# Ux design

### Voorbeeld

### Te gebruiken bij:

-

### Eigenschappen:

-

### Info

#### beginnen

1. **Probleemstelling**: bepalen welk probleem je wilt oplossen voor jouw gebruikers met deze applicatie en wat de vier á vijf belangrijkste kern-functionaliteiten zijn.
2. **Use cases**: je kiest vier kernfunctionaliteiten uit die je gaat uitwerken in use case tabellen. Hierin beschrijf je op gedetailleerde wijze welk proces de gebruiker doorloopt om zijn doel te behalen en hoe de applicatie hierop zou moeten reageren. Ook ga je nadenken over *gebruiksvriendelijkheid*: hoe kun je de gebruiker *ondersteunen* bij het behalen van zijn doel? Door het maken van use case tabellen zul je erachter komen dat de applicatie functionaliteiten moet bevatten waar je voorheen nog niet over nagedacht had.
3. **Systeemeisen**: op basis van alle informatie die je uit de use case tabellen hebt ontdekt, ga je een lijst met functionele- en niet functionele eisen opstellen. Hierbij beschrijven functionele eisen *wat* het systeem doet, en niet-functionele eisen vooral *hoe* het systeem dit doet.
4. **Wireframes**: wanneer je een duidelijk beeld hebt van de werking van de applicatie, ga je nadenken welke elementen je op de pagina nodig hebt om dit te bewerkstelligen.
5. **Schermontwerpen**: op basis van de getekende wireframes kun je de schermen gaan ontwerpen. Want wanneer je gaat programmeren zonder dat je weet hoe de pagina eruit moet komen te zien, duurt het tien keer zo lang.

#### User Experience Design

Zoals je inmiddels hebt geleerd, bestaat een applicatie uit logica (onzichtbaar voor de gebruiker) en de **User Interface** (het visuele deel van de applicatie waarmee de gebruiker interacteert). De samenwerking tussen de logica en de UI hebben ontzettend veel invloed op de uiteindelijke gebruikerservaring van de applicatie. Het ontwerpen van een intuïtieve, soepele gebruikerservaring noemen we **User Experience Design**.

Zo ontstonden posities als UX Designer en UX Researcher, die er samen met de ontwikkelaars en ontwerpers voor zorgen dat gebruikers hun haren niet uit hun hoofd hoeven te trekken.

Gebruiksvriendelijkheid kunnen we het best afmeten aan vijf onderdelen:

* **Learnability**: hoe makkelijk is het voor gebruikers om taken uit te voeren als ze het systeem *voor het eerst* gebruiken?
* **Efficiency**: als gebruikers het systeem eenmaal kennen, hoe snel kunnen ze hun taken dan uitvoeren?
* **Memorability**: als gebruikers het systeem een tijdje niet hebben gebruikt, hoe makkelijk is het dan om hun kennis weer terug te halen?
* **Errors**: hoeveel fouten maken gebruikers, hoe ernstig zijn ze en hoe gemakkelijk kunnen ze hun fouten weer terugdraaien?
* **Satisfaction**: is het systeem prettig in gebruik?

#### De heuristieken van Jakob Nielsen

*de “grootvader van de usability”*

stelde in 1995 tien onderwerpen vast waaraan gebruiksvriendelijkheid gemeten kon worden. En hoewel het internet toen pas net was uitgevonden, zijn deze uitgangspunten nog steeds actueel en vormen ze de basis van ieder *usability* onderzoek.

1. **Zichtbaarheid van de status van het systeem**: Het systeem moet gebruikers op de hoogte houden van wat er aan het gebeuren is. Denk bijvoorbeeld aan een wacht-animatie tijdens het laden van de inhoud.
2. **Overeenkomst tussen het systeem en de echte wereld**: Gebruik taal en symbolen die bekend zijn voor de gebruiker en overeenkomen met concepten in de echte wereld, zodat het direct duidelijk is hoe een applicatie gebruikt moet worden. Toen Apple de compas-functionalteit voor de iPhone maakte, hebben ze "echte" kompassen niet voor niets als inspiratiebron gebruikt:

Afbeelding met kompas, apparaat, Meetinstrument, tekst

Automatisch gegenereerde beschrijving

1. **Controle en vrijheid voor de gebruiker**: Bied de gebruiker altijd een ‘nooduitgang’ zodat hij zijn acties kan terugdraaien. Denk bijvoorbeeld aan de undo-functie (ᴄᴛʀʟ + ᴢ) of de prullenbak, waarin verwijderde bestanden komen te staan.
2. **Consistentie en standaardisering**: Zorg dat gebruikers zich niet hoeven af te vragen of verschillende woorden, situaties of acties hetzelfde betekenen. Volg bestaande conventies: het is inmiddels algemeen bekend dat de drie streepjes onder elkaar een menu aanduiden en een tandwiel de instellingen. Maar denk er bijvoorbeeld ook aan dat je het verwijderen van items niet op de ene button "wissen" noemt en op de ander "verwijderen".
3. **Voorkom fouten**: Goede foutmeldingen zijn cruciaal, maar het vermijden van de mogelijkheid dat een gebruiker een fout *kan* maken is nog veel belangrijker. Als het combineren van twee specifieke acties ervoor zorgt dat het systeem crasht, zorg dan dat deze acties in de eerste plaats al niet gecombineerd kunnen worden.
4. **Herkenning boven herinnering:** Verklein de geheugenbelasting zoveel mogelijk door acties en opties direct zichtbaar te maken. Zorg dat functionaliteit niet te diep ‘begraven’ ligt in een systeem waardoor de gebruiker 8 stappen moet onthouden als hij nog eens van de zoekfunctie gebruik wil maken.
5. **Efficiëntie**: Zorg dat het systeem bruikbaar is voor zowel ervaren, als onervaren gebruikers. Bijna iedere applicatie heeft bijvoorbeeld de mogelijkheid om kopiëren en plakken zowel via het menu, als via shortcuts (ᴄᴛʀʟ + ᴄ en ᴄᴛʀʟ + ᴠ) uit te voeren.
6. **Esthetisch en minimalistisch ontwerp**: Zorg dat er nooit informatie getoond wordt die irrelevant is voor het proces dat de gebruiker aan het uitvoeren is. Dit concurreert namelijk met de informatie die wél essentieel is. Als we telkens bij het opslaan van een bestand in WebStorm een pop-up zouden krijgen met ‘Yes! Het bestand is opgeslagen!’ is dat hartstikke leuk, maar wordt ons werkproces steeds onderbroken.
7. **Hulp bij fouten**: Communiceer foutmeldingen in "gewone mensen"-taal en maak duidelijk wat er precies fout is gegaan. Het is ontzettend frustrerend als jouw contactformulier niet verzonden wordt omdat je een verplicht veld leeg hebt gelaten, maar de enige feedback die je krijgt is: “Er is iets misgegaan.”
8. **Hulp en documentatie**: Het is altijd beter als een systeem zo duidelijk is dat documentatie overbodig is, maar soms kan documentatie simpelweg nodig zijn. Zorg, wanneer dit het geval is, dat de gebruiker handleidingen tot zijn beschikking heeft.

##### Use cases

we gebruik maken van **use case tabellen**, omdat deze variant het meest inzicht geeft in de interactie tussen de gebruiker en het systeem. Een use case tabel is een tabel met daarin een reeks eenvoudige stappen die beschrijven hoe de gebruiker zijn doel bereikt. Zo'n doel kan bijvoorbeeld zijn: inloggen bij website X, of: een bestelling doen bij een bezorgdienst.

Use case tabellen helpen je nadenken over wat het systeem moet doen (of kunnen), maar ook over alle alternatieve scenario's die kunnen voorkomen op het moment dat een gebruiker zijn doel probeert te bereiken. Bijvoorbeeld: wat gebeurt er als jouw gebruiker een verkeerd wachtwoord invoert tijdens het inloggen? Wat moet het systeem dan doen? En wat als jouw gebruiker drie keer achter elkaar een verkeerd wachtwoord invoert? Wordt hij dan geblokkeerd, of mag hij het nog een keer proberen?

Use case tabellen vormen uiteindelijk de basis voor het opstellen van functionele- en niet functionele eisen

.

###### Use case elementen

Een use case bestaat uit de beschrijving van de volgende elementen:

* **Het main success scenario**- het scenario waarin niets misgaat. Dit noemen we vaak ook wel gekscherend de *happy flow*’. De happy flow bestaat uit de volgende elementen:
* **Titel**: welke functionaliteit wordt er beschreven?
* **Actor**: wie interacteert er met het systeem in deze use case? Is dat een gewone gebruiker of bijvoorbeeld een admin?
* **Precondities**: wat moet er al gebeurd zijn, of aanwezig zijn, voordat deze use case kan starten? Denk bijvoorbeeld aan een use case die de account-details wijzigt: deze use case zou *nooit* plaats kunnen vinden zonder dat de gebruiker geregistreerd en ingelogd is;
* **Trigger**: wat is de gebeurtenis die ervoor zorgt dat de gebruiker aan de use case begint? Dit is vaak simpeler dan je denkt. Het kan zijn dat de gebruiker op de “wijzig account”-knop drukt waardoor deze stroom gebeurtenissen in gang wordt gezet, of wanneer de gebruiker een bericht-melding ziet en hierop klikt om zijn berichten te bekijken. Een trigger heeft *nooit* te maken met het gevoel van de gebruiker (dus niet: “De gebruiker besloot dat hij wilde weten of het morgen zonnig wordt”);
* **Postconditie**: wanneer alles goed gegaan is, wat moet het eindresultaat dan zijn? Dit kan zijn: de gebruiker is ingelogd, of: de gebruiker heeft een pizza besteld die op zijn adres wordt bezorgd.
* De **alternatieve scenario's** - de scenario's die op specifieke punten van het main success scenario afsplitsen en beschrijven wat er mis kan gaan in die situatie, en hoe het systeem hierop inspeelt.
* **Titel**: welke situatie wordt er beschreven?
* **Nummering**: bij welke stap uit het main success scenario begint dit alternatieve scenario?
* **Postconditie**: op welke plek (van het main success scenario) kan de gebruiker weer instromen?

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

###### Main success scenario opstellen

In het volgende voorbeeld gaan we in meerdere **tussenstappen** toewerken naar de uiteindelijke use case. In dit geval zullen we doen alsof we een wasmachinefabrikant zijn. Om het product goed vorm te geven en duidelijke richtlijnen te hebben voor de systeemeisen, gaan we eerst in kaart brengen hoe we een gebruiker willen laten interacteren met de wasmachine.

Stap 1 - Titel en postconditie

Bedenk welke functionaliteit je zult doorlopen tijdens deze use case, en wat het einddoel van de gebruiker moet zijn. Functionaliteiten waarvoor je een use case tabel voor zou kunnen schrijven zijn bijvoorbeeld: wastrommel schoonmaken of overtollig water afvoeren. In dit geval kiezen we ervoor om de kernfunctionaliteit van de wasmachine in kaart te brengen: de was doen. Dit doel is bereikt wanneer de was - open deur - is gewassen.

Stap 2 – Actor

Het is altijd belangrijk om te beschrijven wie de hoofdrol speelt in de use case. Meestal maken we een onderscheid tussen admins en gewone gebruikers, maar wanneer je bijvoorbeeld een use case tabel zou maken voor het bezorgsysteem Deliveroo, zou je onderscheid maken tussen bezorgers en bestellers. In het geval van onze wasmachine hebben we het over een gebruiker.

Stap 3 – Trigger

Een trigger beschrijft wanneer of waarom deze specifieke use case begint. Dit beschrijven we altijd in **systeemtaal** omdat het ons ondersteunt bij het ontwerpen van het systeem. Het heeft dus niets te maken met de gevoelens of persoonlijkheid van de gebruiker. Een **incorrecte** trigger zou bijvoorbeeld zijn: “De gebruiker besloot dat het maar eens tijd was om de was te gaan doen” of “De gebruiker ging even in het washok kijken en zag toen dat er vuile was lag.”

In ons geval is de trigger voor het doen van de was: “De gebruiker arriveert in de wasruimte”.

In webapplicaties is de trigger vaak simpelweg dat de gebruiker de pagina bezoekt waar de actie kan worden uitgevoerd. Bij een use case voor het maken van een nieuwe blog-post zal de trigger waarschijnlijk zijn: “De gebruiker klikt op ‘Maak nieuwe blogpost-knop” of “Gebruiker navigeert naar pagina ‘Nieuwe post maken’”. Een use case over het verbeteren van spelfouten zal “De gebruiker maakt een spelfout” als trigger hebben. Een use case voor het beantwoorden van chatberichten zal “De gebruiker ziet melding dat hij/zij een nieuwe chatbericht heeft ontvangen” zijn, waardoor hij of zij het berichtencentrum aanklikt.

Stap 4 - Main success scenario

Nu kunnen we de stappen gaan beschrijven die de gebruiker moet doorlopen om een was te draaien. Denk goed na: hoe willen we dat deze ervaring eruitziet? We beschrijven dit door duidelijk aan te geven wat de gebruiker doet, en hoe het systeem hierop reageert.

De gebruiker stopt de vuile was in de machine;

1. De gebruiker zet de wasmachine aan door op de “ON” knop te drukken;
2. De wasmachine maakt een geluid en het display wordt zichtbaar;
3. De gebruiker kiest wasprogramma “Katoen 30 graden”;
4. De wasmachine geeft het gekozen wasprogramma weer;
5. De gebruiker drukt op de “start”-knop om het wasprogramma te starten;
6. De wasmachine start het programma en laat de aflopende tijd in minuten en seconden zien tot het programma klaar zal zijn;
7. Nadat het programma is geëindigd, haalt de gebruiker de was uit de machine.

Ook deze stappen hebben eigenlijk weinig te maken met de gebruiker, de gebruiker is slechts een acteur in ons verhaal om het systeem in kaart te brengen. We laten daarom alle gevoelens van de gebruiker buiten beschouwing. Een **incorrecte** beschrijving van stap 4 zou bijvoorbeeld zijn:

1. De gebruiker vraagt zich af welke temperatuur ze moet kiezen, want er is veel keus. Dan kiest ze maar voor “Katoen 30 graden” omdat dit het meest gangbaar is.

Stap 5 – Precondities

Het is je waarschijnlijk opgevallen dat de gebruiker in ons main success scenario geen wasmiddel in de machine heeft gedaan. Dat komt omdat deze specifieke wasmachine is voorzien van functionaliteit waarin in één keer een grote hoeveelheid wasmiddel kan worden ingeladen, waardoor dit niet bij iedere lading apart gedaan hoeft te worden. De wasmachine doseert het wasmiddel zelf bij iedere wasbeurt. Ook dit moeten we opnemen in onze use case tabel. Dit doen we door aan te geven aan welke condities voldaan moet zijn voor dit scenario überhaupt kan beginnen:

Preconditie 1: Doseersysteem bevat wasmiddel

Maar we hebben nog iets anders belangrijks nodig... kijk maar eens naar stap 1. Precies! Er moet wel (vuile) was zijn om in de machine te stoppen, anders kunnen we geen was wassen. We hebben daarom nóg een preconditie:

Preconditie 2: Er is vuile was

Eindresultaat

Wanneer we dit combineren in een tabel, komen we tot dit eindresultaat:

###### Alternatieve scenario's opstellen

Het is altijd heerlijk als alles loopt zoals gepland, nietwaar? We moeten echter ook realistisch zijn: want vaak loopt alles juist niet zoals we hadden gepland. Dit is waar de alternatieve scenario's voor zijn. Ze beschrijven wat er mis kan gaan bij die specifieke stap, en ook hoe het systeem de gebruiker helpt om weer terug te keren naar het main success scenario.

Laten we eens kijken naar stap 6, waarin de gebruiker het wasprogramma wil starten. Het kan maar net zo zijn dat ze de deur van de wasmachine nét niet goed heeft gesloten bij het inladen...

Stap 1 - Titel en nummering

We noemen het alternatieve scenario altijd naar hetgeen dat fout gaat. Tevens geven we de use case een nummer die correspondeert met het stapnummer uit het main success scenario, zodat het duidelijk is wanneer dit alternatieve scenario begint:

Alternatief scenario 6A: wasmachinedeur is niet dicht

We gebruiken tevens altijd de letter A, omdat dit het eerste alternatieve scenario is voor stap 6. Het zou maar zo kunnen zijn dat er drie dingen mis kunnen gaan bij stap 6. Dan hebben we alternatief scenario 6A, 6B en 6C.

Stap 2 - Alternatief scenario

Laten we ons in de situatie verplaatsen. De deur is niet goed dicht (hoewel onze gebruiker misschien denkt van wel) en de gebruiker probeert het wasprogramma te starten. Hoe denk jij dat de gebruiker hier het beste van op de hoogte gesteld kan worden? Door een geluid, een knipperend lampje of een letterlijke melding op het scherm? De keus is aan jou!

1. De wasmachine laat melding Sluit deur om wasprogramma te starten” zien
2. De gebruiker sluit de deur
3. De wasmachine maakt een bevestigend geluid

Stap 3 - Postconditie

In de meeste gevallen beschrijft de postconditie waar de gebruiker haar weg weer kan vervolgen in het main success scenario. In ons geval is de postconditie dus:

Gebruiker vervolgt main success scenario bij stap 7

Eindresultaat

We hebben zojuist ons eerste alternatieve scenario geschreven:

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

|  |  |  |
| --- | --- | --- |
| **Het main success scenario “***happy flow”* | A**lternative scenario’s** |  |
| Titel | Titel |  |
| Actor | Actor |  |
| Precondities | Precondities |  |
| Trigger | Trigger |  |
| Postconditie | Postconditie |  |

 De **alternatieve scenario's** - de scenario's die op specifieke punten van het main success scenario afsplitsen en beschrijven wat er mis kan gaan in die situatie, en hoe het systeem hierop inspeelt.

 **Titel**: welke situatie wordt er beschreven?

 **Nummering**: bij welke stap uit het main success scenario begint dit alternatieve scenario?

 **Postconditie**: op welke plek (van het main success scenario) kan de gebruiker weer instromen?

###### *Systeemeisen*

Daarom zullen we duidelijke systeemeisen moeten stellen aan onze applicaties voordat we ze daadwerkelijk gaan bouwen. Binnen systeemeisen onderscheiden we **functionele**- en **niet-functionele eisen**.

Functionele eisen

Functionele eisen bepalen het basale gedrag van het systeem, ofwel: **wat** moet het systeem wel of juist niet doen.

als een manier waarop het systeem reageert op input. Als een gebruiker op “Versturen” klikt, wat doet de applicatie dan?

de focus ligt op de *vereisten*. Als er niet aan deze eisen wordt voldaan, werkt de applicatie niet.

Voorbeelden van functionele eisen zijn:

* Winkelmedewerkers voegen nieuwe bestellingen toe door het bestelformulier in te vullen.
* Het systeem checkt direct of het ingevoerde adres geldig is. Zo niet, dan wordt een foutmelding getoond en kan het formulier niet worden verstuurd.
* Wanneer de winkelmedewerker een bestelling wil wijzigen, wordt er automatisch een wijzigingsverzoek naar de manager gestuurd.
* Aanpassingen worden pas van kracht nadat een wijzigingsverzoek is goedgekeurd door de manager.
* Alle bestellingen en wijzigingen worden geregistreerd in een database.
* Als er 24 uur zijn verstreken sinds de laatste database back-up, wordt er automatisch een nieuwe back-up gemaakt door de server.

Niet-Functionele eisen

niet-functionele eisen specificeren **hoe** het systeem dit moet doen.

Niet-functionele eisen hebben geen invloed op de daadwerkelijke functionaliteit van het systeem, dus als er aan geen van deze eisen wordt voldaan werkt de applicatie nog steeds.

waarom zou ik ze dan opstellen?

Niet-functionele eisen definiëren kenmerken die de gebruikerservaring beïnvloeden en leggen de focus daarmee op de *verwachting* van de gebruiker.

* **Gebruiksvriendelijkheid**: hoe gaan gebruikers interacteren met de software? Welke doelgroep bedien je? Bij de website van de Rijksoverheid is het een vereiste dat het lettertype groot en duidelijk leesbaar is, omdat ook gebruikers met slechter zicht de website moeten kunnen gebruiken. In tegenstelling zal de website van een duur kledingmerk zich vooral focussen op een luxe uitstraling en mooie foto's.
* **Betrouwbaarheid**: hoe belangrijk is het dat het systeem altijd in de lucht is? Voor de applicatie die ervoor zorgt dat locaties van de MeldKamer naar de uitrukkende ambulance worden gestuurd is dit belangrijker dan voor een applicatie die restaurantreserveringen bijhoudt.
* **Prestatie**: hoe snel moet het systeem opereren? Wanneer jij een bestemming op Google Maps invoert, mag die berekening natuurlijk niet 5 minuten duren. Dan bedenk je zelf wel wat de snelste route is!
* **Ondersteuning**: is het belangrijk om de gebruikers ondersteuning te bieden bij het gebruik van de applicatie? Bijvoorbeeld door de aanwezigheid van een gebruikershandleiding?
* **Veiligheid**: hoe goed moet de applicatie beveiligd zijn? De niet-functionele eisen op het gebied van veiligheid zullen veel strenger zijn bij het ontwikkelen van DigiD dan bij die van Pinterest.

Voorbeelden van niet-functionele eisen zijn:

* Dankzij de layout kunnen winkelmedewerkers binnen drie klikken een nieuwe bestelling plaatsen (*gebruiksvriendelijkheid*).
* Er is documentatie beschikbaar die beschrijft hoe ontwikkelaars data van onze server kunnen opvragen via de API (*ondersteuning*).
* Er mag geen verlies in snelheid optreden binnen de range van 0 tot 1000 gelijktijdige gebruikers (*prestatie*).
* Er wordt gebruik gemaakt van JSON Web Tokens voor authenticatie (*veiligheid*).
* Het systeem moet een DDOS-aanval kunnen afweren (*veiligheid*).

Vergelijking

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

###### Wireframes

tekenen we **wireframes**: een versimpelde mockup van de interface.

Het is het beste om wireframes met pen en papier te tekenen.

Het voordeel hiervan is dat je erg wendbaar bent, een schets is zo doorgekrast en opnieuw getekend!

slecht

Afbeelding met schets, Kinderkunst, tekening, tekst

Automatisch gegenereerde beschrijving

Goed

Afbeelding met tekst, schets, tekening, papier

Automatisch gegenereerde beschrijving

figma

Het werkt het beste om mee te doen met de tutorials terwijl je ze kijkt. Wordt de video niet goed weergegeven? Bekijk de afspeellijst dan [hier](https://www.youtube.com/watch?v=dXQ7IHkTiMM&list=PLXDU_eVOJTx7QHLShNqIXL1Cgbxj7HlN4&index=2).