|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name : |  |  | Date : |  |

|  |
| --- |
| **General** |

|  |
| --- |
| Skill Level   * **N** = Novice level; does not know much * **B** = Beginner level; know how to write/code/use * **I** = Intermediate level; have few years of experience coding/using and is comfortable * **A** = Advance level; expert in using/coding in (almost) all the aspects and knows the nuances |

1. What version of Java (JDK) that you have worked with?

|  |  |
| --- | --- |
| JDK Version | Skill Level (N/B/I/A) (optional: Version/Release/Years Using) |
|  |  |
|  |  |

2. What are some of the IDEs and your choice of IDE when developing Java applications?

3. What are some of the revision source control system that you have used?

|  |  |
| --- | --- |
| Name | Skill Level (N/B/I/A) (optional: Version/Release/Years Using) |
| Git |  |
| SVN |  |
| Mercurial |  |
|  |  |

4. What are some of the Java build tools that you have used?

|  |  |
| --- | --- |
| Name | Skill Level (N/B/I/A) (optional: Version/Release/Years Using) |
| Apache Maven |  |
| Apache Ant |  |
| Gradle |  |
|  |  |

5. What are some of the unit test frameworks/tools that you have used?

|  |  |
| --- | --- |
| Name | Skill Level (N/B/I/A) (optional: Version/Release/Years Using) |
| JUnit |  |
| Mockito |  |
| Selenium |  |
| JMeter |  |
|  |  |
|  |  |

6. What are some of the application frameworks that you have used?

|  |  |
| --- | --- |
| Name | Skill Level (N/B/I/A) (optional: Version/Release/Years Using) |
| Spring |  |
| Hibernate |  |
| Struts |  |
|  |  |
|  |  |
|  |  |

7. What are some of the Node.js frameworks/technologies that you have worked with?

|  |  |
| --- | --- |
| Name | Skill Level (N/B/I/A) (optional: Version/Release/Years Using) |
| Node.js |  |
| Express.js |  |
| Jasmine |  |
| Jest |  |
|  |  |
|  |  |
|  |  |

8. What are some of the JavaScript based frameworks that you have used?

|  |  |
| --- | --- |
| Name | Skill Level (N/B/I/A) (optional: Version/Release/Years Using) |
| JQuery |  |
| TypeScript |  |
| Angular |  |
| React |  |
|  |  |
|  |  |
|  |  |

9. What are some of the CSS based frameworks that you have used?

|  |  |
| --- | --- |
| Name | Skill Level (N/B/I/A) (optional: Version/Release/Years Using) |
| Bootstrap |  |
| Foundation |  |
| Tailwind CSS |  |
| Bulma |  |
|  |  |
|  |  |
|  |  |

10. What kind of JEE (now Jakarta EE) application servers or web servers that you have used?

|  |  |
| --- | --- |
| Name | Skill Level (N/B/I/A) (optional: Version/Release/Years Using) |
| Tomcat |  |
| JBoss AS |  |
|  |  |
|  |  |

11. What kind of database systems that you have worked with?

|  |  |
| --- | --- |
| Name | Skill Level (N/B/I/A) (optional: Version/Release/Years Using) |
| MySQL |  |
| Oracle RDBMS |  |
| MongoDB |  |
|  |  |
|  |  |

12. What kind of CI/CD systems that you have worked with?

|  |  |
| --- | --- |
| Name | Skill Level (N/B/I/A) (optional: Version/Release/Years Using) |
| Jenkins |  |
| SonarQube |  |
| Docker |  |
|  |  |
|  |  |

13. What kind of Cloud platforms that you have worked with?

|  |  |
| --- | --- |
| Name | Skill Level (N/B/I/A) (optional: Version/Release/Years Using) |
| Amazon Web Services (AWS) |  |
| Microsoft Azure |  |
| Google Cloud Platform (GCP) |  |
|  |  |
|  |  |

|  |
| --- |
| **Coding Tasks** |

**Guidelines**

1. Complete as many tasks as possible, within the stipulated time frame (~ 1 to 2 mandays). Candidate that can complete more tasks will be ranked higher.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task A | Task B | Task C | Task D | Result |
| Completed |  |  |  | Rank as Junior level |
| Completed | Completed |  |  | Rank as Mid level |
| Completed |  | Completed |  | Rank as Mid level |
| Completed | Completed | Completed |  | Rank as Senior level |
| Completed | Completed | Completed | Completed | Rank as Senior level |

* 1. Junior level is expected to finish "Task A".
  2. Mid level is expected to finish "Task A" and either one of "Task B", "Task C"
  3. Senior level is expected to finish all tasks ("Task B" and "Task C" are compulsory).
  4. "Task A" is the foundation/cornerstone, where the other tasks built on top on it, to provide extended functionalities.

1. After completing the code implementations, provide **only** the project source code (**minus** the binaries: \*.war, \*.jar, \*.ear), that can be built from scratch and run. Remove/delete the "target", "bin", "dist", "node\_modules" folder(s).
2. Grading criteria:
   1. Correctness (functioning, bug free, performant)
      1. Comprehension - understand the requirement and produce code accordingly.
      2. Bug free - the coding is done without logic bugs.
      3. Performant - able to execute within acceptable timeframe.
   2. Style (code formatting, naming convention, readability)
      1. Code formatting - spaces, indentation, braces, new lines.
      2. Naming convention - variables, methods, classes are named meaningfully.
      3. Readability - clear and organized logic flow, easy to read and maintain by others.
   3. Comprehensiveness (flexible, reusable, validation, unit tests)
      1. Methods/Functions/Classes/Designs are flexible; cater for reusability and future extension.
      2. Has proper validation and able to handle edge cases/scenarios.
      3. Has unit tests to validate functionalities.

|  |
| --- |
| **Overview** |

Git clone the latest source code from the repository (switch to the "main" branch)

* <https://github.com/konkeong/fourtitude-interviewq-jumble.git>

Use and fill up the "java application developer.docx" from the checkout repository, as that will be the latest/updated copy.

This "**Jumble**" project is about implementing a word reconstruction game, similar to [Scrabble™](https://en.wikipedia.org/wiki/Scrabble) .

From the provided list of words (src/main/resources/words.txt), pick a random word of at least 6 letters, ask the user to use at least 3 letters from the word to construct a new word.

Each letter can be used only once in the newly constructed word. The new word must not be the same as the original word.

For example, if the given word is "circle", then all the possible words are:

* cleric
* ice
* ire
* lei
* lice
* lie
* lire
* relic
* rice
* rile

Game is ended when all the possible words are guessed correctly.

The project has the initial (and incomplete) structure, and built using Maven, JDK 8, [JUnit 5](https://junit.org/junit5/), [Spring Boot](https://spring.io/projects/spring-boot) (2.7.18) framework, [Thymeleaf](https://www.thymeleaf.org/) templating engine, and [Springdoc](https://springdoc.org/).

**Addition**

* Candidate is allowed to change the JDK/Java version from 8 to higher, but not lower.
* Candidate is allowed to upgrade the Spring Boot to higher/latest version, but not lower.
* Candidate is allowed to add new library(s) (add dependencies to "pom.xml"), if required; e.g: Lombok, PowerMock, Eclipse Collections, etc.
* Candidate is allowed to add other components/sub systems, if required; e.g.: database, Kafka, Testcontainers, GraphQL, etc.

|  |
| --- |
| **Task A: Implementing Core Game Engine** |

Estimate completion time:

* 2 hours for Senior level
* 4 hours for Mid level
* 8 hours for Junior level

"Task A" is to implement the core logic for the game engine "JumbleEngine.java".

Activities:

1. Implement the code logic in all the (7) methods in "JumbleEngine.java". Each method will have requirements written in the Javadoc section.
   1. String scramble(String word) {}
   2. Collection<String> retrievePalindromeWords() {}
   3. String pickOneRandomWord(Integer length) {}
   4. boolean exists(String word) {}
   5. Collection<String> wordsMatchingPrefix(String prefix) {}
   6. Collection<String> searchWords(Character startChar, Character endChar, Integer length) {}
   7. Collection<String> generateSubWords(String word, Integer minLength) {}
2. Must pass all the unit tests in "JumbleEngineTest.java".
3. Higher score is given, if the lookup/searching implementation is fast and performant.

After each implementation, manual testing can be done using the provided console-based application. Run the "main()" method in "ConsoleApp.java".

|  |
| --- |
| **Task B: Implementing Web Interface for Playing Game** |

Estimate completion time:

* 2 hours for Senior level
* 4 hours for Mid level

"Task B" is to implement the Web interface/pages for playing the word game.

Activities:

1. Implement the code logic in all the (5) methods in "RootController.java". Each method will have requirements written in comment section.
   1. doPostScramble()
   2. doPostExists()
   3. doPostPrefix()
   4. doPostSearch()
   5. doPostSubWords()
2. Must pass all the unit tests in "RootControllerTest.java".
3. Implement the code logic in all the (2) methods in "GameWebController.java". Each method will have requirements written in comment section.
   1. doGetNew()
   2. doPostPlay()
4. Must pass all the unit tests in "GameWebControllerTest.java".
5. Must validate all the HTML request input properly.

After implementation, start the Spring Boot application "JumbleApplication.java", to test the functionalities from a web browser at "http://localhost:8080/".

The functionalities of "RootController.java" is corresponding to the "ConsoleApp.java" (in "Task A").

The functionalities of "GameWebController.java" is corresponding to the "GuessWord.java" (in "Task A").

|  |
| --- |
| **Task C: Implementing REST API for Game Engine** |

Estimate completion time:

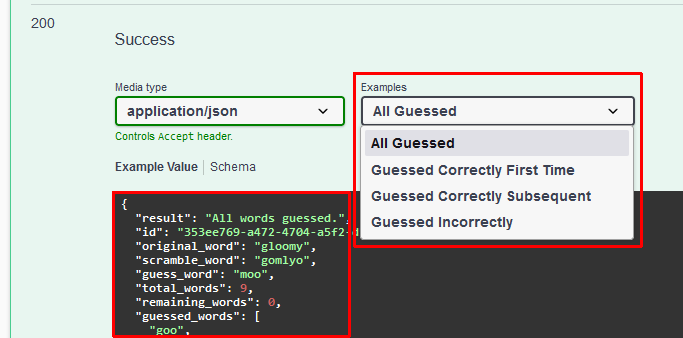
* 2 hours for Senior level
* 4 hours for Mid level

"Task C" is to implement a REST API endpoint and expose the functionalities of paying the word game.

Activities:

1. Implement the code logic in all the (2) methods in "GameApiController.java". Each method will have requirements written in Javadoc section and comment section inside method body. The specification ("api-docs.yaml") in [OpenAPI](https://www.openapis.org/) format, can be viewed from "http://localhost:8080/api-docs".
   1. ResponseEntity<GameGuessOutput> newGame() {}
   2. ResponseEntity<GameGuessOutput> playGame(@RequestBody GameGuessInput input) {}
2. Must validate all the input of request body ("GameGuessInput") properly.
3. Implement the code logic in all the (7) methods in "GameApiControllerTest.java" unit test. Each method will have requirements written as comment inside method body.
   1. whenCreateNewGame\_thenSuccess()
   2. givenMissingId\_whenPlayGame\_thenInvalidId()
   3. givenMissingRecord\_whenPlayGame\_thenRecordNotFound()
   4. givenCreateNewGame\_whenSubmiNullWord\_thenGuessedIncorrectly()
   5. givenCreateNewGame\_whenSubmitWrongWord\_thenGuessedIncorrectly()
   6. givenCreateNewGame\_whenSubmitFirstCorrectWord\_thenGuessedCorrectly()
   7. givenCreateNewGame\_whenSubmitAllCorrectWord\_thenAllGuessed()
4. Must pass all the unit tests in "GameApiControllerTest.java".

Visit "http://localhost:8080/swagger-ui/index.html" which auto generate a UI interface to test the REST API functionalities. Refer to the dropdown of the "**Responses Examples**" for the expected output of JSON body for different cases/scenarios.



Candidate may use the provided collection file ("interviewq-jumble.postman\_collection.json") and import into [Postman](https://www.postman.com/) to perform testing.

|  |
| --- |
| **Task D: Standalone Web Client to Consume Game REST API** |

Estimate completion time:

* 4 hours for Senior level
* 8 hours for Mid level

"Task D" is to provide a Web interface built with a modern JavaScript based framework web client, consuming the functionalities of REST API endpoint (implemented in "Task C"). This is a new/standalone project separate from the above Spring Boot project (another folder of source code).

Choose any one of the modern JavaScript based framework:

* [Angular](https://angular.io/)
* [React](https://react.dev/)
* [Vue.js](https://vuejs.org/)

Candidate can couple (or not) the Web UI with any one of the modern CSS frameworks, to provide a web responsive presentation, so that it looks well on mobile devices.

* [Bootstrap](https://getbootstrap.com/)
* [Foundation](https://get.foundation/)
* [Tailwind CSS](https://tailwindcss.com/)
* [Bulma](https://bulma.io/)

The Web interface will follow the functionalities of "GameWebController.java" in "Task B".

1. Can call to REST API endpoint.
2. Can create new game board/state.
   1. Show the interface accordingly.
3. Allow user to play current game board/state until end of game.
   1. Provide HTML form to submit input guessing word.
4. Has client-side validation, whenever necessary.
5. Able to display server-side success / error message.
6. Has unit tests ([Jasmine](https://jasmine.github.io/), [Jest](https://jestjs.io/), [Vitest](https://vitest.dev/)), to verify the game mechanics.

**Optionally** (not part of the task requirement) candidate can extend the game play with other functionalities (not limited to):

* Has a scoring/point system.
* Limit the number of guesses to be around 5 or 8 guesses, even if all the possible words are a lot (>= 5). And build a difficulty level around the number of guesses required.
* Put in a time-based element in the game play. Correct word within X seconds score higher.
* Deduct point for wrong guess and/or duplicate guess.
* Enhance the backend (server-side) with a proper database, to store history and game played by each user.

|  |
| --- |
| **Grading Sheet** |

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Sub Task | Point (0-5) | Remarks |
| Task A | Engine#scramble |  |  |
| Engine#palindrome |  |  |
| Engine#randomWord |  |  |
| Engine#wordExists |  |  |
| Engine#wordsMatchPrefix |  |  |
| Engine#searchWords |  |  |
| Engine#generateSubWords |  |  |
| JumbleEngineTest |  |  |
| Task B | Root#postScramble |  |  |
| Root#postExists |  |  |
| Root#postPrefix |  |  |
| Root#postSearch |  |  |
| Root#postSubWords |  |  |
| RootControllerTest |  |  |
| Game#getNew |  |  |
| Game#postPlay |  |  |
| GameWebControllerTest |  |  |
| Task C | Api#newGame |  |  |
| Api#playGame |  |  |
| ApiTest#newGameSuccess |  |  |
| ApiTest#missingGameId |  |  |
| ApiTest#missingGameRecord |  |  |
| ApiTest#submitNullWord |  |  |
| ApiTest#submitWrongWord |  |  |
| ApiTest#submitCorrectWord |  |  |
| ApiTest#submitWordAndEndGame |  |  |
| Task D | Web responsive presentation |  |  |
| Correct usage of Model |  |  |
| Correct usage of View |  |  |
| Correct usage of Service |  |  |
| Implement Game mechanics |  |  |
| Has unit tests |  |  |
| Correctness | Comprehension |  |  |
| Bug free |  |  |
| Performant |  |  |
| Style | Code formatting |  |  |
| Naming convention |  |  |
| Clear readability |  |  |
| Comprehensiveness | Reusable design/structure |  |  |
| Handle edge cases/scenarios |  |  |
| Pass all unit tests |  |  |
| Total | |  |  |