← Cypress Fundamentals

Mobile Testing with WebDriverIO →

Advanced Cypress & API Testing

Table of Contents

Custom Commands and Fixtures

Creating Custom Commands

Custom commands make tests more readable and maintainable by encapsulating common actions.

Basic Custom Command:

```
// cypress/support/commands.js
Cypress.Commands.add('login', (email, password) => {
```

```
  cy.get('button[type="submit"]').click()
})

// Use in test
cy.login('user@example.com', 'password123')
```

Command with Options:

```
Cypress.Commands.add('login', (email, password, options = {}) => {
  const { navigate = true, validate = true } = options

  if (navigate) {
    cy.visit('/login')
  }

  cy.get('#email').clear().type(email)
  cy.get('#password').clear().type(password)
  cy.get('button[type="submit"]').click()

  if (validate) {
    cy.url().should('not.include', '/login')
  }
})

// Usage with options
cy.login('user@test.com', 'pass123', { validate: false })
```

Command Chaining:

```
// cypress/support/commands.js
Cypress.Commands.add('getByCy', (selector) => {
  return cy.get(`[data-cy="${selector}"]`)
})

// Use in test (chainable)
cy.getByCy('submit-button').click()
cy.getByCy('email-input').type('test@example.com')
```

Parent Commands:

```
  cy.contains('Logout').click()
  cy.url().should('include', '/login')
})

cy.logout()
```

Child Commands:

```
// Command that operates on previous subject
Cypress.Commands.add('clearSession', { prevSubject: true }, (subject) => {
  cy.clearCookies()
  cy.clearLocalStorage()
  return cy.wrap(subject)
})

// Use
cy.visit('/').clearSession()
```

Dual Commands:

```
// Can be used with or without subject
Cypress.Commands.add('console', { prevSubject: 'optional' }, (subject, mes
  if (subject) {
    console.log(message, subject)
    return subject
  } else {
    console.log(message)
  }
})

// Both work
cy.get('button').console('Button element:')
cy.console('No subject')
```

Advanced Custom Command Examples

Login with Session:

```
    cy.get('#email').type(email)
    cy.get('#password').type(password)
    cy.get('button[type="submit"]').click()
    cy.url().should('include', '/dashboard')
  })
})

// Session is cached - login happens once per unique email/password
cy.loginWithSession('user@test.com', 'password')
```

Fill Form Command:

```
Cypress.Commands.add('fillForm', (formData) => {
  Object.keys(formData).forEach(key => {
    const value = formData[key]
    const field = cy.get(`[name="${key}"]`)

    if (value.type === 'select') {
      field.select(value.value)
    } else if (value.type === 'checkbox') {
      if (value.checked) {
        field.check()
      }
    } else {
      field.type(value)
    }
  })
})

// Usage
cy.fillForm({
  firstName: 'John',
  lastName: 'Doe',
  email: 'john@example.com',
  country: { type: 'select', value: 'US' },
  terms: { type: 'checkbox', checked: true }
})
```

API Request Wrapper:

```
    url: Cypress.env('apiUrl') + url,
    body: body,
    headers: {
      'Authorization': `Bearer ${Cypress.env('authToken')}`,
      'Content-Type': 'application/json'
    },
    failOnStatusCode: false
  })
})

// Usage
cy.apiRequest('GET', '/users/1').then(response => {
  expect(response.status).to.eq(200)
})
```

Wait for Element and Perform Action:

```
Cypress.Commands.add('waitAndClick', (selector, timeout = 10000) => {
  cy.get(selector, { timeout })
    .should('be.visible')
    .and('not.be.disabled')
    .click()
})

cy.waitAndClick('[data-cy="submit"]')
```

Upload File Command:

```
Cypress.Commands.add('uploadFile', (selector, fileName, fileType = 'text/p
  cy.get(selector).selectFile({
    contents: Cypress.Buffer.from('file contents'),
    fileName: fileName,
    mimeType: fileType,
  })
})

cy.uploadFile('input[type="file"]', 'test.pdf', 'application/pdf')
```

TypeScript Support for Custom Commands

```
// cypress/support/commands.ts
declare global {
  namespace Cypress {
    interface Chainable {
      login(email: string, password: string): Chainable<void>
      getByCy(selector: string): Chainable<JQuery<HTMLElement>>
      fillForm(formData: Record<string, any>): Chainable<void>
      apiRequest(method: string, url: string, body?: any): Chainable<Respo
    }
  }
}

export {}
```

Fixture Management

Complex Fixture Structure:

```
// cypress/fixtures/test-data.json
{
  "users": {
    "admin": {
      "email": "admin@test.com",
      "password": "admin123",
      "role": "admin",
      "permissions": ["read", "write", "delete"]
    },
    "user": {
      "email": "user@test.com",
      "password": "user123",
      "role": "user",
      "permissions": ["read"]
    }
  },
  "products": [
    {
      "id": 1,
      "name": "Product 1",
      "price": 29.99,
      "stock": 100
    },
    {
      "id": 2,
```

```
    }
  ],
  "config": {
    "apiUrl": "https://api.example.com",
    "timeout": 10000
  }
}
```

Load and Use:

```
describe('Tests with Fixtures', () => {
  let testData

  before(() => {
    cy.fixture('test-data.json').then(data => {
      testData = data
    })
  })

  it('login as admin', () => {
    const admin = testData.users.admin
    cy.login(admin.email, admin.password)
  })

  it('verify products', () => {
    cy.visit('/products')
    testData.products.forEach(product => {
      cy.contains(product.name).should('be.visible')
    })
  })
})
```

Dynamic Fixtures:

```
// cypress/fixtures/generate-data.js
const faker = require('@faker-js/faker')

function generateUser() {
  return {
    firstName: faker.name.firstName(),
    lastName: faker.name.lastName(),
    email: faker.internet.email(),
```

```
}

function generateUsers(count) {
  return Array.from({ length: count }, () => generateUser())
}

module.exports = {
  generateUser,
  generateUsers,
}
```

Use Dynamic Fixtures:

```
 const { generateUsers } = require('../fixtures/generate-data')

describe('User Registration', () => {
  const users = generateUsers(5)

  users.forEach((user, index) => {
    it(`register user ${index + 1}`, () => {
      cy.visit('/register')
      cy.get('#firstName').type(user.firstName)
      cy.get('#lastName').type(user.lastName)
      cy.get('#email').type(user.email)
      cy.get('#password').type(user.password)
      cy.get('button').click()

      cy.contains('Registration successful').should('be.visible')
    })
  })
})
```

Data-Driven Testing in Cypress

Array-Based Data-Driven Tests

Simple iteration:

```
  { email: 'admin@test.com', password: 'admin', expected: 'Admin Panel' },
]

loginCredentials.forEach((credentials) => {
  it(`login as ${credentials.email}`, () => {
    cy.visit('/login')
    cy.get('#email').type(credentials.email)
    cy.get('#password').type(credentials.password)
    cy.get('button').click()

    cy.contains(credentials.expected).should('be.visible')
  })
})
```

Validation Testing:

```
 const invalidInputs = [
  { field: 'email', value: 'invalid-email', error: 'Invalid email format'
  { field: 'email', value: '', error: 'Email is required' },
  { field: 'password', value: '123', error: 'Password too short' },
  { field: 'password', value: '', error: 'Password is required' },
]

invalidInputs.forEach((test) => {
  it(`should validate ${test.field}: ${test.error}`, () => {
    cy.visit('/register')
    cy.get(`#${test.field}`).type(test.value)
    cy.get('#submit').click()

    cy.get(`.error-${test.field}`).should('contain', test.error)
  })
})
```

Fixture-Based Data-Driven Tests

Test data fixture:

```
 // cypress/fixtures/registration-tests.json
{
  "validUsers": [
```

```
      "password": "SecurePass123",
      "age": 30,
      "expectedMessage": "Registration successful"
    },
    {
      "name": "Jane Smith",
      "email": "jane@test.com",
      "password": "SecurePass456",
      "age": 25,
      "expectedMessage": "Registration successful"
    }
  ],
  "invalidUsers": [
    {
      "name": "",
      "email": "test@test.com",
      "password": "pass",
      "age": 20,
      "expectedError": "Name is required"
    },
    {
      "name": "Test User",
      "email": "invalid-email",
      "password": "pass",
      "age": 20,
      "expectedError": "Invalid email"
    }
  ]
}
```


Use in tests:

```
import registrationTests from '../fixtures/registration-tests.json'

describe('User Registration Tests', () => {
  context('Valid Registrations', () => {
    registrationTests.validUsers.forEach((user) => {
      it(`register ${user.name}`, () => {
        cy.visit('/register')
        cy.get('#name').type(user.name)
        cy.get('#email').type(user.email)
        cy.get('#password').type(user.password)
        cy.get('#age').type(user.age.toString())
```

```
      })
    })
  })

  context('Invalid Registrations', () => {
    registrationTests.invalidUsers.forEach((user) => {
      it(`reject: ${user.expectedError}`, () => {
        cy.visit('/register')
        if (user.name) cy.get('#name').type(user.name)
        if (user.email) cy.get('#email').type(user.email)
        if (user.password) cy.get('#password').type(user.password)
        cy.get('button[type="submit"]').click()

        cy.get('.error-message').should('contain', user.expectedError)
      })
    })
  })
})
```

CSV Data Source

Read CSV in Cypress:

```
 // cypress/support/csv-helper.js
const parse = require('csv-parse/sync')

Cypress.Commands.add('parseCsv', (filePath) => {
  return cy.readFile(filePath).then((csv) => {
    return parse.parse(csv, {
      columns: true,
      skip_empty_lines: true,
    })
  })
})
```

CSV fixture:

```
 # cypress/fixtures/users.csv
name,email,password,role
John Doe,john@test.com,pass123,user
Jane Smith,jane@test.com,pass456,admin
```

Use in test:

```
describe('CSV Data Tests', () => {
  let users

  before(() => {
    cy.parseCsv('cypress/fixtures/users.csv').then((data) => {
      users = data
    })
  })

  it('create users from CSV', () => {
    users.forEach((user) => {
      cy.request('POST', '/api/users', user).then((response) => {
        expect(response.status).to.eq(201)
      })
    })
  })
})
```

Test Matrix Pattern

Combinatorial testing:

```
 const browsers = ['chrome', 'firefox', 'edge']
const viewports = [
  { name: 'mobile', width: 375, height: 667 },
  { name: 'tablet', width: 768, height: 1024 },
  { name: 'desktop', width: 1920, height: 1080 },
]

viewports.forEach((viewport) => {
  describe(`Tests on ${viewport.name}`, () => {
    beforeEach(() => {
      cy.viewport(viewport.width, viewport.height)
    })

    it('should display responsive layout', () => {
      cy.visit('/')
      cy.get('.header').should('be.visible')

      if (viewport.name === 'mobile') {
```

```
      }
    })
  })
})
```

Network Request Interception with cy.intercept()

Basic Interception

Intercept GET request:

```
it('intercept API call', () => {
  cy.intercept('GET', '/api/users').as('getUsers')

  cy.visit('/users')
  cy.wait('@getUsers').then((interception) => {
    expect(interception.response.statusCode).to.eq(200)
    expect(interception.response.body).to.be.an('array')
  })
})
```

Intercept with URL pattern:

```
// Match all user endpoints
cy.intercept('GET', '/api/users/*').as('getUserById')
cy.intercept('POST', '/api/users').as('createUser')
cy.intercept('PUT', '/api/users/*').as('updateUser')
cy.intercept('DELETE', '/api/users/*').as('deleteUser')
```

Intercept with regex:

```
cy.intercept('GET', /\/api\/products\/\d+/).as('getProduct')
cy.intercept({
  method: 'GET',
  url: /\/api\/users\?page=\d+/
```

Stubbing Responses

Return mock data:

```
it('stub API response', () => {
  cy.intercept('GET', '/api/users', {
    statusCode: 200,
    body: [
      { id: 1, name: 'John Doe', email: 'john@test.com' },
      { id: 2, name: 'Jane Smith', email: 'jane@test.com' },
    ],
  }).as('getUsers')

  cy.visit('/users')
  cy.wait('@getUsers')

  cy.contains('John Doe').should('be.visible')
  cy.contains('Jane Smith').should('be.visible')
})
```

Stub with fixture:

```
it('stub with fixture', () => {
  cy.intercept('GET', '/api/products', {
    fixture: 'products.json'
  }).as('getProducts')

  cy.visit('/products')
  cy.wait('@getProducts')
})
```

Dynamic response:

```
it('dynamic stub response', () => {
  cy.intercept('GET', '/api/users/*', (req) => {
    const userId = req.url.split('/').pop()

    req.reply({
      statusCode: 200,
      body: {
```

```
      },
    })
  }).as('getUser')

  cy.visit('/users/123')
  cy.wait('@getUser')
})
```

Modifying Requests

Add headers:

```
cy.intercept('POST', '/api/users', (req) => {
  req.headers['x-custom-header'] = 'custom-value'
  req.continue()
})
```

Modify request body:

```
cy.intercept('POST', '/api/users', (req) => {
  req.body.createdBy = 'test-suite'
  req.body.timestamp = new Date().toISOString()
  req.continue()
})
```

Modify response:

```
cy.intercept('GET', '/api/users', (req) => {
  req.continue((res) => {
    // Add extra field to each user
    res.body = res.body.map(user => ({
      ...user,
      testField: 'added-by-test'
    }))
  })
})
```

```
it('handle 404 error', () => {
  cy.intercept('GET', '/api/users/999', {
    statusCode: 404,
    body: {
      error: 'User not found'
    }
  }).as('getUser')

  cy.visit('/users/999')
  cy.wait('@getUser')

  cy.contains('User not found').should('be.visible')
})
```

Simulate 500 error:

```
it('handle server error', () => {
  cy.intercept('POST', '/api/users', {
    statusCode: 500,
    body: {
      error: 'Internal server error'
    }
  }).as('createUser')

  cy.visit('/users/new')
  cy.get('#name').type('Test User')
  cy.get('#email').type('test@example.com')
  cy.get('button').click()

  cy.wait('@createUser')
  cy.get('.error-message').should('contain', 'server error')
})
```

Network failure:

```
it('handle network failure', () => {
  cy.intercept('GET', '/api/users', {
    forceNetworkError: true
  }).as('getUsers')
```

```
  cy.contains('Network error').should('be.visible')
})
```

Delay Simulation

Slow response:

```
it('test loading state', () => {
  cy.intercept('GET', '/api/users', (req) => {
    req.reply((res) => {
      res.delay = 2000  // 2 second delay
    })
  }).as('getUsers')

  cy.visit('/users')

  // Loading indicator should appear
  cy.get('.loading-spinner').should('be.visible')

  cy.wait('@getUsers')

  // Loading should disappear
  cy.get('.loading-spinner').should('not.exist')
})
```

Advanced Interception Patterns

Conditional stubbing:

```
cy.intercept('GET', '/api/users', (req) => {
  const page = new URL(req.url).searchParams.get('page')

  if (page === '1') {
    req.reply({ fixture: 'users-page-1.json' })
  } else if (page === '2') {
    req.reply({ fixture: 'users-page-2.json' })
  } else {
    req.reply({ statusCode: 404 })
  }
```

Request/Response logging:

```
cy.intercept('**/api/**', (req) => {
  cy.log(`Request: ${req.method} ${req.url}`)

  req.continue((res) => {
    cy.log(`Response: ${res.statusCode}`)
    cy.log(`Body: ${JSON.stringify(res.body)}`)
  })
})
```

Track all API calls:

```
const apiCalls = []

beforeEach(() => {
  cy.intercept('**/api/**', (req) => {
    req.continue((res) => {
      apiCalls.push({
        method: req.method,
        url: req.url,
        status: res.statusCode,
        duration: res.duration
      })
    })
  })
})

afterEach(() => {
  cy.log(`Total API calls: ${apiCalls.length}`)
  apiCalls.forEach(call => {
    cy.log(`${call.method} ${call.url} - ${call.status} (${call.duration}m
  })
})
```

API Testing with Cypress

```
it('GET request', () => {
  cy.request('GET', 'https://jsonplaceholder.typicode.com/users/1')
    .then((response) => {
      expect(response.status).to.eq(200)
      expect(response.body).to.have.property('name')
      expect(response.body).to.have.property('email')
    })
})
```

POST request:

```
it('POST request', () => {
  cy.request({
    method: 'POST',
    url: 'https://jsonplaceholder.typicode.com/users',
    body: {
      name: 'John Doe',
      email: 'john@example.com',
      username: 'johndoe'
    }
  }).then((response) => {
    expect(response.status).to.eq(201)
    expect(response.body).to.have.property('id')
    expect(response.body.name).to.eq('John Doe')
  })
})
```

With headers:

```
it('request with headers', () => {
  cy.request({
    method: 'GET',
    url: '/api/protected',
    headers: {
      'Authorization': 'Bearer ' + Cypress.env('authToken'),
      'Content-Type': 'application/json'
    }
  }).then((response) => {
    expect(response.status).to.eq(200)
```

Complete CRUD Operations

Create:

```
it('create user', () => {
  cy.request('POST', `${Cypress.env('apiUrl')}/users`, {
    name: 'Test User',
    email: 'test@example.com'
  }).then((response) => {
    expect(response.status).to.eq(201)
    expect(response.body).to.have.property('id')

    // Save ID for later tests
    Cypress.env('userId', response.body.id)
  })
})
```

Read:

```
it('get user', () => {
  const userId = Cypress.env('userId')

  cy.request('GET', `${Cypress.env('apiUrl')}/users/${userId}`)
    .then((response) => {
      expect(response.status).to.eq(200)
      expect(response.body.name).to.eq('Test User')
    })
})
```

Update:

```
it('update user', () => {
  const userId = Cypress.env('userId')

  cy.request('PUT', `${Cypress.env('apiUrl')}/users/${userId}`, {
    name: 'Updated Name',
    email: 'updated@example.com'
  }).then((response) => {
    expect(response.status).to.eq(200)
```

Delete:

```
it('delete user', () => {
  const userId = Cypress.env('userId')

  cy.request('DELETE', `${Cypress.env('apiUrl')}/users/${userId}`)
    .then((response) => {
      expect(response.status).to.eq(204)
    })

  // Verify deleted
  cy.request({
    method: 'GET',
    url: `${Cypress.env('apiUrl')}/users/${userId}`,
    failOnStatusCode: false
  }).then((response) => {
    expect(response.status).to.eq(404)
  })
})
```

Authentication Patterns

Login and save token:

```
it('login and get token', () => {
  cy.request({
    method: 'POST',
    url: `${Cypress.env('apiUrl')}/auth/login`,
    body: {
      email: 'user@example.com',
      password: 'password123'
    }
  }).then((response) => {
    expect(response.status).to.eq(200)
    expect(response.body).to.have.property('token')

    // Save token for subsequent requests
    Cypress.env('authToken', response.body.token)
  })
```

Use token in requests:

```
it('access protected endpoint', () => {
  cy.request({
    method: 'GET',
    url: `${Cypress.env('apiUrl')}/users/me`,
    headers: {
      'Authorization': `Bearer ${Cypress.env('authToken')}`
    }
  }).then((response) => {
    expect(response.status).to.eq(200)
  })
})
```

API + UI Hybrid Testing

Setup with API, test with UI:

```
describe('Product Tests', () => {
  let productId

  before(() => {
    // Create test data via API
    cy.request('POST', `${Cypress.env('apiUrl')}/products`, {
      name: 'Test Product',
      price: 29.99,
      stock: 100
    }).then((response) => {
      productId = response.body.id
    })
  })

  it('view product in UI', () => {
    cy.visit(`/products/${productId}`)
    cy.contains('Test Product').should('be.visible')
    cy.contains('$29.99').should('be.visible')
  })

  after(() => {
    // Cleanup via API
    cy.request('DELETE', `${Cypress.env('apiUrl')}/products/${productId}`)
  })
```

Verify UI action with API:

```
it('create product via UI, verify with API', () => {
  // UI action
  cy.visit('/products/new')
  cy.get('#name').type('New Product')
  cy.get('#price').type('49.99')
  cy.get('button').click()

  cy.contains('Product created').should('be.visible')

  // Get product ID from URL
  cy.url().should('include', '/products/')
  cy.url().then((url) => {
    const productId = url.split('/').pop()

    // Verify via API
    cy.request('GET', `${Cypress.env('apiUrl')}/products/${productId}`)
      .then((response) => {
        expect(response.status).to.eq(200)
        expect(response.body.name).to.eq('New Product')
        expect(response.body.price).to.eq(49.99)
      })
  })
})
```

Response Validation

Status code validation:

```
cy.request({
  method: 'GET',
  url: '/api/users',
  failOnStatusCode: false
}).then((response) => {
  expect(response.status).to.be.oneOf([200, 304])
})
```

Response headers:

```
})
```

Response body structure:

```
cy.request('/api/users/1').then((response) => {
  expect(response.body).to.have.all.keys('id', 'name', 'email', 'username'
  expect(response.body.id).to.be.a('number')
  expect(response.body.name).to.be.a('string')
  expect(response.body.email).to.match(/^.+@.+\..+$/)
})
```

Array validation:

```
cy.request('/api/users').then((response) => {
  expect(response.body).to.be.an('array')
  expect(response.body).to.have.length.greaterThan(0)

  response.body.forEach(user => {
    expect(user).to.have.property('id')
    expect(user).to.have.property('name')
    expect(user).to.have.property('email')
  })
})
```

Environment Variables and Plugins

Environment Variables

Define in cypress.config.js:

```
module.exports = {
  e2e: {
    baseUrl: 'http://localhost:3000',
    env: {
```

```
      timeout: 10000
    }
  }
}
```

cypress.env.json:

```
{
  "apiUrl": "https://api.staging.example.com",
  "authToken": "secret-token",
  "testUserId": 12345
}
```

Access in tests:

```
const apiUrl = Cypress.env('apiUrl')
const timeout = Cypress.env('timeout')

cy.visit(apiUrl + '/users')
```

Set dynamically:

```
Cypress.env('authToken', 'new-token')
```

Command line:

```
npx cypress run --env apiUrl=https://prod.api.com,debug=true
```

Different environments:

```
// package.json
{
  "scripts": {
    "cy:dev": "cypress open --env ENV=development",
    "cy:staging": "cypress run --env ENV=staging",
    "cy:prod": "cypress run --env ENV=production"
```

Plugins

Setup Node Events:

```
 // cypress.config.js
const { defineConfig } = require('cypress')

module.exports = defineConfig({
  e2e: {
    setupNodeEvents(on, config) {
      // Plugin code here

      // Task example
      on('task', {
        log(message) {
          console.log(message)
          return null
        },

        queryDb(query) {
          // Database query logic
          return result
        }
      })

      return config
    }
  }
})
```

Use tasks:

```
 it('use task', () => {
  cy.task('log', 'Custom log message')
  cy.task('queryDb', 'SELECT * FROM users').then((result) => {
    cy.log(result)
  })
})
```

Popular Plugins:

```
npm install --save-dev cypress-file-upload
```

```
// cypress/support/e2e.js
import 'cypress-file-upload'

// Usage
cy.get('input[type="file"]').attachFile('example.json')
```

2. cypress-xpath:

```
npm install --save-dev cypress-xpath
```

```
// cypress/support/e2e.js
require('cypress-xpath')

// Usage
cy.xpath('//button[text()="Submit"]').click()
```

3. @testing-library/cypress:

```
npm install --save-dev @testing-library/cypress
```

```
// cypress/support/e2e.js
import '@testing-library/cypress/add-commands'

// Usage
cy.findByRole('button', { name: /submit/i }).click()
cy.findByLabelText('Email').type('test@example.com')
```

Introduction to API Performance Testing

```
it('measure response time', () => {
  const start = Date.now()

  cy.request('/api/users').then((response) => {
    const duration = Date.now() - start

    cy.log(`Response time: ${duration}ms`)
    expect(duration).to.be.lessThan(1000)  // Under 1 second
  })
})
```

Using cy.intercept():

```
it('measure with intercept', () => {
  cy.intercept('GET', '/api/users').as('getUsers')

  cy.visit('/users')

  cy.wait('@getUsers').then((interception) => {
    const duration = interception.response.duration
    cy.log(`API response time: ${duration}ms`)
    expect(duration).to.be.lessThan(500)
  })
})
```

Multiple Request Timing

Sequential requests:

```
it('measure multiple requests', () => {
  const timings = []

  cy.request('/api/users').then((response) => {
    timings.push({ endpoint: '/users', duration: response.duration })
  })

  cy.request('/api/products').then((response) => {
    timings.push({ endpoint: '/products', duration: response.duration })
```

```
    timings.push({ endpoint: '/orders', duration: response.duration })
  })

  cy.wrap(timings).then(() => {
    const total = timings.reduce((sum, t) => sum + t.duration, 0)
    const average = total / timings.length

    cy.log(`Total time: ${total}ms`)
    cy.log(`Average time: ${average}ms`)

    expect(average).to.be.lessThan(1000)
  })
})
```

Load Testing Pattern

Concurrent requests:

```
it('concurrent request test', () => {
  const requests = []
  const count = 10

  for (let i = 0; i < count; i++) {
    requests.push(
      cy.request('/api/users')
    )
  }

  // All requests execute in parallel
  cy.wrap(Promise.all(requests)).then((responses) => {
    responses.forEach((response, index) => {
      expect(response.status).to.eq(200)
      cy.log(`Request ${index + 1}: ${response.duration}ms`)
    })

    const durations = responses.map(r => r.duration)
    const average = durations.reduce((a, b) => a + b, 0) / durations.lengt
    const max = Math.max(...durations)

    cy.log(`Average: ${average}ms, Max: ${max}ms`)
    expect(max).to.be.lessThan(2000)
  })
```

Performance Thresholds

Define SLA thresholds:

```
 const SLA_THRESHOLDS = {
  fastEndpoint: 200,    // ms
  normalEndpoint: 500,
  slowEndpoint: 2000
}

it('verify SLA compliance', () => {
  cy.request('/api/users').then((response) => {
    expect(response.duration).to.be.lessThan(SLA_THRESHOLDS.fastEndpoint)
  })

  cy.request('/api/analytics').then((response) => {
    expect(response.duration).to.be.lessThan(SLA_THRESHOLDS.slowEndpoint)
  })
})
```

Practice Exercises

Exercise 1: Custom Commands Library

Task: Create comprehensive custom command library

```
 // cypress/support/commands.js

// Create these commands:
// 1. cy.login(email, password, options)
// 2. cy.logout()
// 3. cy.fillForm(formData)
// 4. cy.getByCy(selector)
// 5. cy.apiRequest(method, url, body)
// 6. cy.waitForLoad()
// 7. cy.clearSession()
```

```
describe('Custom Commands', () => {
  it('use all commands', () => {
    cy.clearSession()
    cy.login('user@test.com', 'password')
    cy.waitForLoad()
    cy.getByCy('dashboard').should('be.visible')
    cy.logout()
  })
})
```

Exercise 2: Data-Driven Login Tests

Task: Create comprehensive data-driven test suite

```
 // Create fixture: cypress/fixtures/login-test-cases.json
/*
{
  "valid": [
    { "email": "admin@test.com", "password": "admin123", "role": "admin" }
    { "email": "user@test.com", "password": "user123", "role": "user" }
  ],
  "invalid": [
    { "email": "wrong@test.com", "password": "wrong", "error": "Invalid cr
    { "email": "", "password": "test", "error": "Email required" },
    { "email": "test@test.com", "password": "", "error": "Password require
  ]
}
*/

// Implement tests for:
// 1. All valid login scenarios
// 2. All invalid login scenarios
// 3. Verify appropriate messages/redirects
// 4. Verify role-based access

// Your implementation
```

Exercise 3: Network Interception Suite

Task: Complete network interception tests

```
    // 2. Visit users page
    // 3. Wait for request
    // 4. Verify request headers
    // 5. Verify response structure

    // Your implementation
  })

  it('stub API response', () => {
    // 1. Intercept GET /api/products
    // 2. Return mock products
    // 3. Verify mock data appears in UI

    // Your implementation
  })

  it('simulate API error', () => {
    // 1. Intercept POST /api/orders
    // 2. Return 500 error
    // 3. Verify error handling in UI

    // Your implementation
  })

  it('test loading state', () => {
    // 1. Intercept request with delay
    // 2. Verify loading spinner appears
    // 3. Verify spinner disappears after response

    // Your implementation
  })
})
```

Exercise 4: API Testing Suite

Task: Complete CRUD API tests

```
describe('API Testing Suite', () => {
  let userId

  it('create user', () => {
    // POST /api/users
```

```
  it('get user', () => {
    // GET /api/users/:id
    // Verify user data
    // Your implementation
  })

  it('update user', () => {
    // PUT /api/users/:id
    // Verify updated data
    // Your implementation
  })

  it('delete user', () => {
    // DELETE /api/users/:id
    // Verify deleted (404)
    // Your implementation
  })

  it('list users with pagination', () => {
    // GET /api/users?page=1&limit=10
    // Verify pagination works
    // Your implementation
  })
})
```

Exercise 5: Hybrid UI + API Testing

Task: Combine UI and API testing

```
describe('Hybrid Testing', () => {
  it('setup with API, test with UI', () => {
    // 1. Create user via API
    // 2. Create products via API
    // 3. Login via UI
    // 4. Add products to cart via UI
    // 5. Verify cart via API
    // 6. Complete checkout via UI
    // 7. Verify order via API

    // Your implementation
  })
```

```
    // 3. Verify user data matches

    // Your implementation
  })
})
```

Exercise 6: Performance Testing

Task: Measure and validate performance

```
describe('Performance Tests', () => {
  it('measure API response times', () => {
    // Test multiple endpoints
    // Measure response times
    // Calculate average/min/max
    // Verify SLA compliance

    // Your implementation
  })

  it('concurrent request test', () => {
    // Make 10 concurrent requests
    // Measure all response times
    // Verify all successful
    // Check performance degradation

    // Your implementation
  })

  it('page load performance', () => {
    // Intercept all network requests
    // Track timing for each
    // Calculate total page load time
    // Verify under threshold

    // Your implementation
  })
})
```

Exercise 7: Complete E2E Workflow

```
describe('E2E User Journey', () => {
  it('complete shopping flow', () => {
    // 1. Intercept API calls
    // 2. Register new user
    // 3. Verify via API
    // 4. Login
    // 5. Browse products
    // 6. Add to cart
    // 7. Verify cart via API
    // 8. Checkout
    // 9. Complete payment
    // 10. Verify order via API
    // 11. Check order history

    // Use custom commands
    // Use fixtures for data
    // Use intercepts for network
    // Verify with API

    // Your implementation
  })
})
```

Summary

In Day 9, you mastered:

✅ Custom Commands

- Creating reusable commands
- Parent, child, dual commands
- TypeScript support
- Command libraries

✅ Data-Driven Testing

- Array-based iteration
- Fixture-based tests

✅ cy.intercept()

- Request interception
- Response stubbing
- Error simulation
- Delay testing
- Request/response modification

✅ API Testing

- cy.request() usage
- CRUD operations
- Authentication patterns
- API + UI hybrid testing
- Response validation

✅ Environment & Plugins

- Environment variables
- Plugin configuration
- Popular plugins
- Task creation

✅ Performance Testing

- Response time measurement
- Concurrent requests
- SLA validation
- Performance thresholds

Key Takeaways

- **Custom commands** increase maintainability

- **cy.request()** makes API testing simple
- **Hybrid approach** combines best of both worlds
- **Performance testing** ensures SLAs

Best Practices

1. Build comprehensive command library
2. Use fixtures for test data
3. Intercept network calls for speed
4. Combine UI and API testing
5. Set environment-specific configs
6. Measure performance consistently
7. Create reusable patterns

Next Steps

- Master all custom commands
- Build data-driven suite
- Practice network interception
- Complete API test coverage
- Implement hybrid patterns
- Measure performance
- Complete all exercises

End of Day 9 Documentation