riddles cli test

Create a terminal-based Riddle Game in JavaScript using ES Modules.
The game should run entirely synchronously.

The game must:

1. Ask the player for their name (see Appendix A).
2. Load all riddles from files.
3. Show riddles one by one.
4. Wait for the correct answer before moving to the next riddle.
5. Measure how long it takes the player to solve each riddle. (use Date.now() or performance.now() - read about it mdn)
6. Show the total time and the average time per riddle at the end.

---

2. Folder Structure

Suggested folder structure - you can do it differently, as long as its organized and logical:

- `riddle-game/`
    - `app.js` – main file that runs the game
    - `riddles/` – folder containing all riddle files
        - `r1.js`
        - `r2.js`
        - …
    - `utils/` – folder for helper (utility) functions

You may add more utility files if needed, as long as the required behavior exists.

---

3. Riddle File Format

Each file in the `riddles/` folder should export a single riddle object.

Example (`riddles/r1.js`):

```
export default {
  id: 1,
  name: "Easy Math",
  taskDescription: "What is 5 + 3?",
  correctAnswer: "8"
};
```

Requirements:

1. All riddle files must export a default object.
2. All riddle objects must have at least:
    - `id`
    - `name`
    - `taskDescription`

- `correctAnswer`

---

4. Required Utility Code

You should organize your code into logical utility modules, based on topic.

The following functions and data structures are required.

4.1 Player Data and Utility Functions

You must work with a simple player object:

```
{

  name: "",

  times: [] // array of durations per riddle in seconds

}
```

Required functions:

1. `createPlayer(name)`
    - Input: player name (string).
    - Output: a new player object with:
        - `name` set to the given name.
        - `times` initialized as an empty array.

2. `addSolveTime(player, seconds)`
    - Input: a player object and a number of seconds.
    - Behavior: pushes the given time (in seconds) into `player.times`.

3. `showStats(player)`
    - Input: a player object.
    - Behavior:
        - Calculates total time
        - Calculates average time per riddle (total divided by number of riddles).

        - Prints both values to the terminal **in seconds**.

You may add more folders/files/helper functions if you want, but these must exist and work correctly.

4.2 Riddle Utility Functions

Required functions:

1. `askRiddle(riddleObj)`
    - Input: a riddle object.
    - Behavior:
        - Displays the riddle's name and task description.
        - If the riddle has a `choices` field (see section 6), display the choices as numbered options.
        - Asks the user for an answer using synchronous input (see Appendix A).
        - Repeats asking until the answer is correct.
            - For regular riddles: compare the input directly to `correctAnswer` (string comparison).
            - For multiple-choice riddles: compare the chosen option number to the correct one.

2. `measureSolveTime(fn)`
    - Input: a function (for example, a function that calls `askRiddle`).
    - Behavior:
        - Stores the current time before running the function.
        - Runs the function.
        - Stores the current time after it finishes.
        - Returns the difference **in seconds**.
    - You will use this to measure how long each riddle takes to solve.

You may add extra helper functions if needed, but these must exist and work correctly.

---

5. Game Flow (app.js)

`app.js` is the main file that coordinates the game.

Required steps:

1. Show a welcome message.
2. Ask the user for their name using synchronous input (appendix A).
3. Create a player object using `createPlayer(name)`.
4. Import all riddles using a single import (one statement, not one per file).
    - import AllRiddles from ‘path/to/file/ - good
    - import riddle1 from ‘/path/to/file1’

      import riddle2 from ‘/path/to/file2’ - bad
5. For each riddle:
    - Use `measureSolveTime()` to measure how long it takes to solve it.
        - Inside the function you pass to `measureSolveTime`, call `askRiddle(riddle)`.
    - Get the returned duration.
    - Call `addSolveTime(player, duration)` to store the time.

6. After all riddles are solved:
    - Call `showStats(player)` to print total and average time.

Important:

- The game must not move to the next riddle until the current one is solved correctly.
- The main logic should be clear and follow the order described above.

---

6. Multiple-Choice Riddles

Some riddles may be multiple choice.

A multiple-choice riddle object should look like this:

```
export default {

  id: 4,

  name: "Animal Sound",

  taskDescription: "Which of the following makes a 'moo' sound?",

  choices: ["Dog", "Cow", "Cat"],

  correctAnswer: "1" // the index of the correct answer

};
```

Requirements:

1. If a riddle has a `choices` array:
    - Show the available choices
2. The user should answer by typing the number of the choice.
3. `askRiddle()` must check that:
    - The user input matches the `correctAnswer` (for example `"2"`).
    - If wrong, ask again until correct.

---

7. Example Output

Example (you do not have to match this exactly, but the behavior should be similar):

- Welcome to the Riddle Game!
- What is your name? Bob

Riddle 1: Easy Math
What is 5 + 3? → 8
Correct!

Riddle 2: Mystery
I speak without a mouth. What am I? → echo
Correct!

Great job, Bob!
Total time: 72 seconds
Average per riddle: 36 seconds

8. README Requirements

Include a `README` file in the project.

The README must include:

1. Your full name.
2. Your ID.
3. Your class.
4. A short description of the project and how you implemented it.
5. A description of one challenge you had and how you solved it.

---

9. General Code Quality Rules

1. Use clear and meaningful variable and function names.
2. Place files and functions in logical locations.
3. Avoid repeating code (DRY – Don’t Repeat Yourself).
4. Keep each function focused on a single responsibility.
5. Do not leave commented-out code in the final submission.
6. Do not leave unnecessary console.log statements (only keep logs that are required).

Points will be deducted for violations of these rules.

---

10. Tips for Working

1. Planning first:
    - Plan your data structures (player object, riddle objects).
    - Plan your utility functions and their responsibilities.
    - Plan the step-by-step algorithm for the game flow and for each feature.
2. Time management:
    - Identify easy and hard parts of the task.
    - Start with the easier parts to ensure basic functionality is working.
3. Small steps:
    - Break the task into small subtasks.
    - After writing a few lines of code, run and test them.
4. Validation and debugging:
    - Test each part before moving on.
    - Use debugging and logs while developing, but remove unnecessary logs before submission.

---

11. Bonus Challenges (Optional – Extra Points Only)

These are optional and can only add points, not reduce them.

1. Add a `difficulty` field to each riddle object (for example: `"EASY"`, `"MEDIUM"`, `"HARD"`).
2. Before the game starts, allow the player to:
    - Sort riddles by difficulty and then play them.
    - Choose to play only riddles of a specific difficulty.
    - Choose to play all riddles up to a certain difficulty (for example, if the player chooses `"MEDIUM"`, show `EASY` and `MEDIUM` riddles).

You may implement these options in any clean and logical way.

---

12. Appendix A – Synchronous User Input

You must use a synchronous input method from npm.

Option 1 – `readline-sync`: `https://www.npmjs.com/package/readline-sync`:

1. Create `package.json` using `npm init`.
2. If you use ES modules, set `"type": "module"` in `package.json`.
3. Install:
    - `npm install readline-sync`

4. Use in code:

```
import readline from "readline-sync";

const name = readline.question("What is your name?
");

console.log(`Hello, ${name}!`);
```

Option 2 – analiza-sync
https://www.npmjs.com/package/analiza-sync:

1. Install:
    - npm install analiza-sync
2. Use in code:

```
import input from "analiza-sync";

const name = input("What is your name? ");

console.log(`Hello, ${name}!`);
```

You can choose either option

Good Luck !