Automated Test Cucumber

Geautomatiseerd regressietest framework voor Team Stark

H.H. Aslan
03-04-2019

Inhoudsopgave

Keuzes

Hieronder een toelichting van de verschillende keuzes die gemaakt zijn.

Cucumber

De keuze om met “Cucumber” is gevallen omdat we hierdoor meerdere lagen van het ontwikkelteam kunnen aanspreken. Door gebruik te maken van feature files kunnen we onze testscenario’s duidelijk aan de business uitleggen. De stepdefinitions geven aan welke stappen er worden door lopen tijdens deze scenario’s, deze stepdefinitions kunnen gemakkelijk door zowel de tester, als de analisten worden opgezet. De derde laag bestaat uit de implementatie van de methodes die tijdens stepdefinitions worden gebruikt.

Protractor

Protractor is het onderliggende framework waarmee de verschillende testen kunnen worden gestart. Deze is natuurlijk is zeer geschikt, om te kunnen gaan met Angular applicaties.

Javascript/Typescript

De code die gebruikt wordt om de bovenliggende keuzes soepel met elkaar te laten werken is Javascript en typescript. Hiervoor is gekozen omdat de kennis binnen het team aanwezig is en de huidige applicaties in dezelfde taal zijn ontwikkeld.

Framework

Hieronder uitleg hoe je het framework kunt gebruiken, en hoe het framework is opgezet.

Cucumber

Cucumber bestaat uit drie lagen, voordat je deze lagen doorneemt een korte introductie over het opstellen van scenario's:

Opstellen scenario's

Scenario's opstellen voor cucumber heeft te maken met het gedrag die door de eindgebruiker word uitgevoerd. Dit gedrag wordt vertaald naar Gherkin taal wat bestaat uit given, when ,then.

Given (gegeven):
Dit is de plek waar je op dat moment begeeft. Bijvoorbeeld:
Gegeven ik op de home pagina ben.

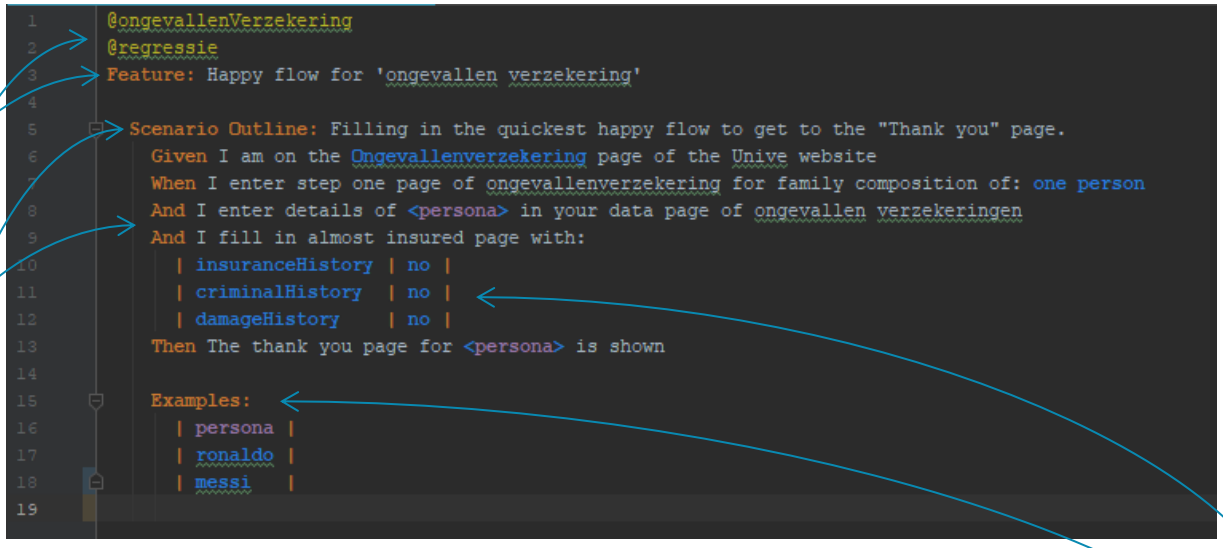
When (als):
Wanneer ik een actie uitvoer. Bijvoorbeeld:
Als ik klik op zorgverzekering.

Then (dan):
De verificatie en de daadwerkelijke test wordt in deze stap uitgevoerd.
Dan zie ik de titel Zorgverzekering op de zorg pagina.

Feature file

Feature files worden opgezet door gherkin language, wat herkenbaar is aan hun Given, When, Then opzet.

Voorbeeld:



```
@ongevallenVerzekering
@regressie
Feature: Happy flow for 'ongevallen verzekering'

Scenario Outline: Filling in the quickest happy flow to get to the "Thank you" page.
    Given I am on the Ongevallenverzekering page of the Unive website
    When I enter step one page of ongevallenverzekering for family composition of: one person
    And I enter details of <persona> in your data page of ongevallen verzekeringen
    And I fill in almost insured page with:
      | insuranceHistory | no |
      | criminalHistory  | no |
      | damageHistory    | no |
    Then The thank you page for <persona> is shown

    Examples:
      | persona |
      | ronaldo |
      | messi   |
```

Dit zijn tags die worden aangegeven met een @ door deze tekst in je testen te plaatsen kun je ze specifiëren en uiteindelijk aanroepen bijvoorbeeld in je pipeline.

De feature staat voor de omschrijving van de testen die je uitvoert, deze tekst komt uiteindelijk terug in je rapportage.

“Scenario Outline:” Hierin wordt beschreven wat de scenario doet, je kunt ook “Scenario” gebruiken maar dan kun je geen examples tabel plaatsen.

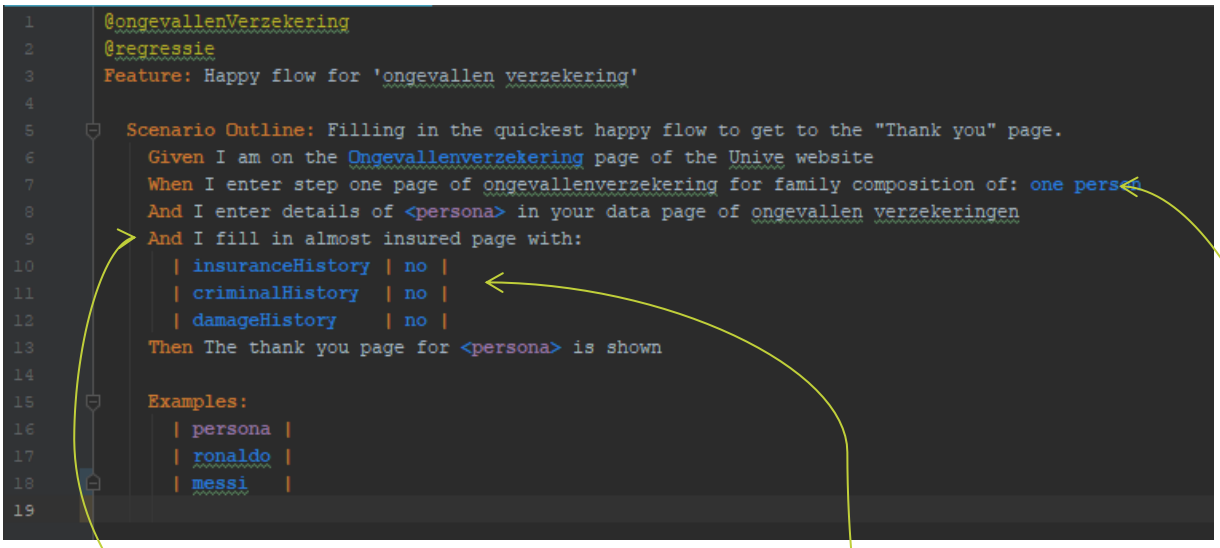
Als je extra handelingen wilt uitvoeren kun je deze altijd er naast beschrijven in “And”.

Datatable: in een datatable kun je voor een regel extra variabelen opgeven, deze variabelen kunnen uiteindelijk in de steps worden meegenomen.

Examples: in de examples is de bovenste regel wat in de voorbeeld als persona word beschreven wordt per regel uitgevoerd als test. Dat houd in dat de volledige test één keer voor persona: ronaldo wordt uitgevoerd en daarna volledig opnieuw voor persona messi.

Stepdefinitions

Step definitions wordt ook wel de gluecode genoemd. Dit houd in dat de scenario's in de feature files worden gecombineerd met de implementatie.



```
@ongevallenVerzekering
@regressie
Feature: Happy flow for 'ongevallen verzekering'

  Scenario Outline: Filling in the quickest happy flow to get to the "Thank you" page.
    Given I am on the Ongevallenverzekering page of the Unive website
    When I enter step one page of ongevallenverzekering for family composition of: one perso
    And I enter details of <persona> in your data page of ongevallen verzekeringen
    And I fill in almost insured page with:
      | insuranceHistory | no |
      | criminalHistory  | no |
      | damageHistory    | no |
    Then The thank you page for <persona> is shown

    Examples:
      | persona |
      | ronaldo |
      | messi   |
```

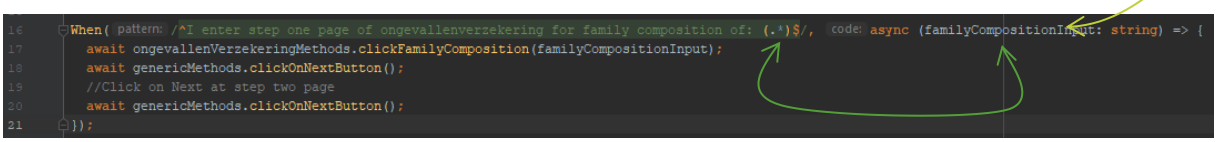
```
When( pattern: /^I fill in almost insured page with:$/, code: async (data) => {
  const dataTable = data.rowsHash();
  await genericMethods.selectInsuranceHistory(dataTable.insuranceHistory, genericEnum.EMPTY);
  await genericMethods.selectCriminalHistory(dataTable.criminalHistory);
  await genericMethods.selectDamageHistory(dataTable.damageHistory);
  await genericMethods.clickOnFinishButton();
});
```

De stepdefinitions zijn opgezet zie bovenstaand voorbeeld. Hier zie je de tekst van de feature file terugkomen. Op deze manier herkent de code van cucumber dat die deze stappen moet uitvoeren bij het tegenkomen van die tekst.

Hierbij is gekozen voor een voorbeeld met een datatable. Je ziet dat de datatable word gedefinieerd in de stepdefinition door gebruik te maken van data en data.rowsHash dit doet niks anders dan per regel de variabele opslaan.

"insuranceHistory" is de naam van de variabele en "no" de waarde. De waarde kan je weer uitlezen voor de methodes waar je het gaat gebruiken: dataTable.* bij sterretje zet je de naam van de variabele die je wilt gebruiken.

De stepdefinitions voer je voor elke feature file uit en elke regel van je Given, when and then. Hieronder nog een voorbeeld met één variabele: één variabele kun je aangeven met: (.*)



```
When( pattern: /^I enter step one page of ongevallenverzekering for family composition of: (.*)$/, code: async (familyCompositionInput: string) => {
  await ongevallenVerzekeringMethods.clickFamilyComposition(familyCompositionInput);
  await genericMethods.clickOnNextButton();
  //Click on Next at step two page
  await genericMethods.clickOnNextButton();
});
```

Implementatie

Onder de steps word de daadwerkelijke implementatie gecodeerd. Deze code is geschreven in typescript. Door gebruik te maken van selenium kunnen de verschillende element worden herkend. Zie onderstaand voorbeeld:

```
async clickOnElement(selector: string) {
  await this.waitForElementNotVisible(genericElements.loader, browser.getPageTimeout);
  await this.waitForElementIsVisible(selector, browser.getPageTimeout);
  const elementToClick: ElementFinder = element(by.css(selector));
  await browser.controlFlow().execute( command: () => {
    browser.executeScript('arguments[0].scrollIntoView({block: \'center\'})', elementToClick);
  });
  await browser.wait((ec.elementToBeClickable(elementToClick)), browser.getPageTimeout).then(() => {
    elementToClick.click();
  })
}
```

Pageobjects

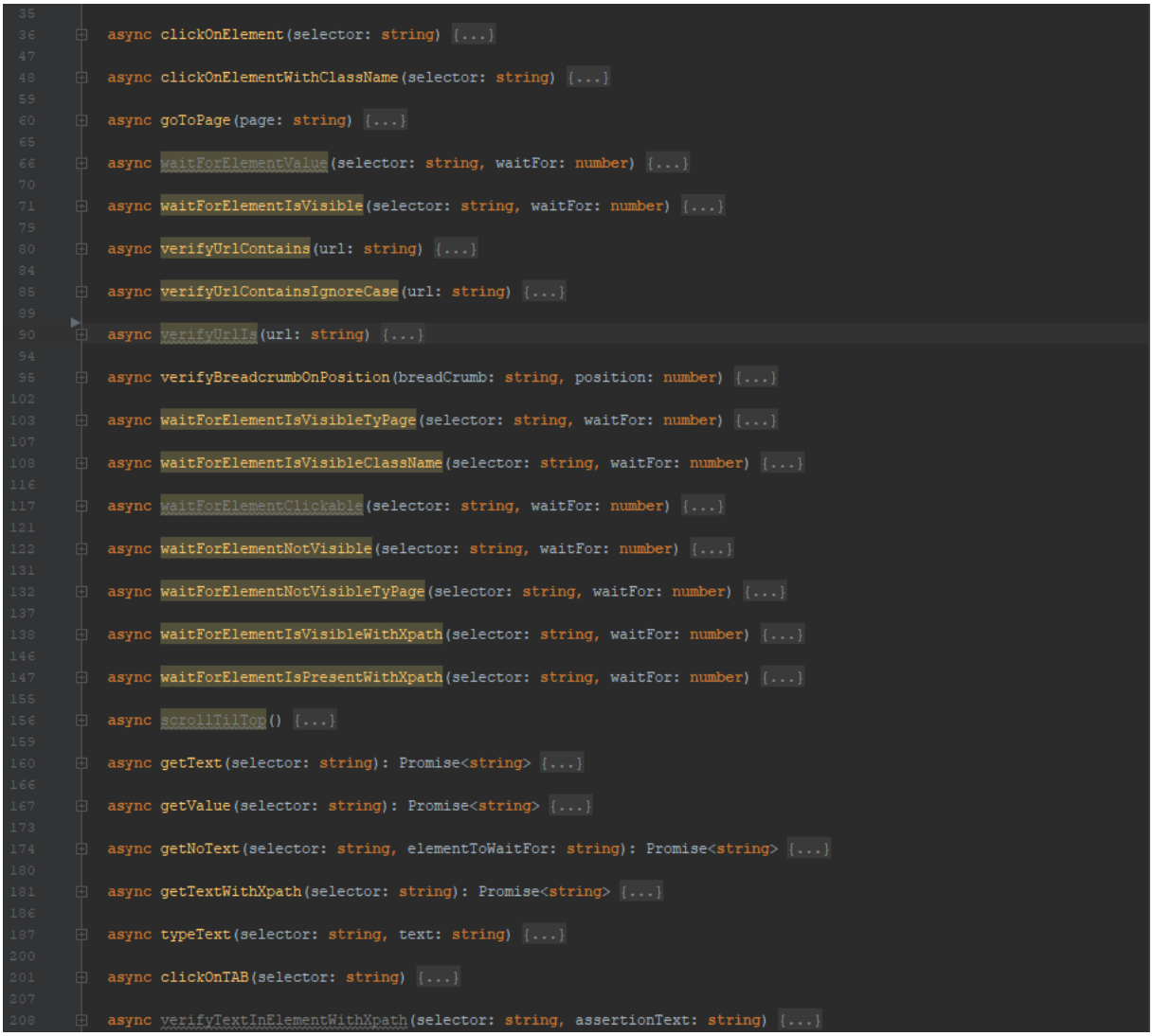
Pageobjects zijn de pagina's waar de verschillende elementen worden gedefinieerd. Deze elementen kunnen dan uiteindelijk via de methodes worden aangesproken om er op te klikken of om mee te verifiëren. Voorbeeld:

```
//HISTORY
insuranceHistoryNoElement: string = '[data-label-id*="LA_IF2535_3710"] .radioList > label:nth-child(1)';
insuranceHistoryYesElement: string = '[data-label-id*="LA_IF2535_3710"] .radioList > label:nth-child(2)';
insuranceHistoryYesExplanationElement: string = '#_Form_IF2535_33524';

criminalHistoryYesElement: string = '[data-label-id*="LA_IF2535_3711"] .radioList > label:nth-child(2)';
criminalHistoryNoElement: string = '[data-label-id*="LA_IF2535_3711"] .radioList > label:nth-child(1)';

damageHistoryYesElement: string = '[data-label-id*="LA_IF2535_33953"] .radioList > label:nth-child(2)';
damageHistoryNoElement: string = '[data-label-id*="LA_IF2535_33953"] .radioList > label:nth-child(1)';

//SIDEBAR
sideBarElement: string = '//*[@class="rbcontainer"]//*';
```

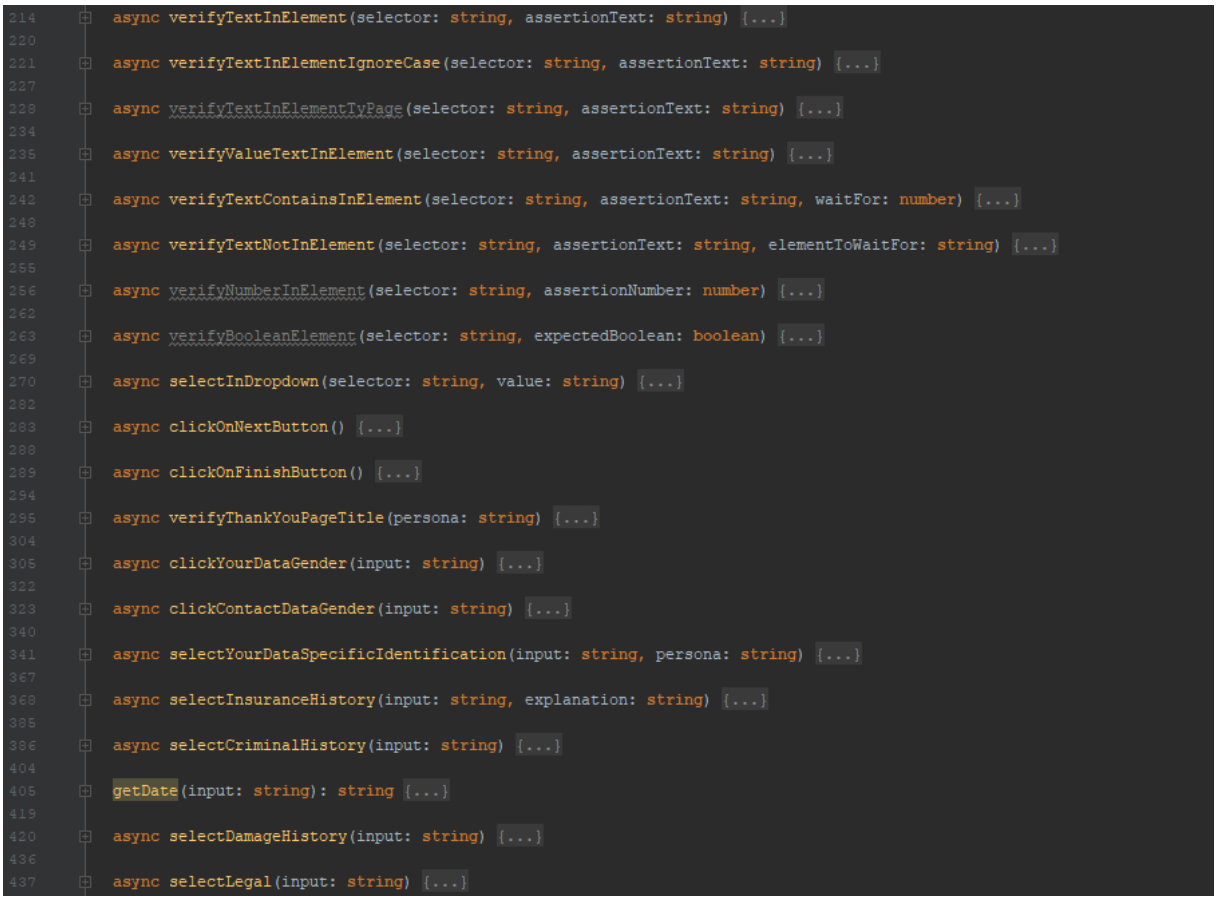
Methodes

Via de steps kunnen verschillende methodes worden aangeroepen.

Genericmethods



```
async clickOnElement(selector: string) {...}

async clickOnElementWithClassName(selector: string) {...}

async goToPage(page: string) {...}

async waitForElementValue(selector: string, waitFor: number) {...}

async waitForElementIsVisible(selector: string, waitFor: number) {...}

async verifyUrlContains(url: string) {...}

async verifyUrlContainsIgnoreCase(url: string) {...}

async verifyUrlIs(url: string) {...}

async verifyBreadcrumbOnPosition(breadCrumb: string, position: number) {...}

async waitForElementIsVisibleTyPage(selector: string, waitFor: number) {...}

async waitForElementIsVisibleClassName(selector: string, waitFor: number) {...}

async waitForElementClickable(selector: string, waitFor: number) {...}

async waitForElementNotVisible(selector: string, waitFor: number) {...}

async waitForElementNotVisibleTyPage(selector: string, waitFor: number) {...}

async waitForElementIsVisibleWithXpath(selector: string, waitFor: number) {...}

async waitForElementIsPresentWithXpath(selector: string, waitFor: number) {...}

async scrollTilTop() {...}

async getText(selector: string): Promise<string> {...}

async getValue(selector: string): Promise<string> {...}

async getNoText(selector: string, elementToWaitFor: string): Promise<string> {...}

async getTextWithXpath(selector: string): Promise<string> {...}

async typeText(selector: string, text: string) {...}

async clickOnTAB(selector: string) {...}

async verifyTextInElementWithXpath(selector: string, assertionText: string) {...}
```



Deze methoden kunnen als volgt worden aangeroepen:

await genericMethods.clickOnNextButton();

sommige van deze methoden hebben extra input nodig, voorbeeld van de meest gebruikte methoden:

await genericMethods.clickOnElement(“element met css slector”);

await genericMethods.waitForElementIsVisible(“element met css selector”, “milliseconden voor wachten”);

Als je wilt dat waitFor de standaard gebruikt van 60000 kun je dat aangeven door gebruik te maken van browser.getPageTimeout

Persona

In de code zijn persona's voor gedefinieerd. Een lijst met persona's:

```
export class PersonaData {

  ronaldo: Persona = new Persona( firstName: 'Cristiano',  initials: 'C', prefix.NO_PREFIX,  lastName: 'Ronaldo',  birthday: '10-05-1987',  birthplace: 'Deventer',  zipcode: '7412XW',  houseNumber: '91',
    houseNumberAddition.NO_HOUSENUMBER_ADDITION, gender.MALE,  phoneNumber: '0601234587', specificIdentification.PASSPORT,  specificIdentificationNumber: 'ACP26N',  eMailAddress: 'ronaldo@unive.nl',
    accountNumber: 'NL05INGB0661095088',  bsn: '218333754',  profession: 'model',  kvkNumber: '1234567890', durationEntrepreneur._6,  height: '185',  weight: '85');
  messi: Persona = new Persona( firstName: 'Lionel',  initials: 'L', prefix.NO_PREFIX,  lastName: 'Messi',  birthday: '05-09-1965',  birthplace: 'Apeldoorn',  zipcode: '7412TV',  houseNumber: '44',
    houseNumberAddition.NO_HOUSENUMBER_ADDITION, gender.FEMALE, phoneNumber.NO_PHONENUMBER, specificIdentification.DRIVER_LICENSE,  specificIdentificationNumber: '12345567890',
    eMailAddress: 'messi@unive.nl',  accountNumber: 'NL05INGB0661095088',  bsn: '218333754',  profession: 'footballer',  kvkNumber: '1234567890', durationEntrepreneur._9,  height: '176',  weight: '76');
  salah: Persona = new Persona( firstName: 'Mohammed',  initials: 'M', prefix.NO_PREFIX,  lastName: 'Saleh',  birthday: '15-06-1992',  birthplace: 'Enschede',  zipcode: '7412XW',  houseNumber: '31',
    houseNumberAddition: 'A', gender.MALE,  phoneNumber: '1234567890', specificIdentification.ID_CARD,  specificIdentificationNumber: 'HDNG68A',  eMailAddress: 'salah@unive.nl',
    accountNumber: 'NL05INGB0661095088',  bsn: '218333754',  profession: 'beast',  kvkNumber: '1234567890', durationEntrepreneur._3,  height: '167',  weight: '50');
  neymar: Persona = new Persona( firstName: 'Neymar',  initials: 'N', prefix: 'da', lastName: 'Silva',  birthday: '05-02-1992',  birthplace: 'Arnhem',  zipcode: '7412XW',  houseNumber: '12',
    houseNumberAddition: 'ABC', gender.FEMALE, phoneNumber.NO_PHONENUMBER, specificIdentification.SOMETHING_ELSE, specificIdentification.NO_NUMBER,  eMailAddress: 'neymar@unive.nl',
    accountNumber: 'NL05INGB0661095088',  bsn: '218333754',  profession: 'diver',  kvkNumber: '1234567890', durationEntrepreneur._7,  height: '190',  weight: '98');
  pogba: Persona = new Persona( firstName: 'Paul',  initials: 'PL', prefix.NO_PREFIX,  lastName: 'Pogba',  birthday: '15-03-1993',  birthplace: 'Zwolle',  zipcode: '7412TV',  houseNumber: '69',
    houseNumberAddition.NO_HOUSENUMBER_ADDITION, gender.MALE,  phoneNumber: '1234567890', specificIdentification.ID_CARD,  specificIdentificationNumber: '897623',  eMailAddress: 'pogba@unive.nl',
    accountNumber: 'NL05INGB0661095088',  bsn: '218333754',  profession: 'beast',  kvkNumber: '1234567890', durationEntrepreneur._MIN1,  height: '209',  weight: '110');

```

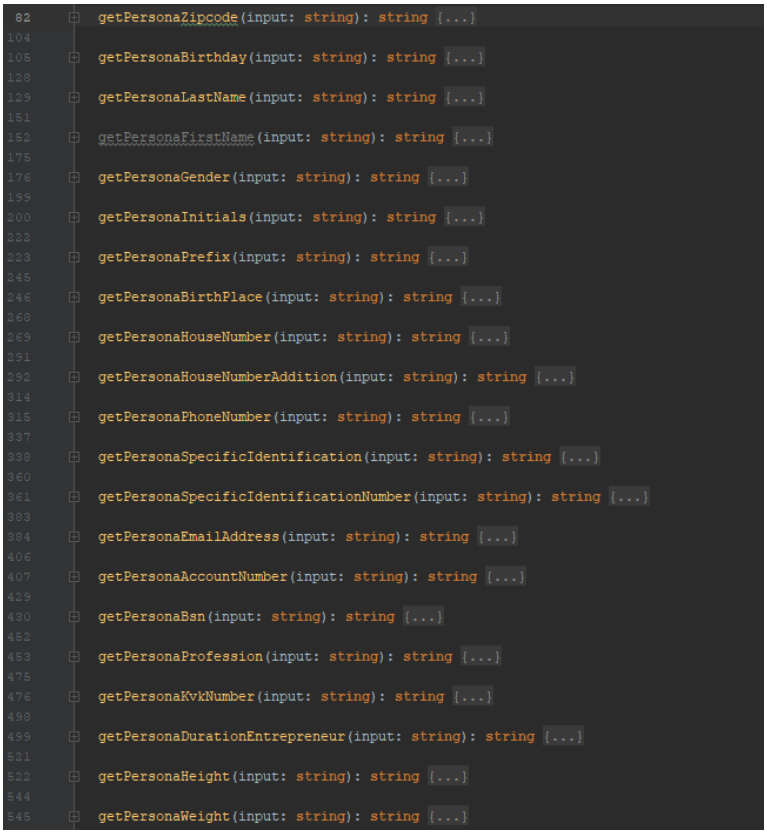
Gebruiken: import PersonaData:

```
import {PersonaData} from "../persona/persona";
```

Definieer personaData:

```
let personaData: PersonaData = new PersonaData();
```

Nu kun je gebruik maken van alle personaData die voor zijn gegeneerd:



```
getPersonaZipcode(input: string): string {...}

getPersonaBirthday(input: string): string {...}

getPersonaLastName(input: string): string {...}

getPersonaFirstName(input: string): string {...}

getPersonaGender(input: string): string {...}

getPersonaInitials(input: string): string {...}

getPersonaPrefix(input: string): string {...}

getPersonaBirthPlace(input: string): string {...}

getPersonaHouseNumber(input: string): string {...}

getPersonaHouseNumberAddition(input: string): string {...}

getPersonaPhoneNumber(input: string): string {...}

getPersonaSpecificIdentification(input: string): string {...}

getPersonaSpecificIdentificationNumber(input: string): string {...}

getPersonaEmailAddress(input: string): string {...}

getPersonaAccountNumber(input: string): string {...}

getPersonaBsn(input: string): string {...}

getPersonaProfession(input: string): string {...}

getPersonaKvkNumber(input: string): string {...}

getPersonaDurationEntrepreneur(input: string): string {...}

getPersonaHeight(input: string): string {...}

getPersonaWeight(input: string): string {...}
```

Je kunt gebruik maken van deze data door de volgende commando te gebruiken:
personaData.getPersonaLastName("ronaldo"))

Voertuigen

Op dezelfde manier zijn er ook standaard voertuigen gedefinieerd. Hieronder de lijst met voertuigen.

Boot

```
export class BoatWithName {

  azzam: Boat = new Boat(boatNamesEnum.AZZAM, boatNamesEnum.AZZAM, typeBoatEnum.SLOEP, constructionYear: '2018', materialEnum.KUNSTSTOF_POLYESTER, motorBrandName: 'Kawasaki', motorNumber: '12345',
    motorConstructionYear: '2018', motorPurchaseYear: '2018', fuelTypEnum.DIESEL, power: '100', value: '15000', length: '15');

  getBoatName(input: string): string {...}

  getBoatBrandName(input: string): string {...}

  getBoatType(input: string): string {...}

  getBoatConstructionYear(input: string): string {...}

  getBoatMaterial(input: string): string {...}

  getBoatMotorBrandName(input: string): string {...}

  getBoatMotorNumber(input: string): string {...}

  getBoatMotorConstructionYear(input: string): string {...}

  getBoatMotorPurchaseYear(input: string): string {...}

  getBoatFuelType(input: string): string {...}

  getBoatPower(input: string): string {...}

  getBoatValue(input: string): string {...}

  getBoatLength(input: string): string {...}
```

Zakelijke auto

```
export class BusinessCarWithLicensePlate {

  BUSINESS_CAR_48VDS3: BusinessCar = new BusinessCar(licensePlates.BUSINESS_CAR_48VDS3, brandName: 'Fiat', brandType: 'DUCATO', constructionYear: '2007', model: '35L 3.0 MJ MH2 GV',
    bodyType: "Bestelauto", fuelType: 'Diesel', firstAdmission: '12-09-2001', reportingCode: '0000', weight: '1496');

  getCarFuelType(input: string): string {...}

  getCarBodyType(input: string): string {...}

  getCarModel(input: string): string {...}

  getCarConstructionYear(input: string): string {...}

  getCarBrandType(input: string): string {...}

  getCarBrandName(input: string): string {...}

  getCarFirstAdmission(input: string): string {...}

  getCarReportingCode(input: string): string {...}

  getCarWeight(input: string): string {...}
```

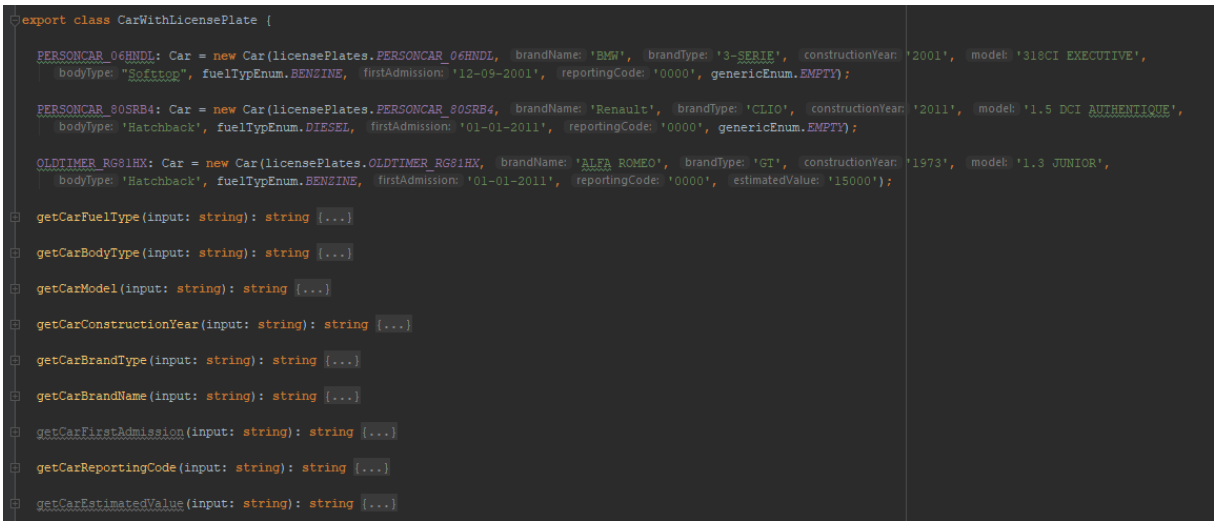
Camper

```
export class CamperWithLicensePlate {

  CAMPER_BLJL41: Camper = new Camper(licensePlates.CAMPER_BLJL41, brandName: 'SPARTAN', brandModel: 'CRUISE-MASTER', newPrice: '15000', meldCode: '0000');


  getCamperBrandName(input: string): string {...}

  getCamperBrandModel(input: string): string {...}

  getCamperNewPrice(input: string): string {...}

  getCamperMeldCode(input: string): string {...}
```

Bromfiets

```
export class MopedWithLicensePlate {

  MOPED_12FRP3: Moped = new Moped(licensePlates.MOPED_12FRP3, brandName: 'RIEJU', vehicleKindEnum.BROMFIETS, constructionYear: '2005', version: 'RR', );
  MOPED_F169NS: Moped = new Moped(licensePlates.MOPED_F169NS, brandName: 'VESPA', vehicleKindEnum.SNORSCOOTER, constructionYear: '2013', version: 'SPRINT S 4T', );


  getMopedBrandName(input: string): string {...}

  getMopedModel(input: string): string {...}

  getMopedConstructionYear(input: string): string {...}

  getMopedVersion(input: string): string {...}
```

Personenauto



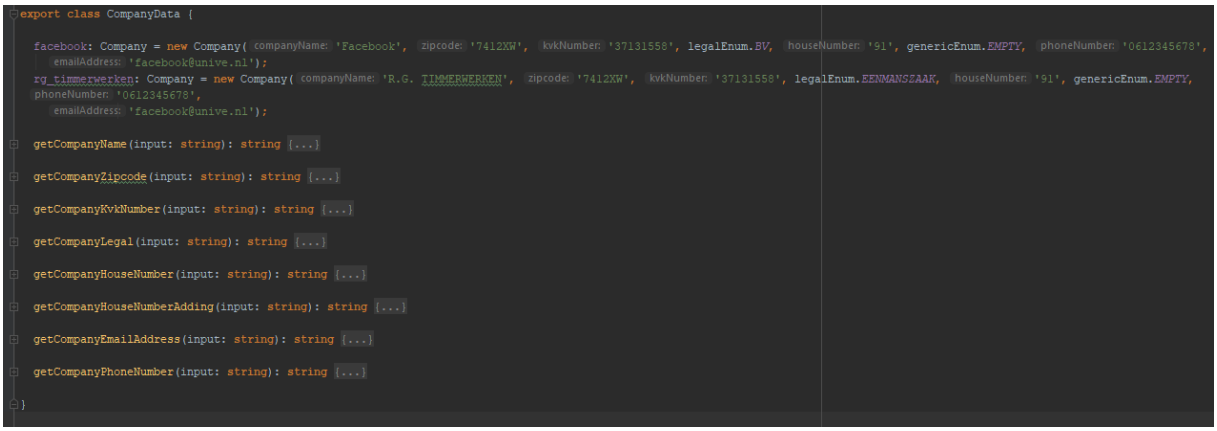
```
export class CarWithLicensePlate {

  PERSONCAR_06HNDL: Car = new Car(licensePlates.PERSONCAR_06HNDL,  brandName: 'BMW',  brandType: '3-SERIE',  constructionYear: '2001',  model: '318CI EXECUTIVE',
    bodyType: "Softtop", fuelTypEnum.BENZINE,  firstAdmission: '12-09-2001',  reportingCode: '0000', genericEnum.EMPTY);

  PERSONCAR_80SRB4: Car = new Car(licensePlates.PERSONCAR_80SRB4,  brandName: 'Renault',  brandType: 'CLIO',  constructionYear: '2011',  model: '1.5 DCI AUTHENTIQUE',
    bodyType: 'Hatchback', fuelTypEnum.DIESEL,  firstAdmission: '01-01-2011',  reportingCode: '0000', genericEnum.EMPTY);

  OLDTIMER_RG81HX: Car = new Car(licensePlates.OLDTIMER_RG81HX,  brandName: 'ALFA ROMEO',  brandType: 'GT',  constructionYear: '1973',  model: '1.3 JUNIOR',
    bodyType: 'Hatchback', fuelTypEnum.BENZINE,  firstAdmission: '01-01-2011',  reportingCode: '0000',  estimatedValue: '15000');

  getCarFuelType(input: string): string {...}

  getCarBodyType(input: string): string {...}

  getCarModel(input: string): string {...}

  getCarConstructionYear(input: string): string {...}

  getCarBrandType(input: string): string {...}

  getCarBrandName(input: string): string {...}

  getCarFirstAdmission(input: string): string {...}

  getCarReportingCode(input: string): string {...}

  getCarEstimatedValue(input: string): string {...}
```

Motor

```
export class MotorWithLicensePlate {

  MOTOR_MPTT99: Motor = new Motor(licensePlates.MOTOR_MPTT99,  brandName: 'SUZUKI', vehicleKindEnum.MOTOR,  constructionYear: '2004',  version: 'GSX R 750',  price: '2999');
  QUAD_97XFTK: Motor = new Motor(licensePlates.QUAD_97XFTK,  brandName: 'hsun', vehicleKindEnum.QUAD,  constructionYear: '2007',  version: 'HS150S',  price: '3999');
  TRIKE_21PLN1: Motor = new Motor(licensePlates.TRIKE_21PLN1,  brandName: 'courage trike', vehicleKindEnum.TRIKE,  constructionYear: '2011',  version: 'CT100TDI TIP',  price: '4960');

  getMotorBrandName(input: string): string {...}

  getMotorModel(input: string): string {...}

  getMotorConstructionYear(input: string): string {...}

  getMotorVersion(input: string): string {...}

  getMotorPrice(input: string): string {...}

}
```

Bedrijven

Ook voor bedrijven zijn ze voor gedefinieerd.



```
export class CompanyData {

  facebook: Company = new Company( companyName: 'Facebook',  zipcode: '7412XW',  kvkNumber: '37131558', legalEnum.BV,  houseNumber: '91', genericEnum.EMPTY,  phoneNumber: '0612345678',
    emailAddress: 'facebook@unive.nl');
  rg_timmerwerken: Company = new Company( companyName: 'R.G. TIMMERWERKEN',  zipcode: '7412XW',  kvkNumber: '37131558', legalEnum.EENMANSZAAK,  houseNumber: '91', genericEnum.EMPTY,
  phoneNumber: '0612345678',
    emailAddress: 'facebook@unive.nl');

  getCompanyName(input: string): string {...}

  getCompanyZipcode(input: string): string {...}

  getCompanyKvkNumber(input: string): string {...}

  getCompanyLegal(input: string): string {...}

  getCompanyHouseNumber(input: string): string {...}

  getCompanyHouseNumberAdding(input: string): string {...}

  getCompanyEmailAddress(input: string): string {...}

  getCompanyPhoneNumber(input: string): string {...}

}
```

Enum

Enum is kort voor enumeration, is een variabel type. Enums gebruiken wij om duidelijk te maken welke datatype er wordt gebruikt en dat het op één plek word beheerd.

Bijvoorbeeld:

```
export enum typeBoatEnum {
  MOTORBOOT = 'motorboot',
  ZEILBOOT = 'zeilboot',
  SLOEP = 'sloep',
  RUBBERBOOT = 'rubberboot',
  ROEIBOOT_KANO = 'roeiboot kano',
  SPEEDBOOT = 'speedboot',
  CATAMARAN = 'catamaran'
}

export enum boatNamesEnum {
  AZZAM = 'azzam'
}
```

Gebruik is als volgt, je definieert de enum zoals hierboven. Daarna kun je de type opgeven ‘typeBoatEnum’ door een punt achter te zetten krijg je de verschillende mogelijkheden.

typeBoatEnum.MOTORBOOT

Als je bovenstaande enum gebruikt wordt er motorboot ingevuld overal waar deze enum wordt aangesproken. Hierdoor hoef je als je de tekst van MOTORBOOT wilt wijzigen op één plek te veranderen. Dit maakt het overzichtelijker en beheerbaar.

Error afhandeling

Er word op twee manieren gebruik gemaakt van Error afhandeling:

1. Try en catch functionaliteit. Dit houd in dat er een actie word uitgevoerd als deze actie een exception geeft komt die in de Catch terecht die uiteindelijk een Error throwth met een duidelijke message en de test laat falen.

```
async waitForElementIsVisible(selector: string, waitFor: number) {
  try {
    const selectorToWaitFor: ElementFinder = element(by.css(selector));
    await browser.wait(ec.visibilityOf(selectorToWaitFor), waitFor);
  } catch (e) {
    throw new Error( message: 'Element with selector: ' + selector + ', is not visible');
  }
}
```

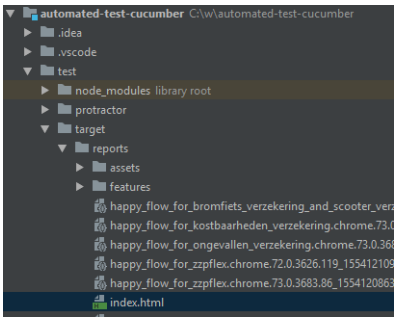
2. In de switch methode worden er verschillende cases doorlopen, als uiteindelijk de input niet voldoet aan wat er in de cases staan komt die switch functie uiteindelijk in de default terecht die een error throwth met een duidelijke message.

   Onderstaand voorbeeld geeft weer dat er twee cases zijn in de switch functie. YES en NO, als je een andere input levert voor deze methode dan YES en NO zal deze case uiteindelijk in de default terecht komen met het bericht dat de input die je ingevoerd hebt voor de methode niet wordt herkend.

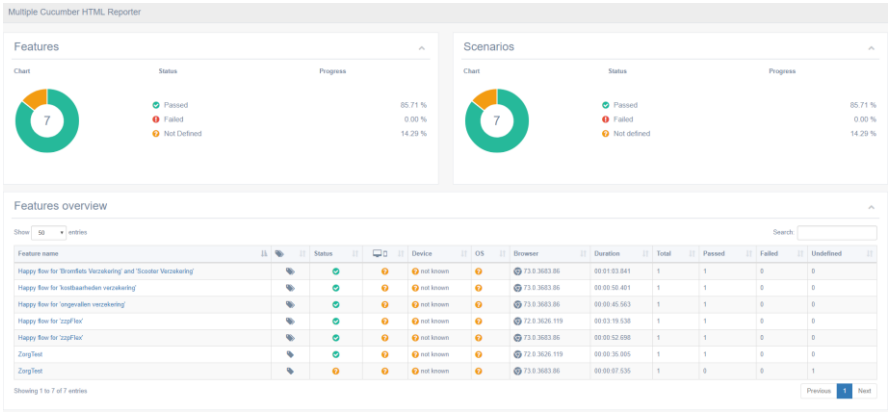
```
async selectCriminalHistory(input: string) {
  await this.waitForElementIsVisible(genericElements.criminalHistoryNoElement, browser.getPageTimeout);
  switch (input) {
    case genericEnum.YES: {
      await this.clickOnElement(genericElements.criminalHistoryYesElement);
      await this.waitForElementIsVisible(genericElements.lightBoxClickElement, browser.getPageTimeout);
      await this.clickOnElement(genericElements.lightBoxClickElement);
      break;
    }
    case genericEnum.NO: {
      await this.clickOnElement(genericElements.criminalHistoryNoElement);
      break;
    }
    default: {
      throw new Error( message: 'The input: "" ' + input + ' ""  you have entered for "" ' + this.constructor.name + ' "" is not recognized as a command');
    }
  }
}
```

Rapportage

Elke keer nadat je een test heb gedraaid wordt er een rapport gegenereerd. Deze kun je vinden in de target folder.

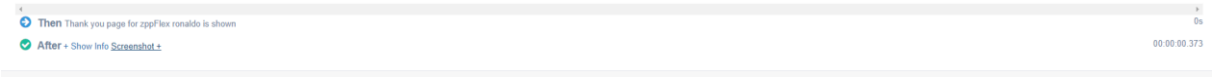


Door de index.html te openen in een browser krijg je het rapport tevoorschijn.



Als er iets fout gaat wordt er standaard een screenshot gemaakt en de foutmelding getoond:

Hier zien we de error terugkomen die in de switch statement is gedefinieerd. Bij after zien we een knop met screenshot door hierop te klikken kan de screenshot worden geopend op het moment dat de test fout is gegaan. Dit is handig voor debug redenen om achter te halen wat er fout is gegaan.



Config

In de config file is gedefinieerd waar het gestart moet worden en hoe, in een hoofdstuk verder kun je lezen hoe je deze bestanden direct kunt aanspreken.

Suites: deze kun je hier definieren en aanvullen.

Headless: om de testen te draaien zonder opstarten van een browser. Aanzette door commentaar te verwijderen.

Selenium server: de testen draaien op selenium server hiervoor moet je wel directconnect uit commentariëren.

Handleiding

Voor installatie

Onderstaande programma's zijn nodig op je computer voordat je het framework lokaal kunt gebruiken:

- Node.js - > https://nodejs.org/en/
  Dit is nodig om protractor en alle node elementen op te halen.
- Git -> https://git-scm.com/downloads
  Dit is nodig om de repository op te halen.
- IDE -> bijvoorbeeld: https://www.jetbrains.com/idea/download/#section=windows of https://www.jetbrains.com/idea/download/#section=windows
  Dit is nodig om de testen te editen, en het overzichtelijker te maken. IntelliJ is een betaalde tool (voorkeur). Visual code is een gratis tool dit zijn de twee grote spelers ook kun je natuurlijk andere editors gebruiken naar wens.

Opzetten

Als eerste moet je het project uitchecken met GIT.
https://code.do.unive.nl/test/automated-test-cucumber
Op bovenstaande pagina kun je de repository vinden van de testen. Via Git kun je deze uitchecken, als je hiervoor een applicatie nodig hebt zou je SourceTree kunnen gebruiken:

- https://www.sourcetreeapp.com/

Nadat je deze repository hebt uitgecheckt kun je een terminal openen. In deze terminal voer je de volgende handelingen uit:

- Ga naar de directory "/test"
  cd ./test
- Voer de volgende commando uit om alle Node elementen te installeren.
  npm install
- Voer de volgende commando uit om alle webdriver elementen te updaten:
  node node_modules/protractor/bin/webdriver-manager update

Nu ben je klaar om nieuwe testen te schrijven en testen te kunnen starten.

Nieuwe test schrijven

Om tot een volledig en goede test te komen is onderstaande stappenplan geschreven.

Stap 1

Schrijf de feature file uit, bepaal hier de given, when then scenario's. Met bijbehorende tags data tabellen en Examples.

Plaats deze feature file in de juiste map, is het een aanvulling op een bestaande testgeval zet het in de juiste bestaande feature file.

Stap 2

Definieer de step definitions, maak een nieuwe bestand aan waar nodig. Nadat je de stepdefinitions heb gedefinieerd kun je deze natuurlijk invullen.

Voorbeeld:

```
When( pattern: /^I enter step one page of ongevallenverzekering for family composition of: (.*)$/, code: async (familyCompositionInput: string) => {
  await ongevallenVerzekeringMethods.clickFamilyComposition(familyCompositionInput);
  await genericMethods.clickOnNextButton();
  //Click on Next at step two page
  await genericMethods.clickOnNextButton();
});
```

Gebruik in eerste instantie zoveel mogelijk de genericMethods. Heb je meerdere gevallen voor je test schrijf dan een nieuwe methode met de waarde in de switch case.

Stap 3

Run de test per regel kijk of die doet wat die moet doen, nadat dit is gedaan ga door met de volgende step. De einde is de verificatie, voer verkeerde gegevens in en controleer dat het fout gaat.

Testen starten

Hoe de testen gestart kunnen worden komen ook terug in de Readme.md binnen het project.

Start alles

protractor protractor.conf.js

Specifier met tags

protractor protractor.conf.js --cucumberOpts.tags="@zzpFlex"

Suites starten

protractor protractor.conf.js --suite zzpFlex

Omgeving wijzigen

protractor protractor.conf.js --cucumberOpts.tags="@zzpFlex" --params.env.environment=tstProj

Regressie set starten

Dit zorgt dat de testen headless worden gedraaid en met 6 instanties tegelijkertijd. Houd er rekening mee als je deze set draait lokaal op je computer dat je tijdens het testen je machine niet tot nauwelijk kunt gebruiken.

protractor protractor.regression.conf.js

Verschillende browsers

Om je regressie set voor Microsoft Edge te starten is het nodig om deze webdriver eerst lokaal te starten. Dat kun je doen met de volgende commando.

node node_modules/protractor/bin/webdriver-manager start --edge "MicrosoftWebDriver.exe"

Regressieset op 3 verschillende browsers

protractor protractor.browsers.conf.js

Om je testen op één browsers te starten kun je de volgende commando gebruiken. Dit start je test met de tag “zzpFlex” op “firefox”. Wil je andere tags of andere browser gebruiken wijzig deze dan in de juiste tag en browser. Voor Edge gebruik: “MicrosfotEdge”

protractor protractor.browsers.conf.js --cucumberOpts.tags="@zzpFlex" --capabilities.browserName=firefox --multiCapabilities

Pipeline (jenkins)

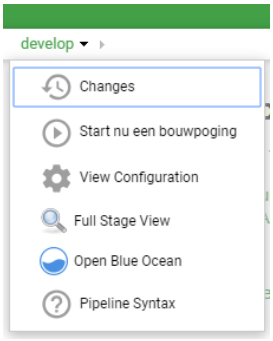
Handmatig:

Om via Jenkins de pipeline automatisch te starten ga je eerst naar:

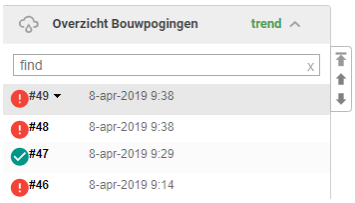
https://jenkins.do.unive.nl/job/Automated%20test/job/regression/job/develop/

Hierin vind je de verschillende bouwpogingen die zijn uitgevoerd.

Als je op het pijltje boven bij develop klikt krijg je een menu, hier kan je klikken op Start nu een bouwpoging dit zorgt ervoor dat de volledige regressieset op de selenium server wordt gedraaid.



Nadat de pipeline klaar is met het runnen van de regressietest kun je de report downloaden, door te klikken op de nummer van de laatste bouwpoging. In onderstaand geval is het nummer 49.



Klik daarn op “Build Artifacts” dan kun je dan klikken op “alle bestanden in een zip-archief” dit geeft je de mogelijkheid om de report te downloaden.