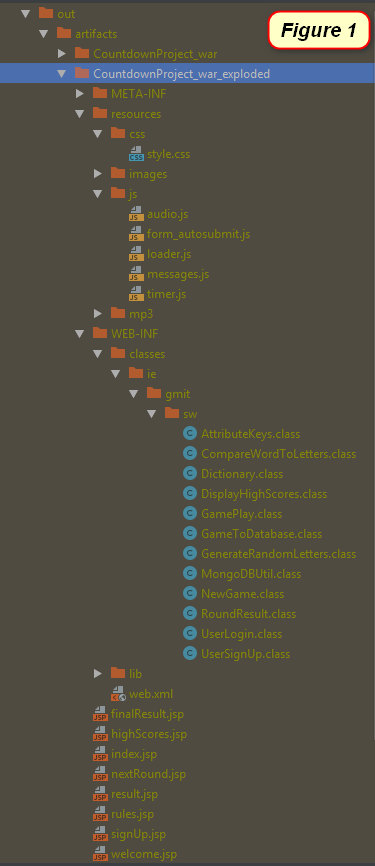
Countdown on a 5-inch android screen :

A screenshot of a cell phone

Description automatically generated

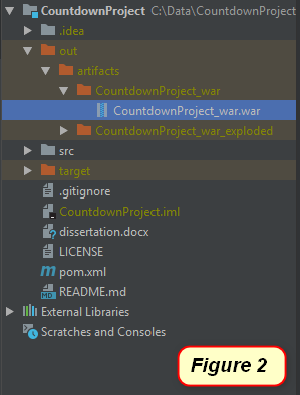
Deployment and running the application

For our application we have decided to use Apache Tomcat Server technology as it is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies.

A web application can be deployed to the server as:

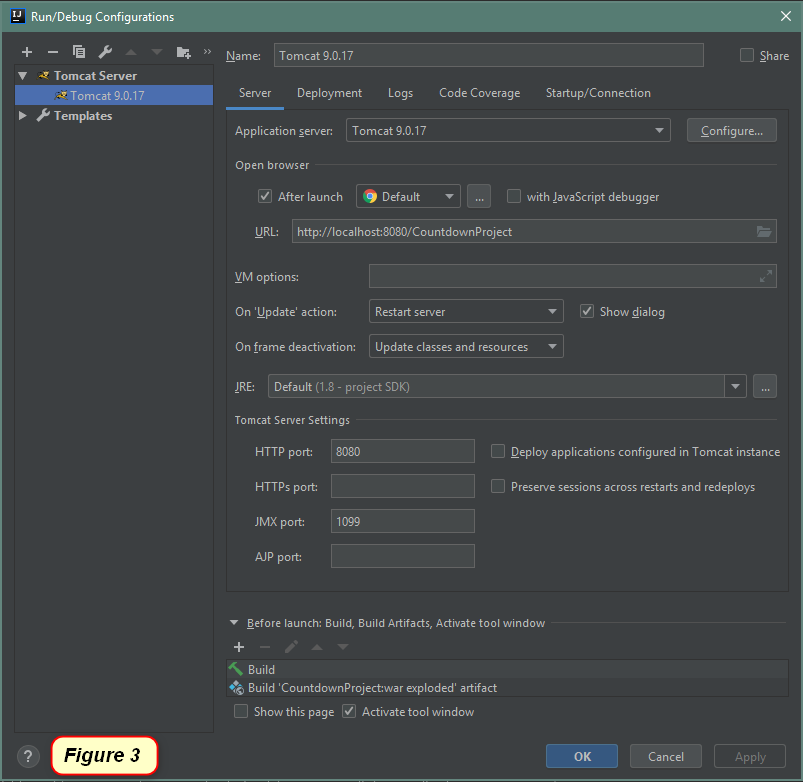
- an exploded directory where files and folders are presented in the file system as separate items (figure 1)

- a WAR file (Web archive) which contains all the required files packed to one archive container that has an extension .war (figure 2)

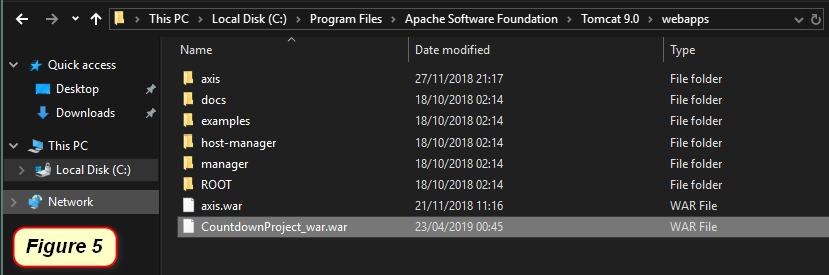


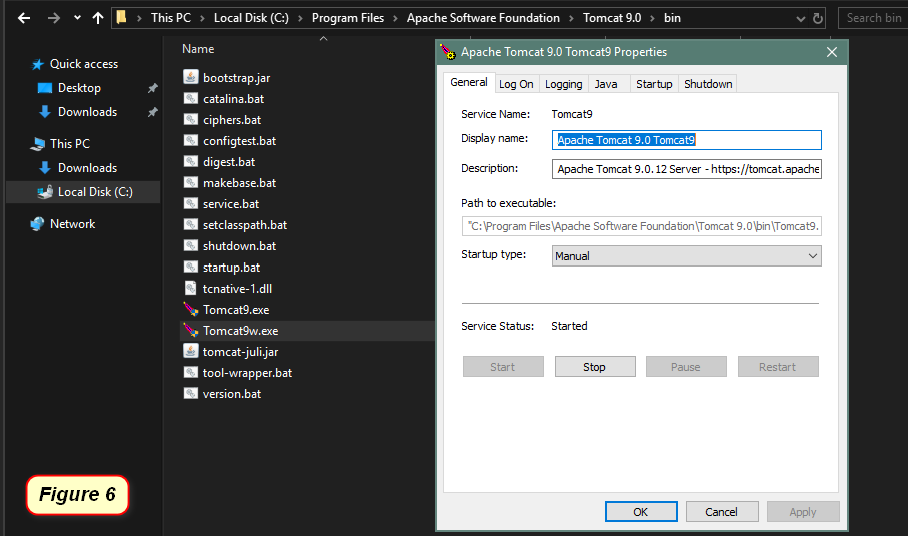
These deployments are called artifacts, they are the layout of our project output, that contain our structural elements of the applications, such as: compilation output for our modules, libraries included in module dependencies, collections of resources (web pages, images, descriptor files, etc.) and individual files, directories and archives. Therefore, they need to be configured before the application can be deployed to the Apache Tomcat server.

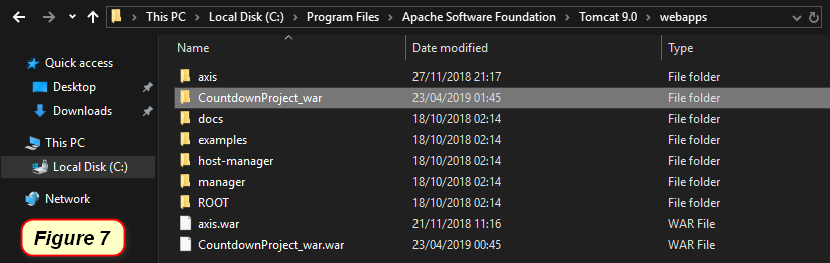
IntelliJ allows as to configure an artifact from existing project, by clicking on the project structure and specifying the artifact type, name, and output directory. In our case we have created both exploded artifact and web archive artifact (war).

The exploded artifact allows us to run application in our IDE(IntelliJ) that is automatically linked to Apache Tomcat Server as we added it to the project configuration (figure 3, 4).

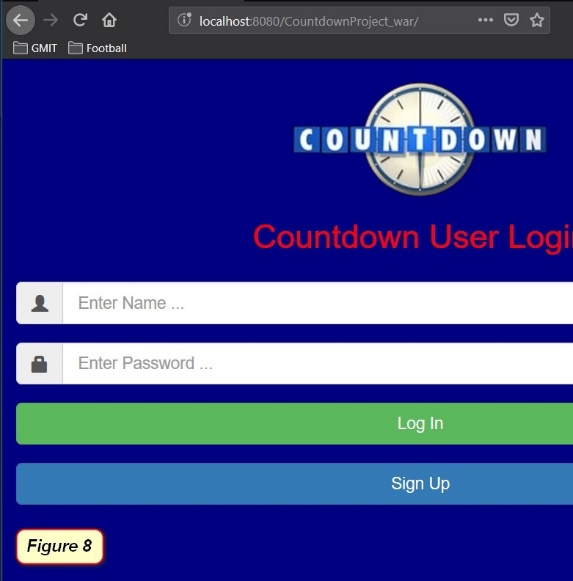


The web archive artifact can be used as standalone package that we copy to our Apache Tomcat Server webapps folder (figure 5) and start the Tomcat9w.exe service located in bin folder (figure 6).



Once the Tomcat9w.exe service is running, we go back to the Tomcat webapps folder and we can see that Tomcat deployed the application automatically (figure 7).

For starting the application, we can now open the browser and navigate to address bar and type: [http://localhost:8080/CountdownProject\_war](http://localhost:8080/CountdownProject_war/) (figure 8).

[](http://localhost:8080/CountdownProject_war)

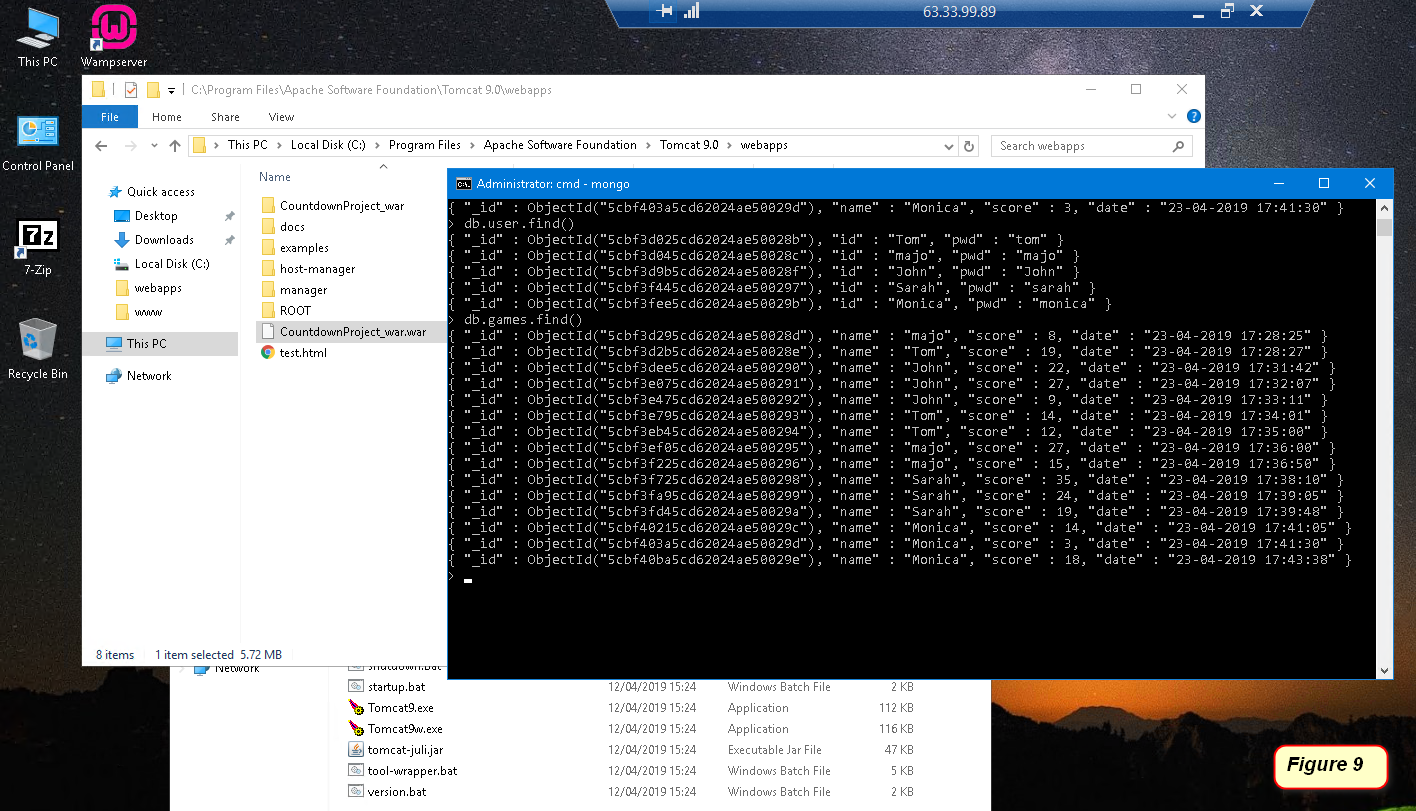
**Web hosting -**

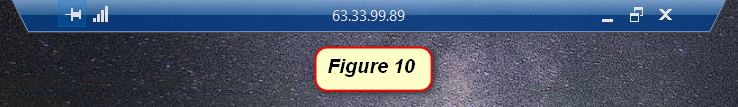
Web hosting is a service that allows organizations and individuals to post a website or web page on to the Internet. A web host, or web hosting service provider, is a business that provides the technologies and services needed for the website or webpage to be viewed in the Internet. Websites are hosted, or stored, on special computers called servers.

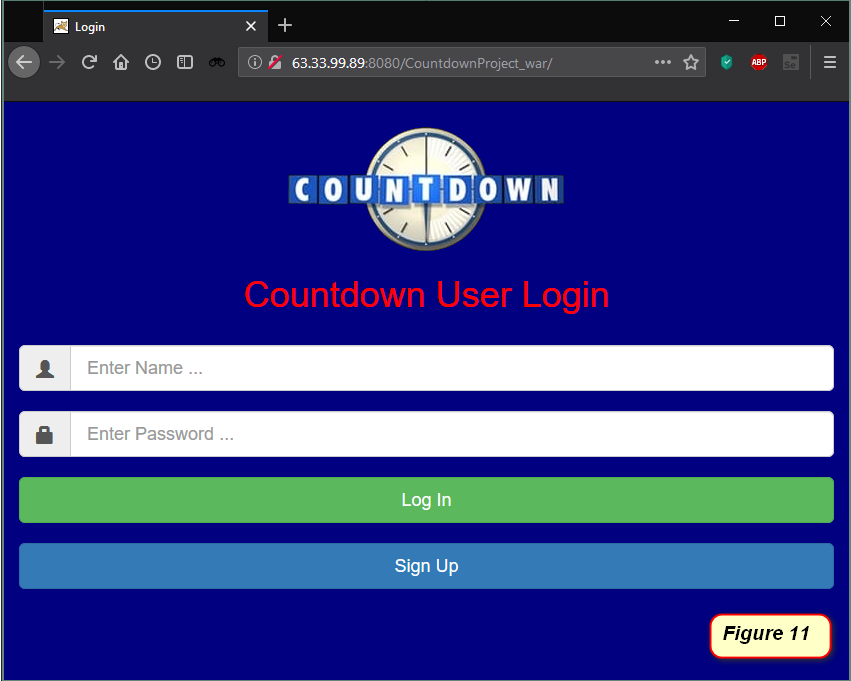
**Amazon Web Service -**

We chose Amazon Web Service as our cloud hosting service for our application, because it offers low cost, high levels of reliability and fast performance. It provides us with complete control of our computing resources and lets us run our application on Amazon’s proven computing environment. It allows to run our application with the same level of speed and performance as we would run it on our local computer, but with the advantage of multiple user to be connected at the same time to play the game. Our application is not going to be used for any commercial purposes so other great advantages of Amazon service such as networking, managing storage, resizable compute capacity and scalability is not going to be an issue.

**Amazon EC2 -**

We created an EC2 virtual server instance, which is a virtual server in Amazon’s Elastic Compute Cloud for running applications on the Amazon Web Services infrastructure. We installed Microsoft Windows Server 2016 as an operating system and then installed all required software that is needed for running the application, such as Apache Tomcat, Java SE Development Kit 8, mongoDB database and transferred our application over to the server (Figure 9). We also had to make some additional settings in the Amazon security policy and server firewall inbound rules to allow the server to communicate with the application through the HTTP 8080 port, which is the default port used for a personally hosted web service.

We can see that the IP address for accessing the server is showing on the top of the server screen that is accessed through the personal computer (Figure 10).

[](http://63.33.99.89:8080/CountdownProject_war)The steps for running the application on Amazon MS Windows Server 2016 are the same as for running it on the personal computer described above, but instead of accessing it locally, we navigate to the browser address bar and type: <http://63.33.99.89:8080/CountdownProject_war> (Figure 11).

Methodology

Development methodology

Version control

Testing methodology

*A close up of a logo

Description automatically generated*

For our application we used black box testing, which is a software testing method that examines the functionality of an application without peering into its internal structures or workings.

Simulating lost connection to mongoDB database

By default, the mongoDB starts running at the computer start-up, we can simulate database failures scenarios by turning it off in computer services. This allows us to show what may happen in the case of the database going down unexpectedly during the game.

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

For the login page we have tested all possibilities of:

* Entering a correct name and correct password
  + **Result** – User’s name and login is checked against the database and the user to redirected to the welcome page to start the game.
* Entering a correct username but an incorrect password
  + **Result** - User is not allowed access to game and is kept on the current page with an error message, “Please enter correct Login details”.
* Entering an incorrect username but a correct password
  + **Result** - User is not allowed access to game and is kept on the current page with an error message, “Please enter correct Login details”.
* Leaving the Username and password fields empty
  + **Result** - - User is not allowed access to game and is kept on the current page with an error message, “Name and Password cannot be empty”.
* If the database is unreachable
  + **Result** – The user won’t be allowed to login or sign up so will be unable to progress to the game and will see the error message “Database Connection Error”.
* Clicking on sigh up button will redirect the user to sign up page
  + **Result** - The user is redirected to sign up page in all test instances

For the signup page we have tested all possibilities of:

* Entering an existing name with any password
  + **Result** - the database is checked for usernames of the same name and when the record is found the user is kept on the current page with the error, “User with this name already exists”.
* Entering an existing name with no password
  + **Result** - the database is checked for usernames of the same name and when the record is found the user is kept on the current page with the error, “Name and Password cannot be empty”.
* Entering a new username and a password
  + **Result** - User is successfully signed up and their name and passwords are recorded to the database, because after next login the user is successfully redirected to welcome page.
* Leaving the Username and password fields empty
  + **Result** - User is not allowed access to game and is kept on the current page with an error message, “Name and Password cannot be empty”.
* If the database is unreachable
  + **Result** - The user won’t be allowed to login or sign up so will be unable to progress to the game and will see the error message “Database Connection Error”.
* Clicking on back button will redirect the user to login page
  + **Result** - The user is redirected to login page in all test instances

For the welcome page we have tested all possibilities of checking:

* The login or signup name are the same as the username displayed on the welcome page.
  + **Result** -Name appears correctly in all instances.
* That user is redirected to game play page after clicking on the new game button
  + **Result** –User is redirected to game play page in all tests.

For game play page we have tested all possibilities of:

* Entering letters that appear in the random letters as a valid word and then clicking at the rules page to check to see if that score is correct.
  + **Result\_1** - We enter the word “fix” and the letters F and X are present in the random letters, so we get a result with the score 12. F is worth 4 points and X is worth 8 points, which is correct.
  + **Result\_2** –If we enter any valid word that all or some of the letters are present in the random letters, we get a result with correct points in all conducted tests.
* Entering letters that appear in the random letters as an invalid word to see if that score is correct
  + **Result** - We enter the word “hygop”, all letters are present, but the word is not a valid word, so we get a result with the score 0, which is correct.
* Entering letters that does not appear in the random letters as a valid word to check to see if that score is correct
  + **Result** - We enter the word “buy”, none of the letters are present in the random letters, we get a result with the score 0, which is correct.
* Entering no letters (leaving the field blank) to see if the score is correct
  + **Result** - We enter no word, we get a result with the score 0, which is correct.
* Entering any other characters other than words, including numbers, special character or combinations of letters and other characters to see if that score is correct
  + **Result** - Entering any special characters, such $, % @; or entering any numbers such as 1, 4, 789; or entering any combinations of word and characters such as all%, say123, rubbing3\* are considered invalid and yield 0 points, which is correct.
* Entering letters that constructed of letter longer than 10 characters to see if it is possible
  + **Result** - We cannot enter word longer than 10 characters, which is correct.
* Checking if user is redirected to result page after 30 seconds if no word is entered
  + **Result** - User is redirected to result page after 30 seconds if no word is entered.
* Checking if after no entering any word and clicking at the submit button the user is redirected to result page
  + **Result** - User is redirected to result page after entering no word and clicking on submit button., which is correct.
* Checking if after no entering any word and clicking at the submit button we get correct score
  + **Result** - User gets score 0, which is correct.

For round result page we have tested all possibilities of:

* Displaying the correct logged in or signed up name
  + **Result** - Correct user’s name is displayed in every test instance.
* Displaying the correct round number of currently played round and all subsequent rounds
  + **Result** - The round numbers are displayed correctly for current round and all subsequent rounds in every test instance.
* Displaying the correct guess word that user enters in the currently played round and all subsequent rounds
  + **Result** - Correct guess as entered by the user is displayed for current round and for all subsequent rounds in every test instance.
* Displaying the correct round score for the word entered by the user when adding up letters points as described on rules page; in the currently played round and all subsequent rounds
  + **Result\_1** - Correct round score for the word entered by the user is displayed for current round and for all subsequent rounds in every test instance.
  + **Result\_2** -If the user enters invalid word, the round score displays 0 in current round and in all subsequent rounds in every test instance, which is correct.
* Displaying the same round score and current game score for round 1 as it is the first round
  + **Result** - Round score and current game score for round 1 match up in every test instance.
* Displaying the correct current game score when adding up currently played round for all subsequent rounds
  + **Result** - Game score displays correct value, for example when current game score is 14 and currently played round is 5, it adds up to 19, which is correct.

For finalResult page we have tested all possibilities of:

* Displaying the correct logged in or signed up name
  + **Result** - Correct user’s name is displayed in every test instance.
* Displaying the correct round number of currently played round
  + **Result** - The round numbers are displayed correctly for current round in every test instance.
* Displaying the correct guess word that user enters in the currently played round
  + **Result** - Correct guess as entered by the user is displayed for current round in every test instance.
* Displaying the correct round score for the word entered by the user when adding up letters points as described on rules page for the currently played round
  + **Result\_1** - Correct round score for the word entered by the user is displayed for current round.
  + **Result\_2** -If the user enters invalid word, the round score displays 0, which is correct.
* Displaying the correct final game score when adding up currently played round
  + **Result** - Final game score displays correct value, for example when current game score from previous rounds is 20 and currently played round is 3, it adds up to 23, which is correct.
* Displaying the message “Data Saved Successfully”, when the user enters to final result page, which means data has been successfully saved to database
  + **Result\_1** - Displaying the message “Data Saved Successfully” in every test instance (tests with database services switched off are excluded in these tests).
  + **Result\_2** -When the user’s final score is high enough to be among 10 best users scores, she/he can see their name, score and date on high score page.
* Displaying the message “Data Save Failed”, when the user enters to final result page, which means database is unreachable
  + **Result** - Displaying the message “Data Save Failed” in every test instance (when testing with database services turned off).
* Clicking on log out button redirects the user to login page
  + **Result** - Clicking on log out button redirects the user to login page in every test instance.
* Clicking on new game button redirects the user to welcome page and displays the correct name
  + **Result** - Clicking on new game button redirects the user to welcome page and displays the correct name in every test instance.

For high scores page we have tested all possibilities of:

* Displaying 10 highest scores out of all users
  + **Result** - When the user’s final score is high enough to be among 10 best scores, she/he can see their name, score and date on high score page, which pushes out the 10th score to 11th place, which means it will not be displayed on the page anymore, testes successfully many times.
* Displaying 10 highest scores out of all users - sorted by score
  + **Result** – the page is displaying 10 highest scores out of all users sorted by score successfully in every test instance.

Other tests, including:

* Checking if every user is presented with correct game rounds
  + **Result** - Every user is presented with correct game rounds; tested with 2, 3 and 5 rounds.
* Checking if clicking on rules and high scores buttons in welcome page, game page (including every round), result page (including after every round) and final page display rules and high score pages correctly by opening in new tab in the browser
  + **Result** - clicking on rules and high scores buttons open rules and high score pages correctly in new browser tabs in all above-mentioned tests.
* Checking if Countdown logo is correctly displayed on every page, including rules and high score pages
  + **Result** - Countdown logo is correctly displayed on every page, including rules and high score pages.
* Checking if audio is playing in every instance of game play round
  + **Result** - Audio is playing in every instance of game play round
* Checking if timer is displaying from 30 seconds down to 0 in every game play round
  + **Result** - Timer is displaying from 30 seconds down to 0 in every game play round.
* Checking when timer reaches 0 seconds, the game play is automatically redirected to result page
* **Result** - The game play page is automatically redirected to result page after the timer reaches 0 seconds.
* Checking if timer is in sync with audio in every game play round
  + **Result** – Audio is behind in first round of the game in some tests, round 2 and round 3 are in sync in every test.
* Checking if loader screen is working on all pages
  + **Result** - Loader screen is working on all pages.
* Checking if loader icon is aligned correctly with Countdown logo when the browser is in full-screen, half-screen and minimum mode
  + **Result** - Loader icon is aligned correctly in all above-mentioned tests.

Technology Review

///////////////////TODO//////////////

Talk about spring boot and how it was set up

Maven and all the dependencies

Yarn, we used but removed it, why.

Talk about babel and webpack and webpack cli, bundle.min.js.

**Unused Technology**

This is the technology we have research and used at some of the development but has either been removed or replaced as we switched to a different structure for the App.

**JSP** - JavaServer Pages (JSP) is a technology that we decided to go with in the end. We based our decision on the following reasons

* Knowledge and experience with HTML
* Knowledge and experience with Java
* Documentation was clear to understand
* The game was programmed in Java when we were testing it out so working with JSP would be reasonably straight forward
* We had experience using Tomcat to serve up pages before thanks to a pervious module in this course

JSP is similar to PHP and ASP, but it uses the Java programming language. To deploy and run JavaServer Pages, a compatible web server with a servlet container, such as Apache Tomcat or Jetty, is required.

**JSTL** - is a standard tag library provided by Oracle to carry out common tasks (such as looping, formatting and more). It is a set of tag libraries for use on JavaServer pages (JSPs) and it is designed to simplify the JSP development.

**WhatsApp** –

We used the free to use messaging service WhatsApp to keep in contact outside of college hours about any ideas or problems we had.

**Outlook** –

We used our college issued email to set up meetings and to talk with our supervisor Martin Kenirons when we couldn’t meet or if we had to set up a different meet time.

**TeamViewer** –

We used this free software which lets one-person view or control another persons PC over the internet. This was very handy when trying to explain an idea to a teammate.

**GitHub** –

We used GitHub as our version control. We picked GitHub because we both have pro accounts from college, and this is a service we have used many times over the years in college.

**Java 8** –

We used Java as we had an idea of how we could build the game in Java from the start but we didn’t fully understand how to make it a WebApp so this presented the perfect opportunity for us to expand our Java knowledge.

**CSS** –

Cascading style sheet, we used CSS to style some of the elements in our WebApp but most of the styling present is from bootstrap

**Bootstrap** –

Bootstrap is a framework which is used for HTML and CSS based design templates. It also has support for JavaScript plugins.

**JavaScript** –

We used JavaScript to add some of the features in our App like the mp3 players which plays the countdown music as the player plays the game.

**MongoDB** –

MongoDB is a NoSQL database which uses JSON like documents. We choose MongoDB over using a SQL database because TODO

**ReactJS** –

**Yarn** –

**Springboot** –

**Maven** –

*Conclusion*

We chose countdown because we thought it was a good game to try to implement in a WebApp. The rules are easy to understand from a logic perspective and we were able to change them around a bit to offer a bit more variety in our scoring system. We felt it would be a good demonstration of the skills we have picked up over the last 4 years of college.

If we were to do this project again there are a few things we might look to change:

Gameplay

Firstly, for our scoring system we would look to improve the way we had out points. We would possibly look at implementing a multiplier based on the number of letters used from the random letters and multiply any player score by this value thus offering the players who use more letters a better chance at a higher score.

*Learning outcomes*

Better understanding of :

Java – We coded the logic of our game in Java, any part where a piece of data needs to be changed or manipulated it is done so by Java code in our app.

JSP – We used JSP as our web pages. This is what the player sees and interacts with when playing the game.

Java Servlets – We used servlets to serve up the webpages in conjunction with TomCat.

API calls in Java and Python – We have two pieces of code done that both make an API call to the online Oxford English dictionary. We got to experience the difference between making the same call in two different programming languages.

*Future Investigation*

We didn’t have time during this development cycle but in future we would look at adding a numbers game to the WebApp.

We could split the screen into two section on the welcome.jsp page

A screenshot of a cell phone

Description automatically generated

Have one side say words and one say numbers and new game buttons under each. Alternatively, we could do it more like the TV show and mix in the words and numbers rounds to give an overall score at the end.

We would need a system similar to our random letter generator where we select 6 random numbers in certain ranges, give the player a choice between large, medium and small numbers and have the range set for the large at between 100 and 999. Medium at between 10 and 99 and small between 1 and 9.

After the player picks their number, we would need a math’s formula to come up with a number for the target that the player must hit or get as close as they can.

There is a good article in the link below about the math’s of the countdown numbers game that would be very useful if we decide to develop this app further.

<http://datagenetics.com/blog/august32014/index.html>

We could also look into adding a private challenge a friend mode where you can create a game ID and when two players enter that ID, they are paired up with each other.

Appendix

GitHub URL : <https://github.com/DuffyTJ89/CountdownProject>

*References*

Limitations

<https://getbootstrap.com/docs/3.0/layout/grid/>

<http://www.howto-expert.com/how-to-get-https-setting-up-ssl-on-your-website/>

Deployment and Running

<https://www.jetbrains.com/help/idea/working-with-artifacts.html>

<https://www.jetbrains.com/help/idea/configure-web-app-deployment.html>

<https://www.jetbrains.com/help/idea/creating-and-running-your-first-java-ee-application.html>

<http://tomcat.apache.org/>

<https://tomcat.apache.org/tomcat-8.0-doc/deployer-howto.html>

<https://www.doteasy.com/web-hosting-articles/what-is-web-hosting.cfm>

<https://aws.amazon.com/websites/>

<https://aws.amazon.com/ec2/>

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>

<https://searchaws.techtarget.com/definition/Amazon-EC2-instances>

Technology Review

<https://www.tutorialspoint.com/jsp/jsp_environment_setup.htm>

<https://www.javatpoint.com/jstl>