Part one: Add JSDOC comments and observe in VS Code

Step 1: Add JSDoc to admin.ts - add the part in bold

```
/**
 * Represents a quiz question.
 * @typedef {Object} QuizQuestion
 * @property {string} question - The text of the question.
 * @property {string[]} choices - List of answer choices.
 * @property {string} answer - The correct answer.
 */
interface QuizQuestion {
  question: string;
  choices: string[];
  answer: string;
}

/** @type {QuizQuestion[]} List of quiz questions loaded from the API.
*/
let quizData: QuizQuestion[] = [];

/**
 * Type guard to check if an element is an HTMLInputElement.
 * @param {Element | null} el
 * @returns {el is HTMLInputElement}
 */
function isInput(el: Element | null): el is HTMLInputElement {
  return el instanceof HTMLInputElement;
}

/**
 * Type guard to check if an element is an HTMLFormElement.
 * @param {Element | null} el
 * @returns {el is HTMLFormElement}
 */
function isForm(el: Element | null): el is HTMLFormElement {
  return el instanceof HTMLFormElement;
}

/**
 * Type guard to check if an element is an HTMLDivElement.
 * @param {Element | null} el
 * @returns {el is HTMLDivElement}
 */
function isDiv(el: Element | null): el is HTMLDivElement {
  return el instanceof HTMLDivElement;
}

/**
 * Retrieves an element by ID and asserts its type.
```

```
 * @template T
 * @param {string} id - The DOM element ID
 * @returns {T} - The typed element
 * @throws Will throw an error if the element is not found
 */
function getElement<T extends HTMLElement>(id: string): T {
  const el = document.getElementById(id);
  if (!el) throw new Error(`Element with ID '${id}' not found.`);
  return el as T;
}

// Load quiz data from API and render
fetch('/api/quiz')
  .then(res => res.json())
  .then((data: QuizQuestion[]) => {
    quizData = data;
    renderQuizAdmin();
  })
  .catch(err => {
    console.error('Failed to load quiz:', err);
  });

/**
 * Renders the quiz data to the admin interface.
 */
function renderQuizAdmin(): void {
  const listDiv = getElement<HTMLDivElement>('quiz-admin-list');
  listDiv.innerHTML = '';

  quizData.forEach((q, index) => {
    const div = document.createElement('div');
    div.className = 'border p-3 mb-3';

    /** Editable Question */
    const questionInput = document.createElement('input');
    questionInput.type = 'text';
    questionInput.className = 'form-control mb-2';
    questionInput.value = q.question;
    questionInput.oninput = () => {
      quizData[index].question = questionInput.value;
    };
    div.appendChild(questionInput);

    /** Editable Choices */
    const choicesInput = document.createElement('input');
    choicesInput.type = 'text';
    choicesInput.className = 'form-control mb-2';
    choicesInput.value = q.choices.join(', ');
    choicesInput.oninput = () => {
```

```
      quizData[index].choices = choicesInput.value.split(',').map(c =>
c.trim());
    };
    div.appendChild(choicesInput);

    /** Editable Answer */
    const answerInput = document.createElement('input');
    answerInput.type = 'text';
    answerInput.className = 'form-control mb-2';
    answerInput.value = q.answer;
    answerInput.oninput = () => {
      quizData[index].answer = answerInput.value;
    };
    div.appendChild(answerInput);

    listDiv.appendChild(div);
  });
}
```

How you can use JSDoc effectively in your workflow:

1. Use it for IntelliSense in your IDE (like VS Code)

After you add JSDoc:

- Hover over functions/variables to see docs.
- Autocomplete improves: param hints, return types, etc.
- Type checking works even in JavaScript if you turn it on.

To enable type checking in a JS project:
- Add a tsconfig.json file:

```
{
  "compilerOptions": {
    "allowJs": true,
    "checkJs": true,
    "noEmit": true,
    "target": "ES2020"
  },
  "include": ["./**/*.js"]
}
```

Or in a specific file, add this to the top:

```
// @ts-check
```

2. Use it for generating documentation

If you want to turn your comments into real HTML docs:

```
npm install -g jsdoc
```

```
jsdoc admin.ts -d docs
```

This will generate a folder called docs/ with browsable documentation.

3. Use JSDoc in TypeScript for clarity and public APIs. Even in .ts files, JSDoc is helpful for:

- Adding descriptions to public APIs.
- Declaring custom types with @typedef and @property.
- Explaining complex generics.

4. JSDoc is ideal when:

- Teaching code or handing off to other devs.
- Working in mixed teams (TS + JS).
- Needing explanations inline instead of separate docs.

Workflow Demo:

If you open admin.ts in VS Code now:

- Hover over renderQuizAdmin → you'll see the JSDoc description.
- If you type getElement(it will suggest <T extends HTMLElement>.

If You Want to Use JSDoc for Strict Type Checking in JavaScript Projects

At the top of your .js files (if you're not using TypeScript), add this:

```
// @ts-check
```

That turns on TypeScript-style checking in plain JavaScript files using JSDoc.

If you're using a project with lots of .js files, create a tsconfig.json file:

```
npx tsc --init
```

Then edit it like this:

```
{
  "compilerOptions": {
    "allowJs": true,
    "checkJs": true,
    "noEmit": true,
    "target": "ES2020"
  },
  "include": ["./**/*.js"]
}
```

See JSDoc in Action in VS Code
1. Open the File in VS Code

Open your admin.ts file (or .js file) in VS Code.
2. Hover Over a Function Name

Try hovering over:

```
renderQuizAdmin()
```

You should see a tooltip: "Renders the quiz admin panel: editable questions, choices, and answers."

Other options to hover over:

```
getElement

isInput

quizData (hover over the word quizData)

fetch(...) line —
hover over .then(res => res.json()) to see inferred types
```

3. Type Something Slowly and Use Autocomplete

Try this inside a function:

```
quizData[0].
```

After typing the dot (.), you should see:

```
question

choices

answer
```

This comes from your interface and JSDoc.