

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](#_Toc6913189)

[Introduction 3](#_Toc6913190)

[Limitations 4](#_Toc6913191)

[Known bugs 4](#_Toc6913192)

[Research: 5](#_Toc6913193)

[How the application works 6](#_Toc6913194)

[Full run with correct words 21](#_Toc6913195)

[Running the application 30](#_Toc6913196)

[Methodology 31](#_Toc6913197)

[Technology Review 32](#_Toc6913198)

[*Conclusion* 35](#_Toc6913199)

[*Learning outcomes* 36](#_Toc6913200)

[*Future Investigation* 37](#_Toc6913201)

[Links 38](#_Toc6913202)

[*References* 38](#_Toc6913203)

*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.

Notes: Amazon Idea:

In order to deploy the app to amazon servers we could install it the same way as we use it locally i.e install eclipse, install mongoDb, install python, install java.

Or we could generate war file and execute it in command prompt on amazon server. The war file contains the web application that can be deployed on any servlet/jsp container. The .war file contains jsp, html, javascript and other files necessary for the development of web applications.

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.

Known bugs

The player can use the back button in the browser to go back in a round and re-enter a new word.

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.

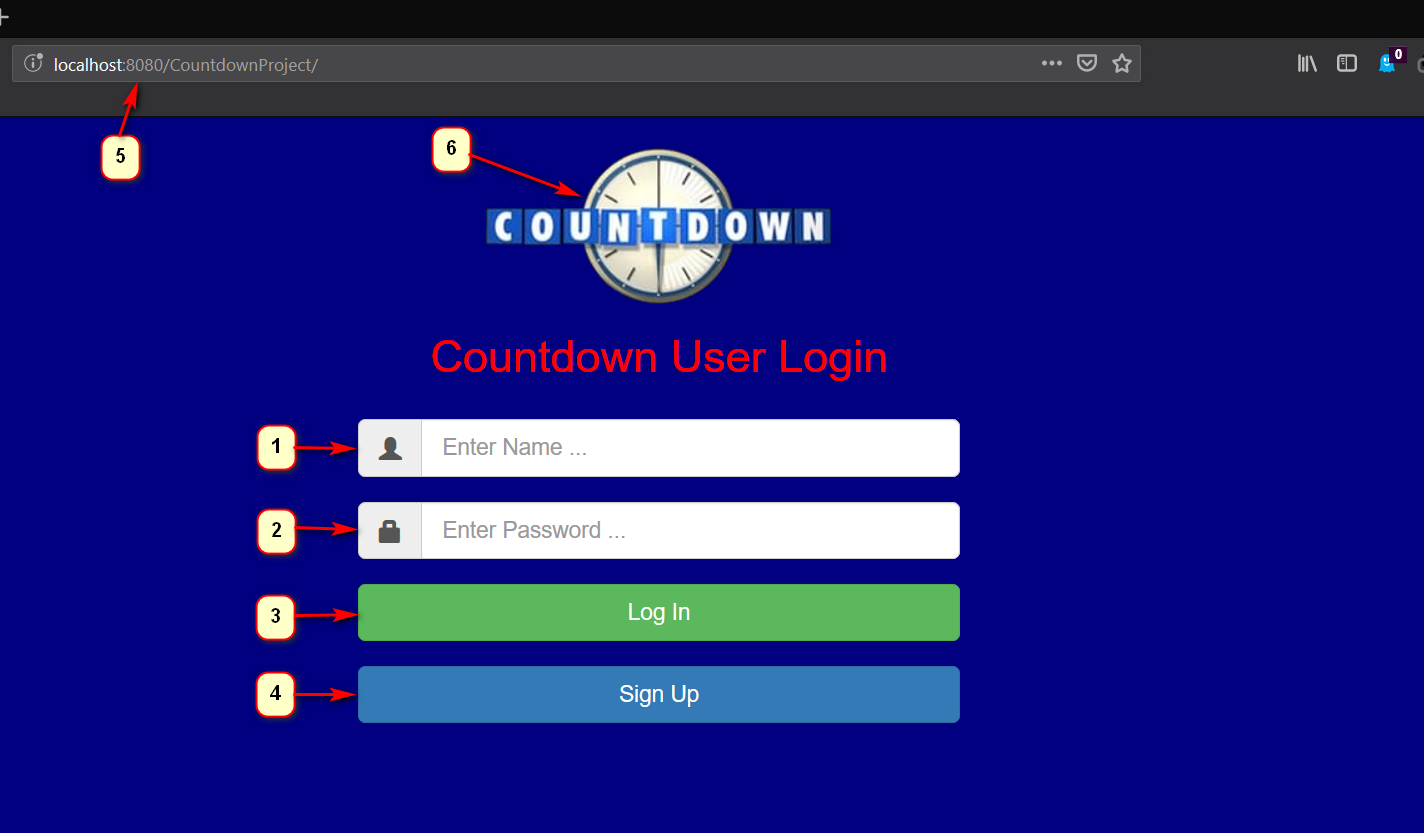
How the application works and code review

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.

The web app is hosted on Amazon web services and is reachable at the following URL:

<http://63.33.99.89:8080/CountdownProject_war/>



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.

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

A screenshot of a cell phone

Description automatically generated

1. 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.
2. 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.
3. Submit button to commit the username and password to the database
4. 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

So, 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

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

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

A close up of a sign

Description automatically generated

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.

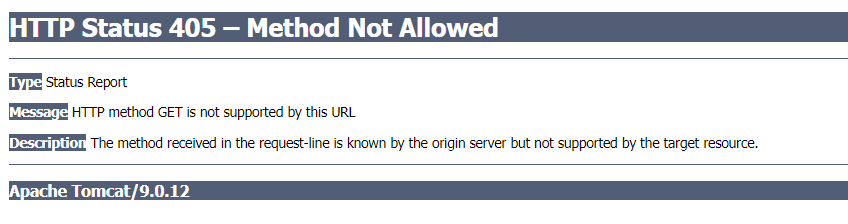
A screenshot of a cell phone

Description automatically generated

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.

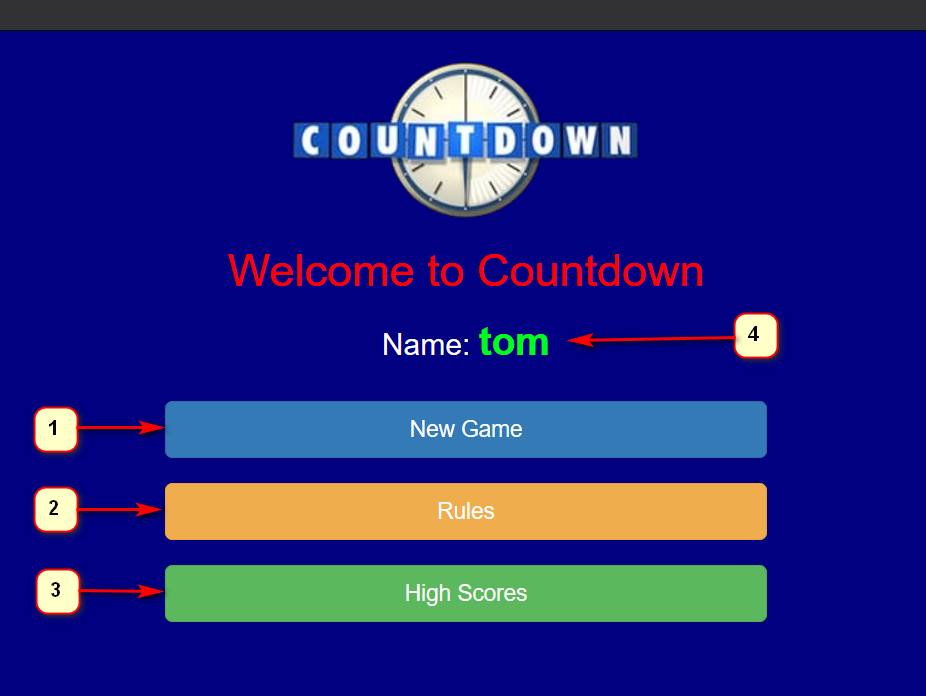
The URL of the next page is <http://63.33.99.89:8080/CountdownProject_war/UserLogin>

Trying to access that URL without login gives this error



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

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

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 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 submits their word, we take it and compare it to the random letters in the following method.

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.

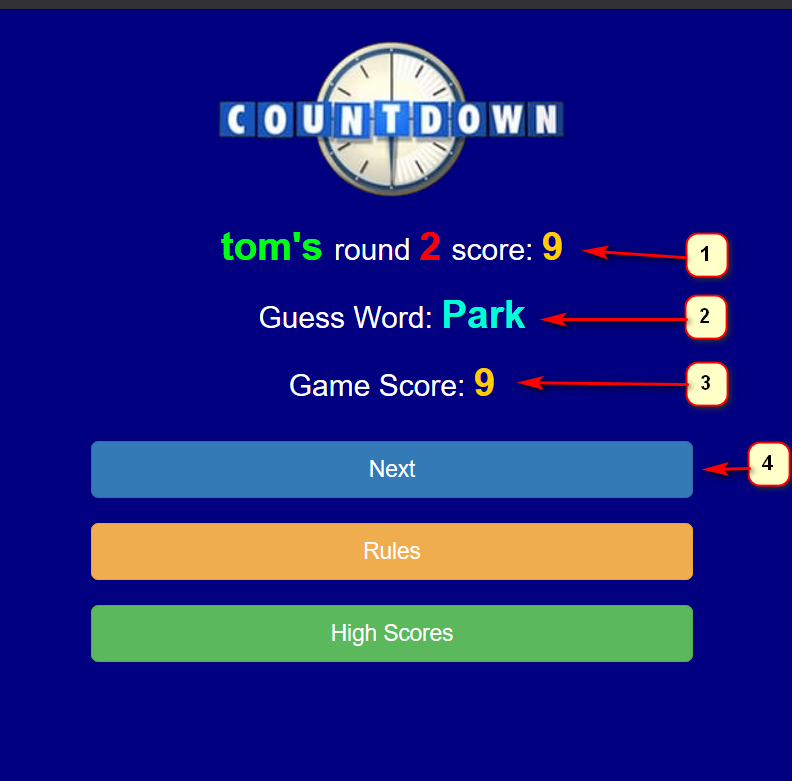
$(document).ready(function() {  
 setTimeout(function() {  
 $("#word\_form").submit();  
 }, 31000);  
});

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

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

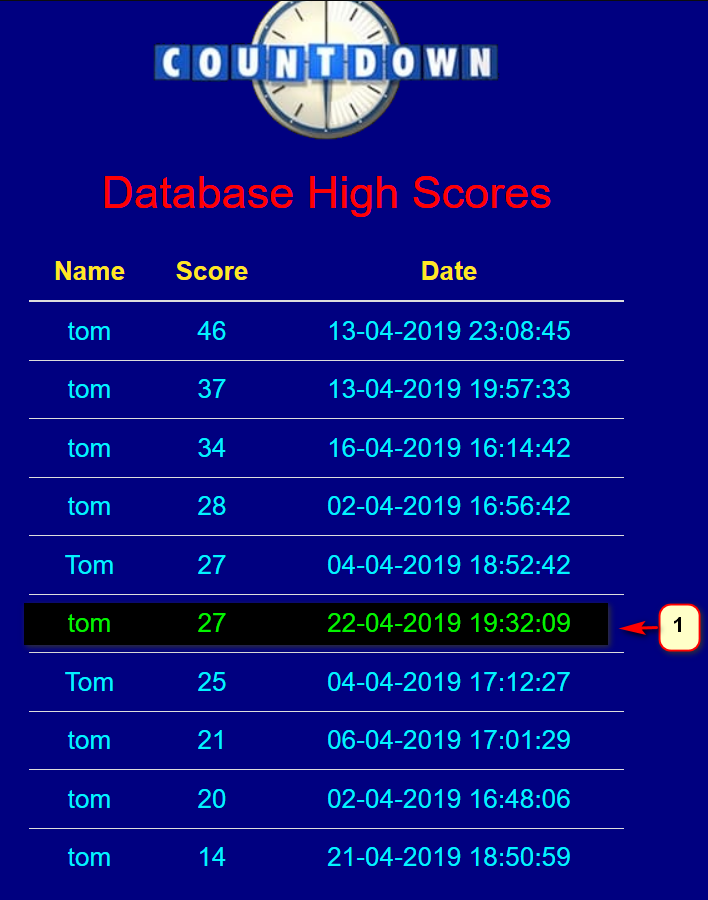
Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated



1. New record in the database

Game also runs on mobile but is not fully optimized so buttons are kind of small.

Countdown on a 5-inch android screen :

A screenshot of a cell phone

Description automatically generated

Deployment and running the application

Methodology

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.

NPM and node.js, why we use. JavaScript runtime built on chrome’s V8 javascript engine.

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

HTML, CSS we created a JS folder separately.

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.

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>

Links

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

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

*References*