

Assignment 4: Online Hangman Game (25% of course total)

- Due date is **7 August, 11:59pm**. Refer to Canvas for any updates.
- Submit deliverables to the corresponding assignment folder on Canvas.
- Make sure you also complete the corresponding online quiz at CourSys to receive full portion of the coding part. Refer to Course Outline and Marking Guide for details.
- Late penalty is 10% per calendar day (each 0 to 24 hour period past due), max. 2 days late (20%)
- This assignment is to be done individually. Do not show another student your code, do not copy code from another person/online. Please post all questions on Canvas.
- You may use general ideas you find online and from others, but your solution must be your own.
- You may use any tool for development. But we will be grading your submission using IntelliJ IDEA Community, so make sure your code compiles and runs there. We support only IntelliJ.

Description

In this assignment you are going to create an online html-based Hangman game using Spring Boot and Thymeleaf (a template engine that simplifies building of web applications). Players can go to a specific webpage using their web browser to play the game ([https://en.wikipedia.org/wiki/Hangman_\(game\)](https://en.wikipedia.org/wiki/Hangman_(game))).

Your system (the game server) must respond to the following path requests from the player (browser):

URL	Description
localhost:8080/welcome	Display the “welcome.html” webpage and include the following: <ul style="list-style-type: none"> • Title of the webpage: “CMPT 213 Online Hangman Game”. • A welcome message. • Information about you as the creator (name, id, email). • Rules of the game. • A button redirecting the player to the “game.html” webpage (using the “th:action” attribute tag of Thymeleaf).
localhost:8080/game	Display the “game.html” webpage and include the following: <ul style="list-style-type: none"> • Title of the webpage: “CMPT 213 Online Hangman Game”. • A message indicating the ID of the current game. • The status of the game (one of: Active/Won/Lost). • The progress of the game, starting with all “_” (spaced appropriately so it’s clear to the player how many letters are in that word). Review the letter(s) that the player has guessed correctly. • Information about the game (i.e., number of guesses made, number of incorrect guesses). • A way for the player to guess a letter (e.g., a textbox to fill in the letter and a button to confirm). • A link taking the player back to the “welcome.html” webpage. This allows the player to start a new game. <p>*note that this page is supposed to be redirected from the “welcome.html” page through a POST request. Directly typing this URL is a GET request and your code doesn’t have to handle it.</p>

localhost:8080/game/id	<p>Display the “game.html” webpage with the game indicated by id refer to the localhost:8080/game entry for what to include.</p> <p>The system will only check for non-existing id. If that happens, throw a <code>GameNotFoundException</code> (create one yourself by extending <code>RuntimeException</code>) and respond with status 404. Also redirect the player to the “gamenotfound.html” and include the following:</p> <ul style="list-style-type: none"> • A message indicating the game is not found. • A link taking the player back to the “welcome.html” webpage, allowing the player to start a new game.
-------------------------------	---

More details:

- Everytime the player is redirected from the “welcome.html” to the “game.html” webpage, a new game (with a different ID) is created along with it a new word is randomly selected.
 - The random selection of a word is achieved by the game server loading a text file of English words and randomly pick a word from them (see the provided file, which should be placed inside the `src/` folder along with `main/` and `test/`).
- Each game is modelled by a Game object, which includes at minimum the following information:
 - ID of the game. Generated by the controller using `AtomicLong` (like Assignment 3).
 - The word the player is guessing.
 - Number of guesses made by the player since the creation of the game.
 - Number of incorrect guesses made by the player since the creation of the game.
 - Status of the game, which is one of: Active/Won/Lost.
- The player will be redirected to a “gameover.html” webpage the next time they make a guess and they win or lose, showing them the appropriate message (win or lose) along with the correct word for that game. Note that if the player goes to an inactive game (already won/lose) using **localhost:8080/game/id**, they will be redirected to the “gameover.html” webpage.
- The player can guess a correct letter multiple times, which will just add to the number of guesses. However, each time an incorrect letter is guessed, it will be counted towards the number of incorrect guesses, regardless of whether this letter has been guessed before. **Empty guesses are not counted towards any guesses.**
- The max number of incorrect guesses is 7 for each game.
- Do not use any Javascript in this application.

Using Spring Boot and Thymeleaf

Spring Boot is a tool for Java to quickly program a web service.

You will be using the Gradle build system in IntelliJ to import all the necessary dependencies for Spring Boot to work. In the build.gradle file, use the following code*:

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:2.2.2.RELEASE")
    }
}
```

```
plugins {  
    id 'java'  
    id 'org.springframework.boot' version '2.2.2.RELEASE'  
    id 'io.spring.dependency-management' version '1.0.7.RELEASE'  
}  
group 'OnlineHangman'  
version '1.0-SNAPSHOT'  
sourceCompatibility = 1.8  
  
repositories {  
    mavenCentral()  
}  
dependencies {  
    compile("org.springframework.boot:spring-boot-starter-web")  
    compile("org.springframework.boot:spring-boot-starter-thymeleaf")  
    compile("org.springframework.boot:spring-boot-devtools")  
}
```

*Note that in the current IntelliJ version (2020.1) the default gradle wrapper version is not compatible with JDK 14. To keep using JDK 14 as with other projects of the course, first follow the instructions at the 1:00 mark of the “Setting up Spring Boot in IntelliJ” video to download the latest gradle (6.5). Once you have the gradle project created in IntelliJ, go to File > Settings > Gradle to point your project to the new gradle, along with using JDK14 for Gradle JVM. After that you will have to manually create a source directory (src) in the project by right-clicking the project name and choose New > Directory, which brings up four directories (src/main/java, src/main/resources, src/test/java, src/test/resources). Choose all four of them for completeness, and read this for reference: <https://stackoverflow.com/questions/49710330/src-folder-not-created-when-creating-simple-intellij-java-gradle-project>. Lastly, set the content of the build.gradle file as shown, and click on the reload icon (or by opening the Gradle panel on the right) to load all the Spring Boot resources.

Then follow the instructions at the 6:00 mark of Dr. Brian Fraser’s “Setting up Spring Boot in IntelliJ” video (<https://www.youtube.com/watch?v=he63dwZdhOM>) to create an Application class and onwards.

This time we will be adding HTML formatting functionality to Spring Boot. To do so note the additional dependency of “org.springframework.boot:spring-boot-starter-thymeleaf”. In addition, to help with auto-reloading UI in browser, we add one more dependency “org.springframework.boot:spring-boot-devtools” along with the following settings in the “src/main/resources/application.properties” file and a plugin to perform live-reload (this auto-reload part is however optional).

Another change you are going to do is to use the @Controller instead of @RestController when creating the controller class. This is because @RestController causes the Spring Boot to do some auto serialization with the return objects, which will interfere with Thymeleaf redirecting the response to the html page.

The Model-View-Controller (MVC) Design Pattern and The Thymeleaf Engine

One of the main learning objectives in this assignment is to practice the MVC design pattern, with some class design thinking you need to perform. Hence, we are only going to provide you with some high-level requirements for the game. The “Required Object Structure” section below will give you a head start with the pattern, but you are expected to complete the rest.

The Model component is the part where the system represents the data independent on the user interface. Most of the game logic (e.g., check if a guess is correct or not) should be done there.

The View component is the part where information about how the system presents the data to the user. In this assignment this is going to be the html webpages.

The Controller component is the part where the handling of commands is done. In this assignment it takes in the HTTP requests from the view, converts them into method calls to the model, and relays the results back to the view. This is supported by Spring Boot using the `@Controller` annotation.

Lastly, to make it a bit more fun/challenging we have added in the use of Thymeleaf so you get to learn about this engine by looking for references and APIs. In addition, **some of the marks will go to how well you have created the interface of the game**. Here are some tips about Thymeleaf:

- Thymeleaf uses html templates to build the webpages when responding to requests. Put the html webpages in the “templates” folder under the “resources” folder (which is at the same level as the java folder). For CSS files and other files such as images, put them in the “static” folder (see Figure 1).
- Thymeleaf provides access to the resources through the Controller component by using attributes. The basic idea is for each controller method you pass in a Model parameter and add objects as attributes. Then at the html template you can use tags such as “th:text” and “th:ref” together with `@{}`, `${}`, and `*{}` to specify what you want to access.
 - The reverse is also true. That is, your controller can also access information from the html templates. This is achieved by passing in an object (annotated with `@ModelAttribute(“name”)`) to the controller method. With the getter and setter methods implemented in that object, your controller method can then use the getter to retrieve the values submitted through an html form by a POST request. For details in form handling, refer to “Form handling using Spring Boot with Thymeleaf” in the “Useful Resources” section.

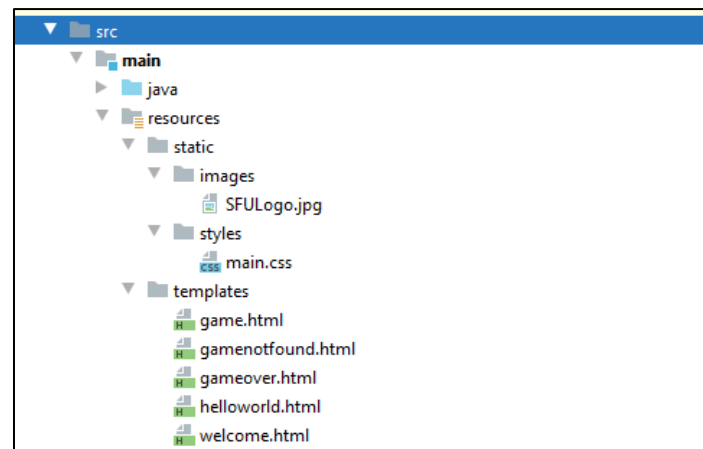


Figure 1. File structure of webpages using Thymeleaf.

To help you start using Thymeleaf, we have created a skeleton project (only the source code and other necessary files under the `src/` directory provided to illustrate the file structure). Pay attention to how the HangmanController class works as the Controller that interacts with the Message class (Model) and the helloworld.html template (View). Note how the message object is passed and accessed as a Thymeleaf attribute.

Required Object Structure

Your code must be structured as follows:

- Use a package hierarchy:
 - Package `ca.cmpt213.a4.onlinehangman`
 - The `Application` class acting as the starting point of the program.
 - Package `ca.cmpt213.a4.onlinehangman.controllers`
 - The `HangmanController` class for handling all the get/post requests
 - The `GameNotFoundException` class for representing the exception.
 - Package `ca.cmpt213.a4.onlinehangman.model`
 - The `Game` class representing a game, along with any classes that support the function of the application.
- Put View components in the resources folder hierarchy:
 - Folder `resources/static/images`
 - All the images you want your game to display.
 - Folder `resources/static/styles`
 - All the CSS files you want your game to use.
 - Folder `resources/templates`
 - All the html files your game is going to display.

Overall Coding Requirements

- Your code must conform to the programming style guide for this course; see course website.
- All classes must have a class-level JavaDoc comment describing the purpose of the class.
- Include your student information (name, student ID, email) as comment in all files.

Submission Instructions

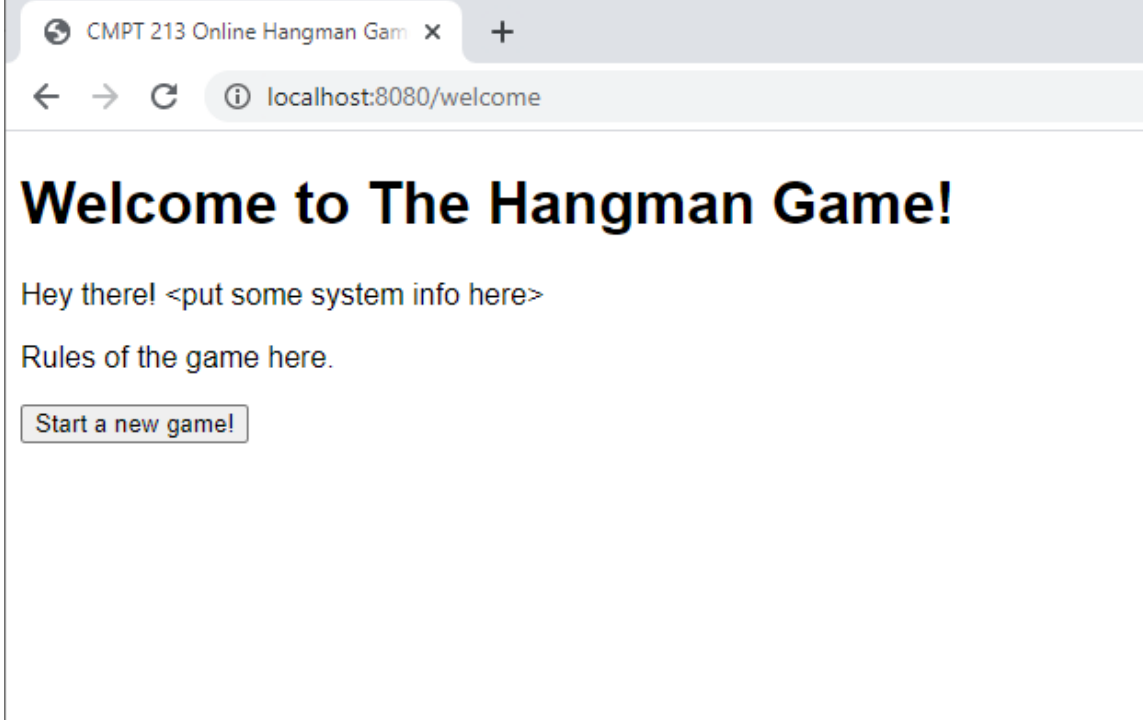
- Submit a ZIP file of your projects to the corresponding folder on Canvas. Name it using this format: **<firstname_lastname>_<studentID>_Assignment4.zip**. Note that you don't need to include '<' & '>', and studentID is the 9-digit number on your student ID card. See course website for directions on creating and testing your ZIP file for submission. If you have any difficulties in submitting your file on Canvas, email it to the instructor.
- Save a copy in a secure location for safe keeping and do not modify it after the deadline.
- If you use any libraries they have to be included in the project as well.
- All submissions will automatically be compared for unexplainable similarities

Useful Resources

- Creating a Gradle project in IntelliJ: <https://www.jetbrains.com/help/idea/gradle.html>
- Form handling using Spring Boot with Thymeleaf: <https://www.codejava.net/frameworks/spring-boot/spring-boot-thymeleaf-form-handling-tutorial>, <https://atacomsian.com/blog/spring-boot-thymeleaf-form-handling>
- Setting up Spring Boot Devtools (optional): <https://howtodoinjava.com/spring-boot2/developer-tools-module-tutorial>
- Setting up the LiveReload plugin (optional): <http://livereload.com/extensions/>

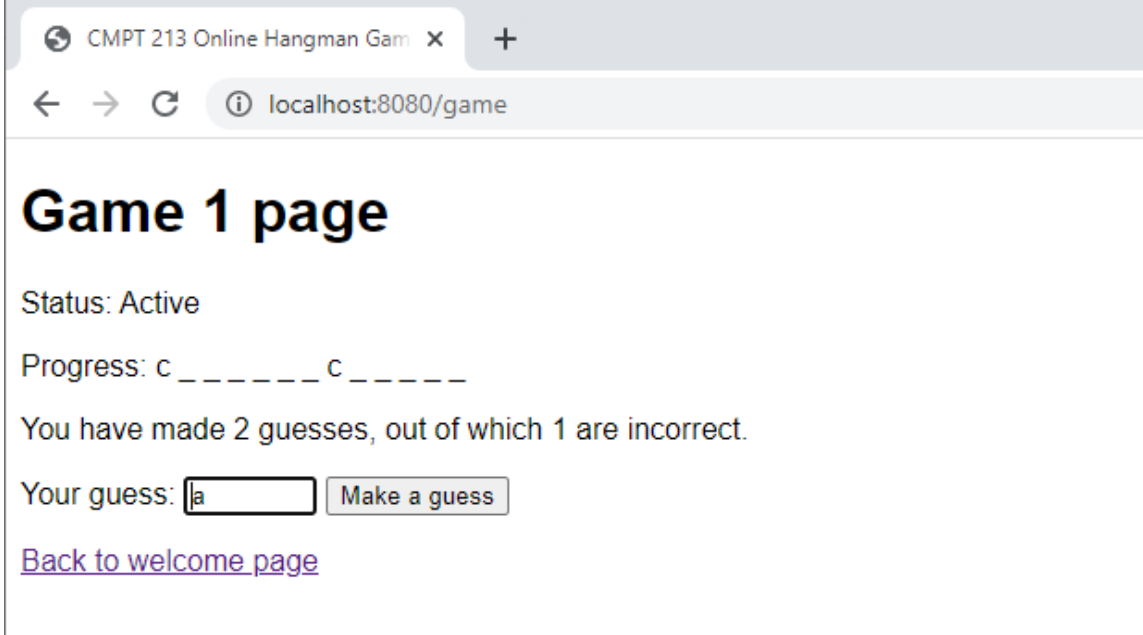
Sample Input/Output

Note: This is bare minimal. You are supposed to make it look much better.



The screenshot shows a web browser window with the title "CMPT 213 Online Hangman Gam" and a single tab. The address bar shows "localhost:8080/welcome". The page content includes a large heading "Welcome to The Hangman Game!", a greeting "Hey there! <put some system info here>", a line "Rules of the game here.", and a button labeled "Start a new game!".

Welcome page of the game.



The screenshot shows a web browser window with the title "CMPT 213 Online Hangman Gam" and a single tab. The address bar shows "localhost:8080/game". The page content includes a heading "Game 1 page", the status "Status: Active", the progress "Progress: c _ _ _ _ _ c _ _ _ _ _", the text "You have made 2 guesses, out of which 1 are incorrect.", a form for "Your guess:" with an input field containing "a" and a "Make a guess" button, and a link "Back to welcome page".

Player guessed 'c', and then 'a' (ignore the singular and plural here).