JEST DOCUMENTATIONIntroduction to JESTJEST is a framework used for JavaScript testing, it is built on top of ‘ Jasmine ‘ and is a popular framework for performing unit testing. Christoph Nakazawa, the brain of JEST developed it so that it provides support and simplicity in testing heavy web apps.JEST is mainly used for performing unit testing for REACT and React Native applications. Along with unit testing JEST can also be used for component testing.Unit testing is not very helpful for front-end software or components, as configuration of unit testing for front-end components requires more effort and is very time consuming. This is where JEST comes in handy, as it reduces the time consumption and complexity to a massive extent.Installation of JSET •U4”är YARNyarn add --dev jest•U4”är å Öç Ò –ç7F ÆÂ Ò×6 ve-dev jestUsing JEST In NPMFirst open package.json file and change the configuration of the file as follows"67 ipts": {™™™™™’ ’ 'FW7B# "jest"™™™™™™ ’ ÒÉ™™™™™™First create a Test fileLet's addition of two numbers as a test case, so create a file sum.js file—7VÒæ§4gVæ7F–öâ add(a,b){™™™™™™™™™™™™•&WGW n a+b™™™™™—Ù™™™™™™™™™™™™’òöPxpor create a file name sum.test.js’ 7VÒçFW7Bæ§2òò –× ÷ t your sum functionconst add= require('./add);test("adding 4and 2 expect to be 6",() =>
{ expect(add(4,2)).toBe(6);});In this file we take two parameters 4 and 2 and send it to the add function in sum.js to check if the function is working properly or not. We expect the output to be 6 if the output is 6 then the given function is working properly if not the test has failed i.e the taken function is not working properly.Execution of a test fileWe can execute the test file in two ways if we want to execute a single file we use the command –ç Ò FW7B 7VÒçFW7Bæ§4–b pe want to execute all the test files we use the commandnpm test JEST MatchersA matcher is used for creating assertions in combination with the expect keyword, we want to compare the output of our test to a value which is the output we are expecting. toBe : It matches the objects.toEqual : It matches the value of the object to the return value.TruthinessIn tests you need to sometimes differentiate between undefined, null, false.toBeNull : matches only nulltoBeUndefined : matches only undefinedtoBedefined : It is the opposite of toBeUndefinedtoBeTruthy : matches anything if the if statement is executed as truetoBeFalsy : matches anything if the if statement is executed as falseExample: • test(‘null’ , () =>{Const n = nullexcept(n).toBeNull()})NumberstoBeGreaterThan : If the return value of the function is greater than the expected valuetoBeGreaterThanOrEqual : If the return value of the function is greater or equal to the value expectedtoBeLessThan : If the return value of the function is less than the expected valuetoBeLessThanOrEqual : If the return value of the function is less than or equal to the value expectedExample: —FW7B, ‘ four minus 2 ‘, () =>{Const n = 4 - 2except(n).toBeGreaterThan(1)})StringstoMatch: It is used check strings against regular expressionExample: —FW7B, ‘ there is a christ in christopher ‘ , () => {except(‘christopher’).toMatch(/christ/)})Arrays and iterablestoContain : You can check if an item is present in an array or not iterablesExample: ™const anime = [‘ Naruto ‘, ‘ Baruto ‘, ‘Zero’, ‘Suzaku’, ‘Goku’] —FW7B, ‘ Naruto is mentioned in the list ‘ , () => {except(anime).toContain(‘Naruto’)})Testing Asynchronous CodeIn JavaScript we can run code asynchronously, so when we test JEST needs to know when the testing is completed so that it can move to the next text. JEST has several ways to handle thisPromises •&WGW n a promise from your test, and Jest will wait for that promise to resolve. If the promise is rejected, the test will fail.Example:™math.jsexport function

multiply(a, b) { —&WGW n new Promise((resolve, reject) => { ™if (typeof a !== 'number' || typeof b !== 'number') { ™—&V¦V7B†æPw Error('Both arguments must be numbers')); ™ } else { ™—&W6öÇ`e(a * b); ™ } —Ò“·Ù™–Ö F,çFW7Bæ§6–× ÷ t { multiply} from './math';test('mu should resolve with the correct result', () => { ’ &WGW n multiply(2, 3).then((result) => { ™expect(result).toBe(6); —Ò“·Ò“·FW7B,vÐultiply should reject with an error for non-numeric arguments', () => { expect.assertions(1); return multiply(2, 'not a number').catch((error) => { expect(error.message).toBe('Both arguments must be numbers'); });});Async and wait™Alternatively, you can use async and await in your tests. To write an async test, use the async keyword in front of the function passed to test.Example:™math.jsexport function divideAsync(a, b) { —&WGW n new Promise((resolve, reject) => { ™if (b === 0) { ™—&V¦V7B†æPw Error('Cannot divide by zero')); ™} else { ™ resolve(a / b); ™} ’ Ò“·Ù™–Ö F,çFW7Bæ§6–× ÷ t { divideAsync } from './math';test('divideAsync should resolve with the correct result', async () => { const result = await divideAsync(10, 2); expect(result).toBe(5);});test('divideAsync should reject with an error for division by zero', async () => { expect.assertions(1); try { await divideAsync(10, 0); } catch (error) { expect(error.message).toBe('Cannot divide by zero'); }});CallbackExample:™math.jsexport function subtractAsync(a, b, callback) { • setTimeout(() => { ™if (typeof a !== 'number' || typeof b !== 'number') { ™• callback(new Error('Both arguments must be numbers')); ™ } else { ™–6 ÆÆ& 0k(null, a - b); ™ } ’ ÒÂ “·Ù™–Ö F,çFW7Bæ§6–× ÷ t { subtractAsync } from './math';test('subtractAsync should return the correct result via callback', (done) => { —7V'G actAsync(10, 3, (error, result) => { ™expect(error).toBeNull(); ™expect(result).toBe(7); ™done(); ’ Ò“·Ò“·FW7B,w7V'G actAsync should return an error via callback for non-numeric arguments', (done) => { —7V'G actAsync(10, 'not a number', (error, result) => { ™ expect(error).toBeInstanceOf(Error); ™expect(result).toBeUndefined(); ™done(); —Ò“·Ò“µW6–ær Öö0k in JESTMock functions allow you to test the connectivity between two codes by actually erasing the actual implementation of a function.There are two ways to create mock functions:Either by creating a mock function to use in the test code.Writing a manual mock to override a module dependency.All mock functions have this special .mock property which is where the data about how the function has been called and what the function returned is kept. The .mock property also tracks the value of this for each call. ™™™™™