Hangman Challenge

Last updated June 28, 2017

Instructions

Bad Rabbit is providing you with this software development challenge as a paid project with a budget of 6 hours. This development scenario is intended to be a simplified, realistic sample project testing for the attributes and aptitudes of a senior-level software engineer equipped to face modern challenges with a web-based approach.

Part of the rationale for this pre-employment challenge is to test your attention to detail and capacity to work independently. **Please read carefully!**

1. **Read** and understand this challenge and its requirements. Ask for clarification if needed.

2. **Design** a solution to meet all of the requirements. Please create at least a short write-up articulating your design ***before*** you begin programming.

3. **Implement** your solution using any programming language with a freely-available cross-platform compiler/interpreter/runtime environment. We need to be able to run it.

4. **Validate** your solution by writing a quick explanation of how your solution meets each numbered requirement from the "Requirements" section of this document.

5. **Submit** your design document, source code, and validation at least 24 hours before your in-person interview.

6. **Present** your design--including your written design document--and a working demonstration of your code at your in-person interview. Please host it on a publicly-accessible web server if you have time to do so.

If you have not fulfilled any of the requirements, include a plan and time estimate for extending your solution to fulfill the unmet requirements in your presentation.

Overview

Implement a hangman game that can be played in a web browser. Use the "**Rules**" from this document. The game must be implemented in a manner that is secure against tampering and inspection (see "**Requirements**" for specifics).

Rules

1. Randomly choose a word from the list: RABBIT, BUNNY, CARROT, LETTUCE, BURROW, FLUFFY, FLOPPY, LITTER, PELLETS. Do not re-use a word that has been used by a prior game in a series.

2. Display a blank line for each letter in the secret word

3. Display the number of guesses remaining (up to eight)

4. The player may guess single letters or, **once per game**, the entire word.

5. On the eighth incorrect guess the game is over.

6. When the player guesses all letters (or the correct word) they have won.

Requirements

1. **Rules**: Implement the game according to the "**Rules**" above.

2. **New Game**: Present a button to start a new game with a new random word after the player wins. Do not reuse words. When all words have been guessed, the game is over!

3. **Bookmarkable Turns**: A player must be able to bookmark a particular turn and use that bookmark return to that same turn and resume guessing. In this way, players can sort of 'cheat' by testing a different path, but this is ok.

4. **Sharing**: Players must be able to share a game with friends by sending the URL for the current turn. For example, a player might share the current URL and say, "I was playing hangman and I only had two guesses left. Can you solve it?" The original player should then be able to continue playing without impact on the URL shared. The recipients of the URL must be able to start playing from the point in the game that the URL was copied and sent.

5. **Undo**: There must be infinite, non-destructive undo. Unless there is no previous turn, clicking the Undo button will redirect the browser to the URL for the previous turn (even if that turn was made by a different player as in a shared game). Undo must go back to previous words (if any) and work both with single-letter and whole-word guesses.

6. **Standards**: The game **must** be playable using **any** standards-compliant web browser.

7. **Security**: The game **must** be secure against ***all*** forms of client-side tampering. It **must not** be possible for a malicious user to determine what word is being guessed or what letters (or past words) have been guessed other than through the intended user interface. When a player shares a URL it **must not** be possible to tell ***anything*** about the game from the URL. That would ruin the surprise!

8. **Proxy-friendly**: Your solution **must** work when run behind a caching HTTP reverse-proxy that is listening on port 80 on the same server.

9. **No other software/storage**: Your server-side solution **must not** rely on any other software running on the server other than the HTTP reverse proxy. Your game **must** work without the proxy. **Do not** use databases or disk storage. It’s OK to lose everything when the server process exits.

We hope you have fun and look forward to meeting you in person!