

# Testing using Cypress and Cucumber in JavaScript

- [Overview](#)
- [Requirements](#)
- [Description](#)
  - [Using Jira and Xray as master](#)
  - [Using Git or other VCS as master](#)
- [FAQ and Recommendations](#)
- [References](#)

## Overview

In this tutorial, we will create some UI tests as Cucumber Scenario(s)/Scenario Outline(s) and use [Cypress](#) to implement the tests in JavaScript.

### Source-code for this tutorial

Code is available in [GitHub](#); the repo contains some auxiliary scripts.

## Requirements

- Node.js
- cucumber-json-formatter
- npm packages
  - cypress
  - [@badeball/cypress-cucumber-preprocessor](#)

## Description

For the purpose of this tutorial, we'll use a [dummy website](#) (source-code [here](#)) containing just a few pages to support login/logout kind of features; we aim to test precisely those features.

### Login Page

Please input your user name and password and click the login button.

User Name:

Password:

We need to configure Cypress to use the new [@badeball/cypress-cucumber-preprocessor](#), which provides the ability of understanding .feature files and also of producing Cucumber JSON reports.

This is done in Cypress' main configuration file, where you can also define the base URL of the website under test, the regex of the files that contain the test scenarios (i.e. <...>.feature files). Other options may be defined (e.g for bypassing chromeWebSecurity, additional reporters, the ability to upload results to Cypress infrastructure in the cloud, etc).

#### /cypress.config.js

```
const { defineConfig } = require('cypress')
const createBundler = require("@bahmutov/cypress-esbuild-preprocessor");
const createEsbuildPlugin = require('@badeball/cypress-cucumber-preprocessor/esbuild').createEsbuildPlugin;
const addCucumberPreprocessorPlugin = require('@badeball/cypress-cucumber-preprocessor').
addCucumberPreprocessorPlugin;

async function setupNodeEvents(on, config) {
  await addCucumberPreprocessorPlugin(on, config);

  on(
    "file:preprocessor",
    createBundler({
      plugins: [createEsbuildPlugin(config)],
    })
  );

  // Make sure to return the config object as it might have been modified by the plugin.
  return config;
}

module.exports = defineConfig({
  e2e: {
    baseUrl: "https://robotwebdemo.herokuapp.com/",
    specPattern: "**/*.feature",
    excludeSpecPattern: [
      "*.js",
      "*.md"
    ],
    chromeWebSecurity: false,
    projectId: "bfi83g",
    supportFile: false,
    setupNodeEvents
  }
})
```

The configuration of @badeball/cypress-cucumber-preprocessor

can either be done on a JSON file `.cypress-cucumber-preprocessorrc.json` or within `package.json`.

Next, you may find an example of the contents of `package.json`.

## package.json

```
{
  "name": "tutorial-js-cypress-cucumber",
  "version": "1.0.0",
  "description": "An example for Cypress and Cucumber usage using Robot login demo website",
  "main": "index.js",
  "scripts": {
    "cypress:open:local": "CYPRESS_ENV=localhost npm run cypress:open",
    "cypress:open:prod": "CYPRESS_ENV=production npm run cypress:open",
    "cypress:open": "npx cypress open",
    "test:local": "CYPRESS_ENV=localhost npm run test --spec 'cypress/integration/**/*.feature'",
    "test:prod": "CYPRESS_ENV=production npm run test",
    "test": "npx cypress run --spec 'features/**/*.feature'",
    "test:debug:local": "CYPRESS_ENV=localhost npm run test:debug",
    "test:debug:prod": "CYPRESS_ENV=production npm run test:debug",
    "test:debug": "npx cypress run --headed --browser chrome --env TAGS='@e2e-test' --spec 'cypress/integration/**/*.feature'"
  },
  "author": "Xblend",
  "license": "BSD-3-Clause",
  "dependencies": {
    "axios": "^0.18.0",
    "fs-extra": "^7.0.1",
    "glob": "^7.1.7"
  },
  "devDependencies": {
    "@badeball/cypress-cucumber-preprocessor": "^13.0.2",
    "@bahmutov/cypress-esbuild-preprocessor": "^2.1.3",
    "@cypress/webpack-preprocessor": "latest",
    "cypress": "^10.8.0",
    "esbuild": "^0.14.45",
    "eslint": "^5.13.0",
    "eslint-config-airbnb-base": "^12.1.0",
    "eslint-config-prettier": "^2.9.0",
    "eslint-plugin-cypress": "^2.11.3",
    "eslint-plugin-import": "^2.23.4",
    "eslint-plugin-prettier": "^2.6.0",
    "husky": "^1.3.1",
    "lint-staged": "^8.1.3"
  },
  "cypress-cucumber-preprocessor": {
    "json": {
      "enabled": true,
      "formatter": "/usr/local/bin/cucumber-json-formatter",
      "output": "cucumber-report.json"
    }
  },
  "husky": {
    "hooks": {
      "pre-commit": "lint-staged"
    }
  },
  "lint-staged": {
    "*.js": [
      "eslint",
      "git add"
    ]
  }
}
```

We need to have the `cucumber-json-formatter` tool, which can be downloaded from the [respective GitHub repository](#). Make sure you pick the correct binary for your environment.

This tool is necessary to convert the Cucumber messages protobuf (\*.ndjson file) report generated by `@badeball/cypress-cucumber-preprocessor`

#### example of Bash script to download hte cucumber-json-formatter tool for Linux

```
wget https://github.com/cucumber/json-formatter/releases/download/v19.0.0/cucumber-json-formatter-linux-amd64 -
O /usr/local/bin/cucumber-json-formatter
chmod +x /usr/local/bin/cucumber-json-formatter
```

In case you need to interact with Xray REST API at low-level using scripts (e.g. Bash/shell scripts), this tutorial uses an auxiliary file with the credentials (more info in [Global Settings: API Keys](#)).

#### Example of cloud\_auth.json used in this tutorial

```
{ "client_id": "215FFD69FE4644728C72180000000000", "client_secret":
"1c00f8f22f56a8684d7c18cd6147ce2787d95e4da9f3bfb0af8f020000000000" }
```

Before moving into the actual implementation, we need to decide is which workflow we'll use: do we want to use Xray/Jira as the master for writing the declarative specification (i.e. the Gherkin based Scenarios), or do we want to manage those outside using some editor and store them in Git, for example?



#### Learn more

Please see [Testing in BDD with Gherkin based frameworks \(e.g. Cucumber\)](#) for an overview of the possible workflows.

The place that you'll use to edit the Cucumber Scenarios will affect your workflow. There are teams that prefer to edit Cucumber Scenarios in Jira using Xray, while there others that prefer to edit them by writing the .feature files by hand using some IDE.

## Using Jira and Xray as master

This section assumes using Xray as master, i.e. the place that you'll be using to edit the specifications (e.g. the scenarios that are part of .feature files).

The overall flow would be something like this:


1. create Scenario/Scenario Outline as a Test in Jira; usually, it would be linked to an existing "requirement"/Story (i.e. created from the respective issue screen)
2. implement the code related to Gherkin statements/steps and store it in Git, for example
3. generate .feature files based on the specification made in Jira
4. checkout the code from Git
5. run the tests in the CI
6. import the results back to Jira


Usually, you would start by having a Story, or similar (e.g. "requirement"), to describe the behavior of a certain feature and use that to drive your testing.


If you have it, then you can just use the "Create Test" on that issue to create the Scenario/Scenario Outline and have it automatically linked back to the Story/"requirement".


Otherwise, you can create the Test using the standard (issue) Create action from Jira's top menu.


## As a user, I can login the application


 Attach

 Create subtask

 Link issue



 Test Coverage



### Description

As a user, I can login the application

### Test Coverage

...


Create new Sub Test Execution


Create new Test


No Tests are associated with this issue.


UNCOVERED


## As a user, I can logout the application


 Attach

 Create subtask

 Link issue



 Test Coverage



### Description

As a user, I can logout the application

### Test Coverage

...

Create new Sub Test Execution

Create new Test

No Tests are associated with this issue.

UNCOVERED

In this case, we'll create a Cucumber Scenario.

We need to create the Test issue first and fill out the Gherkin statements later on in the Test issue screen.

Create issue

Import issuesConfigure fields

Project

Calculator (CALC)

Issue Type

Test

Some issue types are unavailable due to incompatible field configuration and/or workflow associations.

Summary

Valid Login

Components

None

Attachment

Drop files to attach, or [browse](#).

Description

Style B I U A A Link Image Bulleted List Numbered List Mention + More

Tests As a user, I can logout the application

Linked Issues

tests

Issue

Create another

Create

Cancel

Projects / Calculator / CALC-634

## Description

Tests As a user, I can logout the application

## Linked issues



tests

CALC-633

As a user, I can logout the application

↑

TO DO

## Test Details



Manual

Generic

Cucumber

Exploratory



There are no steps defined.

Create Step

Open Dialog

Import

Test Repository

## Description

Tests As a user, I can logout the application

## Linked issues



tests

 CALC-633 As a user, I can logout the application



TO DO

## Test Details



Cucumber



Test Repository

Scenario

- 1 Given browser is opened to login page
- 2 When user "demo" logs in with password "mode"
- 3 Then welcome page should be open



After the Test is created it will impact the coverage of related "requirement", if any.

The coverage and the test results can be tracked in the "requirement" side (e.g. user story). In this case, you may see that coverage changed from being UNCOVERED to NOTRUN (i.e. covered and with at least one test not run).

As a user, I can login the application

Attach

Create subtask

Link issue

Test Coverage

Description

As a user, I can login the application

Linked issues

is tested by

CALC-634 Valid Login

↑ TO DO

Test Coverage

Calculate the Test Coverage for the following scopes.

Latest

Version

Test Plan

Test Environment

All Environments

▼

→

NOTRUN

Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
↑ .. TO DO	CALC-634	Valid Login	→ TO DO
Prev	1	Next	

Additional tests could be created, eventually linked to the same Story or linked to another one (e.g. logout).

The related statement's code is managed outside of Jira and stored in Git, for example.

In Cypress, tests related code is mainly stored under `cypress/integration` directory, which itself contains several other directories. In this case, we've organized the assets as follows:

- `cypress/support/step_definitions`: step implementation files, in JavaScript.

cypress/support/step\_definitions/login.js

```
import { Given, When, Then } from "@badeball/cypress-cucumber-preprocessor";
import LoginPage from '../../pages/login-page';
import LoginResultsPage from '../../pages/login-results-page';

Given(/^browser is opened to login page$/, () => {
  LoginPage.visit();
});

When('user {string} logs in with password {string}', (username, password) => {
  LoginPage.enter_username(username);
  LoginPage.enter_password(password);
  LoginPage.pressLogin();
});

Then(/^welcome page should be open$/, () => {
  LoginResultsPage.expect().toBeSuccessful();
});

Then(/^error page should be open$/, () => {
  LoginResultsPage.expect().toBeUnsuccessful();
});
```



- **cypress/support/step\_definitions/logout.js**

```
import { Given, When, Then } from "@badeball/cypress-cucumber-preprocessor";

import WelcomePage from '../../pages/welcome-page';
import LogoutResultsPage from '../../pages/logout-results-page';

Given(/^user is on the welcome page$/, () => {
  WelcomePage.visit();
});

When('user chooses to logout', () => {
  WelcomePage.pressLogout();
});

Then(/^login page should be open$/, () => {
  LogoutResultsPage.expect().toBeSuccessful();
});
```

- cypress/integration/pages: abstraction of different pages, somehow based on the page-objects model

- **cypress/integration/pages/login.js**

```
import LoginResultsPage from './login-results-page';

const USERNAME_FIELD = 'input[id=username_field]';
const PASSWORD_FIELD = 'input[id=password_field]';
const LOGIN_BUTTON = 'input[type=submit]';
const LOGIN_TEXT = 'LOGIN';

class LoginPage {
  static visit() {
    cy.visit('/');
  }

  static enter_username(username) {
    cy.get(USERNAME_FIELD)
      .type(username);
  }

  static enter_password(password) {
    cy.get(PASSWORD_FIELD)
      .type(password);
  }

  static pressLogin() {
    cy.get(LOGIN_BUTTON).contains(LOGIN_TEXT)
      .click();
    return new LoginResultsPage();
  }
}

export default LoginPage;
```

◦ **cypress/integration/pages/login-results-page.js**

```
const RESULT_HEADER = 'h1';

class LoginResultsPage {
  static expect() {
    return {
      toBeSuccessful: () => {
        cy.get(RESULT_HEADER).should('have.text', 'Welcome Page')
      },

      toBeUnsuccessful: () => {
        cy.get(RESULT_HEADER).should('have.text', 'Error Page')
      },
    };
  }
}

export default LoginResultsPage;
```

◦ **cypress/integration/pages/logout-results-page.js**

```
const RESULT_HEADER = 'h1';

class LogoutResultsPage {
  static expect() {
    return {
      toBeSuccessful: () => {
        cy.get(RESULT_HEADER).should('have.text', 'Login Page')
      },
    };
  }
}

export default LogoutResultsPage;
```

◦ **cypress/integration/pages/welcome-page.js**

```
import LoginPage from './login-page';

const LOGOUT_LINK = 'a';
const LOGOUT_TEXT = 'logout';

class WelcomePage {
  static visit() {
    cy.visit('/welcome.html');
  }

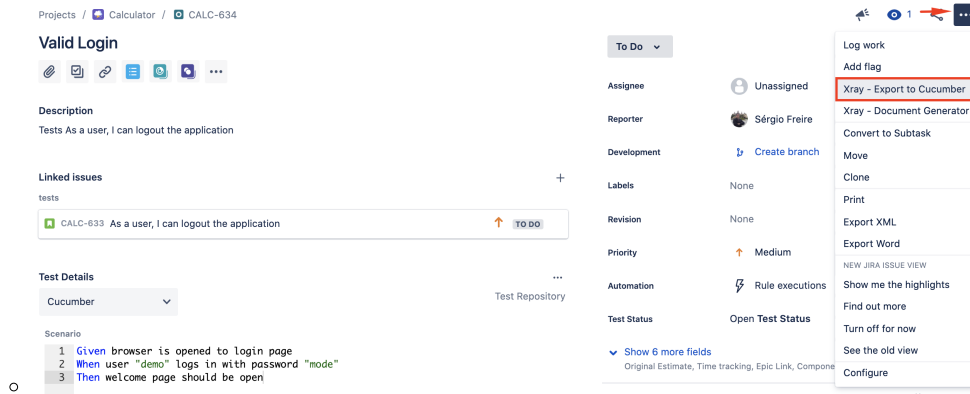
  static pressLogout() {
    cy.get(LOGOUT_LINK).contains(LOGOUT_TEXT)
      .click();
    return new LoginPage();
  }
}

export default WelcomePage;
```

You can then export the specification of the test to a Cucumber .feature file via the REST API, or the **Export to Cucumber** UI action from within the Test /Test Execution issue or even based on an existing saved filter. A plugin for your CI tool of choice can be used to ease this task.

So, you can either:

- use the UI



- use the REST API (more info [here](#))

- **example of a shell script to export/generate .features from Xray**

```
#!/bin/bash

token=$(curl -H "Content-Type: application/json" -X POST --data @"cloud_auth.json" https://xray.
cloud.getxray.app/api/v2/authenticate| tr -d '"')
curl -H "Content-Type: application/json" -X GET -H "Authorization: Bearer $token" "https://xray.
cloud.getxray.app/api/v2/export/cucumber?keys=CALC-632;CALC-633" -o features.zip

rm -rf features/*.feature
unzip -o features.zip -d features
```

- use one of the available CI/CD plugins (e.g. see an example of [Integration with Jenkins](#))

We will export the features to a new directory named `features/` on the root folder of your Cypress project (we'll need to tell Cypress to use this folder).

After being exported, the created .feature(s) will contain references to the Test issue key, eventually prefixed (e.g. "TEST\_") depending on an [Xray global setting](#), and the covered "requirement" issue key, if that's the case. The naming of these files is detailed in [Generate Cucumber Features](#).

## features/2\_CALC-632.feature

```
@REQ_CALC-632
Feature: As a user, I can login the application
  #As a user, I can login the application

  #Tests As a user, I can logout the application
  @TEST_CALC-634
  Scenario: Valid Login
    Given browser is opened to login page
    When user "demo" logs in with password "mode"
    Then welcome page should be open
  #Tests As a user, I can logout the application
  @TEST_CALC-635
  Scenario: Invalid Login
    Given browser is opened to login page
    When user "dummy" logs in with password "password"
    Then error page should be open
  @TEST_CALC-636
  Scenario Outline: Login With Invalid Credentials Should Fail
    Given browser is opened to login page
    When user "<username>" logs in with password "<password>"
    Then error page should be open

    Examples:
      | username | password |
      | invalid  | mode     |
      | demo     | invalid  |
      | invalid  | invalid  |
      | demo     | mode     |
```

## features/1\_CALC-633.feature

```
@REQ_CALC-633
Feature: As a user, I can logout the application
  #As a user, I can logout the application

  @TEST_CALC-637
  Scenario: Valid Logout
    Given user is on the welcome page
    When user chooses to logout
    Then login page should be open
```

To run the tests and produce Cucumber JSON reports(s), we can either use `npm` or `cypress` command directly.

```
npm run test

# or instead...

node_modules/cypress/bin/cypress run --spec 'features/**/*.feature'
```

This will produce one Cucumber JSON report.

After running the tests, results can be imported to Xray via the REST API, or the **Import Execution Results** action within the Test Execution, or by using one of the available CI/CD plugins (e.g. see an example of [Integration with Jenkins](#)).

### import\_results\_cloud.sh

```
#!/bin/bash

BASE_URL=https://xray.cloud.getxray.app
token=$(curl -H "Content-Type: application/json" -X POST --data @"cloud_auth.json" "$BASE_URL/api/v2/authenticate" | tr -d '\n')
curl -H "Content-Type: application/json" -X POST -H "Authorization: Bearer $token" --data @"cucumber-report.json" "$BASE_URL/api/v2/import/execution/cucumber"
```



#### Which Cucumber endpoint/"format" to use?

To import results, you can use two different endpoints/"formats" (endpoints described in [Import Execution Results - REST](#)):

1. the "standard cucumber" endpoint
2. the "multipart cucumber" endpoint

The standard cucumber endpoint (i.e. `/import/execution/cucumber`) is simpler but more restrictive: you cannot specify values for custom fields on the Test Execution that will be created. This endpoint creates new Test Execution issues unless the Feature contains a tag having an issue key of an existing Test Execution.

The multipart cucumber endpoint will allow you to customize fields (e.g. Fix Version, Test Plan), if you wish to do so, on the Test Execution that will be created. Note that this endpoint always creates new Test Executions (as of Xray v4.2).

In sum, if you want to customize the Fix Version, Test Plan and/or Test Environment of the Test Execution issue that will be created, you'll have to use the "multipart cucumber" endpoint.

A new Test Execution will be created (unless you originally exported the Scenarios/Scenario Outlines from a Test Execution).

Execution results [1604501240267]

Attach

Create subtask

Link issue

Tests

Description

Add a description...

Tests

Create Test

+ Add

Overall Execution Status

TOTAL TESTS: 4

3 PASSED 1 FAILED

Filters

100

Columns

Rank	Key	Summary	Test Type	Status	Actions
<input type="checkbox"/> 1	CALC-634	Valid Login	Cucumber	PASSED	<div></div> ...
<input type="checkbox"/> 2	CALC-635	Invalid Login	Cucumber	PASSED	<div></div> ...
<input type="checkbox"/> 3	CALC-636	Login With Invalid Credentials Should Fail	Cucumber	FAILED	<div></div> ...
<input type="checkbox"/> 4	CALC-637	Valid Logout	Cucumber	PASSED	<div></div> ...

Prev

1

Next

Total 4 issues

One of the tests fails (on purpose).

The execution screen details of the Test Run will provide overall status information and Gherkin statement-level results, therefore we can use it to analyze the failing test.

## Execution results [1604501240267]

[Attach](#)
[Create subtask](#)
[Link issue](#)
[Tests](#)

### Description

Add a description...

### Tests

[Create Test](#)
[Add](#)

### Overall Execution Status

TOTAL TESTS: 4

3 PASSED
1 FAILED

Rank	Key	Summary	Test Type	Status	Actions
<input type="checkbox"/> 1	CALC-634	Valid Login	Cucumber	PASSED	...
<input type="checkbox"/> 2	CALC-635	Invalid Login	Cucumber	PASSED	...
<input type="checkbox"/> 3	CALC-636	Login With Invalid Credentials Should Fail	Cucumber	FAILED	...
<input type="checkbox"/> 4	CALC-637	Valid Logout	Cucumber	PASSED	...

Prev 1 Next Total 4 issues

Jira Software
Your work
Projects
Filters
Dashboards
People
Apps
Create

Search

### Execution Details

Test Description
None

Test Issue Links (1)
tests
CALC-632 As a user, I can login the application

Custom Fields
There are no Test Run Custom Fields defined.

Test Details
Test Type: Cucumber
Scenario Type: Scenario Outline
Scenario:

```

1 Given browser is opened to login page
2 When user "<username>" logs in with password "<password>"
3 Then error page should be open
4
5 Examples:
6 | username | password |
7 | invalid | mode |
8 | demo | invalid |
9 | invalid | invalid |
10 | demo | mode |

```

Examples

<username>	<password>	Duration	Status
invalid	mode	1s 788ms	PASSED
demo	invalid	3s 677ms	PASSED
invalid	invalid	11s 119ms	PASSED
demo	mode	8s 866ms	FAILED

A given example can be expanded to see all Gherkin statements and, if available, it is possible to see also the attached screenshot(s).

Jira Software Your work Projects Filters Dashboards People Apps Create

Custom Fields

There are no Test Run Custom Fields defined.

Test Details

Test Type: Cucumber  
Scenario Type: Scenario Outline  
Scenario:

```
1 Given browser is opened to login page
2 When user "<username>" logs in with password "<password>"
3 Then error page should be open
4
5 Examples:
6 | username | password |
7 | invalid  | mode     |
8 | demo     | invalid  |
9 | invalid  | invalid  |
10 | demo     | mode     |
```

Examples

<username>	<password>	Duration	Status
invalid	mode	1s 788ms	PASSED
demo	invalid	3s 617ms	PASSED
invalid	invalid	11s 119ms	PASSED
demo	mode	8s 866ms	FAILED

Steps

Given browser is opened to login page 3 secs PASSED

When user "demo" logs in with password "mode" secs PASSED

Then error page should be open (1) evidence-0.png secs FAILED

AssertionError: Timed out retrying: expected '<html>' to have text 'Error Page', but the text was 'Welcome Page'

+ expected - actual

- 'Welcome Page'

+ 'Error Page'

at Object.toBeUnsuccessful (https://robotwebdemo.herokuapp.com/\_cypress/tests?p=features/2\_CALC-632.feature:134:33)  
at Context.eval (https://robotwebdemo.herokuapp.com/\_cypress/tests?p=features/2\_CALC-632.feature:26:41)  
at Context.resolveAndRunStepDefinition (https://robotwebdemo.herokuapp.com/\_cypress/tests?p=features/2\_CALC-632.feature:10674:9)  
at Context.eval (https://robotwebdemo.herokuapp.com/\_cypress/tests?p=features/2\_CALC-632.feature:10015:35)

Note: in this case, the bug was on the Scenario Outline example which was using a valid username/password combination.

Results are reflected on the covered item (e.g. Story). On its issue screen, coverage now shows that the item is OK based on the latest testing results, that can also be tracked within the Test Coverage panel below.

Projects / Calculator / CALC-632

### As a user, I can login the application

Attach Create subtask Link issue Test Coverage

Description

As a user, I can login the application

Linked issues

is tested by

CALC-634 Valid Login	↑ TO DO
CALC-635 Invalid Login	↑ TO DO
CALC-636 Login With Invalid Credentials Should Fail	↑ TO DO

Test Coverage

Calculate the Test Coverage for the following scopes.

Latest Version Test Plan

Test Environment

All Environments

NOK

Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
↑ TO DO	CALC-634	Valid Login	PASSED
↑ TO DO	CALC-635	Invalid Login	PASSED
↑ TO DO	CALC-636	Login With Invalid Credentials Should Fail	FAILED

Using Git or other VCS as master



You can edit your .feature files using your IDE outside of Jira (eventually storing them in your VCS using Git, for example) alongside with remaining test code.

In any case, you'll need to synchronize your .feature files to Jira so that you can have visibility of them and report results against them.

The overall flow would be something like this:

1. look at the existing "requirement"/Story issue keys to guide your testing; keep their issue keys
2. specify Cucumber/Gherkin .feature files in your IDE supporting Cypress and store it in Git, for example
3. implement the code related to Gherkin statements/steps and store it in Git, for example
4. import/synchronize the .feature files to Xray to provision or update corresponding Test entities
5. export/generate .feature files from Jira, so that they contain references to Tests and requirements in Jira
6. checkout the Cypress related code from Git
7. run the tests in the CI
8. import the results back to Jira

Usually, you would start by having a Story, or similar (e.g. "requirement"), to describe the behavior of a certain feature and use that to drive your testing.

Projects / Calculator / CALC-632

## As a user, I can login the application

 Attach  Create subtask  Link issue   Test Coverage 

### Description

As a user, I can login the application

### Test Coverage

...

Create new Sub Test Execution

Create new Test

No Tests are associated with this issue.

UNCOVERED

Projects / Calculator / CALC-633

## As a user, I can logout the application

 Attach  Create subtask  Link issue   Test Coverage 

### Description

As a user, I can logout the application

### Test Coverage

...

Create new Sub Test Execution

Create new Test

No Tests are associated with this issue.

UNCOVERED

Having those to guide testing, we could then move to Cypress to describe and implement the Cucumber test scenarios.

In Cypress, tests related code is mainly stored inside the `cypress/integration` directory, which itself contains several other directories. In this case, we've organized the assets as follows:

- `cypress/support/step_definitions`: step implementation files, in JavaScript.

- **cypress/support/step\_definitions/login.js**

```
import { Given, When, Then } from "@badeball/cypress-cucumber-preprocessor";
import LoginPage from '../../../pages/login-page';
import LoginResultsPage from '../../../pages/login-results-page';

Given(/^browser is opened to login page$/, () => {
  LoginPage.visit();
});

When('user {string} logs in with password {string}', (username, password) => {
  LoginPage.enter_username(username);
  LoginPage.enter_password(password);
  LoginPage.pressLogin();
});

Then(/^welcome page should be open$/, () => {
  LoginResultsPage.expect().toBeSuccessful();
});

Then(/^error page should be open$/, () => {
  LoginResultsPage.expect().toBeUnsuccessful();
});
```

- **cypress/support/step\_definitions/logout.js**

```
import { Given, When, Then } from "@badeball/cypress-cucumber-preprocessor";

import WelcomePage from '../../../pages/welcome-page';
import LogoutResultsPage from '../../../pages/logout-results-page';

Given(/^user is on the welcome page$/, () => {
  WelcomePage.visit();
});

When('user chooses to logout', () => {
  WelcomePage.pressLogout();
});

Then(/^login page should be open$/, () => {
  LogoutResultsPage.expect().toBeSuccessful();
});
```

- cypress/integration/pages: abstraction of different pages, somehow based on the page-objects model

◦ **cypress/integration/pages/login.js**

```
import LoginResultsPage from './login-results-page';

const USERNAME_FIELD = 'input[id=username_field]';
const PASSWORD_FIELD = 'input[id=password_field]';
const LOGIN_BUTTON = 'input[type=submit]';
const LOGIN_TEXT = 'LOGIN';

class LoginPage {
  static visit() {
    cy.visit('/');
  }

  static enter_username(username) {
    cy.get(USERNAME_FIELD)
      .type(username);
  }

  static enter_password(password) {
    cy.get(PASSWORD_FIELD)
      .type(password);
  }

  static pressLogin() {
    cy.get(LOGIN_BUTTON).contains(LOGIN_TEXT)
      .click();
    return new LoginResultsPage();
  }
}

export default LoginPage;
```

◦ **cypress/integration/pages/login-results-page.js**

```
const RESULT_HEADER = 'h1';

class LoginResultsPage {
  static expect() {
    return {
      toBeSuccessful: () => {
        cy.get(RESULT_HEADER).should('have.text', 'Welcome Page')
      },

      toBeUnsuccessful: () => {
        cy.get(RESULT_HEADER).should('have.text', 'Error Page')
      },
    };
  }
}

export default LoginResultsPage;
```

◦ **cypress/integration/pages/logout-results-page.js**

```
const RESULT_HEADER = 'h1';

class LogoutResultsPage {
  static expect() {
    return {
      toBeSuccessful: () => {
        cy.get(RESULT_HEADER).should('have.text', 'Login Page')
      },
    };
  }
}

export default LogoutResultsPage;
```

◦ **cypress/integration/pages/welcome-page.js**

```
import LoginPage from './login-page';

const LOGOUT_LINK = 'a';
const LOGOUT_TEXT = 'logout';

class WelcomePage {
  static visit() {
    cy.visit('/welcome.html');
  }

  static pressLogout() {
    cy.get(LOGOUT_LINK).contains(LOGOUT_TEXT)
      .click();
    return new LoginPage();
  }
}

export default WelcomePage;
```

- cypress/integration/login: Cucumber .feature files, containing the tests as Gherkin Scenario(s)/Scenario Outline(s). Please note that each "Feature: <..>" section should be tagged with the issue key of the corresponding "requirement"/story in Jira. You may need to add a prefix (e. g. "REQ\_") before the issue key, depending on an [Xray global setting](#).

#### ○ cypress/integration/login/login.feature

@REQ\_CALC-6232

Feature: As a user, I can login the applicaiton

Scenario: Valid Login

Given browser is opened to login page  
When user "demo" logs in with password "mode"  
Then welcome page should be open

Scenario: Invalid Login

Given browser is opened to login page  
When user "dummy" logs in with password "password"  
Then error page should be open

Scenario Outline: Login With Invalid Credentials Should Fail

Given browser is opened to login page  
When user "<username>" logs in with password "<password>"  
Then error page should be open

Examples:

username	password
invalid	mode
demo	invalid
invalid	invalid
demo	mode

#### ○ cypress/integration/login/logout.feature

@REQ\_CALC-633

Feature: As a user, I can logout the application

Scenario: Valid Logout

Given user is on the welcome page  
When user chooses to logout  
Then login page should be open

Before running the tests in the CI environment, you need to import your .feature files to Xray/Jira; you can invoke the REST API directly or use one of the available plugins/tutorials for CI tools.

#### example of a shell script to import/synchronize Scenario(s)/Background(s) from .features to Jira and Xray

```
#!/bin/bash

BASE_URL=https://xray.cloud.getxray.app
PROJECT=CALC

zip -r features.zip cypress/integration/ -i \*.feature

token=$(curl -H "Content-Type: application/json" -X POST --data @"cloud_auth.json" "$BASE_URL/api/v2/authenticate" | tr -d '\n')
curl -H "Content-Type: multipart/form-data" -H "Authorization: Bearer $token" -F "file=@features.zip" "$BASE_URL/api/v2/import/feature?projectKey=$PROJECT"
```



#### Please note

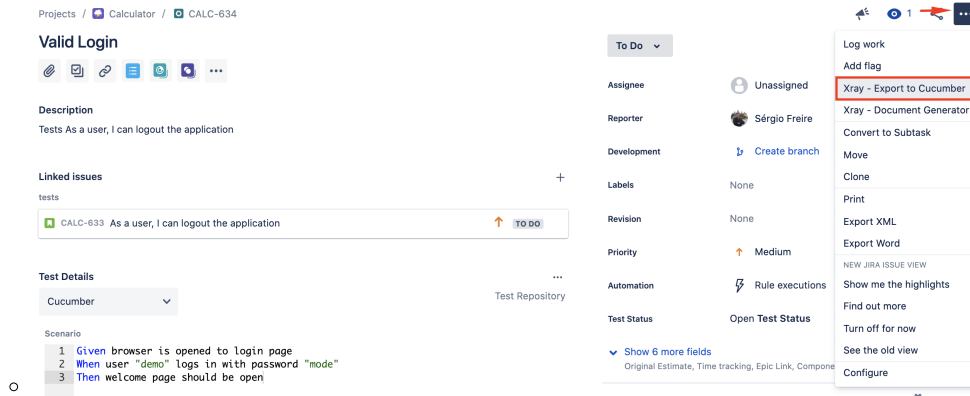
Each Scenario of each .feature will be created as a Test issue that contains unique identifiers, so that if you import once again then Xray can update the existent Test and don't create any duplicated tests.

Afterward, you can export those features out of Jira based on some criteria, so they are properly tagged with corresponding issue keys; this is important because results need to contain these references.

You can then export the specification of the test to a Cucumber .feature file via the REST API, or the **Export to Cucumber** UI action from within the Test /Test Execution issue or even based on an existing saved filter. A plugin for your CI tool of choice can be used to ease this task.

So, you can either:

- use the UI



- use the REST API (more info [here](#))

#### example of a shell script to export/generate .features from Xray

```
#!/bin/bash

BASE_URL=https://xray.cloud.getxray.app
token=$(curl -H "Content-Type: application/json" -X POST --data @"cloud_auth.json" $BASE_URL/api/v2/authenticate | tr -d ' ')
curl -H "Content-Type: application/json" -X GET -H "Authorization: Bearer $token" "$BASE_URL/api/v2/export/cucumber?keys=CALC-632;CALC-633" -o features.zip

rm -rf features/*.feature
unzip -o features.zip -d features
```

- use one of the available CI/CD plugins (e.g. see an example of [Integration with Jenkins](#))

For CI only purpose, we will export the features to a new temporary directory named `features/` on the root folder of your Cypress project (we'll need to tell Cypress to use this folder). Please note that while implementing the tests, .feature files should be edited inside the `cypress/integration/login` folder, in this case;

After being exported, the created .feature(s) will contain references to the Test issue keys, eventually prefixed (e.g. "TEST\_") depending on an [Xray global setting](#), and the covered "requirement" issue key, if that's the case. The naming of these files is detailed in [Generate Cucumber Features](#).

## features/2\_CALC-632.feature

```
@REQ_CALC-632
Feature: As a user, I can login the application
  #As a user, I can login the application

  #Tests As a user, I can logout the application
  @TEST_CALC-634
  Scenario: Valid Login
    Given browser is opened to login page
    When user "demo" logs in with password "mode"
    Then welcome page should be open

  #Tests As a user, I can logout the application
  @TEST_CALC-635
  Scenario: Invalid Login
    Given browser is opened to login page
    When user "dummy" logs in with password "password"
    Then error page should be open

  @TEST_CALC-636
  Scenario Outline: Login With Invalid Credentials Should Fail
    Given browser is opened to login page
    When user "<username>" logs in with password "<password>"
    Then error page should be open

    Examples:
      | username | password |
      | invalid  | mode     |
      | demo     | invalid  |
      | invalid  | invalid  |
      | demo     | mode     |
```

To run the tests and produce Cucumber JSON reports(s), we can either use `npm` or `cypress` command directly.

```
npm run test

# or instead...

node_modules/cypress/bin/cypress run --spec 'features/**/*.feature'
```

This will produce one Cucumber JSON report.

After running the tests, results can be imported to Xray via the REST API, or the **Import Execution Results** action within the Test Execution, or by using one of the available CI/CD plugins (e.g. see an example of [Integration with Jenkins](#)).

### Example of shell script to import results (e.g. `import_results_cloud.sh`)

```
#!/bin/bash

BASE_URL=https://xray.cloud.getxray.app
token=$(curl -H "Content-Type: application/json" -X POST --data @"cloud_auth.json" "$BASE_URL/api/v2/authenticate" | tr -d '\n')
curl -H "Content-Type: application/json" -X POST -H "Authorization: Bearer $token" --data @"cucumber-report.json" "$BASE_URL/api/v2/import/execution/cucumber"
```



### Which Cucumber endpoint/"format" to use?

To import results, you can use two different endpoints/"formats" (endpoints described in [Import Execution Results - REST](#)):

1. the "standard cucumber" endpoint
2. the "multipart cucumber" endpoint

The standard cucumber endpoint (i.e. `/import/execution/cucumber`) is simpler but more restrictive: you cannot specify values for custom fields on the Test Execution that will be created. This endpoint creates new Test Execution issues unless the Feature contains a tag having an issue key of an existing Test Execution.

The multipart cucumber endpoint will allow you to customize fields (e.g. Fix Version, Test Plan), if you wish to do so, on the Test Execution that will be created. Note that this endpoint always creates new Test Executions (as of Xray v4.2).

In sum, if you want to customize the Fix Version, Test Plan and/or Test Environment of the Test Execution issue that will be created, you'll have to use the "multipart cucumber" endpoint.

A new Test Execution will be created (unless you originally exported the Scenarios/Scenario Outlines from a Test Execution).

Projects / Calculator / CALC-639

## Execution results [1604502815636]

Attach Create subtask Link issue Tests ...

### Description

Add a description...

### Tests

Create Test + Add

### Overall Execution Status

TOTAL TESTS: 4

3 PASSED 1 FAILED

Rank	Key	Summary	Test Type	Status	Actions
<input type="checkbox"/> 1	<a href="#">CALC-634</a>	Valid Login	Cucumber	<span>PASSED</span>	...
<input type="checkbox"/> 2	<a href="#">CALC-635</a>	Invalid Login	Cucumber	<span>PASSED</span>	...
<input type="checkbox"/> 3	<a href="#">CALC-636</a>	Login With Invalid Credentials Should Fail	Cucumber	<span>FAILED</span>	...
<input type="checkbox"/> 4	<a href="#">CALC-637</a>	Valid Logout	Cucumber	<span>PASSED</span>	...

Prev 1 Next Total 4 issues

One of the tests fails (on purpose).

The execution screen details of the Test Run will provide overall status information and Gherkin statement-level results, therefore we can use it to analyze the failing test.



## Execution results [1604502815636]

[Attach](#) [Create subtask](#) [Link issue](#) [Tests](#) [...](#)

## Description

Add a description...

## Tests

[Create Test](#)[+ Add](#)

## Overall Execution Status

TOTAL TESTS: 4

3 PASSED 1 FAILED

Rank	Key	Summary	Test Type	Status	Actions
1	CALC-634	Valid Login	Cucumber	PASSED	<a href="#">View</a> <a href="#">...</a>
2	CALC-635	Invalid Login	Cucumber	PASSED	<a href="#">View</a> <a href="#">...</a>
3	CALC-636	Login With Invalid Credentials Should Fail	Cucumber	FAILED	<a href="#">View</a> <a href="#">...</a>
4	CALC-637	Valid Logout	Cucumber	PASSED	<a href="#">View</a> <a href="#">...</a>

Prev 1 Next

Total 4 issues

## Execution Details

## Test Description

None

## Test Issue Links (1)

tests [CALC-632](#) As a user, I can login the application [↑](#) [TO DO](#)

## Custom Fields

There are no Test Run Custom Fields defined.

## Test Details

Test Type:	Cucumber
Scenario Type:	Scenario Outline
Scenario:	<pre>1 Given browser is opened to login page 2 When user "&lt;username&gt;" logs in with password "&lt;password&gt;" 3 Then error page should be open 4 5 Examples: 6   username   password   7   invalid    mode       8   demo       invalid    9   invalid    invalid    10   demo       mode      </pre>

## Examples

<username>	<password>	Duration	Status
invalid	mode	1s 788ms	PASSED
demo	invalid	3s 617ms	PASSED
invalid	invalid	11s 119ms	PASSED
demo	mode	8s 866ms	FAILED

A given example can be expanded to see all Gherkin statements and, if available, it is possible to see also the attached screenshot(s).

Jira Software Your work Projects Filters Dashboards People Apps Create

Custom Fields

There are no Test Run Custom Fields defined.

Test Details

Test Type: Cucumber  
Scenario Type: Scenario Outline  
Scenario:

```
1 Given browser is opened to login page
2 When user "<username>" logs in with password "<password>"
3 Then error page should be open
4
5 Examples:
6 | username | password |
7 | invalid  | mode     |
8 | demo     | invalid  |
9 | invalid  | invalid  |
10 | demo     | mode     |
```

Examples

<username>	<password>	Duration	Status
invalid	mode	1s 788ms	PASSED
demo	invalid	3s 617ms	PASSED
invalid	invalid	11s 119ms	PASSED
demo	mode	8s 866ms	FAILED

Steps

Given browser is opened to login page 3 secs PASSED

When user "demo" logs in with password "mode" secs PASSED

Then error page should be open (1) evidence-0.png secs FAILED

AssertionError: Timed out retrying: expected '<h1>' to have text 'Error Page', but the text was 'Welcome Page'

+ expected - actual

- 'Welcome Page'

+ 'Error Page'

at Object.toBeUnsuccessful (https://robotwebdemo.herokuapp.com/\_cypress/tests?p=features/2\_CALC-632.feature:134:33)  
at Context.eval (https://robotwebdemo.herokuapp.com/\_cypress/tests?p=features/2\_CALC-632.feature:26:41)  
at Context.resolveAndRunStepDefinition (https://robotwebdemo.herokuapp.com/\_cypress/tests?p=features/2\_CALC-632.feature:10674:9)  
at Context.eval (https://robotwebdemo.herokuapp.com/\_cypress/tests?p=features/2\_CALC-632.feature:10015:35)

Note: in this case, the bug was on the Scenario Outline example which was using a valid username/password combination.

Results are reflected on the covered item (e.g. Story). On its issue screen, coverage now shows that the item is OK based on the latest testing results, that can also be tracked within the Test Coverage panel below.

Projects / Calculator / CALC-632

### As a user, I can login the application

Attach Create subtask Link issue Test Coverage

Description

As a user, I can login the application

Linked issues

is tested by

CALC-634 Valid Login	↑ TO DO
CALC-635 Invalid Login	↑ TO DO
CALC-636 Login With Invalid Credentials Should Fail	↑ TO DO

Test Coverage

Calculate the Test Coverage for the following scopes.

Create new Sub Test Execution Create new Test

Latest Version Test Plan

Test Environment

All Environments

NOK

Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
↑ TO DO	CALC-634	Valid Login	PASSED
↑ TO DO	CALC-635	Invalid Login	PASSED
↑ TO DO	CALC-636	Login With Invalid Credentials Should Fail	FAILED

If we change the specification (i.e. the Gherkin scenarios), we need to import the .feature(s) once again.

Therefore, in the CI we always need to start by importing the .feature file(s) to keep Jira/Xray on synch.

## FAQ and Recommendations

Please see [this page](#).

## References

- [Cypress](#)
- [Cypress documentation](#)
- [@badeball/cypress-cucumber-preprocessor](#)