Cypress Configuration and Setup for QA Dashboard

This guide explains how to configure Cypress to work with your QA Dashboard and covers project organization and reporter setup.

Project Structure

First, let's organize your project with a clear structure:

```
juice-shop-qa/
├── cypress/
│   ├── integration/
│   │   ├── cypress-test-examples.js       # Your existing test files
│   │   └── cypress-user-management-tests.js
│   ├── fixtures/
│   │   └── test-data.json                 # Test data
│   ├── plugins/
│   │   └── index.js                       # Cypress plugins
│   ├── results/                           # Test results will be stored here
│   ├── reports/                           # HTML reports will be generated here
│   └── support/
│       ├── commands.js                    # Custom commands
│       └── index.js
├── dashboard/
│   ├── frontend/                          # React dashboard frontend
│   └── backend/                           # Express API server
│       └── server.js                      # Backend implementation
├── cypress.json                           # Cypress configuration
├── package.json                           # Project dependencies
└── README.md
```

Required Dependencies

Add these to your `package.json`:

```json
{
  "name": "juice-shop-qa",
  "version": "1.0.0",
  "description": "QA Testing Dashboard for OWASP Juice Shop",
  "scripts": {
    "cy:run": "cypress run",
    "cy:open": "cypress open",
    "test": "cypress run",
    "report:merge": "mochawesome-merge cypress/results/*.json > cypress/results/mochaw
    "report:generate": "marge cypress/results/mochawesome.json -f report -o cypress/re
    "test:ci": "npm run cy:run && npm run report:merge && npm run report:generate",
    "start:api": "node dashboard/backend/server.js",
    "start:dashboard": "cd dashboard/frontend && npm start",
    "dev": "concurrently \"npm run start:api\" \"npm run start:dashboard\""
  },
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "fs-extra": "^11.1.1"
  },
  "devDependencies": {
    "concurrently": "^8.2.1",
    "cypress": "^13.3.1",
    "cypress-multi-reporters": "^1.6.3",
    "mocha": "^10.2.0",
    "mochawesome": "^7.1.3",
    "mochawesome-merge": "^4.3.0",
    "mochawesome-report-generator": "^6.2.0"
  }
}
```

Cypress Configuration

Create a `cypress.json` file in the project root with these settings:

```json
{
  "baseUrl": "https://juice-shop.herokuapp.com",
  "viewportWidth": 1280,
  "viewportHeight": 800,
  "video": true,
  "screenshotOnRunFailure": true,
  "reporter": "cypress-multi-reporters",
  "reporterOptions": {
    "reporterEnabled": "mochawesome",
    "mochawesomeReporterOptions": {
      "reportDir": "cypress/results",
      "overwrite": false,
      "html": false,
      "json": true
    }
  },
  "env": {
    "defaultUser": "user@juice-sh.op",
    "defaultPassword": "password123"
  }
}
```

Setting Up Reporters

The configuration above uses the Mochawesome reporter to generate JSON test results that can be parsed by your backend API. Let's set up the plugins to properly generate these reports.

1. Update Cypress Plugins

Create or update your `cypress/plugins/index.js` file:

```javascript
/// <reference types="cypress" />

const fs = require('fs-extra');
const path = require('path');

module.exports = (on, config) => {
  // Delete results folder before run
  on('before:run', async () => {
    const resultsFolder = path.join(__dirname, '..', 'results');
    if (await fs.pathExists(resultsFolder)) {
      await fs.emptyDir(resultsFolder);
    }
  });

  // Add custom task to save screenshots in results
  on('task', {
    saveScreenshot({ name, screenshotData }) {
      const screenshotsFolder = path.join(__dirname, '..', 'results', 'screenshots');
      if (!fs.existsSync(screenshotsFolder)) {
        fs.mkdirSync(screenshotsFolder, { recursive: true });
      }

      const screenshotPath = path.join(screenshotsFolder, `${name}.png`);
      return fs.writeFile(screenshotPath, screenshotData, 'base64')
        .then(() => screenshotPath)
        .catch(err => err.message);
    }
  });

  return config;
};
```

2. Add Support for Custom Commands

Update your `cypress/support/commands.js` file to include some useful utilities:

javascript

```javascript
// Custom command to close welcome banners or cookie notices that might interfere with
Cypress.Commands.add('closeBanners', () => {
  cy.get('body').then($body => {
    // Close welcome banner if it exists
    if ($body.find('button[aria-label="Close Welcome Banner"]').length > 0) {
      cy.get('button[aria-label="Close Welcome Banner"]').click();
    }

    // Accept cookies if the dialog exists
    if ($body.find('button[aria-label="dismiss cookie message"]').length > 0) {
      cy.get('button[aria-label="dismiss cookie message"]').click();
    }
  });
});

// Custom command for login
Cypress.Commands.add('login', (email, password) => {
  // Use default credentials from env if not provided
  const userEmail = email || Cypress.env('defaultUser');
  const userPassword = password || Cypress.env('defaultPassword');

  cy.visit('/');
  cy.closeBanners();
  cy.get('#navbarAccount').click();
  cy.get('#navbarLoginButton').click();
  cy.get('#email').type(userEmail);
  cy.get('#password').type(userPassword);
  cy.get('#loginButton').click();

  // Verify successful login
  cy.get('#navbarAccount').should('be.visible');
});

// Custom command to capture and save a screenshot with the test name
Cypress.Commands.add('saveScreenshot', (name) => {
  cy.screenshot(name, { capture: 'viewport' })
    .then(({ path }) => {
      // Convert the screenshot file to base64
      cy.task('saveScreenshot', {
        name,
        screenshotData: Cypress.Blob.base64StringToBlob(
          cy.readFile(path, 'base64').toString(),
          'image/png'
        )
      });
```

```
    });
});
```

Integrating with the Dashboard

To ensure your Cypress tests work with the dashboard, follow these steps:

1. Make sure all your test files are in the `cypress/integration` directory
2. Run tests with the proper reporters enabled
3. Connect the backend API to the Cypress results directory

Running Tests

You can run tests in several ways:

1. **Interactive mode** with Cypress Test Runner:

   ```
   npm run cy:open
   ```

2. **Headless mode** for CI/CD or the dashboard:

   ```
   npm run cy:run
   ```

3. **Generate a full report** after running tests:

   ```
   npm run test:ci
   ```

Setting Up a Complete Development Environment

To run both the dashboard and tests locally:

1. Install dependencies:

   ```
   npm install
   ```

2. Install frontend dependencies:

   ```
   cd dashboard/frontend
   npm install
   ```

3. Start both the backend API and frontend dashboard:

   ```
   npm run dev
   ```

4. Access the dashboard at http://localhost:3000

Customizing Test Execution

You can run specific test suites by passing the `--spec` parameter to Cypress:

```bash
npx cypress run --spec "cypress/integration/cypress-user-management-tests.js"
```

Or through the API:

```bash
curl -X POST http://localhost:5000/api/tests/run \
  -H "Content-Type: application/json" \
  -d '{"suites": ["cypress-user-management-tests.js"], "browser": "chrome"}'
```

Setting Up Continuous Integration

For GitHub Actions, create a `.github/workflows/cypress.yml` file:

```yaml
name: Cypress Tests

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  cypress-run:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Cypress run
        uses: cypress-io/github-action@v5
        with:
          build: npm run build
          start: npm run start:api
          wait-on: 'http://localhost:5000'
          record: true

      - name: Generate reports
        run: npm run report:merge && npm run report:generate

      - name: Upload reports
        uses: actions/upload-artifact@v3
        with:
          name: cypress-reports
          path: cypress/reports
```

Troubleshooting Common Issues

Issue: Tests Pass in Cypress Runner But Fail in Dashboard

1. Check the error logs in your test results
2. Verify that the selectors are stable and not changing between runs
3. Add wait or retry logic for flaky elements

Issue: Reports Not Generating Correctly

1. Make sure the results directory exists and is writable
2. Check that the Mochawesome reporter is configured correctly
3. Run `npm run report:merge` manually to see if there are any errors

Issue: Dashboard Can't Connect to Backend

1. Verify the backend is running on the expected port
2. Check for CORS issues in the browser console
3. Ensure the API_BASE_URL in the frontend code is correct