CodeCept JS

Site web officiel : https://codecept.io/

Cet outil dispose des capacités pour l'automatisation test Front (IHM Web) & Back (API Rest), le bouchonnage à la volée par interception de requête pour le front, il supporte également le test des applications mobile.

C'est un framework agnostique du driver, que l'on souhaite utiliser Playwright, WebDriver, Puppeteer, TestCafe, Protractor ou Appium : le code de test sera le même.
Il dispose également d'un debugger interactif.

Pré-Requis

Disposer de nodejs v14 minimum.

Limites

L'abstraction multi drivers des bouchons n'est pas exploitable à l'identique sur tous les drivers.

Initialiser le projet

Codecept intègre un installateur tout en un, en une commande cela créé le projet et fourni un cas d'exemple :

```
npx create-codeceptjs .
```

Ensuite il faudra lancer l'initialisation interactive du projet :

```
npx codeceptjs init
```

Cela permet de choisir différents paramètres tel que :

- L'activation de typescript (Non par défaut)
- L'emplacement et le nommage des tests (dossier courant par défaut et fichier se terminant r _test.js)
- Le Driver parmi :
  - Playwright (par défaut)
  - WebDriver
  - Puppeteer
  - REST
  - GraphQL
  - Appium
  - TestCafe
- le dossier contenant logs, capture d'écran et rsudo aptapports (./output par défaut)

- le choix de la localisation coode dans une langue spécifique (aucun par défaut)

Concernant playwright, il sera demandé le navigateur à utiliser : chromium (défaut), firefox, webkit, electron.
Il sera demandé l'URL d'accès à l'application (https://www.francetravail.fr dans cet exemple) ainsi que le choix d'afficher le navigateur. (affiché par défaut)

Nous allons choisir les options par défaut sauf la location où l'on prendra 'fr-FR'.

Il sera demandé le nom de la fonctionnalité testée puis le nom du fichier de tests.

Enfin il faudra installer les dépendances playwright :

```
npx playwright install --with-deps chromium
```

Ensuite deux modes de tests sont possibles :

- Tests simple avec support d'une notion de langage métier/description des tests.
- Tests gherkin pour faire du Behavior Driven Development

Dans tous les cas il sera important de s'attarder sur les concepts intégrés à l'outil pour :

- les bouchons
- le pattern page object
- la gestion de persona

Faire ses premiers tests

Deux types de tests :

- Classique : test classique
- Gherkin/Cucumber : pour faire du béhavior driven develpoment

Voici différents points pour aller plus loin :

- Page Object
- Test d'api
- Bouchons

Pour une utilisation plus avancée :

- Astuces
- Reporting
- Persona
- Continious Integration

Les tests classiques

Ce sont des tests qui ne basent pas sur le concept gherkin/cucumber.

Ils sont rédigés sous forme d'une fonction qui dispose d'une description.

Ils peuvent être associé à une fonctionnalité.

Pour créer un nouveau test :

```
npx codeceptjs gt
```

Il va être demandé :

- La fonctionnalité testée : Page d'accueil
- le nom du fichier : accueil_test.js

Un fichier accueil_test.js contient :

```
Fonctionnalité('Page d\'accueil');

Scénario('test something',  ({ Je }) => {

});
```

On va pouvoir transformer le scénario 'test something' en 'Faire une recherche' :

```
Fonctionnalité('Page d\'accueil');

Scénario('Faire une recherche',  async ({ Je }) => {
    Je.suisSurLaPage('/');
    Je.cliqueSur("#id");
   /*
    ...
   */
    Je.voisDansLeChamp("div.css","Leognan");
});
```

Retour Accueil

Test gherkin/cucumber sous CodeCeptJS

Le prefixe `gherkin` des commandes shell, peut être remplacé par `bdd` pour une saisie plus rapide

Activer le Gherkin / BDD

Pour initialiser la partie BDD (Behavior Driven Development), il suffit de lancer :

```
npx codeceptjs gherkin:init
```

Cela va adapter du projet pour ajouter - un dossier "features" qui contiendra les critères d'acceptation au format gherkin - un dossier "step_definitiions" qui

contiendra le code permettant de lier les critères d'acception a la manipulation de l'application - l'activation de gherkin dans la codecept.conf.js :

```
gherkin: {
  features: './features/*.feature',
  steps: ['./step_definitions/steps.js']
}
```

Exemple de feature :

```
@gherkin @accueil
Feature: Page d'accueil
  En tant que candidat non authentifié
  Je souhaite pouvoir rechercher une offre d'emploi

  @recherche @code_postal
  Scenario: Recerche par code postal
    Given je suis suis sur la page d'accueil
    When je saisi le code postal 33850 et selectionne l'id commune 33238
    And je lances une rechercher
    Then je vois les offres pour la commune de "Léognan"
```

Générer les steps définitions

```
npx codeceptjs gherkin:snippets
```

Cette commande va générer dans step_defintions/steps.js l'intégralité des actions des fichiers feature qui n'existe pas déjà.

La création des steps définition utilise par défaut toutes les fichiers feature présent dans les dossiers de la configuration de codecept et positionne les définitions dans le premier fichier de steps de la configuration.
Il est possible de préciser `--path=` pour donner le fichier de steps de destination et/ou `--feature=` pour cibler un fichier gherkin spécifique.

L'implémentation

Exemple d'implémentation :

```
When("je lances une rechercher", () =>{
  Je.cliqueSur(accueilPage.boutons.recherche);
});
```

Il est possible de passer des paramètre depuis les fichiers feature, pas de besoin d'expression régulière, l'utilisation d'accolades et du type de données:

```
When("je saisi le code postal {int} et sélectionne la commune {string}", (codePostal, commun
  Je.remplisLeChamp(accueilPage.champs.lieu, codePostal);
  Je.cliqueSur(accueilPage.listes.communes(commune));
});
```

Les différents types :

Type	Description
`{int}`	les entiers tel que `71` ou `-19`
`{float}`	les chiffres décimaux `3.6`, `.8` ou `-9.2`
`{word}`	correspond à un seul mot sans espace `banana` (mais pas `banana split`)
`{string}`	une chaine de caractere entre simple ou double quotes `"banana split"` ou `'banana split'` (mais pas `banana split`). les quotes qui encadrent la chaine ne seront pas passées à la variable
`{}` anonymous	correspond à tout (`/.*/`).

Il est possible de n'executer qu'une partie des features (ou des scénarios):

```
npx codeceptjs run --steps --grep "@code_postal"
```

Retour Accueil

Le pattern Page Object sur CodeCept JS

Initialiser un page object

La création est intégrée à l'outil par la commande :

```
npx codeceptjs gpo
```

Cela demandera le nom de la page puis le lieu de stockage.
L'exemple ici utilisera les pages "Accueil" et "ListeOffres" dans le dossier par défaut "pages" et au format "module".

Le shell interactif va créer le fichier "Accueil.js" contenant :

```
const { I } = inject();

module.exports = {

  // insert your locators and methods here
}
```

Si l'on utilise la localisation "FR", il faudra peut-être modifier les injections pour ajouter "Je" :

```
const { I, Je } = inject();
```

Et il ajoutera à codecept.conf.js l'include nécessaire :

```
include: {
    Je: './steps_file.js',
    accueilPage: "./pages/Accueil.js",
  },
```

Pour utiliser le pageobject dans un test classique, il suffira de le rajouter en paramètre du test en plus de la persona (I/Je par défaut) :

```
Scénario('Faire une recherche',  async ({ Je, accueilPage, listeOffresPage, popinCookiesPage
 /*
 ...
 */
});
```

Utiliser un page object

Pour l'utiliser en test gherkin, il faut ajouter l'injection en début du fichier de steps definitions :

```
const { I, Je } = inject();
const { accueilPage, listeOffresPage } = inject();

Given('je suis connecté', () => {
  /*
    ...
  */
});
```

Le Pattern PageOject

Ce concept permet d'avoir des objets qui vont servir a manipuler l'application et mutualiser (permettre de réutiliser) des éléments de l'écran dans plusieurs tests ou scénarios différents.

Le PageObject ne doit contenir que ce qui permet d'identifier les éléments (bouton, champs, image, zone texte, etc.) et de quoi réaliser des actions sur ceux-ci.

Les contrôles (assertions) seront dans les tests car ils sont spécifiques aux tests. Une exception peut-être une méthode permettant d'attendre/manipuler un chargement long/non géré par le framework d'automatisation qui serait utilisée dans **plusieurs** tests.

CodeCept utilisant des actions très simples de manipulation et gérant le concept d'acteur, le PageObject pourra porter uniquement les éléments d'identification qui seront utilisés ensuite dans les tests.

Il n'y a pas d'intérêt à faire un page.clicksurbouton() plutôt qu'un simple Je.clickSur(page.bouton).
Le test portant ainsi l'acteur, cela permet une lisibilité accrue.

Exemple Page-Object sur écran de connexion :

```
module.exports = {
  champs: {
    identifiant: "#identifiant",
    motDePasse: "#password",
  },
  boutons: {
    poursuivre: "#submit",
    seConnecter: "#submit",
  }
}
```

Utilisation du page object dans un test

```
Je.voisLElément(connexionPage.boutons.poursuivre);
Je.remplisLeChamp(connexionPage.champs.identifiant, "mon-login");
Je.cliqueSur(connexionPage.boutons.poursuivre);
Je.remplisLeChamp(connexionPage.champs.motDePasse, "secret");
Je.cliqueSur(connexionPage.boutons.seConnecter);
```

Page Fragment

Pour les cas spécifiques, tel que iframe ou widget, il est possible de créer des page fragement qui permettront de de contraindre les actions DANS le fragment est donner une lisibilité accrue.

Créer le page fragment :

```
npx codeceptjs go --type fragment
```

Exemple de contenu :

```
const { Je } = inject();
// fragments/modale.js
module.exports = {

  racine: '#modal',
  boutons : {
    accepter: '#accept'
  }
}
```

L'utiliser :

```
Scenario('fragment', async ({ j,  modale }) => {
  I.attendsPourVoir(modale.racine);
```

```
  within(modale.racine, function () {
    I.cliqueSur(racine.boutons.accepter);
  })
});
```

Cela reste techniquement un concept de page object avec une propriété racine (root) permettant d'exploiter la fonctionnalité `within` de codeceptjs.

Plus de détails: https://codecept.io/pageobjects/#page-fragments

Retour Accueil

Test d'api

Activation des modules API

Il est nécessaire d'ajouter les dépendances pour les tests d'api en tant que helpers dans `codecept.conf.js` :

```
  helpers: {
    /**
     ...
     * */
    REST: {
      endpoint: 'https://api.francetravail.io',
      prettyPrintJson: true
    },
    JSONResponse: {},
  }
```

Utilisation

Le concept reste le même, en test classique ou gherkin.
Les mots clés ne sont pas encore traduit.
Une gestion simplifiée de bearer est présente.
Exemple :

```
  I.amBearerAuthenticated('123456abcdef');
  I.sendGetRequest('/partenaire/offresdemploi/v2/offres/search?codeROME='+rome);
  I.seeResponseCodeIsSuccessful();
```

Documentation : https://codecept.io/api/
Tout un jeu de simplification existe pour les controles JSON et les codes retour (agrégation des Succes & Echec par exemple).

Assertion Avancées

Il est possible de faire des assertions plus avancées avec une mécanique de call-back :

```
I.seeResponseValidByCallback(({ data, status, expect }) => {
    for(let offre of data.resultats) {
      expect(offre.romeLibelle.toLowerCase()).to.include(metier.toLowerCase())
    }
  })
```

Retour Accueil

Les bouchons

Les bouchons font partis de la partie qui n'a pas de mots clés uniformément implémenté sur les différents drivers à l'heure actuelle.

Méthode native

`MockRoute` est disponible sur Playwright & Puppeteer.
On obtient un objet route qui permet de traiter le fonctionnent: code retour, reponse, modification headers, etc :

```
Acteur.mockRoute('https://www.francetravail.fr/api-accueil/api/lieu/'+recherche, route =>
```

Playwright dispose d'une version simplifiée `MockTraffic`.
Il permet pour une url, de renvoyer une réponse.

```
Acteur.mockTraffic('https://www.francetravail.fr/api-accueil/api/lieu/'+recherche,reponse
```

Helper Tiers : MockRequest

Documentation : https://codecept.io/helpers/MockRequest/

Il s'agit d'un helper tiers qui supporte Puppeteer et WebDriver.

Installer le package :

```
npm i -D @codeceptjs/mock-request
```

Et ajouter le helper dans `codecept.conf.js` :

```
helpers: {
  MockRequestHelper: {
      require: '@codeceptjs/mock-request',
  }
}
```

Il permet de faire des mocks simple ou avancés :

```
I.startMocking(); //Optionnel
```

```
I.mockRequest('/google-analytics/*path', 200);
I.mockRequest('GET', '/api/users', 200);
I.mockServer(server => {
  server.get('https://server.com/api/users*').
    intercept((req, res) => { res.status(200).json(users);
  });
});

I.click('Get users);

I.stopMocking(); //permet de desactiver les bouchons
```

Il dispose également d'une fonctionnalité enregistrement/rejeu qui permet de rapidement créer des bouchons:

```
I.recordMocking(); // lance l'enregistrement

I.replayMocking(); //active le rejeu
```

Retour Accueil

Petits bonus

Playwright

Activer les videos

```
Playwright: {
      ...
      video: true,
    }
```

Activer les traces

```
Playwright: {
      ...
      trace: true,
    }
```

Les traces sont des fichiers zip (un par test), contenant l'intégralité du déroulé avec l'équivalent des options développeur :

- sortie console
- traces réseaux
- actions
- capture du déroulé
- code source

Pour les voir en local :

```
npx playwright show-trace <fichier.zip>
```

En version web avec deux usages :

- Appli PWA restant en local : https://trace.playwright.dev/
- Appli PWA en fournissant l'URL des traces : https://trace.playwright.dev/?trace=http(s):///<fichier.zip

Capture d'écran

Quelques options complémentaires sur les captures d'écran.

De toute la page

```
Playwright: {
      ...
      fullPageScreenshots: true,
    }
```

Mêmee si le test est OK

```
Playwright: {
      ...
      keepTraceForPassedTests: true,
    }
```

Regenerer l'aide à la saisie VSCode

Lancer :

```
npx codeceptjs def
```

Cela régénère le fichier steps.d.ts avec les derniers acteurs/pages incluent dans le projet.

Utiliser différentes configurations

Pour du multi-environnement, différents navigateurs ou autres spécificités:

```
npx codeceptjs run -c configs/ci.conf.js
```

Utiliser un Proxy

Il sera nécessaire de configurer la variable d'environnement :

```
https_proxy=http://mon.proxy:8080
http_proxy=$https_proxy
HTTP_PROXY=$https_proxy
HTTPS_PROXY=$https_proxy
no_proxy=mon-repo.proxy,mon-repo.node
```

```
ELECTRON_GET_USE_PROXY=true
GLOBAL_AGENT_HTTPS_PROXY=$https_proxy
```

Spécificité pour télécharger Playwright

Si vous disposez d'un proxy de repository, il est possible de le spécifier pour playwrigth, exemple proxy artifactory:

```
PLAYWRIGHT_DOWNLOAD_HOST=http://mon-repo.proxy/artifactory/playwright-proxy
```

Changement d'onglet

//!TODO

Les Iframe

Il existe deux possibilités pour manipuler le contenu d'une iframe, soit avec la fonctionnalité `within` :

```
within({frame: "#editor"}, () => {
  I.vois('Page');
});
```

Une fois sorti de `within`, le test reprend le contrôle de la page racine.
Il ne permet cependant pas de gérer une iframe incluse dans une iframe.

Soit avec switchTo() :

```
I.switchTo('#editor');
I.switchTo('iframe');  //Prend l'iframe peu importe ses propriétés
I.vois('sous page');
I.switchTo(); //retour à page racine
```

Il est alors possible de passer d'iframe en sous iframe sans soucis.
Mais il est nécessaire de retourner à la "racine" en utilisant la même méthode sans paramètre.

Réaliser une action non bloquante

Ajouter le plugin à codecept.conf.js :

```
plugins: {
    tryTo: {
        enabled: true
    }
}
```

Utilisation :

```
const cookiesAcceptes = await tryTo(() => Je.click("#pecookies-continue-btn"));
```

La fonction qui l'exécute devra être async.
Le résultat est un booléen, permettant ainsi de savoir si l'essai a réussi ou non.

Retour Accueil

Les rapports

Par défaut le reporting est uniquement la sortie console avec un code retour adapté pour une remonté OK/KO dans une CI/CD

Beaucoup d'option locale ou distante existent : https://codecept.io/reports/

Rapport Cucumber

Installer la dépendance :

```
npm i --save-dev codeceptjs-cucumber-json-reporter
```

JSON

C'est la source pour générer des rapports HTML ou des imports dans les outils de gestions de tests tel qu'XRAY.

Il faut ajuster la `codecept.conf.js` avec le plugin :

```
plugins: {
  cucumberJsonReporter: {
    require: 'codeceptjs-cucumber-json-reporter',
    enabled: true,
    attachScreenshots: true,
    attachComments: true,
    outputFile: 'output/cucumber.json',
    includeExampleValues: true,
    timeMultiplier: 1000000,
  },
}
```

Le fichier sera disponible dans output après l'exécution des tests pour traitement ou import.

Pour ajouter des commentaires dans le rapport on peut utiliser :

```
I.say("mon commentaire")
```

Pour ajouter des captures dans le rapport on peut utiliser :

```
I.saveScreenshot()
```

HTML

Installer la dépendance :

```
npm i -D cucumber-html-reporter
```

Créer un fichier report.js contenant :

```
var reporter = require('cucumber-html-reporter');

var options = {
        theme: 'bootstrap',
        jsonFile: 'output/cucumber.json',
        output: 'output/cucumber_report.html',
        reportSuiteAsScenarios: true,
        scenarioTimestamp: true,
        launchReport: true,
        metadata: {
            "App Version":"0.3.2",
            "Test Environment": "STAGING",
            "Browser": "Chromium",
        },
        failedSummaryReport: true,
    };
    reporter.generate(options);
```

La génération du rapport se fait avec :

```
node report.js
```

Rapport HTML Allure

Pré-Requis

Disposer une machine virtuelle java (https://www.java.com/fr/download/)

Configurer

Couvre le gherkin et les tests classiques

Installer la dépendance :

```
npm install --save-dev allure-codeceptjs
```

```
npm install --save-dev -g allure-commandline
```

Ajouter dans `codecept.js.conf` :

```
  plugins: {
    allure: {
      enabled: true,
      require: "allure-codeceptjs",
    },
  }
```

Commande pour générer le rapport :

```
allure generate --clean  -o ./output/allure
```

En version fichier HTML unique (images autoportées en base64) :

```
allure generate --clean --single-file -o ./output/allure
```

Nettoyer le reporting

Afin de nettoyer les dossiers de reporting, il est possible de créer un script `report_clean.js` :

```
const fs = require('fs');
fs.rmSync("./output", { recursive: true, force: true });
fs.rmSync("./allure-results", { recursive: true, force: true });
```

Et de l'appeler avec:

```
node report_clean.js
```

Retour Accueil

La gestion des acteurs

Codecept intègre une gestion d'acteur. L'acteur par défaut est "I" (Je) et ne dispose que des méthodes standards.

Ajout d'une action personnalisée

Dans actors/CandidatNonConnecté.js, il est possible de rajouter des actions spécifiques pour l'acteur :

```
"use strict";

const { accueilPage, popinCookiesPage } = inject();

module.exports = function() {
  return actor({

    ouvreLeSite: async function() {
      this.suisSurLaPage(accueilPage.url);
      const cookiesAcceptes = await tryTo(() => this.limitTime(10).cliqueSur(popinCookiesPag
    }

  });
}
```

`"use strict";` est nécessaire.

Il faut définir actors/CandidatConnecté.js également avec la cinématique liée à la connexion.

Mettre à jour codecept.conf.js pour les includes :

```
  include: {
    CandidatConnecte: './actors/CandidatConnecté.js',
    CandidatNonConnecte: './actors/CandidatNonConnecté.js',
    Je: './steps_file.js',
    accueilPage: './pages/Accueil.js',
    listeOffresPage: './pages/ListeOffres.js',
    popinCookiesPage: './pages/PopinCookies.js',
    espaceCandidatPage: "./pages/EspaceCandidat.js",
    connexionPage: "./pages/Connexion.js",
  },
```

Dans ce contexte, le I/Je devra être remplacé par “this”. La gestion de l’ouverture du site a était déplacée dans une fonction “ouvrirLeSite” qui permettra en fonction de la persona de gérer l’ouverture de manière adaptée (connecté, non connecté).

Il faut ajuster les steps definitions pour gérer un “Acteur”, et spécifier l’acteur par défaut pour limiter la régression des cas de tests précédent. Puis gérer les cas d’acteur.

```
let Acteur=CandidatNonConnecte;

Given('je suis un candidat connecté', async () => {
  Acteur=CandidatConnecte;
});

Given('je suis un candidat non connecté', async () => {
  Acteur=CandidatNonConnecte;
});

Given('je suis suis sur la page d\'accueil', async () => {
  await Acteur.ouvreLeSite();
});
```

Les features peuvent désormais disposer de deux cas d’usage tout en mutualisant les actions.
Les personnas peuvent aussi être utilisées en parallèles dans un même test en combinaison avec la gestion de session multiples : https://codecept.io/acceptance/#multiple-sessions

Il est aussi possible de récupérer des cookies de la session navigateur pour les tests d’api, en ajoutant dans codecept.conf.js :

```
const { setSharedCookies } = require('@codeceptjs/configure');
// a mettre avant exports.config
```

```
setSharedCookies();
```

Retour Accueil

Continious Integration

Voir dans le dossier `ci`, plusieurs exemples de pipeline sont présents :

- concourse
- gitlab-ci
- jenkins

Le principe est de partir de l'image codeceptjs ou d'une image node, d'installer les dépendances node puis le/les navigateurs du driver. (playwright dans les exemples).

Pour le reporting, il être etre généré dans le même stage (exemple concourse) ou un autre (exemple gitlab-ci).

Retour Accueil