Genereer een gebruikershandleiding

Met je tests

"Hebben we al een handleiding voor de gebruikers?" vraagt de product owner. Terwijl je teamgenoten onder hun bureaus duiken, stamel jij iets over geen tijd, wisselende prioriteiten en de ellenlange backlog. Zonder succes natuurlijk, dus ben jij de vrijwilliger om het Word-document van 96 pagina's te voorzien van 149 screenshots. Vier sprints later lijkt de applicatie totaal niet meer op de screenshots en kan je weer van voren af aan beginnen.

Maar het kan anders! We zijn immers in dit vakgebied gestapt om tijdrovende menselijke handelingen te automatiseren, zodat er tijd overblijft om leuke, waardevolle dingen te maken. Juist software schrijven en complexe problemen oplossen, vinden we leuk.

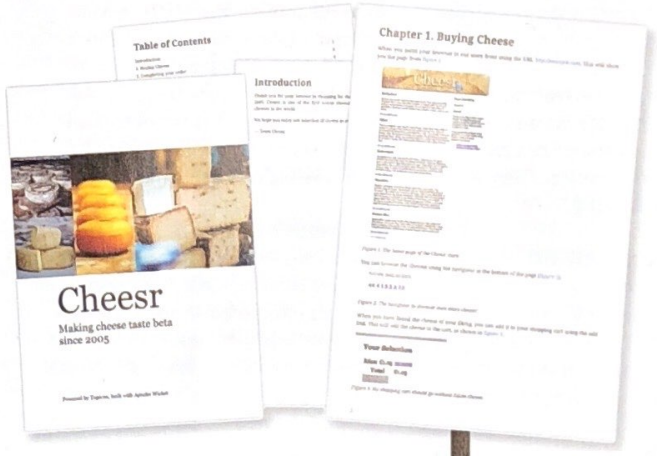
Als ik je nu vertel dat je met de juiste ontwikkeltools automatisch screenshots kunt maken, deze kunt integreren in een mooi opgemaakte handleiding in HTML en PDF-formaat en daarbij direct je applicatie goed kunt testen, dan heb ik je aandacht toch?

In dit artikel laat ik zien hoe je een altijd up-to-date handleiding kunt genereren met AsciiDoctor, Graphene, Arquillian en aShot. Het lost een daadwerkelijk probleem op, het bestaat uit een hoop programmeerwerk, het kan zo complex worden als je zelf wil en is ook nog eens leuk. Om dit te illustreren, gebruik ik een eenvoudig project: een online kaaswinkel.

De kaaswinkel Cheesr

De kaaswinkel '*Cheesr*' is bedacht in 2005, omdat Eelco (mijn toenmalige co-auteur) kaaswinkel "De Brink" in Deventer enorm miste na zijn emigratie naar de VS. Het is een standaard webapplicatie met twee schermen: het selecteren van kazen en het plaatsen van een bestelling. In **Afbeelding 1** zie je een aantal pagina's van de handleiding.

De kaaswinkel gebruikt Maven om de verschillende componenten van het bouwproces aan elkaar te knopen. **Afbeelding 2** illustreert de



Afbeelding 1

stappen, die doorlopen worden om vanaf de broncode naar een webapplicatie te komen, met daarin de handleiding gebundeld.

Nadat de broncode gecompileerd is, worden de tests gedraaid. *Arquillian* zorgt ervoor dat de webapplicatie gestart wordt en het bestuurt de browser met *WebDriver*. Dankzij *Graphene* kunnen we HTML-pagina's als Java objecten beschrijven en deze gebruiken om de tests uit te voeren. De bibliotheek *aShot* maakt de screenshots en zet deze bestanden in de folder **target/screenshots**. Verderop leer je meer over deze technologieën.

In de fase "pre-package" van het bouwproces, vlak voordat de web-applicatie geassembleerd wordt door Maven,

Martijn Dashorst werkt sinds 2004 bij Topicus. Hij lust geen kaas.

converteert *AsciiDoctor* de handleiding van *AsciiDoc* naar HTML en PDF. Uiteindelijk is dan de webapplicatie klaar en zit daarin ook de handleiding als PDF en als HTML-bestand.

Als voorbeeld test de code in **listing 1** dat de checkout-knop niet beschikbaar is, als je nog geen item in je winkelwagen hebt zitten. Daarnaast maakt de test ook direct de eerste screenshots.

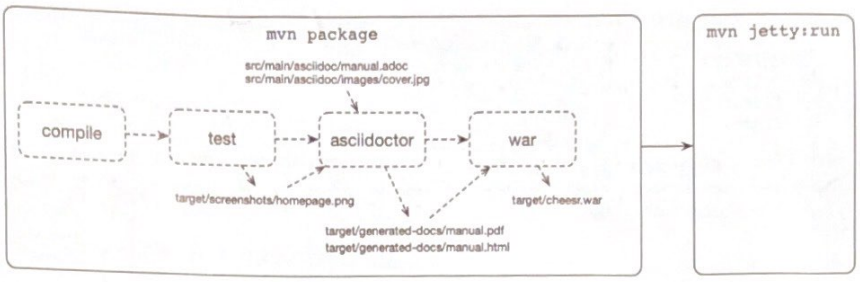
De annotatie **@RunWith(Arquillian.class)** geeft aan dat deze testcase met *Arquillian* gedraaid moet worden. Met Arquillian kun je integratietests uitvoeren, servers starten en stoppen en je applicatiecode deployen. Ook kun je Arquillian gebruiken om je tests tegen een al draaiende remote server uit te voeren, zodat je die niet vanuit je tests hoeft op te zetten.

De methode **createDeployment** met de annotatie **@Deployment** vertelt Arquillian wat er naar de server gedeployed moet worden. Dat is in dit geval een WAR met daarin onze applicatiecode.

Het veld **@Drone WebDriver** browser is een instructie, zodat Arquillian een instantie van *WebDriver* injecteert in de testcase, waarmee we de browser kunnen besturen.

De parameter **@InitialPage GIndex index** van de testmethode geeft aan dat deze test moet beginnen met de pagina **GIndex**. **GIndex** is een representatie van de interactiemogelijkheden van de homepage van Cheesr: een implementatie van het *Page Object Pattern*. Deze class bevat onder andere de locatie (URL) waar de browser naar toe gezonden moet worden. *Graphene* zorgt ervoor dat de **GIndex** instantie correct aangemaakt is en dat de browser de pagina laadt, net voordat de test is begonnen. Je kunt dan meteen interacteren met die pagina, zoals screenshots nemen, op links klikken en uitvragen of bepaalde elementen aanwezig zijn.

Voor nu is het handig om te zien hoe de screenshots van deze test gebruikt worden in de tekst van de handleiding. Het stukje AsciiDoc uit **Listing 2** is een voorbeeld hiervan.



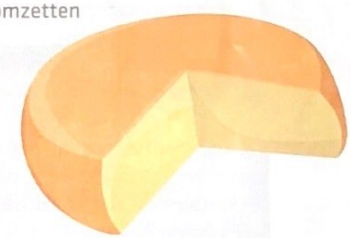
Afbeelding 2: De fases en bestanden, die een rol spelen in het bouwen van de kaaswinkel en de handleiding

De test case uit **listing 1** genereert screenshots met namen als «cheesr-1-home.png». De AsciiDoc van **listing 2** pikt deze op in image elementen zoals **image::cheesr-1-home[]**. Het uiteindelijke resultaat ziet er dan uit, zoals in **Afbeelding 1**.

AsciiDoc en AsciiDoctor

AsciiDoc is een tekstformaat om documenten in te schrijven. AsciiDoc lijkt erg veel op het veel gebruikte Markdown, maar biedt meer mogelijkheden: includes, inhoudsopgave, automatische nummering van secties, waarschuwingen, annotaties bij codevoorbeelden en nog veel meer.

AsciiDoctor is een project dat AsciiDoc kan verwerken en omzetten



```
@RunWith(Arquillian.class)
public class IndexTest {
    @Deployment(testable = false)
    public static WebArchive createDeployment() {
        return CheesrDeployment.createWar();
    }

    @Drone
    private WebDriver browser;

    @Test
    public void step1EmptyCart(@InitialPage GIndex index) {
        Camera.takeScreenshot(browser, "cheesr-1-home.png");

        Camera.takeScreenshot(browser, "cheesr-1-home-navigator.png", By.id
("navigator"));

        // nothing was selected, so the checkout button should not be present
        assertFalse(index.checkoutPresent());
    }
}
```

Listing 1: De eerste test van de kaaswinkel

```
= Buying Cheese

When you point your browser to our store front using the URL _http://example.com_.
This will show you the page from <<cheesr-home>>.

[[cheesr-home, figure 1]]
.The home page of the Cheesr store.
image::cheesr-1-home.png[{half-width}]

You can browse the cheeses using the navigator at the bottom of the page (<<cheesr-
navigator>>).

[[cheesr-navigator, figure 2]]
.The navigator to discover even more cheese!
image::cheesr-1-home-navigator.png[]

When you have found the cheese of your liking, you can add it to your shopping cart
using the _add_ link.
This will add the cheese to the cart, as shown in <<cheesr-cart>>.
```

Listing 2: Een stukje AsciiDoc uit de handleiding van de kaaswinkel

```
src/main/asciidoc
├── artikel.adoc
├── cheesr.adoc
└── images
    ├── build-process.png
    ├── cheesr-cover.jpg
    ├── cheesr-manual.jpg
    └── pageobjects.png
```

Listing 3

naar onder andere PDF, HTML, EPub en DocBook. Er is een commandline tool, die je direct kunt aanroepen, maar ook plug-ins voor Maven en Jekyll (een statische website generator). AsciiDoctor integreert ook met diagramtools, zoals PlantUML en GraphViz. Hiermee kun je UML diagrammen beschrijven in platte tekst.

AsciiDoc is uitermate geschikt om op te slaan in een Git-repository, omdat het een platte tekstformaat is. Je kunt dan je documentatie net zo versioneren als de code van je project en bijvoorbeeld meenemen in code reviews.

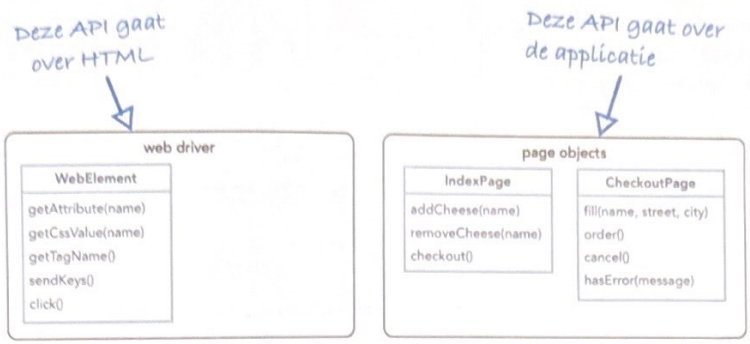
De AsciiDoc inhoud van het voorbeeld-project (waarin ook dit artikel is geschreven) ziet er als volgt uit (zie **LIsting 3**).

Met deze documenten kan AsciiDoctor de handleiding genereren en de screenshots (uit **target/screenshots**) in de handleiding opnemen. De screenshots worden genomen tijdens het uitvoeren van de applicatietests. Dit is opgezet met behulp van Arquillian, WebDriver en Graphene.

WebDriver en Graphene

WebDriver is een technologie om browsers te besturen en uit te lezen. Dit kan fragiele code opleveren als je niet goed oplet: wijzigingen in de structuur van de HTML in de browser kunnen je tests eenvoudig laten falen. Graphene is een schil om WebDriver heen om je applicatie te beschrijven op een hoger abstractieniveau door de (interactie met) HTML te encapsuleren met het *Page Objects* patroon.

In plaats van op zoek te gaan naar een link in het HTML-document om een kaas aan de winkelwagen toe te voegen, roep je de **addCheese** methode aan op je pagina object. Natuurlijk gebeurt achter de schermen alsnog het op zoek gaan in de HTML naar de betreffende link, maar is dat verborgen voor je tests. In **afbeelding 3** zie je wat het verschil is tussen direct werken met de API van WebDriver (en daarmee de HTML) en werken met een Page Objects API.



Afbeelding 3: Het verschil tussen de WebDriver API en Page Objects

Het gebruik van page objects maakt je tests een stuk beter leesbaar. De code in **listing 4** geeft een voorbeeld van het verschil tussen de WebDriver API en het gebruik van Page Objects.

In het voorbeeld van de page objects code is de bedoeling van de test direct duidelijk, terwijl dat in de WebDriver code verborgen zit in het parsen van de HTML structuur. Natuurlijk is dit gewoon het verplaatsen van de WebDriver logica naar een façade, maar dat is nu juist de essentie van het Page Object patroon.

```
// WebDriver code:
List<WebElement> addLinks = browser
    .findElements(By.cssSelector("a[id^=add]"));
addLinks.stream()
    .filter(e -> e.getAttribute("id").contains("edam"))
    .findFirst()
    .click();

Assert.assertTrue(browser.findElement(By.linkText("checkout"))
.isPresent());

// page objects code:
indexPage.addCheese("edam");
Assert.assertTrue(index.checkoutIsPresent());
```

Listing 4: Voorbeeld van het werken met de WebDriver API en Page Objects

```
@Location("")
public class GIndex {
    @FindBy(css = "a[id^=add]")
    private List<GrapheneElement> addLinks;

    @FindBy(css = "input[type=button]")
    private GrapheneElement checkout;

    public void addCheese(String cheese) {
        addLinks.stream()
            .filter(a -> a.getAttribute("id").contains(safeCheeseId(cheese)))
            .findFirst()
            .orElseThrow(() -> new NoSuchElementException("Add " + cheese + " link"))
            .click();
    }

    public boolean checkoutPresent() {
        return checkout.isPresent();
    }

    public By byCart() {
        return By.cssSelector("div[id=cart],input[type=button]");
    }
}
```

Listing 5: Een Page Object voor de Cheesr homepage

De code in **listing 5** laat de implementatie zien van het **GIndex** page object.

```
public static void takeScreenshot(WebDriver browser, String name, By crop) {
    IndentCropper cropper = new IndentCropper(25)
        .addIndentFilter(new MonochromeFilter());

    BufferedImage screenshot = new AShot()
        .imageCropper(cropper)
        .takeScreenshot(browser, browser.findElements(crop))
        .getImage();

    saveScreenshot(name, screenshot);
}
```

Listing 6: Neemt een screenshot van specifieke elementen in de pagina

De code begint met **@Location** om te vertellen waar de pagina zich bevindt in de applicatie: **""** betekent de *root*. Daarna worden alle links verzameld, die een kaas aan de winkelwagen kunnen toevoegen. Tot slot biedt de **addCheese** methode de mogelijkheid om een kaas op basis van de naam toe te voegen door op de bijbehorende link te klikken.

Dankzij Graphene kun je eenvoudig het Page Objects patroon toepassen om een model te maken van je applicatie, zodat je tests leesbaar en onderhoudbaar blijven. Nu rest ons nog het maken van de screenshots. De bibliotheek *aShot* is gemaakt om met behulp van WebDriver screenshots te maken.

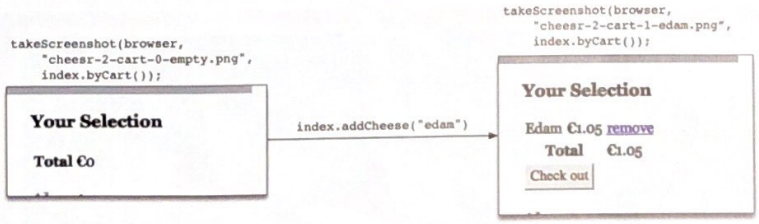
Screenshots met aShot

aShot heeft een vrij eenvoudige API. Je hoeft alleen de WebDriver instantie mee te geven en het element waarvan je de screenshot wilt maken. Je krijgt dan de afbeelding terug. Je kunt wel wat extra zaken instellen en dus is het handig om dit te wrappen in een functie, die je vanuit je tests kunt aanroepen. Je kunt ook meer webelementen aangeven, waarvan een plaatje genomen moet worden. Daarnaast is het ook mogelijk om extra ruimte om de elementen mee te nemen en kun je filters loslaten op de omliggende ruimte, zoals het zwart-wit of het wazig maken (blurren) van de rand. De code in **listing 6** laat zien hoe je dit met aShot voor elkaar krijgt.

```
@Test
public void step2AddToEmptyCart(@InitialPage GIndex index) {
    Camera.takeScreenshot(browser, "cheesr-2-cart-0-empty.png", index.byCart());

    index.addCheese("edam");

    Camera.takeScreenshot(browser, "cheesr-2-cart-1-edam.png", index.byCart());

    // assert that the checkout button is present
    assertTrue(index.checkoutPresent());

    index.removeCheese("edam");
    assertFalse(index.checkoutPresent());
}
```

Listing 7: Een test die gebruik maakt van het page object GIndex en screenshots neemt

Met deze functie is het maken van een screenshot heel eenvoudig geworden. Nu kunnen we tijdens het testen screenshots wegschrijven en deze gebruiken in onze handleiding (zie **Listing 7**).



Afbeelding 4: Twee screenshots van de winkelwagen

Een voorbeeld van de screenshots genomen in deze test zie je in **afbeelding 4**.

Hiermee is het artikel rond. We kunnen nu de applicatie testen, screenshots maken en een handleiding genereren.

Conclusie

We hebben één geïntegreerd geheel, want de documentatie leeft samen met de code in AsciiDoc formaat. We testen onze applicatie via de browser, dankzij Arquillian en Graphene. Tijdens het uitvoeren van de tests maken we screenshots en spitsen deze toe op precies die onderdelen die belangrijk zijn met aShot. Maven voegt al deze stappen samen tot één soepel proces, dat uiteindelijk onze up-to-date handleiding opneemt in onze applicatie.

Met deze setup kun je ook geïnternationaliseerde screenshots maken voor meertalige handleidingen: zet je applicatie in een andere taal en voer de tests opnieuw uit. Nu alleen nog iemand bereid vinden om die 96 pagina's tekst te vertalen... ■

BEKIJK ALLE CODE OP GITHUB

De broncode is beschikbaar op Github: **https://github.com/dashorst/nljug-article-2017**. Het is een Maven project, dus je kunt het direct importeren in je favoriete IDE. In de README.md staan instructies om het project te bouwen en te runnen. In dit artikel sta ik niet stil bij de specifieke versies en configuratie van plug-ins en dependencies. Deze kun je vinden in het project.