Part 1: Folder Structure

Organize your files like this:

```
AdvJSTypeScript/
├── tsconfig.json
├── jest.config.ts
├── package.json
├── ts-src/
│   ├── admin.ts
│   ├── contact.ts
│   ├── filter.ts
│   └── quizquestions.ts
├── public/
│   └── js/          ← Output JS goes here after compiling
└── __tests__/
    ├── admin.test.ts
    ├── contact.test.ts
    ├── filter.test.ts
    └── quizquestions.test.ts
```

Part 2: tsconfig.json

Put this in the root:

```
{
  "compilerOptions": {
    "target": "ES6",
    "module": "ES6",
    "outDir": "./public/js",
    "rootDir": "./ts-src",
    "strict": true,
    "esModuleInterop": true,
    "moduleResolution": "node",
    "forceConsistentCasingInFileNames": true,
    "skipLibCheck": true,
    "lib": ["DOM", "ES6"]
  },
  "include": ["ts-src/**/*"]
}
```

What it does:

Transpiles TS to JS into public/js

Starts compiling from ts-src

Enables strict typing and compatibility with modern JS modules

Recognizes DOM APIs and ES6 features

Part 3: Install Required Packages

Run in terminal:

```
npm init -y
npm install --save-dev typescript ts-jest jest @types/jest
npx ts-jest config:init
```

This creates or updates:

jest.config.ts (you can also use jest.config.js)

Adds everything Jest needs to run TypeScript tests

Part 4: jest.config.ts

Here's a working Jest config for TypeScript:

```
export default {
  preset: 'ts-jest',
  testEnvironment: 'jsdom',
  testMatch: ['**/__tests__/**/*.test.ts'],
  transform: {
    '^.+\\.tsx?$': 'ts-jest'
  },
  moduleFileExtensions: ['ts', 'tsx', 'js'],
};
```

What it does:

Uses ts-jest to transpile TypeScript

Looks for tests ending in .test.ts inside the __tests__ folder

Uses the browser-like jsdom environment so DOM APIs work

Part 5: Test File Naming Rules

To ensure Jest finds your tests:

Do __tests__/file.test.ts not file.test.js

Use .ts if using TypeScript .js won't be transpiled properly

Part 6: Example NPM Scripts in package.json

Add this to your scripts section:

```
"scripts": {
  "build": "tsc",
```

```
  "test": "jest"
}
```

Part 7: Now, write the tests:

__tests__/admin.test.ts

Explanation:

- This checks that your quiz data structure is valid.
- It ensures answer is one of the choices.

```
import { describe, test, expect } from '@jest/globals';

interface QuizQuestion {
  question: string;
  choices: string[];
  answer: string;
}

describe('QuizQuestion Interface', () => {
  test('should match structure', () => {
    const q: QuizQuestion = {
      question: 'What is 2 + 2?',
      choices: ['3', '4', '5'],
      answer: '4',
    };
    expect(q.choices.includes(q.answer)).toBe(true);
  });
});
```

__tests__/filter.test.ts

Explanation:

- This is a unit test of the logic behind your RxJS filter.ts file.
- Mimics how visible rows are filtered in the UI.

```
import { describe, test, expect } from '@jest/globals';

function filterRows(rows: string[], searchTerm: string): string[] {
  return rows.filter(row =>
row.toLowerCase().includes(searchTerm.toLowerCase()));
}

describe('filterRows', () => {
  test('filters rows that include the search term', () => {
    const rows = ['Angular', 'React', 'Vue'];
```

```
    expect(filterRows(rows, 're')).toEqual(['React']);
  });

  test('returns all if search term is empty', () => {
    const rows = ['Angular', 'React', 'Vue'];
    expect(filterRows(rows, '')).toEqual(rows);
  });
});
```

__tests__/contact.test.ts

Explanation:

- Simulates the contact form validation from contact.ts.
- Separates validation logic so it's easy to test without DOM.

```
import { describe, test, expect } from '@jest/globals';

function validateContact(phone: string, email: string, zip: string):
string[] {
  const errors: string[] = [];
  if (!/^\d{10}$/.test(phone)) errors.push('Phone must be 10
digits.');
  if (!/^[\w.-]+@[\w.-]+\.[a-zA-Z]{2,}$/.test(email))
errors.push('Email is invalid.');
  if (!/^\d{5}$/.test(zip)) errors.push('Zip must be 5 digits.');
  return errors;
}

describe('validateContact', () => {
  test('valid data returns no errors', () => {
    expect(validateContact('1234567890', 'test@example.com',
'78701')).toEqual([]);
  });

  test('invalid data returns proper errors', () => {
    expect(validateContact('123', 'invalid', '12')).toEqual([
      'Phone must be 10 digits.',
      'Email is invalid.',
      'Zip must be 5 digits.'
    ]);
  });
});
```

__tests__/quizquestions.test.ts

Explanation:

- Confirms quiz questions are usable by your admin panel.
- Simulates logic used when adding questions.

```
import { describe, test, expect } from '@jest/globals';

interface QuizQuestion {
  question: string;
  choices: string[];
  answer: string;
}

function isQuizValid(q: QuizQuestion): boolean {
  return q.choices.length >= 2 && q.choices.includes(q.answer);
}

describe('isQuizValid', () => {
  test('valid question', () => {
    const question: QuizQuestion = {
      question: 'Capital of France?',
      choices: ['Paris', 'Berlin'],
      answer: 'Paris'
    };
    expect(isQuizValid(question)).toBe(true);
  });

  test('invalid when not enough choices', () => {
    const question: QuizQuestion = {
      question: 'Capital of France?',
      choices: ['Paris'],
      answer: 'Paris'
    };
    expect(isQuizValid(question)).toBe(false);
  });
});
```

Last: Running Everything

1. Compile TypeScript:

```
npm run build
```

Transpiles ts-src/*.ts → public/js/*.js.

2. Run Jest Tests:

```
npm test
```

Runs all *.test.ts files inside __tests__/.