Test Cases Summary Notes

The objective of the summary notes is to:

- Provide a ready reference for commonly used test patterns, assertions and mock setups.

Backend Unit Testing

Code	Explanation
`describe('...', () => {...})`	Groups related tests together into a test suite
`it('...', async () => {...})`	Defines one unit test scenario

`jest.mock('fs', () => ({`
`  promises: {`
`  readFile: jest.fn(),`
`  writeFile: jest.fn(),`
`  }`
`}));`
	Replaces the real fs module to avoid actual disk writes

`fs.readFile.mock`**`Resolved`**`Value(...)`
 	
`E.g.`
`const mockData = JSON.stringify([{`
`  name: 'Test Resource',`
`  location: 'Room 101',`
`  description: 'Projector and screen',`
`  owner: 'user@example.com'`
`}]);`
`fs.readFile.mock`**`Resolved`**`Value(mockData);`
	Simulates successful file read and returns mock content

`fs.readFile.mock`**`Rejected`**`ValueOnce(...)`
 	
`E.g.`
`fs.readFile.mock`**`Rejected`**`ValueOnce({ code: 'ENOENT' });`
	Simulates failure when reading a file

`fs.writeFile.mockResolvedValue()`	Pretend write operation completed with no errors

`fs.writeFile.mockRejectedValueOnce(new Error(...))`
 	
`E.g.`
`fs.writeFile.mockRejectedValueOnce(new Error("Write failed"));`
	Simulate write failure → triggers error handling

`jest.clearAllMocks()`	Resets mock counters between tests for isolation
`const req = { body: {...} }`	Fake Express request object with POST payload
`const req = { params: { id: 1 }}`	Fake route parameter input
`const res = { status: jest.fn().mockReturnThis(), json: jest.fn() }`	Mock Express response with method chaining support, allowing .status().json() chaining like real Express
`await functionName(req, res)`	Executes the function being unit-tested
`const response = res.json.mock.calls[0][0]`	Retrieves JSON response content passed to res.json()
`expect(res.status).toHaveBeenCalledWith(201)`	Ensures correct status returned
`expect(res.json).toHaveBeenCalledWith({...})`	Check returned response body
`expect(response.length).toEqual(1)`	Validate response list size
`expect(response[0].name).toEqual(...)`	Confirms data saved correctly
`expect(response.message).toEqual('...')`	Validate message
`expect(res.status).`**`not`**`.toHaveBeenCalledWith(500)`	Ensure no failure if success expected

Backend API Testing

Code	Explanation
id = ... jsonData = {...} await request(app).post('/route-name').send(jsonData) await request(app).get('/route-name') await request(app).put('/route-name/' + id).sendData(jsonData) await request(app).delete('/route-name/' + id)	Sends an HTTP POST/GET/PUT/DELETE request to the specified route .send(jsonData) includes the payload in the request body (usually JSON)
expect(res.status).toBe(201)	Ensures correct status returned
expect(...).toBe(false)	Ensure deleted item removed
expect(res.body.message).toBe('...')	Validate message
expect(res.body.some(r => r.id === someId)).toBe(true);	Validate that someId exists in response

Frontend UI Testing

Code	Explanation
await page.goto(BASE_URL)	Opens the web app in a real browser
await page.click('button:has-text("Some Text")')	Clicks button by visible text
await page.click('#some-button')	Clicks button by id
await page.fill('#some-id', 'some value')	Enters input field text by id

page.once('dialog', dialog => dialog.**accept**())
 	
Other variations (validate message):
page.once('dialog', dialog => {
  expect(dialog.message()).toBe('Unable to add resource!')
  dialog.accept();
});
	Handles confirmation pop-up - Ok

page.once('dialog', dialog => dialog.**cancel**())	Handles confirmation pop-up - Cancel

page.locator('selector')
 	
E.g. Find a table row inside #tableContent that contains 'some text'
page.locator('#tableContent tr', { hasText: 'some text' })
	Find elements on the page

await page.waitForSelector('#resourceModal', {state:'hidden'})	Waits until modal disappears
await row.waitFor({state: 'visible'})	Ensures row rendered in table
await expect(...).toBeVisible()	Validate element to be visible on page
await expect(...).toHaveText(...)	Validate content
await expect(locator).toHaveCount(1)	Validate number of matches
await expect(row.locator('td').nth(0)).toHaveText('some text');	Validates that the first cell (td) contains the stated text

await page.route('**/some-route', route => {
  route.fulfill({
  status: 500,
  contentType: 'application/json',
  body: JSON.stringify({ message: 'Simulated backend error' }),
  });
});
	Simulate error from backend API call

End of Summary Notes