

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](#_Toc534969412)

[Introduction 3](#_Toc534969413)

[Links 3](#_Toc534969414)

[Running the application 4](#_Toc534969415)

[Methodology 5](#_Toc534969416)

[Technology Review 6](#_Toc534969417)

[*Conclusion* 7](#_Toc534969418)

[*Learning outcomes* 8](#_Toc534969419)

[*Future Investigation* 9](#_Toc534969420)

[*References* 10](#_Toc534969421)

*Acknowledgements*

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 web pages locally, we also have deployed the app to Amazon server where it can be available to access from anywhere.

When the user enters a word, we use a python script to make 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 python script finds the word in the dictionary, then Java continues to score the letters used in the word.

At the end of 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 top 10 players who scored the highest.

We will cover this in more depth in the technology section.

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

Known bugs

Technologies used:

Spring boot

Java 8

Junit4

ReactJS

WebPack

Bootstrap

Yarn

CSS

Html5

JavaScript

MongoDB

Maven

Git/GitHub

Docker

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. Then we included a Python script that makes a call to the

We then use

Spring boot java server connected to mongo DB which talks to a React JS app as the root resource that connects to the HTTP API as a user.

Scrabble online is a HTTP web API that allows its users to create and play games of scrabble via HTTP requests.

There is a react JS front end served as the root resource that allows you to play and start new games while also keeping track of all in-game progress.

Java Server

Provides various HTTP endpoints which facilitate managing scrabble games programmatically via HTTP

React front end

React JS front end allows users to play and manage their ongoing scrabble game via a convenient web application.

System requirements:

Standard Modern web browser such as Chrome, Firefox, Safari and Edge etc.

Links

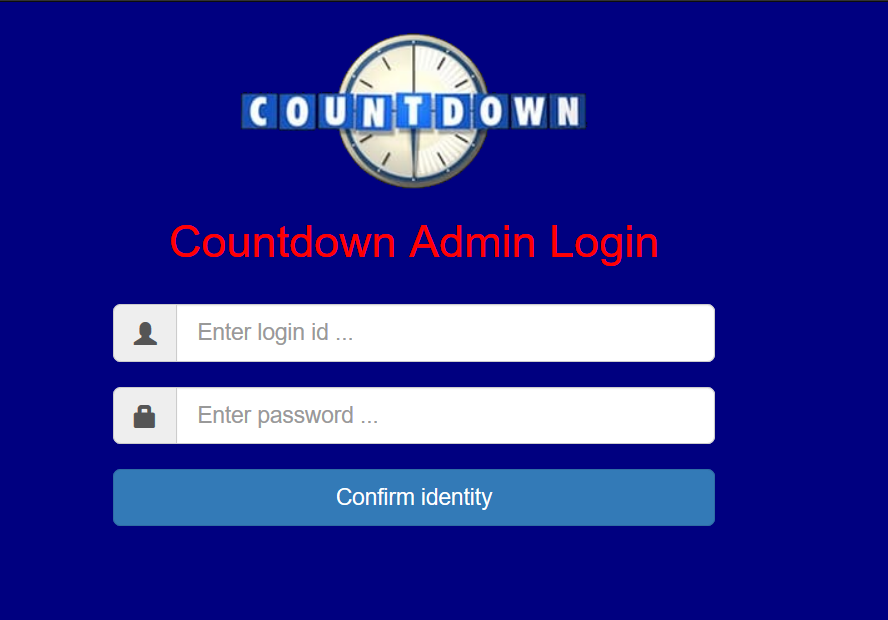
GitHub URL : <https://github.com/DuffyTJ89/CountdownProject>

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

How the application works

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.

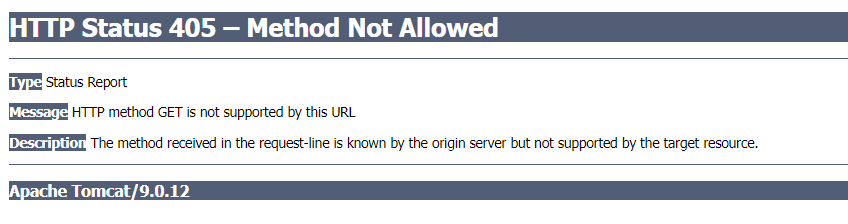


We use mongoDB as our database and in this we have users set up which can log into the system. This provides a security feature for our application. Should the user fail to enter the correct details they will be kept on the index page, if they enter a login that is kept in our database, we will progress them to the welcome.jsp page.

There is a security feature here where even if you know the URL of the next page the system will block you from accessing it if you have no provided the required login credentials.

The URL of the next page is <http://localhost:8080/CountdownProject/adminLogin>

Trying to access that URL without login gives this error



Login with the correct details and the system will progress you to welcome.jsp where you will be prompted to enter the name you want to ne known as in the game.



This name will be passed to the Java servlet and the set as a session variable to be used throughout the application. We will pass this name, timestamp and the score the user gets at the end of the game to our database so we can keep track of all the scores our users attain.

Before starting a game a user can check the rules or previous high scores by clicking either of the buttons displayed.

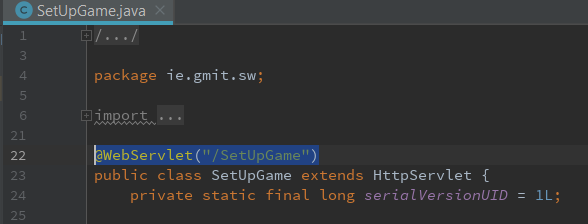
Rules opens a new window and we display to the user the rules.jsp page which contains the rules we have set out for our game.



Back on the welcome page, when the user clicks submit on their name the first Java servlet (called SetUpGame.Java) will be called by an action on the button seen in code below.



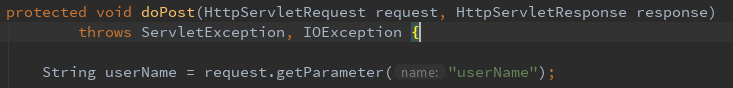
We have used bootstrap to style our pages which can be seen above. We create a user form which the user will enter their name into and when they click New Game button the action part of that form will look for a servlet which has the mapping for “SetUpGame”.



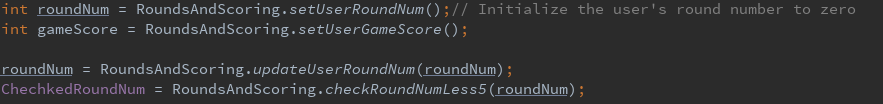
SetUpGame.Java is our first servlet. A servlet is a class which responds to a particular type of network request - most commonly an HTTP request.

@webServlet(“/SetUpGame”) matches the action we were looking for, so this servlet called SetUpGame gets used. In this servlet we make the first of our calls to the Java backend but first we deal with the username which was set up on the welcome.jsp page.

In the doPost method we have a HttpServletRequest, this is an object that has been passed as an argument to the doPost method. From this we can get the parameter passed across which is ID’d by “username”.



Next as this is called setUpGame.java it is obviously used to set up variables and counters we will need to use to get the game functioning correctly.



In the java class rounds and scoring we have methods which we use to make sure the round number (roundNum) and the game score (gameScore) are set to zero at the start of a new game.

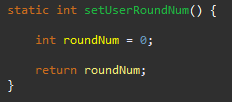
Then we can use request to get the parameter is has by assessing it through its variable name and setting it to a new variable for use later.

String yourName = request.getParameter("userName");

Next, we call the first methods from our Java backend. These methods generally have one job and will return something back to the servlet. The first method call is on a class we use to set up and keep track of certain details during the game e.g. track a player score over the game.

int roundNum = RoundsAndScoring.*setUserRoundNum*();

This method initializes the user’s round to zero at the start of the game, inside the RoundsAndScoring class it uses this setUserRoundNum method:



This is important because we play the game for 5 rounds so we need to keep track of the rounds throughout the game. We start the user out on zero and every time a round is started, we use another method in this class which we will see later to update the current round number.

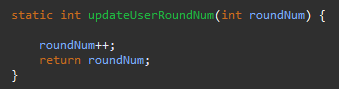
We do the same as above with the user’s gameScore. This is how we will keep track of the amount of points the player has scored while playing the game.

int gameScore = RoundsAndScoring.*setUserGameScore*();

Next, we make our first call to update the user round as we will soon be displaying the jsp page for the first round back to the user. We pass the roundNum parameter into the method, currently roundNum would be zero.

roundNum = RoundsAndScoring.*updateUserRoundNum*(roundNum);

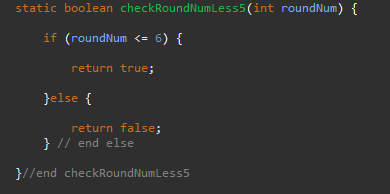
This calls the method below which increases the roundNum by 1.



Then we have a method which checks if the roundNum is less than or equal to 5 because after 5 rounds we want to end the game.

ChechkedRoundNum = RoundsAndScoring.*checkRoundNumLess5*(roundNum);

The method inside rounds and scoring looks like this



The update round method and check if its in the 5 rounds will be called at every round in the app.

We use the returned value (either true or false) from checkRoundLess5 to determine what should be displayed to the user next. If we are in the 5 rounds, we want to generate the random letters and display that to the player.

First, we check if we are inside the 5 rounds with an if statement

if (ChechkedRoundNum == true) {

This returns true until the last round so we move onto generating the random letters. The random letters are handled in a method in the class file GenerateRandomLetters.

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

The class file looks like this:



Going through this method we can see what each line does.

String[] abArr = { "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" };

This creates an array of strings which are just single letters from the alphabet.

We put them into a List of strings so we can use Collections with the list below.

List<String> abList = Arrays.*asList*(abArr);

This method acts as bridge between array-based and collection-based APIs, in combination with Collection.toArray(). The returned list is serializable and allows us to use RandomAccess.

We can then shuffle this list using Collections.shuffle. Java docs describes the method as follows

“Randomly permutes the specified list using a default source of randomness”.

We need to randomly jumble up our letters before selecting 10 to display so this is how we will do it.

Collections.*shuffle*(abList);

We then need to get a smaller version of the randomized list because we want to only display 10 letters to the user and the current list has the full alphabet randomized. We use subList and ask for the letters in the 0 position up to but not including the 10th position.

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

Running the application

Methodology

Technology Review

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.

*Conclusion*

We selected scrabble as we wanted ourselves and make use of all the skills we have learned during our time in GMIT. As the project progressed, we realised we needed to experiment with technologies and research them to figure the best way of implementing our idea.

The technologies we settled on where the ones we found were best suited for our projects. React and Node.js to develop the backend instead of Spring boot. The reason for this was needing to restart the server any time a change was made to the UI slowed us down greatly.

This is a project where it is hard to know where to finish as you can almost always improve it in some way. In the end we decided to stop at

*Learning outcomes*

*Future Investigation*

*References*