Jest   
  
sum.test.js – to run this file you must install jest library in your project folder.

Important: jest library ni package.json file lo “scripts” lo “test” value changed to “jest”

**✅ Basic ga install cheyyali:**

**npm install --save-dev jest**

👉 Jest ni install cheyyadam.  
--save-dev ante idi development lo matrame use avuthundi, final app lo kaadu.

**npm init jest@latest**

👉 Jest ni setup cheyyadam – jest.config.js ane configuration file create chesthundi.

**✅ JS Babel tho run cheyyali ante:**

JavaScript lo konni advanced features (latest syntax) browser ki or Jest ki ardam kaakapovachu. So:

**Babel enti?**

👉 Babel ante **JavaScript compiler**. Modern JS ni older version ga marchi, Jest or browser ki ardam ayyela chesthundi.

**✅ Babel kosam install cheyyalsina packages:**

**npm install --save-dev babel-jest @babel/core @babel/preset-env**

**1. babel-jest:**

👉 Jest and Babel madhya bridge laga pani chesthundi.  
Jest ki import or modern JS syntax kanipinchinappudu, babel-jest vaalla adi translate cheysthundi.

**2. @babel/core:**

👉 Babel yokka main brain/motor.  
Translation logic ekkada run cheyyali ani idi decide chesthundi.

**3. @babel/preset-env:**

👉 Idi chepthundi: "Nee code ni ela convert cheyyali browser ki support undetattu."  
Modern JS ni old JS ki easy ga convert cheyyadam.

**✅ TypeScript use chesthunnappudu:**

**npm install --save-dev @babel/preset-typescript**

👉 TypeScript code ni JS ga marchadaniki Babel ki idi help chesthundi.

**npm install --save-dev ts-jest**

👉 Jest ki TypeScript ni understand cheyyali ante idi kavali.  
Idi Jest ki chepthundi: “Nuvvu TypeScript ni handle chey.”

**✅ Extras for smooth testing:**

**npm install --save-dev @jest/globals**

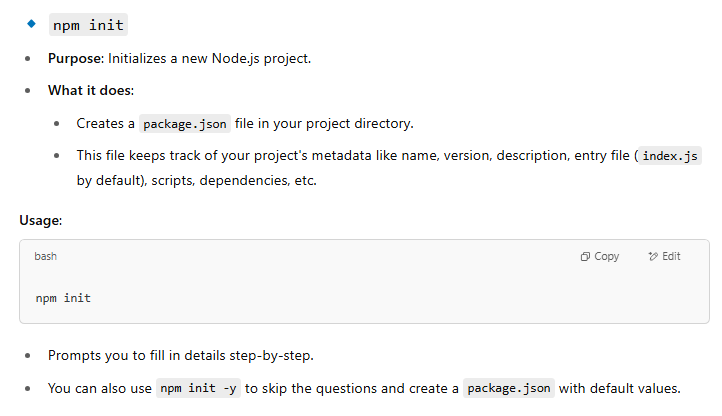
👉 Jest lo expect, test etc. ane functions ni separate ga import cheyyali ante idi use chestaru.

**npm install --save-dev @types/jest**

👉 TypeScript lo Jest tho panicheyyali ante idi mandatory.  
Idi Jest functions ki type information provide chesthundi (TypeScript ki ardham ayyela).

**💡 Summary table:**

| **Command** | **Purpose** |
| --- | --- |
| jest | Testing framework |
| babel-jest | Jest ki Babel syntax ardam cheyyadaniki |
| @babel/core | Babel core logic |
| @babel/preset-env | Modern JS → Old JS conversion |
| @babel/preset-typescript | TypeScript → JS (via Babel) |
| ts-jest | TypeScript + Jest integration |
| @jest/globals | Jest functions ni individual ga use cheyyadam |
| @types/jest | TypeScript ki Jest types ivvadam |

**  
Jest Installation and configuration:**

1. **Run npm install --save-dev jest** – Which installs jest library
2. **Run npm init** – creates package.json file in your project folder.

This file keeps track of your project's metadata like name, version, description, entry file (index.js by default), scripts, dependencies, etc.

**Or**

**Run npm init -y -** to skip the questions and create a package.json with default values.

1. **Run npm install** – which installs packages specified in package.json
2. **Run npm install packagename -** Downloads the specified package(s) from the npm registry and adds them to **node\_modules** folder
3. **Add following to package.json**

{  
 "scripts": {  
 "test": "jest"  
 }  
}

1. **Run npm init jest@latest -** Jest will ask you a few questions and will create a basic configuration file
2. Install **babel** package (compiles/convert one version of JS to another version) when your JS runtime environment does not understand modern JavaScript code.
   1. Run **npm install --save-dev babel-jest @babel/core @babel/preset-env**
   2. **if you're using Babel with Jest**, then configure **Babel** to target your current version of Node by creating a **babel.config.js** file in the root of your project.

module.exports = {

presets: [['@babel/preset-env', {targets: {node: 'current'}}]],

};

**NOTE:** which Node.js వర్షన్‌కు సరిపోయేలా transpile cheyyali anedi Babel needs to know. If you **don’t set targets.node: 'current'**, which means **ఏ వర్షన్ కోసం?** అనేది మీరు చెప్పకపోతే, **Compile your code for an older version of Node or browsers by default**, **Add unnecessary polyfills** or **fail to compile your code properly**.

**Remember:** **babel-jest** is automatically installed when installing **Jest** and will automatically transform files if a babel configuration file exists in your project.

To avoid this behaviour, you can explicitly reset the “**transform** “configuration option at **jest.config.js:**

module.exports = {

transform: {},

};

**This configuration means:** Don’t use Babel. Don’t auto-transform anything — even if **babel.config.js** exists.

1. If your Node.js project has TypeScript file. To make Node.js understand TypeScript code install **npm install --save-dev @babel/preset-typescript.**
2. Then add **@babel/preset-typescript** to the list of presets in your babel.config.js.

module.exports = {

presets: [

['@babel/preset-env', {targets: {node: 'current'}}],

'@babel/preset-typescript',

],

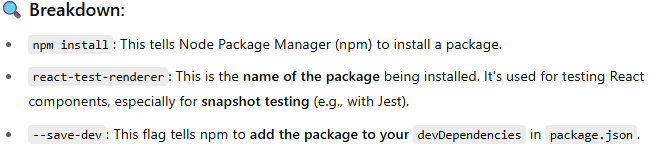
};

1. Run npm install --save-dev @types/jest - **need to install** @types/jest **if you are writing tests in TypeScript** (.ts file) It provides **TypeScript type definitions** for all the Jest globals like : describe, it, test, expect, beforeEach, afterEach

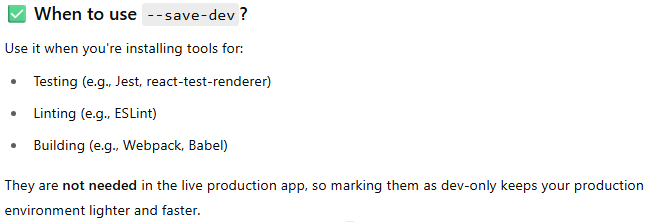
**Snapshot mechanism:**

**Snapshot testing** captures a rendered output of a React component (as JSON) and saves it. On future runs, Jest compares the current output to the stored snapshot. If they differ, the test fails — alerting you that something in the output changed.

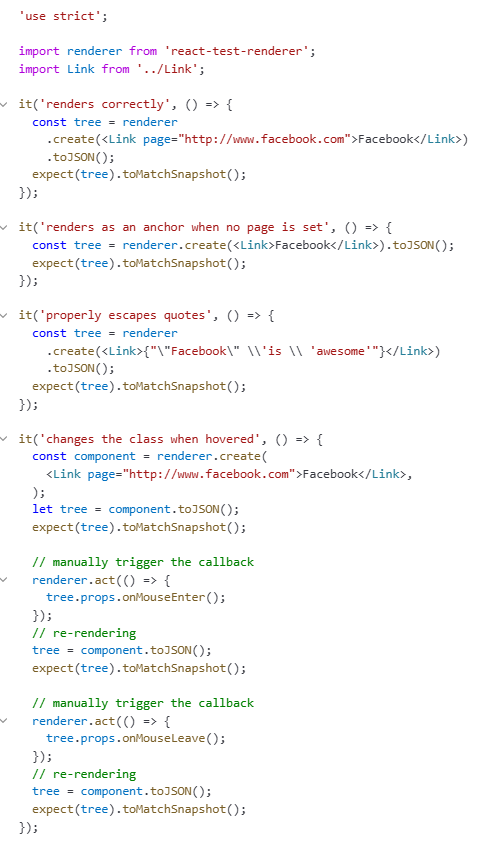
Step-1: npm install --save-dev react-test-renderer











Testcase-1 explanation:   
renderer.create(...): renders the component using `react-test-renderer` (a way to simulate React rendering without a real DOM).

.toJSON() : converts the rendered output to a JSON tree (simplified structure).

expect(tree).toMatchSnapshot(): takes a snapshot (if it’s the first time), or compares it to the previous saved snapshot.

Testcase-2 explanation:

The <Link> component may be reused elsewhere in your app without `page` property like:

                     <Link>Facebook</Link>

Testcase-3 explanation:

The <Link> component may be reused elsewhere in your app with:

                     <Link>{'"Facebook" is awesome'}</Link>

Quotes (" and '), slashes (\), and other symbols can sometimes break HTML if not escaped properly.

Snapshot testing ensures the component safely renders such content without introducing bugs or XSS issues (e.g., malformed DOM nodes).

Testcase-4 explanation:

testing that your <Link> component properly changes its CSS class when hovered and unhovered.

expect(tree).toMatchSnapshot(): takes a snapshot (if it’s the first time), or compares it to the previous saved snapshot.

This captures the initial state — the class should be "normal" because nothing has happened yet.

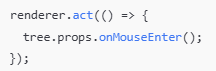


Fig: Simulate mouse hover

renderer.act(...): tells React “simulate an interaction and update the state.”

tree.props.onMouseEnter() is manually triggering the `onMouseEnter` event handler defined inside the Link component.

This simulates the hover effect.



Fig: Capture snapshot after hover

React re-renders due to state change (status becomes 'hovered').

A new snapshot is taken to verify the output changed correctly — the className should now be "hovered".

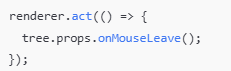


Fig: Simulate mouse leave

This simulates the **mouse moving out**, triggering the onMouseLeave event.



Fig: Capture snapshot after mouse leaves

Now the component should re-render again.

The className should go back to "normal".

Snapshot comparison ensures the class returns to its original state.

Snapshot Testing Advantage Here,

If any logic breaks (e.g., setStatus isn't triggered), one of these snapshots will **fail**, alerting you to the problem