Creating a comprehensive API automation framework in Cypress to handle various types of requests and responses involves setting up reusable functions, configuring Cypress, and structuring the project to handle different environments and scenarios. Below is an example framework that provides a foundation for API automation, covering GET, POST, PUT, DELETE requests, and response handling.

### Project Setup

1. **Install Cypress**:

npm install cypress --save-dev

**Create the Directory Structure**:

cypress/

integration/

api-tests/

getRequest.spec.js

postRequest.spec.js

putRequest.spec.js

deleteRequest.spec.js

support/

commands.js

index.js

apiCommands.js

**Define API Commands in apiCommands.js**:

Create reusable functions for API requests.

// cypress/support/apiCommands.js

Cypress.Commands.add('apiGet', (url, headers = {}) => {

return cy.request({

method: 'GET',

url: url,

headers: headers,

});

});

Cypress.Commands.add('apiPost', (url, body, headers = {}) => {

return cy.request({

method: 'POST',

url: url,

body: body,

headers: headers,

});

});

Cypress.Commands.add('apiPut', (url, body, headers = {}) => {

return cy.request({

method: 'PUT',

url: url,

body: body,

headers: headers,

});

});

Cypress.Commands.add('apiDelete', (url, headers = {}) => {

return cy.request({

method: 'DELETE',

url: url,

headers: headers,

});

});

**Include apiCommands.js in index.js**:

Ensure the custom commands are loaded.

// cypress/support/index.js

import './apiCommands';

### Writing API Tests

#### GET Request Test

// cypress/integration/api-tests/getRequest.spec.js

describe('GET Request', () => {

it('should fetch data successfully', () => {

cy.apiGet('/api/endpoint')

.then((response) => {

expect(response.status).to.eq(200);

expect(response.body).to.have.property('key', 'expectedValue');

});

});

});

POST Request Test

// cypress/integration/api-tests/postRequest.spec.js

describe('POST Request', () => {

it('should post data successfully', () => {

const requestBody = {

key: 'value'

};

cy.apiPost('/api/endpoint', requestBody)

.then((response) => {

expect(response.status).to.eq(201);

expect(response.body).to.have.property('key', 'value');

});

});

});

PUT Request Test

// cypress/integration/api-tests/putRequest.spec.js

describe('PUT Request', () => {

it('should update data successfully', () => {

const requestBody = {

key: 'newValue'

};

cy.apiPut('/api/endpoint/1', requestBody)

.then((response) => {

expect(response.status).to.eq(200);

expect(response.body).to.have.property('key', 'newValue');

});

});

});

DELETE Request Test

// cypress/integration/api-tests/deleteRequest.spec.js

describe('DELETE Request', () => {

it('should delete data successfully', () => {

cy.apiDelete('/api/endpoint/1')

.then((response) => {

expect(response.status).to.eq(204);

});

});

});

### Running the Tests

1. **Open Cypress Test Runner**:

npx cypress open

1. **Run Tests**: Select the test files from the Cypress Test Runner interface.

### Advanced Configuration

1. **Environment Variables**: Use cypress.json to define environment variables for different environments (e.g., dev, staging, prod).

{

"baseUrl": "https://api.yourservice.com",

"env": {

"apiToken": "your\_api\_token\_here"

}

}

**Handling Authentication**: If your API requires authentication, you can add a command to handle the login and store the token.

// cypress/support/apiCommands.js

Cypress.Commands.add('apiLogin', (username, password) => {

cy.request({

method: 'POST',

url: '/api/login',

body: {

username: username,

password: password

}

}).then((response) => {

expect(response.status).to.eq(200);

Cypress.env('apiToken', response.body.token); // Save token to environment variables

});

});

Use the stored token in your requests:

Cypress.Commands.add('apiGetWithAuth', (url) => {

return cy.request({

method: 'GET',

url: url,

headers: {

'Authorization': `Bearer ${Cypress.env('apiToken')}`

}

});

});

Example Authentication Test

// cypress/integration/api-tests/authRequest.spec.js

describe('Authenticated Requests', () => {

before(() => {

cy.apiLogin('username', 'password');

});

it('should fetch data with authentication', () => {

cy.apiGetWithAuth('/api/secure-endpoint')

.then((response) => {

expect(response.status).to.eq(200);

expect(response.body).to.have.property('secureKey', 'secureValue');

});

});

});

This framework provides a robust starting point for API automation using Cypress, covering various types of requests, response validation, and authentication. Adjust the structure and commands as needed to fit your specific use case.