[**1. Propose**](#_4u6649gkrdn9) **2**

[**2. Automation Test Requirement**](#_dumehz27afo8) **3**

[**3. Automation Test Approach**](#_egjvi6bqv68j) **3**

[3.1. Scope of automation](#_y28bcn8hq9vl) 3

[3.2. Automation tools](#_j7fmuvw1usmo) 3

[3.2.1. Using Visual Code](#_6hokn2ao88un) 3

[3.2.2. Node.js and npm](#_4xwykdtn0wsp) 4

[3.3. Automation process](#_d9syrc615g3x) 4

[**4. Checkout source code from GitHub**](#_nr55bvl5ft0q) **4**

[**5. Automation Test Framework**](#_1y8x098n5u2r) **5**

[5.1. Overview Framework](#_ydhgd2ags9tw) 5

[5.2. Code Review Checklist](#_30mnp43i7ncw) 5

[5.2.1. Naming convention](#_g3w5ycc6bdg) 5

[5.2.2. Coding best practices](#_czojxlv65qc6) 6

[5.3. Coding Standard](#_n8026718cvrq) 6

[5.4. Code structure](#_920m8ohjco98) 8

[5.4.1. Feature files](#_lxp7q6lrw39g) 8

[5.4.2. Page-Object](#_yb0lekud5met) 9

[5.4.2.1. Keywords](#_58sl5a9851vm) 9

[5.4.2.2. Common](#_vqf6d69wse52) 9

[5.4.2.3. Specific Page](#_lwmor4ozmwt) 10

[5.4.3. Step-definitions](#_5jaavtdxedtw) 11

[5.4.3.1. Common Steps](#_ydja57humvsn) 11

[5.4.3.2. Specific Steps (page)](#_s4qihsp04wm7) 12

[5.4.4. Support](#_akfdm39h41m1) 13

[5.4.4.1. Hook](#_l6r801ljqrcr) 13

[5.4.4.2. World](#_te7gg15urcny) 15

[5.4.5. Report](#_cr2q2u3hbg3w) 15

[5.4.6. Execution](#_sjgiv0uyzne4) 17

[5.4.7. CI/CD](#_wzr62jsycrjs) 19

[5.4.8. Dependencies](#_d5ro6mxd8c3b) 21

[**6. Reference**](#_og473k7lblbx) **23**

[6.1. async/await](#_nzwmpnf7u3wr) 23

[6.2. Constructor](#_mv2z78xc3m2t) 24

[6.3. setWorldConstructor](#_fq6h9wgbmmo8) 24

[6.4. promisify](#_irmyc28vey5s) 25

[6.5. World term](#_ep1rd1s7fgf8) 25

[6.6. fillTemplate](#_mvyq2i9wpf0x) 25

# Propose

This procedure is to help members gains the knowledge on the automation UI project with the following stacks:

* Coding style by Javascript
* Define BDD by Cucumber
* Automation framework by Selenium
* Multiple report
* Execution by batch
* Setup CI/CD by GitHub Action

# Automation Test Requirement

# Automation Test Approach

## Scope of automation

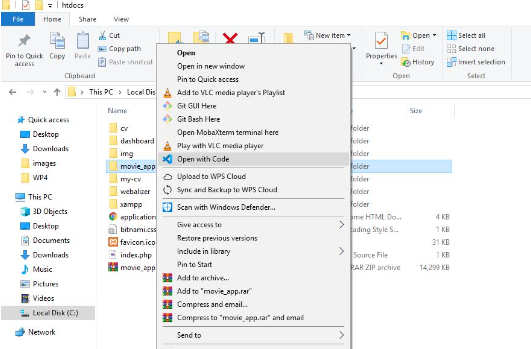
## Automation tools

### Using Visual Code

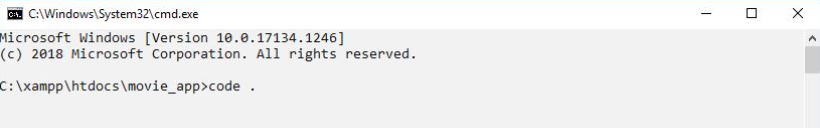
Download the Visual Studio Code from <https://code.visualstudio.com/download> and install

To open a folder in Visual Studio Code there are two ways possible.

1. Right Click on the Folder inside the folder area and Click on open with code.



1. The Second way to open with code can be using a terminal. Go to your folder path using cmd and just hit “code .” .Your folder will be open in Visual Studio Code



### Node.js and npm

Download Node.js from <https://nodejs.org/en/download/> and install

Make sure you have Node and NPM installed by running simple commands to see what version of each is installed and to run a simple test program:

**Test Node.** To see if Node is installed, open the Windows Command Prompt, Powershell or a similar command line tool, and type node -v. This should print a version number, so you’ll see something like this v0.10.35.

**Test NPM.** To see if NPM is installed, type npm -v in Terminal. This should print NPM’s version number so you’ll see something like this 1.4.28

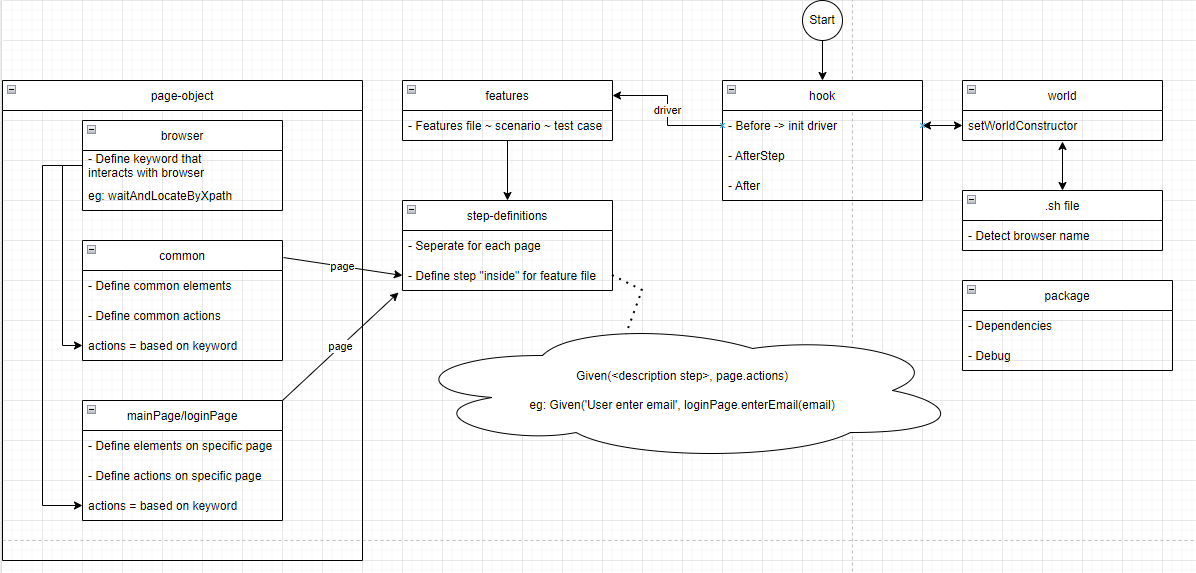
## Automation process

# Checkout source code from GitHub

Source code from GitHub repo: <https://github.com/kms-technology/lni-auto-ui-selenium>

# Automation Test Framework

## Overview Framework



## Code Review Checklist

### Naming convention

* Use self-explanatory names for everything (interfaces, classes, methods, variables etc.).
* Use Pascal casing for all public member, type, and namespace names consisting of multiple words. Use Camel casing for parameter names.
* Name classes and structs with nouns or noun phrases. Name methods with verbs or verb phrases.
* Prefix interface names with the letter I, to indicate that the type is an interface.
* Suffix asynchronous method name by "Async".

### Coding best practices

* Write methods which do one and only one job.
* Public methods must validate their input arguments.
* Report execution failures by throwing exceptions.
* Write comments which describe what the code does and/or why it does that, not how the code works.
* There should be an explanation for any code that is commented out.
* No hard coding, use constants/configuration values.
* Never, ever, use a magic number/string.
* Avoid multiple if/else blocks
* Logging: Do not log the critical information (e.g. credit card number, SSN).Consider encrypt critical information in log if the information logging is required.
* Remove unused code from the source file instead of commenting it out.

## Coding Standard

| Control | Prefix | Example |
| --- | --- | --- |
| Label | lbl | lblUsername |
| TextBox | txt | txtPassword |
| DataGrid | dg | dgResults |
| Button | btn | btnSave |
| ImageButton | ibtn | ibtnSave |
| Hyperlink | lnk | lnkHomePage |
| DropDownList | ddl | ddlCompany |
| ListBox | lst | lstCompany |
| DataList | dlst | dlstAddress |
| Repeater | rep | repSection |
| Checkbox | chk | chkRemember |
| RadioButton | rdo | rdoSex |
| RadioButtonList | rdo | rdoAgeGroup |
| Image | img | imgLogo |
| Panel | pan | panSection |
| PlaceHolder | plh | plhHeader |
| Calendar | cal | calDateTime |
| Adrotator | adr | adtBanner |
| Table | tbl | tblResults |
| Validators | val | valCreditCardNumber |
| ValidationSummary | vals | valsErrors |
| Combo box | cbo | cboEnglish |
| Common dialog | dlg | dlgFileOpen |
| Date picker | dtp | dtpPublished |
| Form | frm | frmEntry |
| Frame | fra | fraLanguage |
| Menu | mnu | mnuFileOpen |
| Text Area | txa | txaDescription |

## Code structure

### Feature files

* By using the Cucumber feature, define the scenarios/test case by BDD

Feature: Demo for LNI training

Background: Open browser and navigate to main page

Given A user visits 'https://phptravels.net'

Then Page title is 'PHPTRAVELS | Travel Technology Partner - PHPTRAVELS'

Scenario: User can login successfuly

When User click Account button

And User click Customer Login button

Then Page title is 'Login - PHPTRAVELS'

When set:

| userName | password |

| user@phptravels.com | demouser |

And User type '${userName}' into email

And User type '${password}' into password

And User click Login button

Then Page title is 'Dashboard - PHPTRAVELS'

### Page-Object

#### Keywords

* Define keywords that represent for "actions to elements on UI"
* The intrinsic of the keyword is to use Selenium

async waitUntilElementIsNotVisible(xpath) {

const element = await self.waitUntilElementLocated.call(this, xpath);

return this.driver.wait(until.elementIsNotVisible(element), timeout);

}

async waitClick(xpath) {

const element = await self.waitUntilElementIsClickable.call(this, xpath);

element.click();

}

#### Common

* Define common elements that able to use whenever/wherever
  + Elements detected by xpath/selector …
* Define common actions that able to use whenever/wherever (almost the interact with the common elements)
  + Interact with page and based on element
  + Interact with the page without the element
  + The intrinsic of action is to use keyword

const chai = require('chai');

const keywords = require('./keywords');

const {assert} = chai;

const icnLoading = `(//div[@id='preloader'])[1]`;

async checkPageTitle(title) {

await keywords.waitUntilTitleIs.call(this, title);

const actualTitle = await this.driver.getTitle();

this.attach(`Actual page title: ${actualTitle}`);

this.attach(`Expected page title: ${title}`);

assert.equal(actualTitle, title);

}

async waitLoading() {

await keywords.waitUntilElementIsNotVisible.call(this, icnLoading);

}

#### Specific Page

* Define elements that available to use for the specify page
  + Elements detected by xpath/selector …
* Define actions for specific page
  + Interact with page and based on element
  + Interact with the page without the element
  + The intrinsic of action is to use common/keyword

const keywords = require('./keywords');

const common = require('./common');

const txtEmail = `//span[contains(@class,'user')]/../input[@name='email']`;

const txtPassword = `//input[@name='password']`;

const btnLogin = `//button/span[.='Login']`;

module.exports = {

/\*\*

\* Type user email into the email text box.

\* @param {string} email The user email.

\*/

async typeEmail(email) {

await keywords.setText.call(this, txtEmail, email);

this.attach(`User enter email: ${email}`);

},

/\*\*

\* Type user password into the password text box.

\* @param {string} password The user password.

\*/

async typePassword(password) {

await keywords.setText.call(this, txtPassword, password);

this.attach(`User enter password: ${password}`);

},

/\*\*

\* Click the Login button.

\*/

async clickBtnLogin() {

await common.waitLoading.call(this);

await keywords.waitClick.call(this, btnLogin);

},

};

### Step-definitions

#### Common Steps

* Common steps that can be used for other features
* Define the common step by using the common action in Page Object (refer common)

const {Given, Then} = require('@cucumber/cucumber');

const common = require('../page\_objects/common');

Given('A user visits {string}', common.navigateToPage);

Then('Page title is {string}', common.checkPageTitle);

Then('The page URL is {string}', common.checkPageURL);

#### Specific Steps (page)

* Steps that used only in the specific page/feature
* Define the step by using the action in Page Object (refer common or specific page)

const {When} = require('@cucumber/cucumber');

const {filltemplate} = require('@ln-maf/core');

const loginPage = require('../page\_objects/loginPage');

require('@ln-maf/core/parameter\_types');

require('@ln-maf/validations');

const fillTemplate = filltemplate;

/\*\*

\* Returns the value of the variable if it exists in this.results.

\* @param {string} variable the variable to check

\* @param {Object} scenario the scenario to check

\* @return {Object} the value of the variable if it exists in this.results.

\* Returns the variable itself if variable does not contain "${}"

\*/

function getVal(variable, scenario) {

if (!scenario.results) {

scenario.results = {};

}

return fillTemplate(variable, scenario.results);

}

When('User type {string} into email', async function(email) {

email = getVal(email, this);

await loginPage.typeEmail.call(this, email);

});

When('User type {string} into password', async function(password) {

password = getVal(password, this);

await loginPage.typePassword.call(this, password);

});

When('User click Login button', loginPage.clickBtnLogin);

### Support

#### Hook

Hooks are used for setup and teardown the environment before and after each scenario. *Hooks* allows us to better manage the code workflow and helps us to reduce the code redundancy. We can say that it is an unseen step, which allows us to perform our scenarios or tests.

In the world of testing, you must have encountered the situations where you need to perform the prerequisite steps before testing any test scenario. This prerequisite can be anything from:

* Starting a webdriver
* Setting up DB connections
* Setting up test data
* Setting up browser cookies
* Navigating to certain page
* or anything before the test

In the same way, there are always after steps as well of the tests like:

* Killing the webdriver
* Closing DB connections
* Clearing the test data
* Clearing browser cookies
* Logging out from the application
* Printing reports or logs
* Taking screenshots on error
* or anything after the test

See the [API reference](https://github.com/cucumber/cucumber-js/blob/4c80df1a25c3bb25dc57d65ab8e5ee842a469826/docs/support_files/api_reference.md) for the specification of the first argument passed to hooks. Multiple *Before* hooks are executed in the order that they were defined. Multiple *After* hooks are executed in the reverse order that they were defined.

const {Before, AfterStep, After, Status} = require('@cucumber/cucumber');

Before(function() {

return this.driver.manage().window().maximize();

});

AfterStep(async function({result}) {

const world = this;

if (result.status === Status.FAILED) {

await this.driver.takeScreenshot().then((screenShot) => {

world.attach(screenShot, 'base64:image/png');

});

}

});

After(function() {

return this.driver.quit();

});

#### World

*World*, is an isolated scope for each scenario, exposed to the steps and most hooks as this. It allows you to set variables in one step and recall them in a later step. All variables set this way are discarded when the scenario concludes. It is managed by a world class, either the default one or one you create. Each scenario is given a new instance of the class when the test starts, even if it is a retry run.

The world is not available to the hooks BeforeAll or AfterAll as each of these executes outside any particular scenario.

You might also want to have methods on your world that hooks and steps can access to keep their own code simple. To do this, you can write your own world implementation with its own properties and methods that help with your instrumentation, and then call setWorldConstructor to tell Cucumber about it:

const {setWorldConstructor} = require('@cucumber/cucumber');

function CustomWorld({attach, parameters}) {

this.attach = attach;

this.parameters = parameters;

this.driver = new webdriver.Builder()

.withCapabilities(capabilities)

.build();

}

setWorldConstructor(CustomWorld);

### Report

Multiple Cucumber HTML Reporter is a reporting module for Cucumber to parse the JSON output to a beautiful report. The difference between all the other reporting modules on the market is that this module has:

* A quick overview of all tested features and scenarios.
* A features overview that can hold multiple runs of the same feature / runs of the same feature on different browsers / devices.
* A features overview that can be searched / filtered / sorted.
* A feature(s) overview with metadata of the used browser(s) / devices.

Install this module locally with the following command:

npm install multiple-cucumber-html-reporter

Save to dependencies or dev-dependencies:

npm install multiple-cucumber-html-reporter --save

npm install multiple-cucumber-html-reporter --save-dev

Compatibility

Multiple Cucumber HTML Reporter now works with CucumberJS 1, 2, 3 and 4.

Dependency

"multiple-cucumber-html-reporter": "^3.0.1"

Report config

#!/usr/bin/env node

const report = require('multiple-cucumber-html-reporter');

const os = require('os');

const {startTime} = process.env;

const {endTime} = process.env;

const browserName = process.env.BROWSER;

const paramPath = process.env.param;

const customData = {

title: 'Run info',

data: [

{label: 'Execution Start Time:', value: startTime},

{label: 'Execution End Time:', value: endTime},

],

};

const platformMap = {

darwin: 'osx',

window: '10',

};

let platformName = os.platform();

if (platformMap[os.platform()]) {

platformName = platformMap[os.platform()];

}

const metadata = {

device: 'Local test machine',

browser: {

name: browserName,

},

platform: {

name: platformName,

version: os.release(),

},

};

report.generate({

jsonDir: `./report/${paramPath}`,

reportPath: `./report/${paramPath}`,

displayDuration: true,

metadata,

customData,

reportName: 'Generate Events Report',

saveCollectedJSON: true,

openReportInBrowser: true,

});

### Execution

export param=$1

rm -rf ./report/$param/

mkdir -p ./report/$param/

export startTime=$(date)

if [[ $param = "all" ]]; then

export BROWSER="edge"

node node\_modules/@cucumber/cucumber/bin/cucumber-js -f json:report/$param/edge.json src/features/

export BROWSER="firefox"

node node\_modules/@cucumber/cucumber/bin/cucumber-js -f json:report/$param/firefox.json src/features/

export BROWSER="chrome"

node node\_modules/@cucumber/cucumber/bin/cucumber-js -f json:report/$param/chrome.json src/features/

else

export BROWSER=$param

node node\_modules/@cucumber/cucumber/bin/cucumber-js -f json:report/$param/cucumber\_report.json src/features/

fi

result=$?

export endTime=$(date)

node multiReport.js

exit $result

* export NAME=VALUE
  + To create a new variable, use the export command by a variable name and its value.
* rm -rf command
  + Delete directory forcefully. By giving that command it will delete that particular file having read-only access without asking for permission.
* mkdir -p command
  + Create sub-directories of a directory. It will create a parent directory first, if it doesn't exist. But if it already exists, then it will not print an error message and will move further to create sub-directories. This command is most helpful in the case when you don't know whether a directory already exists or not.
* In cucumber-js, Formatters ingest data about your test run in real time and then output content, either to the console or a file, in a useful format.
  + You can specify one or more formats via the format configuration option:
    - In a configuration file { format: ['<TYPE[:PATH]>'] }
    - On the CLI $ cucumber-js --format <TYPE[:PATH]>
  + For each value you provide, TYPE should be one of:
    - The name of one of the built-in formatters (below) e.g. progress
    - A module/package name e.g. @cucumber/pretty-formatter
    - A relative path to a local formatter implementation e.g. ./my-customer-formatter.js
    - An absolute path to a local formatter implementation in the form of a file:// URL
  + If PATH is supplied, the formatter prints to the given file, otherwise it prints to stdout.For example, this configuration would give you a progress bar as you run, plus JSON and HTML report files:
    - In a configuration file { format: ['progress-bar', 'json:cucumber-report.json', 'html:cucumber-report.html'] }
    - On the CLI $ cucumber-js --format progress-bar --format json:cucumber-report.json --format html:cucumber-report.html

| node node\_modules/@cucumber/cucumber/bin/cucumber-js | Using the cucumber-js to run the BDD |
| --- | --- |
| -f | Type format |
| json:report/$param/edge.json | Define the format type that use  $param is the first argument input when run the bash |
| src/features/ | Path of feature files |

Define the cucumber-js to run BDD in package.json

"scripts": {

"cucumber": "node\_modules/@cucumber/cucumber/bin/cucumber-js",

"firefox": "node\_modules/@cucumber/cucumber/bin/cucumber-js -t @firefox",

"chrome": "node\_modules/@cucumber/cucumber/bin/cucumber-js -t @chrome",

"edge": "node\_modules/@cucumber/cucumber/bin/cucumber-js -t @edge"

}

### CI/CD

name: "Full Regression Automation UI"

on:

push:

branches:

- "\*\*"

schedule:

- cron: "0 7 \* \* 0"

workflow\_dispatch:

jobs:

full-regression-ui:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v2

- name: Set up Node.js 14.16.1

uses: actions/setup-node@v2

with:

node-version: "14.16.1"

- name: Set up libraries for browsers

id: setup-libs-chrome

run: |

sudo apt update

sudo apt install -y libgconf-2-4 libatk1.0-0 libatk-bridge2.0-0 libgdk-pixbuf2.0-0 libgtk-3-0 libgbm-dev libnss3-dev libxss-dev libasound2 libx11-xcb1

- name: Set up Google Chrome

uses: browser-actions/setup-chrome@latest

with:

chrome-version: stable

- name: Set up Edge

uses: browser-actions/setup-edge@latest

with:

edge-version: stable

- name: Set up Firefox

uses: browser-actions/setup-firefox@latest

with:

firefox-version: "105.0"

- name: Run Automation Test UI with Chrome browser

id: automation-test-chrome

run: |

npm install

bash runFeature\_FullRegression.sh chrome

- name: Run Automation Test UI with Edge browser

id: automation-test-edge

run: |

bash runFeature\_FullRegression.sh edge

- name: Run Automation Test UI with FireFox browser

id: automation-test-firefox

run: |

bash runFeature\_FullRegression.sh firefox

- name: Upload Artifact

uses: actions/upload-artifact@v2

if: always()

with:

name: FullRegressionUI

path: ${{ github.workspace }}/report/\*\*/\*

retention-days: 2

### Dependencies

The information of third-party tools will be installed and stored in the package.json file

{

"name": "package.json",

"version": "1.0.0",

"description": "Automation for LNI training",

"main": "mainPage.js",

"dependencies": {

"@cucumber/cucumber": "^8.5.2",

"@ln-maf/core": "^1.6.1",

"@ln-maf/validations": "^1.6.2",

"chai": "^4.3.6",

"chromedriver": "^107.0.0",

"geckodriver": "^3.0.2",

"lodash": "^4.17.21",

"multiple-cucumber-html-reporter": "^3.0.1",

"newmsedgedriver": "^107.0.0",

"selenium-webdriver": "^4.4.0",

"string-contains": "^0.1.0"

},

"devDependencies": {

"eslint": "^8.24.0",

"eslint-config-google": "^0.14.0",

"npm": "^8.18.0"

}

}

# Reference

## async/await

**Async / Await** is a JavaScript feature that helps us to work with asynchronous functions in a way that is more interesting and easier to understand. It is built on Promises and is compatible with all API-based Promises. In there:

**Async** - declares an async function (async function someName(){...}).

* Automatically transform a regular function into a Promise.
* When calling the async function it will handle everything and return the result in its function.
* Async allows the use of Await.

**Await** - pauses the execution of async functions. (Var result = await someAsyncCall())

* When placed in front of a Promise, it waits until the Promise ends and returns the result.
* Await only works with Promises, it doesn't work with callbacks.
* Await can only be used inside async functions.

Here is a simple example that will hopefully clear the above:

Let's say we want to get some JSON files from our server. We will write a function that uses the axios library and sends an HTTP GET request to Https://tutorialzine.com/misc/files/example.json . We have to wait for a response from the server, which is an asynchronous HTTP request.

Below we will write the same function in 2 different ways: 1st way with Promises, and 2nd way with Async/Await

| // way 1:  function getJSON() {  // To make the function blocking we manually create a Promise.  return new Promise( function(resolve) {  axios.get('https://tutorialzine.com/misc/files/example.json')  .then( function(json) {  // The data from the request is available in a .then block  // We return the result using resolve.  resolve(json);  });  });  }  // way 2:  // Async/Await approach  // The async keyword will automatically create a new Promise and return it.  async function getJSONAsync() {  // The await keyword saves us from having to write a .then() block.  let json = await axios.get('https://tutorialzine.com/misc/files/example.json');  // The result of the GET request is available in the json variable.  // We return it just like in a regular synchronous function.  return json;  } |
| --- |

Refer for more details <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function>

## Constructor

## setWorldConstructor

Refer for more details <https://github.com/cucumber/cucumber-js/blob/main/docs/support_files/world.md>

## promisify

## World term

## fillTemplate

Refer for more details  
<https://www.npmjs.com/package/fill-template>