Quiz App Project

A complete quiz: questions as data, one question at a time, answer checking, a 15-second timer, score and result screen — live.

Introduction

A quiz looks like a game, but under the screen it is pure series skills: **data as an array of objects**, **one render function** doing all the DOM work (the 3.24 rule), **events** on buttons, **state** in simple variables, and the **timer** from Lesson 3.19 racing every question.

You will build the complete thing: questions shown one at a time, instant right/wrong colours, a 15-second countdown per question, a score, and a result screen with Play Again. *(Roman Urdu: Quiz asal mein data aur state ka khel hai — sawal data mein, screen aik render function se, aur har sawal par 15 second ki ghari.)*

Key points

- **Questions are data** — one array of objects shaped `{ question, options: [...], answer: index }`, where `answer` is the **index** of the correct option: `answer: 1` means the second option is right. Adding a question to the quiz means adding one object — no new code.
- **The screen shows ONE question at a time** — a question heading, its option buttons built fresh from the data, a timer line, and a feedback line. Never all questions on one long page.

  (Roman Urdu: Aik waqt mein sirf aik sawal — sawal bhi data se banta hai, us ke options bhi data se bante hain.)
- **State is two variables** — `current` (which question we are on, starting at 0) and `score` (how many correct answers so far). Everything on screen is a picture of these two plus the data.
- **Clicking an option checks it at once** — correct: the button turns **green** and `score` grows by 1. Wrong: the clicked button turns **red**, and the correct button turns green so the student learns. Then the question **locks** — no second clicks can change it.
- **The 15-second timer guards each question** — `setInterval` counts `timeLeft` down from 15, once per second. At **zero**, the question locks by itself, counts as wrong, and the correct answer is shown.
- **Timers never leak** — `clearInterval(timerId)` runs on **every** answer, on timeout, and on Next, before any new timer starts. Forget it once and two timers fight over the same counter — the countdown speeds up and jumps.

  (Roman Urdu: Har jawab aur har Next par purana timer clear karo — warna do ghariyaan aik saath chal kar countdown kharab kar dengi.)
- **At the end, a result screen** — the quiz area hides, and the result shows the score out of the total, like `You scored 2 out of 3`. **Play Again** resets `current` and `score` to 0 and renders question 1 again — no page refresh needed.
- **One `renderQuestion()` does all the DOM work** — it sets the question text, builds the option buttons, resets the feedback and timer, and starts the countdown. The single-truth rule from the Todo app, again.

Code examples

Example 1 — The complete Quiz App (single HTML file)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Quiz App — Code Scope</title>
  <style>
    body { font-family: Arial, sans-serif; background: #faf9f6; color: #0b0f19; padding: 20px;
}
    .option-btn { display: block; margin: 8px 0; padding: 10px 14px; width: 320px; text-align:
left; cursor: pointer; }
    #timer { font-weight: bold; }
    #nextBtn { margin-top: 12px; padding: 8px 16px; }
  </style>
</head>
<body>
  <h1>JavaScript Quiz</h1>

  <div id="quiz">
    <p id="timer">Time left: 15</p>
    <h2 id="question"></h2>
    <div id="options"></div>
    <p id="feedback"></p>
    <button id="nextBtn" onclick="nextQuestion()">Next</button>
  </div>

  <div id="result" style="display: none;">
    <h2 id="scoreText"></h2>
    <button onclick="restartQuiz()">Play Again</button>
  </div>

  <script>
    const questions = [
      {
        question: "Which keyword creates a variable that can change?",
        options: ["const", "let", "var is not a keyword", "function"],
        answer: 1
      },
      {
        question: "What does console.log(typeof 42) print?",
        options: ["integer", "text", "number", "value"],
        answer: 2
```

```
  },
  {
    question: "Which symbol starts a template literal?",
    options: ["Single quote", "Double quote", "Backtick", "Hash"],
    answer: 2
  }
];

let current = 0;
let score = 0;
let timeLeft = 15;
let timerId = null;
let answered = false;

const questionEl = document.getElementById("question");
const optionsEl = document.getElementById("options");
const feedbackEl = document.getElementById("feedback");
const timerEl = document.getElementById("timer");
const nextBtn = document.getElementById("nextBtn");
const quizEl = document.getElementById("quiz");
const resultEl = document.getElementById("result");
const scoreText = document.getElementById("scoreText");

function renderQuestion() {
  answered = false;
  timeLeft = 15;
  timerEl.textContent = "Time left: 15";
  feedbackEl.textContent = "";
  nextBtn.style.display = "none";

  const q = questions[current];
  questionEl.textContent = "Question " + (current + 1) + ": " + q.question;
  optionsEl.innerHTML = "";

  q.options.forEach((optionText, index) => {
    const btn = document.createElement("button");
    btn.textContent = optionText;
    btn.className = "option-btn";
    btn.onclick = () => checkAnswer(index, btn);
    optionsEl.appendChild(btn);
  });

  clearInterval(timerId);
  timerId = setInterval(() => {
    timeLeft = timeLeft - 1;
```

```
    timerEl.textContent = "Time left: " + timeLeft;
    if (timeLeft <= 0) {
      clearInterval(timerId);
      timeOut();
    }
  }, 1000);
}

function lockQuestion() {
  const buttons = optionsEl.querySelectorAll("button");
  buttons.forEach((btn) => {
    btn.disabled = true;
  });
  const q = questions[current];
  buttons[q.answer].style.background = "green";
  buttons[q.answer].style.color = "white";
  nextBtn.style.display = "inline-block";
}

function checkAnswer(index, clickedBtn) {
  if (answered) {
    return;
  }
  answered = true;
  clearInterval(timerId);

  const q = questions[current];
  if (index === q.answer) {
    clickedBtn.style.background = "green";
    clickedBtn.style.color = "white";
    score = score + 1;
    feedbackEl.textContent = "Correct!";
  } else {
    clickedBtn.style.background = "red";
    clickedBtn.style.color = "white";
    feedbackEl.textContent = "Wrong! The correct answer is: " + q.options[q.answer];
  }
  lockQuestion();
}

function timeOut() {
  if (answered) {
    return;
  }
  answered = true;
```

```
      const q = questions[current];
      feedbackEl.textContent = "Time is over! The correct answer is: " + q.options[q.answer];
      lockQuestion();
    }

    function nextQuestion() {
      clearInterval(timerId);
      current = current + 1;
      if (current < questions.length) {
        renderQuestion();
      } else {
        showResult();
      }
    }

    function showResult() {
      clearInterval(timerId);
      quizEl.style.display = "none";
      resultEl.style.display = "block";
      scoreText.textContent = "You scored " + score + " out of " + questions.length;
    }

    function restartQuiz() {
      current = 0;
      score = 0;
      resultEl.style.display = "none";
      quizEl.style.display = "block";
      renderQuestion();
    }

    renderQuestion();
  </script>
</body>
</html>
```

What it does: This is the complete quiz in one file — copy it, save it as an HTML file, open it, and play. The questions live in one data array (3 questions, each with 4 options and the index of the correct one). `renderQuestion()` shows the current question, builds its option buttons, resets the timer to 15 and starts the countdown. `checkAnswer()` marks right or wrong with colour, updates the score and locks the question. `timeOut()` handles the timer reaching zero. `nextQuestion()` moves on or shows the result, and `restartQuiz()` resets the state for Play Again.

What you will see: the page opens with `JavaScript Quiz`, `Time left: 15` counting down one number per second, `Question 1: Which keyword creates a variable that can change?` and four option buttons: `const`, `let`, `var is not a keyword`, `function`. The Next button is hidden. Click `let`: the button turns green, `Correct!` appears, all buttons disable, the countdown freezes, and the Next button appears. Click

Next: question 2 appears with fresh buttons and the timer back at 15. This time click a wrong option, for example `integer`: your button turns red, the `number` button turns green, and the feedback reads `Wrong! The correct answer is: number`. Click Next, and on question 3 do nothing: the timer counts 15 down to 0 — at zero the `Backtick` button turns green by itself, the feedback reads `Time is over! The correct answer is: Backtick`, and the question locks. Click Next: the quiz hides and the result screen shows `You scored 1 out of 3` (in this exact walkthrough — one correct, one wrong, one timeout), with a Play Again button that restarts at question 1 with the score reset.

Example 2 — Answering, scoring and moving on

```
function checkAnswer(index, clickedBtn) {
  if (answered) {
    return;
  }
  answered = true;
  clearInterval(timerId);

  const q = questions[current];
  if (index === q.answer) {
    clickedBtn.style.background = "green";
    clickedBtn.style.color = "white";
    score = score + 1;
    feedbackEl.textContent = "Correct!";
  } else {
    clickedBtn.style.background = "red";
    clickedBtn.style.color = "white";
    feedbackEl.textContent = "Wrong! The correct answer is: " + q.options[q.answer];
  }
  lockQuestion();
}

function nextQuestion() {
  clearInterval(timerId);
  current = current + 1;
  if (current < questions.length) {
    renderQuestion();
  } else {
    showResult();
  }
}
```

What it does: These two functions from the full app carry the game's logic. `checkAnswer` first checks the `answered` guard — a locked question ignores all further clicks. It stops the timer, then compares the clicked index with the data's `answer` index: equal means green plus `score + 1`; different means red plus the correct answer spelled out in the feedback. `lockQuestion()` then disables every button and reveals Next.

`nextQuestion` clears the timer again for safety, steps `current` forward, and either renders the next question or — when `current` passes the last question — ends the game with the result.

What you will see: answer question 1 correctly and try clicking other options afterwards: nothing happens — no colour changes, no extra score; the `answered` guard is doing its job. Double-clicking the correct button also cannot raise the score twice, because the first click sets `answered = true` and stops the timer. Press Next on the last question: instead of a question 4 that does not exist, `showResult()` runs — proof that the `current < questions.length` check is the quiz's finish line.

Example 3 — The per-question timer and the result screen

```
// Inside renderQuestion(), a fresh timer starts for every question:
clearInterval(timerId);
timerId = setInterval(() => {
  timeLeft = timeLeft - 1;
  timerEl.textContent = "Time left: " + timeLeft;
  if (timeLeft <= 0) {
    clearInterval(timerId);
    timeOut();
  }
}, 1000);

function timeOut() {
  if (answered) {
    return;
  }
  answered = true;
  const q = questions[current];
  feedbackEl.textContent = "Time is over! The correct answer is: " + q.options[q.answer];
  lockQuestion();
}

function showResult() {
  clearInterval(timerId);
  quizEl.style.display = "none";
  resultEl.style.display = "block";
  scoreText.textContent = "You scored " + score + " out of " + questions.length;
}

function restartQuiz() {
  current = 0;
  score = 0;
  resultEl.style.display = "none";
  quizEl.style.display = "block";
  renderQuestion();
}
```

What it does: The timer block is Lesson 3.19 back at work: each new question first clears any old interval, sets `timeLeft` back to 15 (done in `renderQuestion` just before this block), and starts a fresh 1-second interval that updates the timer line. Reaching zero clears the interval and calls `timeOut()`, which locks the question as wrong — no score — but still shows the correct answer in green via `lockQuestion()`, so the student learns even on a timeout. `showResult()` swaps the two screens by changing `display`, and `restartQuiz()` resets both state variables and renders question 1 — a full restart with no page reload.

What you will see: watch the countdown closely across a whole game: every new question starts at exactly `Time left: 15` and ticks once per second — never faster, never skipping — because the old interval is cleared in three places (answer, Next, and render). If timers leaked, you would see the numbers fall 15, 13, 11 as two intervals share `timeLeft`; in this code that never happens. Let the last question time out, click Next, and the result screen replaces the quiz completely. Click Play Again: the result hides, question 1 is back, timer at 15 — and finishing this second game with all three correct shows `You scored 3 out of 3`.

Quick recap

- **Questions are data** — `{ question, options, answer: index }`; state is just `current` and `score`; one `renderQuestion()` does all the DOM work.
- Clicking checks at once: **green + score for correct, red plus the correct answer shown for wrong** — then the question locks so it cannot be farmed.
- A **15-second** `setInterval` guards each question; `clearInterval` on every answer, timeout and Next stops leaks; at the end, the **result screen** shows score out of total and Play Again resets the state.

Common mistakes

1. **Not clearing the interval** — moving to the next question while the old timer still runs leaves two intervals counting the same `timeLeft`, so the clock falls twice as fast and timeouts fire mid-question. The fix: `clearInterval(timerId)` on every answer, every Next, and before starting every new timer.
2. **Letting students click again and again** — without a lock, a wrong click followed by a right click still gives the point, and fast clicking can raise the score more than once per question. The fix: an `answered` flag checked at the top of `checkAnswer` and `timeOut`, plus disabling the buttons in `lockQuestion()`.
3. **Storing the answer as text instead of an index** — comparing the clicked button's text with an answer string breaks on small spelling or space differences, and duplicate option texts become ambiguous. The fix: store `answer` as the **index** of the correct option and compare indexes — `index === q.answer` — numbers never have typos.

Practice

1. Add two more questions to the `questions` array in the same object shape, then play the full game and confirm the result line reports the score out of **5**. No other code should need to change — if it does, find out why.
2. Show a live score line during the quiz — a paragraph that `renderQuestion()` and `checkAnswer()` keep updated in this shape: `Score: 1`. Then make Play Again appear **also** after a timeout on the last question without needing Next, by calling `showResult()` in place of the Next button in that one case.

Practice quiz

1. How are quiz questions stored in the Quiz App?

- As separate HTML pages, one per question

- As one array of objects shaped { question, options: [...], answer: index }, where answer is the index of the correct option
- As plain text with no structure
- In localStorage only

Answer: As one array of objects shaped { question, options: [...], answer: index }, where answer is the index of the correct option

2. In the Quiz App, what happens the moment you click a wrong option?

- Nothing, you can keep clicking until you get it right
- The clicked button turns red, the correct button turns green, and the question locks so no second clicks can change it
- The app restarts
- Your score goes up anyway

Answer: The clicked button turns red, the correct button turns green, and the question locks so no second clicks can change it

3. How long is each question's timer, and when is it cleared?

- 60 seconds, cleared only at the end
- 15 seconds via setInterval, and clearInterval runs on every answer, on timeout, and on Next, before any new timer starts
- 5 seconds, never cleared
- There is no timer

Answer: 15 seconds via setInterval, and clearInterval runs on every answer, on timeout, and on Next, before any new timer starts

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/js-quiz-app-project · More free tools and notes at https://scopetreasure.com