**REPORT PORTAL DEPLOY NODE JS**

Being an internal app with strictly limited access, there are no information about to step-by-step integration of this software. However, there are some actions what should be done basing on pure empiric observations - according to that, we will observe a RP integration to already installed and running service.

We will use deployment to project what executes with protractor.

First of all, after initializing project we need to install some plugins and libraries for proper integration of the RP. There is a package.json file:

|  |
| --- |
| {    "name": "project Name",    "version": "1.0.0",    "description": "",    "main": "index.js",    "scripts": {      "test:dev": "cross-env NODE\_ENV=dev RP\_ENABLED=true TEST\_HOST=local protractor config/conf.js --suite complexAppSuite",      "prepare": "npx webdriver-manager update --ignore\_ssl=true",      "postinstall": "webdriver-manager update --versions.standalone=3.14.0 --gecko=false",      "longApp:dev5": "cross-env NODE\_ENV=dev5 APP\_TYPE=long RP\_ENABLED=true TEST\_HOST=local protractor config/conf.js --specs=tests/resubmitAppE2E\_LongApp.spec.js",      "shortApp:dev5": "cross-env NODE\_ENV=dev5 APP\_TYPE=short RP\_ENABLED=true TEST\_HOST=local protractor config/conf.js --specs=tests/resubmitAppE2E\_ShortApp.spec.js",      "longApp:dev": "cross-env NODE\_ENV=dev APP\_TYPE=long RP\_ENABLED=true TEST\_HOST=local protractor config/conf.js --specs=tests/resubmitAppE2E\_LongApp.spec.js",      "shortApp:dev": "cross-env NODE\_ENV=dev APP\_TYPE=short RP\_ENABLED=true TEST\_HOST=local protractor config/conf.js --specs=tests/resubmitAppE2E\_ShortApp.spec.js",      "longApp:qa": "cross-env NODE\_ENV=qa1 APP\_TYPE=long RP\_ENABLED=true TEST\_HOST=local protractor config/conf.js --specs=tests/resubmitAppE2E\_LongApp.spec.js",      "shortApp:qa": "cross-env NODE\_ENV=qa1 APP\_TYPE=short RP\_ENABLED=true TEST\_HOST=local protractor config/conf.js --specs=tests/resubmitAppE2E\_ShortApp.spec.js",      "longApp:uat": "cross-env NODE\_ENV=uat APP\_TYPE=long RP\_ENABLED=true TEST\_HOST=local protractor config/conf.js --specs=tests/resubmitAppE2E\_LongApp.spec.js",      "shortApp:uat": "cross-env NODE\_ENV=uat APP\_TYPE=short RP\_ENABLED=true TEST\_HOST=local protractor config/conf.js --specs=tests/resubmitAppE2E\_ShortApp.spec.js"    },    "author": "author name",    "license": "ISC",    "dependencies": {},    "devDependencies": {      "@babel/core": "^7.8.4",      "@babel/preset-env": "^7.8.4",      "@babel/register": "^7.8.3",      "babel-preset-env": "^1.7.0",      "babel-preset-es2015": "^6.24.1",      "babel-register": "^6.26.0",      "chai": "^4.2.0",      "cross-env": "^6.0.3",      "deasync": "^0.1.19",      "eslint": "^6.8.0",      "eslint-config-airbnb": "^18.0.1",      "eslint-config-airbnb-base": "^14.0.0",      "eslint-config-node": "^4.0.0",      "eslint-plugin-import": "^2.20.0",      "eslint-plugin-jsx-a11y": "^6.2.3",      "eslint-plugin-node": "^11.0.0",      "eslint-plugin-react": "^7.14.3",      "eslint-plugin-react-hooks": "^1.7.0",      "mocha": "^6.2.2",      "mocha-multi-reporters": "^1.1.7",      "mochawesome": "^4.1.0",      "mochawesome-merge": "^1.0.7",      "mochawesome-report-generator": "^3.1.5",      "protractor": "^5.4.2",      "reportportal-client": "5.2.3",      "rp-mocha-reporter": "^2.0.7",      "underscore": "^1.9.2",      "winston": "^3.2.1"    }  } |

So after initializing project we should:

1. **REMOVE** *node\_modules* folder
2. Insert this content above into **package.json** file(completely clear it out before)
3. Execute command **NPM INSTALL**

After installation will be completed, we can proceed further.

In our root we need to create ***config*** folder, where we need to place a file named **reportPortalConf.js**

|  |
| --- |
| require("dotenv").config();  const env = process.env.NODE\_ENV.toUpperCase();  const host = process.env.TEST\_HOST.toUpperCase();  export const config = {    token: process.env.RP\_TOKEN || "ToKen from your RP profile",    endpoint: process.env.RP\_ENDPOINT || "Link to your version of RP",    launch: process.env.RP\_LAUNCH ?      `${process.env.RP\_LAUNCH} - ${env}` : `YOUR CUSTOM NAME - ${env}`,    project: process.env.RP\_PROJECT || "NAME\_OF YOUR PROJECT ON RP BOARD(YOUR PERSONAL OR THE NAME OF THE TEAM OF YOUR PROJECT – ALL WHAT YOU HAVE AN OPENED ACCESS)",    mode: process.env.RP\_MODE || "DEFAULT",    description: process.env.RP\_DESCRIPTION || `Run ${host} ${env}`,    attachScreenshots: process.env.RP\_SCREENSHOT || true,    showPassedHooks: false,    waitTime: +process.env.RP\_WAITTIME || 30000  }; |

Then, in root folder we need to create folders named – ***testReports*** and ***testResult***.

After that we need to form **conf.js** file. There we put this:

|  |
| --- |
| **require("@babel/register");**  **console.log(`PROCESS CWD ${process.cwd()}`);**  const suites = require('../config/suites/index');  **const {**  **RPLauncher**  **} = require("../tests/utils/rpReporter/RPLauncher");**  exports.config = {    directConnect: true,    capabilities: {      browserName: "chrome",      chromeOptions: {        args: [          // disable chrome's wakiness          "--disable-infobars",          "--window-size=1800,900",          "--disable-extensions",          "verbose",          "log-path=/tmp/chromedriver.log",          "--disable-web-security --user-data-dir",          "--allow-running-insecure-content",          // "--headless",          "--allow-cross-origin-auth-prompt",          "--no-sandbox",          "--disableChecks",        ],        prefs: {          // disable chrome's annoying password manager          "profile.password\_manager\_enabled": false,          credentials\_enable\_service: false,          password\_manager\_enabled: false,        },      },    },  **beforeLaunch: async () => {**  **if (process.env.RP\_ENABLED) {**  **process.env.RP\_LAUNCH\_ID = (await RPLauncher.startRPLaunch()).id;**  **}**  **},**  **afterLaunch: async () => {**  **if (process.env.RP\_ENABLED) {**  **await RPLauncher.closeRPLaunch();**  **}**  **},**    framework: "mocha",    specs: ["../tests/\*.js"],    suites: {      complexAppSuite: './tests/resubmitAppE2E\_ShortApp.spec.js',    },  **mochaOpts: {**  **slow: 3000,**  **reporter: "mocha-multi-reporters",**  **reporterOptions: {**  **reporterEnabled: "./tests/utils/rpReporter/mocha-reportportal-reporter, mochawesome",**  **mochawesomeReporterOptions: {**  **overwrite: false,**  **reportDir: "./testResult",**  **html: false,**  **json: true,**  **inline: true,**  **reportFilename: "mochawesome"**  **}**  **},**  **timeout: 600000,**  **exit: true,**  **force: true,**  **colors: true,**  **verbose: true,**  **},**  }; |

Bold things are obviously relates to RP, but there is no 100% sure that non-bold things are not.

Then, if we have ***tests*** folder in root, we need to create a folder ***utils*** there, and in this ***utils*** folder we need to create a folder ***rpReporter***. After that we need to place there three files:

**File 1. mocha-reportportal-reporter.js**

|  |
| --- |
| /\* eslint-disable indent \*/  /\* eslint-disable no-console \*/  const SpecificUtils = require("./specificUtils");  const RPClient = require("reportportal-client");  const mocha = require("mocha");  const \_ = require("underscore");  const deasync = require("deasync");  const {    config  } = require("../../../config/reportPortalConf");  const STATUS = {    PASSED: "PASSED",    FAILED: "FAILED",    STOPPED: "STOPPED",    SKIPPED: "SKIPPED",    RESETED: "RESETED",    CANCELLED: "CANCELLED"  };  const LEVEL = {    ERROR: "ERROR",    TRACE: "TRACE",    DEBUG: "DEBUG",    INFO: "INFO",    WARN: "WARN",    EMPTY: ""  };  const TYPE = {    SUITE: "SUITE",    STORY: "STORY",    TEST: "TEST",    SCENARIO: "SCENARIO",    STEP: "STEP",    BEFORE\_CLASS: "BEFORE\_CLASS",    BEFORE\_GROUPS: "BEFORE\_GROUPS",    BEFORE\_METHOD: "BEFORE\_METHOD",    BEFORE\_SUITE: "BEFORE\_SUITE",    BEFORE\_TEST: "BEFORE\_TEST",    AFTER\_CLASS: "AFTER\_CLASS",    AFTER\_GROUPS: "AFTER\_GROUPS",    AFTER\_METHOD: "AFTER\_METHOD",    AFTER\_SUITE: "AFTER\_SUITE",    AFTER\_TEST: "AFTER\_TEST"  };  function resolveReportPortalPromise(promise) {    let asyncDone = false;    promise.then(      () => {        asyncDone = true;      },      () => {        asyncDone = true;        console.log("Error occured on resolving Report Portal promise");      },    );    deasync.loopWhile(function () {      return !asyncDone;    });  }  function MochaRPReporter(runner, options) {    let Base = mocha.reporters.Base;    Base.call(this, runner);    // let reporterOpts = options.reporterOptions || {};    // config = Object.assign({    //   attachScreenshots: true,    //   showPassedHooks: false,    //   token: "",    //   endpoint: "",    //   project: "",    //   waitTime: 30000    // }, config);    let enabled;    try {      enabled = \_.has(process.env, "RP\_ENABLED") ? (process.env.RP\_ENABLED === "true") : false;      console.log(enabled);      console.log("RP\_ENABLED: " + process.env.RP\_ENABLED);    } catch (err) {      console.error(`Failed to load config. Error: ${err}`);    }    let client = new RPClient(config);    let tempLaunchId = null;    let launchStatus = STATUS.PASSED;    let suitStatus = STATUS.PASSED;    let testStatus = STATUS.PASSED;    let hookStatus = STATUS.PASSED;    let parentIds = [];    let promise = Promise.resolve(null);    let promiseArr = [];    let finish = false;    this.done = (failures, exit) => {      while (!finish) {        deasync.runLoopOnce();      }      exit && exit(failures > 0 ? 1 : 0);    };    let setLaunchId = (id) => {      tempLaunchId = id;    };    let getParentId = () => {      if (!parentIds.length) {        return null;      }      return parentIds[parentIds.length - 1];    };    let getHookParentId = () => {      if (!parentIds.length) {        return null;      }      return parentIds[parentIds.length - 2];    };    let setParentId = (id) => {      parentIds.push(id);    };    let removeParent = () => {      parentIds.pop();    };    let cutStringLength = (str, length) => {      return str.length > length ? `${str.substring(0, length - 3)}...` : str;    };    let checkNameLength = (name) => {      if (name === "") {        return "Empty IT name!";      } else {        return cutStringLength(name, 256);      }    };    let getHookType = (test) => {      let hType;      let t = test.title;      switch (true) {        case t.startsWith("\"before each\""):          hType = TYPE.BEFORE\_METHOD;          break;        case t.startsWith("\"after each\""):          hType = TYPE.AFTER\_METHOD;          break;        case t.startsWith("\"before all\""):          hType = TYPE.BEFORE\_CLASS;          break;        case t.startsWith("\"after all\""):          hType = TYPE.AFTER\_CLASS;          break;        default:          hType = null;      }      return hType;    };    // let startLaunch = async () => {    //   let rpObj = client.startLaunch({});    //   setLaunchId(rpObj.tempId);    //   promise = rpObj.promise;    // };    let connectRPLaunch = async () => {      // BeforeFeatures      if (!tempLaunchId) {        let rpObj = client.startLaunch({          name: checkNameLength(config.launch),          start\_time: client.helpers.now(),          description: !config.description ? "" : config.description,          id: process.env.RP\_LAUNCH\_ID,        });        setLaunchId(rpObj.tempId);        promise = rpObj.promise;        promiseArr.push(promise);      };    };    let finishLaunch = () => {      const p = Promise.all(promiseArr);      p.then(() => {        setTimeout(() => {          finish = true;        }, config.waitTime);        // client.finishLaunch(tempLaunchId, {        //   status: launchStatus        // }).promise.then(() => {        console.log(`[RP Launch] Finishing`);        finish = true;        const promiseFinishAll = client.getPromiseFinishAllItems(tempLaunchId);        resolveReportPortalPromise(promiseFinishAll);        // }, (err) => {        //   finish = true;        //   console.error(err);        // });      });    };    let startSuit = async (suite) => {      // await promise;      let rpObj = client.startTestItem({        type: TYPE.SUITE,        description: suite.fullTitle(),        name: checkNameLength(suite.title)      }, tempLaunchId, getParentId());      setParentId(rpObj.tempId);      promise = rpObj.promise;      promiseArr.push(promise);    };    let finishSuit = async () => {      // await promise;      let rpObj = client.finishTestItem(getParentId(), {        status: suitStatus      }).promise;      removeParent();      promise = rpObj.promise;      promiseArr.push(promise);    };    let startHook = async (test) => {      // await promise;      let rpObj = client.startTestItem({        type: getHookType(test),        name: checkNameLength(test.title)      }, tempLaunchId, getHookParentId());      setParentId(rpObj.tempId);      promise = rpObj.promise;      promiseArr.push(promise);    };    let startTest = async (test) => {      // await promise;      let rpObj = client.startTestItem({        type: TYPE.STEP,        description: test.fullTitle(),        name: checkNameLength(test.title)      }, tempLaunchId, getParentId());      setParentId(rpObj.tempId);      promise = rpObj.promise;      promiseArr.push(promise);    };    let finishTest = async (test) => {      // await promise;      if (test.log) {        client.sendLog(getParentId(), {          message: test.log,          level: LEVEL.INFO        });      }      let rpObj = client.finishTestItem(getParentId(), {        status: testStatus      });      removeParent();      promise = rpObj.promise;      promiseArr.push(promise);    };    let finishFailedTest = async (test) => {      // await promise;      let parentId = getParentId();      if (config.attachScreenshots) {        promise = SpecificUtils.takeScreenshot(test.fullTitle());      }      promise.then((fileObj) => {        if (fileObj) {          if (fileObj.name) {            fileObj.name = fileObj.name.replace(/\//g, "");          }          client.sendLog(parentId, {            message: test.err.message,            level: LEVEL.ERROR          }, fileObj);        }        if (test.err.actual && test.err.expected) {          client.sendLog(parentId, {            message: "Actual:\n" + test.err.actual + "\nExpected:\n" + test.err.expected,            level: LEVEL.ERROR          });        }        if (test.log) {          client.sendLog(parentId, {            message: test.log,            level: LEVEL.INFO          });        }        client.sendLog(parentId, {          message: test.err.stack,          level: LEVEL.TRACE        });        client.sendLog(parentId, {          message: test.body,          level: LEVEL.TRACE        });        let rpObj = client.finishTestItem(parentId, {          status: STATUS.FAILED        });        promise = rpObj.promise;        promiseArr.push(promise);      });      removeParent();    };    runner.on("start", () => {      if (!enabled) return;      launchStatus = STATUS.PASSED;      try {        connectRPLaunch();      } catch (err) {        console.error("Failed to start launch: ", err);      }    });    runner.once("end", () => {      if (!enabled) return;      try {        finishLaunch();      } catch (err) {        console.error("Failed to finish launch: ", err);      }    });    runner.on("suite", (suite) => {      if (!enabled) return;      if (suite.title !== "") {        suitStatus = STATUS.PASSED;        try {          startSuit(suite);        } catch (err) {          console.error("Failed to start suit: ", err);        }      }    });    runner.on("suite end", (suite) => {      if (!enabled) return;      if (suite.title !== "") {        try {          finishSuit();        } catch (err) {          console.error("Failed to finish suit: ", err);        }      }    });    runner.on("hook", (test) => {      if (!enabled) return;      hookStatus = STATUS.PASSED;      if (test.title !== "\"after each\" hook: ret" && config.showPassedHooks) {        try {          startHook(test);        } catch (err) {          console.error("Failed to start hook: ", err);        }      }    });    runner.on("hook end", (test) => {      if (!enabled) return;      if (test.title !== "\"after each\" hook: ret" && config.showPassedHooks) {        try {          finishTest(test);        } catch (err) {          console.error("Failed to finish hook: ", err);        }      }    });    runner.on("pass", (test) => {      if (!enabled) return;      testStatus = STATUS.PASSED;      try {        startTest(test);        finishTest(test);      } catch (err) {        console.error("Failed to add passed test: ", err);      }    });    runner.on("fail", (test) => {      if (!enabled) return;      testStatus = STATUS.FAILED;      suitStatus = STATUS.FAILED;      launchStatus = STATUS.FAILED;      try {        if (test.type === "hook") {          hookStatus = STATUS.FAILED;          if (!config.showPassedHooks) {            startHook(test);          }          finishFailedTest(test);        } else {          startTest(test);          finishFailedTest(test);        }      } catch (err) {        console.error("Failed to add failed test: ", err);      }    });    runner.on("pending", (test) => {      if (!enabled) return;      testStatus = STATUS.SKIPPED;      try {        startTest(test);        finishTest(test);      } catch (err) {        console.error("Failed to add pending test: ", err);      }    });  }  module.exports = MochaRPReporter; |

**File 2. rpLauncher.js**

|  |
| --- |
| /\* eslint-disable no-console \*/  import ReportPortalClient from "reportportal-client";  import {    config  } from "../../../config/reportPortalConf";  const \_ = require("underscore");  // require('dotenv').config();  const cutStringLength = (str, length) => (str.length > length ? `${str.substring(0, length - 3)}...` : str);    class RPlauncher {    constructor() {      try {        this.enabled = \_.has(process.env, "RP\_ENABLED") ? (process.env.RP\_ENABLED === "true") : false;        console.log(this.enabled);        console.log("RP\_ENABLED: " + process.env.RP\_ENABLED);      } catch (err) {        console.error(`Failed to load config. Error: ${err}`);      }      this.RP = new ReportPortalClient(config);    }    async startRPLaunch() {      if (!this.enabled) return;      // BeforeFeatures      const launch = await this.RP.startLaunch({        name: cutStringLength(config.launch, 256),        start\_time: this.RP.helpers.now(),        description: !config.description ? "" : config.description,        tags: !config.tags ? [] : config.tags,      });      this.tempId = launch.tempId;      return launch.promise;    }    async closeRPLaunch() {      if (!this.enabled) return;      const {        promise      } = this.RP.finishLaunch(this.tempId, {        end\_time: this.RP.helpers.now(),      });      await promise;    }  }  export const RPLauncher = new RPlauncher(); |

**File 3. specificUtils.js**

|  |
| --- |
| /\* eslint-disable no-console \*/  const SpecificUtils = {    takeScreenshot(fileName) {      let promiseResolve;      let promise = new Promise((resolve, reject) => {        promiseResolve = resolve;      });      if (browser) {        browser.takeScreenshot().then((png) => {          promiseResolve({            name: fileName,            type: "image/png",            content: png          });        }, (error) => {          console.dir(error);          promiseResolve(null);        });      }      else {        promiseResolve(null);      }      return promise;    }  };  module.exports = SpecificUtils; |

After all this, you **MUST** include into your runnin script those parameters:

**RP\_ENABLED=true TEST\_HOST=local**

So as a full example, it could be like this:

*"shortApp:dev": "cross-env NODE\_ENV=dev APP\_TYPE=short****RP\_ENABLED=true TEST\_HOST=local****protractor config/conf.js --specs=tests/resubmitAppE2E\_ShortApp.spec.js",*

So, after we place all those files and create all necessary environment, we shall run our tests. When we do this,

In those board of RP what we have set in our configuration we can see that our tests runnin. There is no needs to do any other actions for that. If all those code and dependencies will set up correctly, the results will grow up into your RP board.