Hangman

Please implement a game of hangman, which can be played by a user "against" the computer. Hangman is a game where the **secret-keeper** (in this case, the computer) thinks of a word, and the **guesser** (the user) tries to guess it one letter at a time. The **guesser** has six guesses. If the **guesser** guesses a letter which is part of the word, the **secret-keeper** will reveal all occurrences of that letter in the word. If the **guesser** guesses a correct letter such that all letters are now revealed, the game is over and player 2 has won. Instead if player 2 runs out of guesses before the whole word is discovered, the game is over and player 1 has won. Check out https://en.wikipedia.org/wiki/Hangman_(game) for more details.

Game rules

- At the start of the game the computer/secret-keeper will choose a dictionary word
- The guesser loses the game if they guess 6 letters that are not in the secret word
- The guesser wins the game if they guess all letters in the secret word correctly and have not already lost the game per the conditions above

User Interface

Any type of user interface is acceptable (command line, mobile app, web page etc) but the player must have a way of interacting with your game including:

- The length of the secret word is displayed to the guesser (e.g. as a set of underscores)
- As the guesser makes correct guesses, occurrences of the guessed letter in the word are shown while unknown letters are still hidden
- The number of guesses remaining is displayed
- A list of incorrect guesses are displayed

Implementation

- Your program must retrieve a dictionary list of words from the word dictionary REST API provided (see attached documentation)
- You can choose whichever combination of programming languages, tools, frameworks, and libraries you find appropriate within reason (e.g. you can’t use a game framework that implements Hangman)

Extension

Apart from the above requirements, everything else is up to you. We also encourage you to think of these requirements as a starting point, and just use your creativity/imagination to expand the game in any way that you want to showcase your talents and passion for programming. Sample extension ideas include:

- Track users/scores over time and show a leaderboard
- Add support for guessing full words instead of just letters one at a time, and count those against the guesses total
- Add support for phrases instead of just words, and numbers in addition to letters
- Show an actual hangman diagram that gets filled in as the user guesses incorrectly
- Add a configurable “difficulty level” and adjust the words that are used based on the user’s preference
- Anything else that you come up with to make the game more fun/interesting!

Submission

Please submit a compressed folder containing your code, any resources/assets that your code needs, and a README file. This README should explain how an interviewer could run your code, document your thought process and/or code structure, and describe any creative extensions attempted or implemented. There is no prescribed format for the README, but it should be clear and unambiguous in listing all the steps in building, running, and playing the game you built (you should make no assumptions about what software the interviewer has, and err on the side of being explicit). Your interviewers will be engineers, so you can assume a certain level of technical ability as relates to installing what your project requires. Please send your compressed project to **REACH@linkedin.com.**

Hangman APIs

The following APIs are provided for your use when creating your hangman game. The only API required in your game is one call to GET ALL on the word dictionary API. You can use the others at your discretion.

Word Dictionary API

http://linkedin-reach.hagbpyjegb.us-west-2.elasticbeanstalk.com/words

Method	**Response**	**Example**

GET	A plain text response, with one word per line.	`foo`
		`bar`
		`baz`
		`...`
		`quux`

Parameters
(all of these are optional, and applied in the following order)

URL Parameter	**Legal Values**	**Purpose**
`difficulty`	Integer from 1-10	Filters returned words based on the difficulty level provided: 1 is the lowest level and 10 is the highest level.
`minLength`	Integer >= 0	Filters returned words to ensure they are at least as long as the provided length. Providing 0 will return all words, providing a number larger than the length of the longest word will return an empty result.
`maxLength`	Integer >= 0	Filters returned words to ensure they are shorter than the provided length. Providing 0 will return an empty result, providing a

		number larger than the length of the longest word will return all words.
`start`	Integer >= 0	Filters returned words to ignore the first “start” words in the list. Providing a number larger than the number of words that would be returned results in an empty result.
`count`	Integer >= 0	Filters returned words to return up to “count” words. Providing 0 will return an empty result.