Plan van aanpak

# Doelstelling

We gaan Coded UI-tests schrijven voor de Clinical Trials applicatie van Calidos. Dit zijn tests om de werking van de User Interface te controleren (wat je anders handmatig zou doen). Het voordeel van deze geautomatiseerde tests is dat deze veel sneller gaan en alle aspecten dekken (als gebruiker zie je snel iets over het hoofd).

Het testen zelf is echter niet het belangrijkste, aangezien de applicatie in de toekomst nog kan wijzigen en onze tests dan niets meer waard zijn. Het is veel belangrijker dat we een gedetailleerde guideline schrijven waarin beschreven staat hoe de applicatie getest moet worden, waaraan de applicatie moet voldoen om het schrijven van tests zo eenvoudig en efficiënt mogelijk te laten gaan, en welke onderdelen al getest zijn. Zo kan een eventuele opvolger (of ook wijzelf tijdens het proces) bij wijzigingen in de applicatie zeer snel traceren waar de tests moeten aangepast worden en hoe dit moet gebeuren.

# Werkwijze

## Hiërarchie

Eerst en vooral moeten we de hiërarchie van de navigations binnen de applicatie in kaart brengen (wat intussen gebeurd is). Hierin kunnen we zien welke pagina’s naar waar navigeren om zo makkelijker te kunnen vastleggen welke navigations er per pagina getest moeten worden.

## To test

Vervolgens is het de bedoeling pagina per pagina na te gaan welk soort control ’s er zijn, welke handelingen deze kunnen uitvoeren en welke acties er op de pagina mogelijk zijn.

### Checklist

We beginnen met de basispagina’s die de meest gebruikte basisacties/control ’s bevatten (die zo goed als overal zullen terugkomen).

Voor elke gevonden control/actie (met actie bedoelen we handelingen die meer als één control gebruiken om te voltooien) zoeken we dan uit hoe we deze zo efficiënt mogelijk kunnen testen. Deze methode documenteren we dan zodat wij, en onze eventuele opvolgers, deze altijd terug kunnen raadplegen wanneer nodig (voor volgende pagina’s etc.). We documenteren ook voor de pagina die we aan het testen zijn welke control ’s en acties reeds getest zijn, tot we ze allemaal hebben gehad. Op dit moment wordt de pagina als volledig getest verklaard.

Indien we op een volgende pagina weer een nieuwe control/actie tegenkomen, zetten we deze bij onderaan onze checklist met gevonden control ‘s/acties. Het is dan weer opnieuw de bedoeling om uit te zoeken hoe we deze kunnen testen en alle voorgaande pagina’s opnieuw af te gaan en te controleren of ook dit element aanwezig is op de pagina en te testen indien nodig, zodat we de pagina weer als getest kunnen verklaren.

Unieke controls zoeken

& pagina testen

Aanvullen van guideline ‘s

Voorgaande pagina’s aan nieuwe

guideline ’s laten voldoen

*Onze checklist zal er als ongeveer volgt uitzien:*

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Page 1** | **Page 2** | **Page 3** |
| **Control/action A** | tested | To Do | Tested |
| **Control/action B** |  | Tested | Tested |
| **Control/action C** | Problem: … | Tested | Tested |

*Hierbij is B niet van toepassing op Page 1, Page 3 is volledig getest.*

### BaseClass

Als we tijdens het proces merken dat bepaalde stukken code (vooral heel omslachtige/moeilijk te begrijpen code) heel vaak terugkomen, schrijven we hiervoor een methode in een BaseClass, waarvan al onze testprojecten overerven. Hierdoor moeten we slechts één keer het vervelende werk doen en nadien altijd heel makkelijk dezelfde handelingen testen op volgende pagina’s.

### Soorten handelingen

In het begin gaan we ons vooral focussen op de onafhankelijke tests pagina per pagina. Het gaat hier dus vooral om navigations, buttons, back buttons,…

Als we dieper gaan in de applicatie heeft data steeds meer belang in het User-interface gegeven, hierbij.gaan we gebruik maken van datadriven tests. Dit zijn tests die gebruik maken van externe datafiles. We kunnen hiermee dus demodata inladen en pagina’s controleren aan de hand van deze data. Hierbij zal het dan meestal gaan om meer globale tests, die niet op één enkele pagina blijven maar waarbij bepaalde ingevoerde data invloed heeft op verschillende pagina’s. Er moet dus overal gecontroleerd worden of de ingevoerde data de juiste veranderingen heeft aangebracht binnen de applicatie.

## Guideline

### Checklist

In onze volledige test-guideline zal ook de checklist staan zoals eerder vermeld.

### How to test

In deze sectie schrijven we voor elk gevonden element in de checklist hoe dit getest moet worden. Zowel wij als onze opvolgers kunne hiervan dus steeds gebruik maken wanneer we een test moeten schrijven of herschrijven. Dit is het belangrijkste stuk van de guideline wanneer de applicatie zelf gewijzigd wordt. Op de hiërarchie kan je zien wat er veranderd is en waar deze verandering invloed op heeft, en in deze sectie kan je dan terugvinden hoe je toegevoegde onderdelen moet testen. Indien je dit hier niet terugvind moet je dus zelf uitzoeken hoe je dit moet testen, en dan is het weer de bedoeling om terug te gaan naar de checklist en dit element voor elke pagina af te gaan.

Ook gaan we in deze sectie beschrijven hoe Coded UI-tests in het algemeen zijn opgebouwd voor de eventuele opvolger.

### App-vereisten

In de loop van het proces gaan we waarschijnlijk nog meerdere situaties tegenkomen waarbij de manier van werken in de applicatie zelf problemen geeft bij onze tests. We hebben bijvoorbeeld al ontdekt dat bijna geen enkele control in de applicatie een AutomationId bevat, welke één van de belangrijkste properties is om een control correct te identificeren in de UI\_map.

In deze sectie gaan we al deze situaties documenteren voor de developers van de applicatie zelf, zodat zij, indien mogelijk, aanpassingen kunnen maken aan hun applicatie om het testen te vereenvoudigen.