TESTSTRATIE EN -PLAN

SOFTWARE DEVELOPMENT PROJECT 2

Inhoud

[Teststrategie 6](#_Toc102919112)

[Testplan 7](#_Toc102919113)

[Introductie 7](#_Toc102919114)

[Test Use Case 1: Raadplegen categorie 7](#_Toc102919115)

[Resources 7](#_Toc102919116)

[Features 7](#_Toc102919117)

[Methoden 7](#_Toc102919118)

[Assumpties 9](#_Toc102919119)

[Risico’s 9](#_Toc102919120)

[Test matrix 10](#_Toc102919121)

[Test Use Case 2: Toevoegen categorie 11](#_Toc102919122)

[Resources 11](#_Toc102919123)

[Features 11](#_Toc102919124)

[Methoden 11](#_Toc102919125)

[Assumpties 14](#_Toc102919126)

[Risico’s 14](#_Toc102919127)

[Test matrix 15](#_Toc102919128)

[Test Use Case 3: Wijzigen categorie 15](#_Toc102919129)

[Resources 15](#_Toc102919130)

[Features 15](#_Toc102919131)

[Methoden 16](#_Toc102919132)

[Assumpties 18](#_Toc102919133)

[Risico’s 19](#_Toc102919134)

[Test matrix 19](#_Toc102919135)

[Test Use Case 4: Verwijderen categorie 20](#_Toc102919136)

[Resources 20](#_Toc102919137)

[Features 20](#_Toc102919138)

[Methoden 21](#_Toc102919139)

[Assumpties 22](#_Toc102919140)

[Risico’s 22](#_Toc102919141)

[Test matrix 23](#_Toc102919142)

[Test Use Case 5: Registreren gebruiker 24](#_Toc102919143)

[Resources 24](#_Toc102919144)

[Features 24](#_Toc102919145)

[Methoden 24](#_Toc102919146)

[Assumpties 26](#_Toc102919147)

[Risico’s 26](#_Toc102919148)

[Test matrix 27](#_Toc102919149)

[Test Use Case 6: Aanmelden gebruikers 27](#_Toc102919150)

[Test Use Case 7: Aanpassen gebruikersrol 27](#_Toc102919151)

[Test Use Case 8: Raadplegen datasource 28](#_Toc102919152)

[Resources 28](#_Toc102919153)

[Features 28](#_Toc102919154)

[Methoden 28](#_Toc102919155)

[Assumpties 29](#_Toc102919156)

[Risico’s 29](#_Toc102919157)

[Test matrix 30](#_Toc102919158)

[Test Use Case 9: Toevoegen datasource 32](#_Toc102919159)

[Resources 32](#_Toc102919160)

[Features 32](#_Toc102919161)

[Methoden 32](#_Toc102919162)

[Assumpties 34](#_Toc102919163)

[Risico’s 34](#_Toc102919164)

[Test matrix 35](#_Toc102919165)

[Test Use Case 10: Wijzigen datasource 36](#_Toc102919166)

[Resources 36](#_Toc102919167)

[Features 36](#_Toc102919168)

[Methoden 36](#_Toc102919169)

[Assumpties 38](#_Toc102919170)

[Risico’s 38](#_Toc102919171)

[Test matrix 39](#_Toc102919172)

[Test Use Case 11: Verwijderen datasource 40](#_Toc102919173)

[Resources 40](#_Toc102919174)

[Features 40](#_Toc102919175)

[Methoden 40](#_Toc102919176)

[Assumpties 41](#_Toc102919177)

[Risico’s 41](#_Toc102919178)

[Test matrix 42](#_Toc102919179)

[Test Use Case 12: Raadplegen MVO-doelstelling 44](#_Toc102919180)

[Resources 44](#_Toc102919181)

[Features 44](#_Toc102919182)

[Methoden 44](#_Toc102919183)

[Assumpties 45](#_Toc102919184)

[Risico’s 45](#_Toc102919185)

[Test matrix 46](#_Toc102919186)

[Test Use Case 13: Toevoegen MVO-doelstelling 48](#_Toc102919187)

[Resources 48](#_Toc102919188)

[Features 48](#_Toc102919189)

[Methoden 48](#_Toc102919190)

[Assumpties 50](#_Toc102919191)

[Risico’s 50](#_Toc102919192)

[Test matrix 51](#_Toc102919193)

[Test Use Case 14: Wijzigen MVO-doelstelling 52](#_Toc102919194)

[Resources 52](#_Toc102919195)

[Features 52](#_Toc102919196)

[Methoden 52](#_Toc102919197)

[Assumpties 54](#_Toc102919198)

[Risico’s 54](#_Toc102919199)

[Test matrix 55](#_Toc102919200)

[Test Use Case 15: Verwijderen MVO-doelstelling 56](#_Toc102919201)

[Resources 56](#_Toc102919202)

[Features 56](#_Toc102919203)

[Methoden 56](#_Toc102919204)

[Assumpties 57](#_Toc102919205)

[Risico’s 57](#_Toc102919206)

[Test matrix 58](#_Toc102919207)

[Test Use Case 16: Raadplegen cetegorioverzicht 60](#_Toc102919208)

# Teststrategie

Dit document beschrijft hoe we de verschillende use cases in dit project zullen testen, teneinde een kwalitatief eindproduct te verzekeren. Aangezien we de agile methodologie volgen, gebeurt het **testen niet louter aan het einde van het project**.

Enerzijds streeft elk groepslid ernaar om zo vroeg als mogelijk **feedback** te geven tijdens dat het ontwikkelingsproces plaatsvindt. Dat zorgt ervoor dat we wijzigingen niet op het laatste moment moeten doorvoeren, maar dat dit reeds voor tussenproducten kan. Daarmee vermijden we om in tijdsnood te geraken. Indien het mogelijk is, schrijven en testen we de code in paren (“**pair programming**” techniek), wat de code naar een hoger niveau zal tillen. Nadat code klaar staat, voert een aangesteld groepslid een **code review** uit, waarna eventueel refactoring plaatsvindt.

Anderzijds zorgt een set van diverse **types testen** ervoor dat onze applicatie kwalitatief is uitgewerkt, met oog voor de behoeften van de eindklant. Als inspiratiebron, gebruiken we het testkwadrant van Brian Marick (2003) en de testpiramide van Mike Cohn (2009). Naarmate de relevantie voor de uitwerking van de use cases, maken we een keuze uit de onderstaande types.

* Unit en component testen in de IDE (automatisch).
* Functionele testen voor de user stories, met oog voor integraties (automatisch).
* Exploratief of verkennend testen van elke use case (manueel).
* Load of performance testen na elke use case (manueel en/of automatisch).
* Security testen nadat de authenticatie-code is ontwikkeld.
* User interface testen om de correcte werking in de GUI te checken.

Gelet op een efficiënte aanpak, zetten we daarbij maximaal in op het **automatiseren** van het testproces, zodanig dat we regressietesten snel kunnen laten lopen. Dat zorgt ervoor dat we fouten of bugs vroeg ontdekken, waarmee we de **technische schuld** (of “technical debt”) op de lange termijn laag houden.

# Testplan

## Introductie

Dit testplan beschrijft voor elke use case de wijze van testen. Daarbij wordt de gebouwde functionaliteit kort geduid, met aansluitend de **soorten van testen**. De **resultaten** worden aan het einde in een matrix met condities opgesomd, zodat gekend is of de functionaliteit aan de **domeinregels** voldoet, en bijgevolg aan de **verwachtingen** van de eindklant. Indien deze positief zijn, dan kan de code kan worden geïntegreerd, mits eventuele refactoring.

## Test Use Case 1: Raadplegen categorie

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (30 minuten), Java |
| Henri De Bel | 33% (30 minuten), React |
| Christof Remeysen | 33% (30 minuten), Java en React |

### Features

De ontwikkelde features voor deze use case staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Overzichtspagina | Deze toont de categorieën die in de MySQL databank en REST API aanwezig zijn. |
| Detailpagina | Deze geeft de details van de categorie weer: naam, beschrijving, afbeelding. |
| Routing | Deze zorgt ervoor dat je van de overzichtspagina naar de bijbehorende detailpagina kan navigeren. |
| REST API (React)  JPA (Java) | De REST API/JPA ontvangt en behandelt verzoeken van de front-end. De REST API/JPA bevat seeds met mock-data; |
| CDN platform | Dit bevat de gebruikte afbeeldingen (React). |
| MySQL databank | Deze bevat de data van een categorie, inclusief afbeeldingen (Java). |
| CSS stijlen | Deze zorgen voor de opmaak van een scherm:   * React: Chakra. * Java: JavaFX CSS. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Overzichtspagina | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De testers, waarvan er minstens twee een verschillende rol hebben, openen de overzichtspagina en controleren deze. Daarna consulteren ze twee categorieën. Voor React gebeurt dit semiautomatisch met Cypress. |
| Detailpagina | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de detailpagina en controleert deze. Voor React gebeurt dit semiautomatisch met Cypress. |
| Routing | Functioneel  Performance | De functionele testen verlopen exploratief, zowel voor Java als React. De tester controleert of de juiste pagina’s bij navigatie openen.    Voor Java, verifieert de tester de compilatieduur en snelheid van laden op exploratieve wijze. Voor React, zet de tester hiervoor Google Lighthouse en React Profiler in. |
| REST API (Nest.js) | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht en via Postman of de data goed en snel genoeg vanuit de Nest.js API inlaadt. |
| CDN platform | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht of afbeeldingen goed en snel genoeg vanuit Cloudinary inladen. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor Java. De tester verifieert op het zicht of de data en afbeeldingen goed en snel genoeg vanuit de MySQL databank inladen. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen en leesbaarheid van tekst. |

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* De “mock” data is aanwezig in de persistentie laag om de exit criteria te kunnen bereiken.
* Elke rol kan testen. Authenticatie wordt pas in volgende use cases (UC5 en UC6) voorzien.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Risico** | **Impact** | **Mitigatieplan** |
| Afwezigheid van authenticatie (vertrouwelijkheid) | De data is zichtbaar voor alle gebruikers, ook al hebben ze de juiste rol niet. | Groot | De testers zorgen voor fictieve data. Zodra de applicatie live kan, testen we authenticatie en de zichtbaarheid van features voor rollen in detail. |
| Capaciteit en mock-data (performantie) | Het aantal gelijktijdige verzoeken voor de REST API en databank en de hoeveelheid mock-data weerspiegelen geen realistische niveaus inzake laadtijden en performantie. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank en -API uitgevoerd. |
| Dataformaat (consistentie) | De data bevindt zich in een API voor React en in een databank voor Java, welke aanvragen in bepaalde formaten moeten blijven verwerken. | Groot | De testers voeren exploratieve en contract testen voor de API en DAO verzoeken uit. |

### Test matrix

Testen op domeinregels voor use-case 1 (DR\_CATEGORIE):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **FEATURE** | | | |
| **DOMEINREGEL** | Overzichtspagina | Detailpagina | REST API (Nest.js) | MySQL databank |
| Unieke naam | Getest | Getest | Getest | Getest |

Legende matrix:

* Getest: de tester dient de domeinregel tijdens het testen voor de feature te verifiëren.
* Niet getest: de tester dient de domeinregel niet tijdens het testen voor de feature te verifiëren.

De test is geslaagd.

## Test Use Case 2: Toevoegen categorie

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (2u), Java, React |
| Henri De Bel | 33% (2u), React |
| Christof Remeysen | 33% (2u), React en Java |

### Features

De ontwikkelde features voor deze use cases staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Voeg een categorie toe knop | De overzichtspagina heeft een knop voor het creëren van een categorie die via de REST API wordt opgeslagen in de databank. |
| Categorie creatie pagina | Een pop-up geeft velden weer voor het invullen van : naam, beschrijving en afbeelding. |
| Routing | De routing zorgt ervoor dat je van de overzichtspagina naar de bijbehorende categorie creatie pagina  kan navigeren en omgekeerd. |
| REST API (React)  JPA (Java) | De REST API/JPA ontvangt en behandelt verzoeken van de front-end. De REST API/JPA bevat seeds met mock-data; |
| CDN platform | Het CDN platform bevat de gebruikte afbeeldingen (React). |
| Resources folder | Deze folder bevat de afbeeldingen van een categorie (Java). |
| MySQL databank | Deze bevat de data van een categorie, inclusief afbeeldingen (Java). |
| CSS stijlen | De CSS stijlen zorgen voor de opmaak van een scherm:   * React: Chakra, CSS. * Java: JavaFX, CSS. |
| Alerts | Er worden alerts voorzien om de gebruiker te informeren over het slagen/falen van de uitgevoerde taak. |
| Foutmeldingen | Er worden foutmeldingen voorzien bij de inputvelden volgens de domeinregels. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Voeg een categorie toe knop | Unit en component,  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De testers, waarvan er minstens twee een verschillende rol hebben, klikken op de Voeg een categorie toe knop en controleren of de categorie creatie pop-up wordt weergegeven. Voor React gebeurt dit semiautomatisch met Cypress. |
| Categorie creatie pagina | Unit en component,  Functioneel,  Load of performance | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de Categorie creatie pagina en vult vervolgens alle verplichte velden in. Vervolgens de tester kan kiezen om een categorie toe te voegen door op een voeg toe knop te drukken of te annuleren. Voor React gebeurt dit semiautomatisch met Cypress. |
| Routing | Functioneel  Performance | De functionele testen verlopen exploratief, zowel voor Java als React. De tester controleert of de juiste pagina’s bij navigatie openen.    Voor Java, verifieert de tester de compilatieduur en snelheid van laden op exploratieve wijze. Voor React, zet de tester hiervoor Google Lighthouse en React Profiler in. |
| REST API | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht of de data goed en snel genoeg vanuit de Nest.js API wordt opgeslagen en ingeladen. |
| CDN platform | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht of afbeeldingen goed en snel genoeg vanuit Cloudinary worden ingeladen en opgeslagen. |
| Resources folder | Functioneel  Performance | De tester verifieert exploratief of afbeeldingen goed en snel genoeg vanuit de resources folder op de overzichtspagina inladen. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor Java. De tester verifieert op het zicht of de data en afbeeldingen goed en snel genoeg in de MySQL databank worden opgeslagen. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen. |
| Alerts | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.  De tester verifieert exploratief of de juiste alerts getriggerd worden bij het succesvol of niet succesvol toevoegen van een categorie. |
| Foutmeldingen | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.  De tester verifieert exploratief of de juiste foutmeldingen getriggerd worden bij een foutieve invoer volgens de domeinregels. |

#### Verwacht resultaat

De categorie creatie pagina opent correct. In te vullen velden zoals naam, beschrijving en afbeelding worden correct getoond. Bij het toevoegen van een nieuwe categorie indien alle gegevens werden correct ingevuld zal de nieuwe categorie op de overzichtspagina verschijnen. Bij een foutieve invoer krijgt de tester een foutmelding te zien dat zijn gegevens niet correct zijn. Bij het onsuccesvol of succesvol toevoegen van een categorie krijgt de gebruiker een gepaste alert te zien.

#### Beoordeling resultaat

[In te vullen nadat de test is uitgevoerd]

De foutmeldingen werden niet correct getoond. Dit werd in de code aangepast.

#### Remediëring

[In te vullen nadat de re-test is uitgevoerd]

De resultaten zijn voor het React-gedeelte zoals verwacht.

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* Elke rol kan testen. Authenticatie wordt pas in een volgende use case voorzien.
* De “mock” data is aanwezig in de persistentie laag om de exit criteria te kunnen bereiken.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **Risico** | **Beschrijving** | **Impact** | **Mitigatieplan** |
| Afwezigheid van authenticatie (vertrouwelijkheid) | De data is zichtbaar voor alle gebruikers, ook al hebben ze de juiste rol niet. | Groot | De testers zorgen voor fictieve data. |
| Mock-data (performantie) | De mock-data weerspiegelt geen realistische niveaus inzake laadtijden en performantie. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank uitgevoerd. |
| API-data (consistentie) | De data bevindt zich in een API voor React, welke aanvragen in bepaalde formaten moet blijven verwerken. | Groot | De testers voeren exploratieve en contract testen voor de API calls uit. |

### Test matrix

Testen op domeinregels voor use-case 2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | DOMEINREGEL TEST |  |  |  |
| FEATURES | Categorie unieke naam | Categorie naam, beschrijving, afbeelding zijn verplicht | Categorie naam ,beschrijving, afbeelding mogen niet leeg zijn | De naam bestaat minimaal uit 2 hoogstens 50 karakters.  De beschrijving bestaat uit minimaal 5 hoogstens 250 karakters. |
| Categorie creatie pagina | Word getest | Getest | Getest | Getest |

De testen slagen.

## Test Use Case 3: Wijzigen categorie

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (2u), Java, React |
| Henri De Bel | 33% (2u), React |
| Christof Remeysen | 33% (2u), React en Java |

### Features

De ontwikkelde features voor deze use cases staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Wijzig een categorie knop | De overzichtspagina heeft een knop dat een pop-up aanroept voor het wijzigen van een geselecteerde categorie. |
| Categorie wijzigen pop-up pagina | Een pop-up geeft velden weer voor het wijzigen van : naam, beschrijving en afbeelding. |
| Routing | De routing zorgt ervoor dat je van de overzichtspagina naar de bijbehorende categorie wijzigen pagina  kan navigeren en omgekeerd. |
| REST API (React)  JPA (Java) | De REST API/JPA ontvangt en behandelt verzoeken van de front-end. De REST API/JPA bevat seeds met mock-data; |
| CDN platform | Het CDN platform bevat de gebruikte afbeeldingen (React). |
| Resources folder | Deze folder bevat de afbeeldingen van een categorie (Java). |
| MySQL databank | Deze bevat de data van een categorie, inclusief afbeeldingen (Java). |
| CSS stijlen | De CSS stijlen zorgen voor de opmaak van een scherm:   * React: Chakra. * Java: JavaFX CSS. |
| Alerts | Er worden alerts voorzien om de gebruiker te informeren over het slagen/falen van de uitgevoerde taak. |
| Foutmeldingen | Er worden foutmeldingen voorzien bij de inputvelden volgens de domeinregels. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Wijzig een categorie knop | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De testers, waarvan er minstens twee een verschillende rol hebben, klikken op de wijzig een categorie knop en controleren of de categorie wijzigen pop-up pagina wordt weergegeven. Voor React gebeurt dit semiautomatisch met Cypress. |
| Categorie wijzigen pop-up pagina | Unit en component  Functioneel  Load of performance | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de categorie wijzigen pop-up pagina en wijzigt vervolgens de gewenste velden/en. De tester kan het wijzigen van een categorie doorvoeren of annuleren door op de daarvoor bestemde knop te drukken.  Voor React gebeurt dit semiautomatisch met Cypress. |
| Routing | Functioneel  Performance | De functionele testen verlopen exploratief, zowel voor Java als React. De tester controleert of de juiste pagina’s bij navigatie openen.    Voor Java, verifieert de tester de compilatieduur en snelheid van laden op exploratieve wijze. Voor React, zet de tester hiervoor Google Lighthouse en React Profiler in. |
| REST API | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht of de data goed en snel genoeg vanuit de Nest.js API wordt opgeslagen en ingeladen. |
| CDN platform | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht of afbeeldingen goed en snel genoeg vanuit Cloudinary worden ingeladen en opgeslagen. |
| Resources folder | Functioneel  Performance | De tester verifieert exploratief of afbeeldingen goed en snel genoeg vanuit de resources folder op de overzichtspagina inladen. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor Java. De tester verifieert op het zicht of de data en afbeeldingen goed en snel genoeg in de MySQL databank worden gewijzigd. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen , responsive design enz. |
| Alerts | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.  De tester verifieert exploratief of de juiste alerts getriggerd worden bij het succesvol of niet succesvol wijzigen van een categorie. |
| Foutmeldingen | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.  De tester verifieert exploratief of de juiste foutmeldingen getriggerd worden bij een foutieve invoer volgens de domeinregels. |

#### Verwacht resultaat

De categorie wijzigen pop-up pagina opent correct. De te wijzigen velden zoals naam, beschrijving en afbeelding worden correct getoond. Bij het wijzigen van een categorie indien alle gegevens werden correct ingevuld zal de gewijzigde categorie op de overzichtspagina verschijnen. Bij een foutieve invoer krijgt de tester een foutmelding te zien dat zijn gegevens niet correct zijn. Bij het onsuccesvol of succesvol wijzigen van een categorie krijgt de gebruiker een gepaste alert te zien.

#### Beoordeling resultaat

[In te vullen nadat de test is uitgevoerd]

React: niet alle foutmeldingen werden correct getoond. Dit werd in de code aangepast.

#### Remediëring

[In te vullen nadat de re-test is uitgevoerd]

De resultaten zijn voor het React-gedeelte zoals verwacht.

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* Elke rol kan testen. Authenticatie wordt pas in een volgende use case voorzien.
* De “mock” data is aanwezig in de persistentie laag om de exit criteria te kunnen bereiken.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **Risico** | **Beschrijving** | **Impact** | **Mitigatieplan** |
| Afwezigheid van authenticatie (vertrouwelijkheid) | De data is zichtbaar voor alle gebruikers, ook al hebben ze de juiste rol niet. | Groot | De testers zorgen voor fictieve data. |
| Mock-data (performantie) | De mock-data weerspiegelt geen realistische niveaus inzake laadtijden en performantie. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank uitgevoerd. |
| API-data (consistentie) | De data bevindt zich in een API voor React, welke aanvragen in bepaalde formaten moet blijven verwerken. | Groot | De testers voeren exploratieve en contract testen voor de API calls uit. |

### Test matrix

Testen op domeinregels voor use-case 3:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | DOMEINREGEL TEST |  |  |  |
| FEATURES | Categorie unieke naam | Categorie naam, beschrijving, afbeelding zijn verplicht | Categorie naam ,beschrijving, afbeelding mogen niet leeg zijn | De naam bestaat minimaal uit 2 hoogstens 50 karakters.  De beschrijving bestaat uit minimaal 5 hoogstens 250 karakters. |
| Categorie wijzigen pagina | Word getest | Word getest | Word getest | Getest |

De testen slagen.

## Test Use Case 4: Verwijderen categorie

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (2u), Java, React |
| Henri De Bel | 33% (2u), React |
| Christof Remeysen | 33% (2u), React en Java |

### Features

De ontwikkelde features voor deze use cases staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Categorie verwijderen knop | De overzichtspagina heeft een knop om een geselecteerde categorie te verwijderen. |
| REST API (React)  JPA (Java) | De REST API/JPA ontvangt en behandelt verzoeken van de front-end. De REST API/JPA bevat seeds met mock-data; |
| MySQL databank | Deze bevat de data van een categorie, inclusief afbeeldingen (Java). |
| CSS stijlen | De CSS stijlen zorgen voor de opmaak van een scherm:   * React: Chakra. * Java: JavaFX CSS. |
| Alerts | Er worden alerts voorzien om de gebruiker te informeren over het slagen/falen van de uitgevoerde taak.  Er worden alerts voorzien voor het bevestigen/annuleren van een uit te voeren taak. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Categorie verwijderen knop | Unit en component,  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De testers, waarvan er minstens twee een verschillende rol hebben, klikken op de verwijderen een categorie knop en controleren of dat de gewenste categorie is verwijderd.  Voor React gebeurt dit semiautomatisch met Cypress. |
| REST API | Integratie  Performance | De integratietest verloopt exploratief. De tester verifieert op het zicht of de data goed en snel genoeg vanuit de Nest.js API wordt verwijderd en ingeladen. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor Java. De tester verifieert op het zicht of de een categorie goed en snel genoeg uit de MySQL databank wordt verwijderd. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen, responsive design enz. |
| Alerts | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.  De tester verifieert exploratief of de juiste alerts getriggerd worden bij het succesvol of niet succesvol verwijderen van een categorie.  Er wordt een alert voorzien om te bevestigen/annuleren dat de gebruiker een categorie wilt verwijderen. |

#### Verwacht resultaat

Bij het verwijderen van een categorie indien de gebruiker dit taak bevestigd zal de verwijderde categorie niet meer verschijnen op de categorie overzichtspagina. De gebruiker ziet een gepaste alert verschijnen.

Beoordeling resultaat

[In te vullen nadat de test is uitgevoerd]

De resultaten zijn voor het React-gedeelte zoals verwacht.

#### Remediëring

[In te vullen nadat de re-test is uitgevoerd]

/

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* Elke rol kan testen. Authenticatie wordt pas in een volgende use case voorzien.
* De “mock” data is aanwezig in de persistentie laag om de exit criteria te kunnen bereiken.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **Risico** | **Beschrijving** | **Impact** | **Mitigatieplan** |
| Afwezigheid van authenticatie (vertrouwelijkheid) | De data is zichtbaar voor alle gebruikers, ook al hebben ze de juiste rol niet. | Groot | De testers zorgen voor fictieve data. |
| Mock-data (performantie) | De mock-data weerspiegelt geen realistische niveaus inzake laadtijden en performantie. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank uitgevoerd. |
| API-data (consistentie) | De data bevindt zich in een API voor React, welke aanvragen in bepaalde formaten moet blijven verwerken. | Groot | De testers voeren exploratieve en contract testen voor de API calls uit. |

### Test matrix

Testen op domeinregels voor use-case 4:

|  |  |
| --- | --- |
|  | DOMEINREGEL TEST |
| FEATURES | / |
| / | / |

De testen slagen.

## Test Use Case 5: Registreren gebruiker

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (2u), Java, React |
| Henri De Bel | 33% (2u), React |
| Christof Remeysen | 33% (2u), React en Java |

### Features

De ontwikkelde features voor deze use cases staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Registratie pop-up pagina | Een pop-up geeft velden weer voor het invullen van : naam, wachtwoord familienaam en voornaam. |
| Routing | De routing zorgt ervoor dat je van de registratiepagina naar de info pagina kan navigeren en omgekeerd. |
| REST API (React)  JPA (Java) | De REST API/JPA ontvangt en behandelt verzoeken van de front-end. De REST API/JPA bevat seeds met mock-data; |
| MySQL databank | Deze bevat de data van een gebruiker (Java) (React). |
| CSS stijlen | De CSS stijlen zorgen voor de opmaak van een scherm:   * React: Chakra, CSS. * Java: JavaFX, CSS. |
| Alerts | Er worden alerts voorzien om de gebruiker te informeren over het slagen/falen van de uitgevoerde taak. |
| Foutmeldingen | Er worden foutmeldingen voorzien bij de inputvelden volgens de domeinregels. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Registratie pop-up pagina | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de registratie pop-up pagina en vult vervolgens alle verplichte velden in. Vervolgens de tester kan kiezen om zich te registreren of te annuleren. Voor React gebeurt dit semiautomatisch met Cypress. |
| Routing | Functioneel  Performance | De functionele testen verlopen exploratief, zowel voor Java als React. De tester controleert of de juiste pagina’s bij navigatie openen.    Voor Java, verifieert de tester de compilatieduur en snelheid van laden op exploratieve wijze. Voor React, zet de tester hiervoor Google Lighthouse en React Profiler in. |
| REST API | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De REST API (Nest.js) zorgt voor het opslaan van een nieuwe registratie van een gebruiker in de databank (React).  De tester verifieert op het zicht of de registratie verzoeken goed en snel genoeg door de Nest.js API worden uitgevoerd. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor Java. De tester verifieert op het zicht of de data goed en snel genoeg in de MySQL databank worden opgeslagen. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen, responsive design. |
| Alerts | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.  De tester verifieert exploratief of de juiste alerts getriggerd worden bij het succesvol of niet succesvol registreren van een tester. |
| Foutmeldingen | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.  De tester verifieert exploratief of de juiste foutmeldingen getriggerd worden bij een foutieve invoer volgens de domeinregels. |

#### Verwacht resultaat

In te vullen velden zoals gebruikersnaam, wachtwoord, familienaam en voornaam worden correct getoond. Bij een succesvolle registratie zal de tester automatisch aangemeld worden en indien gewenst kan die de dashboard raadplegen. Bij een foutieve invoer krijgt de tester een foutmelding te zien dat zijn gegevens niet correct zijn. Bij een onsuccesvolle of succesvolle registratie krijgt de tester een gepaste alert te zien.

#### Beoordeling resultaat

[In te vullen nadat de test is uitgevoerd]

De foutmeldingen werden niet correct getoond. Dit werd in de code aangepast.

#### Remediëring

[In te vullen nadat de re-test is uitgevoerd]

De resultaten zijn voor het React-gedeelte zoals verwacht.

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* Authenticatie is voorzien. De tester krijgt een default gebruikersrol.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **Risico** | **Beschrijving** | **Impact** | **Mitigatieplan** |
| API-data (consistentie) | De data bevindt zich in een API voor React, welke aanvragen in bepaalde formaten moet blijven verwerken. | Groot | De testers voeren exploratieve en contract testen voor de API calls uit. |

### Test matrix

Testen op domeinregels voor use-case 5:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | DOMEINREGEL TEST |  |  |  |
| FEATURES | Gebruikersnaam is uniek en moet bestaan uit minstens 2 en hoogstens 20 karakters | Wachtwoord moet bestaan uit minstens 5 en hoogstens 20 karakters | Voornaam moet bestaan uit minstens 2 en hoogstens 30 karakters | Familienaam moet bestaan uit minstens 2 en hoogstens 30 karakters |
| Registratie pop-up pagina | Word getest | Getest | Getest | Getest |

## Test Use Case 6: Aanmelden gebruiker

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (2u), Java, React |
| Henri De Bel | 33% (2u), React |
| Christof Remeysen | 33% (2u), React en Java |

### Features

De ontwikkelde features voor deze use cases staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Login pop-up pagina | Een pop-up geeft velden weer voor het invullen van : gebruikersnaam en wachtwoord. |
| Routing | De routing zorgt ervoor dat je van de login naar de info pagina kan navigeren en omgekeerd. |
| REST API (React)  JPA (Java) | De REST API/JPA ontvangt en behandelt verzoeken van de front-end. De REST API/JPA bevat seeds met mock-data; |
| MySQL databank | Deze bevat de data van een gebruiker (Java) (React). |
| CSS stijlen | De CSS stijlen zorgen voor de opmaak van een scherm:   * React: Chakra, CSS. * Java: JavaFX, CSS. |
| Alerts | Er worden alerts voorzien om de gebruiker te informeren over het slagen/falen van de uitgevoerde taak. |
| Foutmeldingen | Er worden foutmeldingen voorzien bij de inputvelden volgens de domeinregels. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Login pop-up pagina | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de login pop-up pagina en vult vervolgens alle verplichte velden in. Vervolgens de tester kan kiezen om zich aan te melden of te annuleren. Voor React gebeurt dit semiautomatisch met Cypress. |
| Routing | Functioneel  Performance | De functionele testen verlopen exploratief, zowel voor Java als React. De tester controleert of de juiste pagina’s bij navigatie openen.    Voor Java, verifieert de tester de compilatieduur en snelheid van laden op exploratieve wijze. Voor React, zet de tester hiervoor Google Lighthouse en React Profiler in. |
| REST API (React)  JPA (Java) | Integratie  Performance | De integratietest verloopt exploratief. De tester verifieert op het zicht of de login verzoeken goed en snel genoeg door de Nest.js API worden uitgevoerd.  De REST API (Nest.js) stuurt een verzoek naar databank voor het ophalen van een bestaande gebruiker tijdens het aanmelden. (React) (JPA voor Java). |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. De tester verifieert op het zicht of de data goed en snel vanuit de databank wordt opgehaald. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen, responsive design. |
| Alerts | Unit en component  Functioneel | De tester verifieert exploratief of de juiste alerts getriggerd worden bij het succesvol of niet succesvol aanmelden van een tester. |
| Foutmeldingen | Unit en component  Functioneel | De tester verifieert exploratief of de juiste foutmeldingen getriggerd worden bij een foutieve invoer volgens de domeinregels. |

#### Verwacht resultaat

In te vullen velden zoals gebruikersnaam en wachtwoord worden correct getoond. Bij het succesvol aanmelden zal de tester indien gewenst de dashboard raadplegen. Bij een foutieve invoer krijgt de tester een foutmelding te zien dat zijn gegevens niet correct zijn. Bij het onsuccesvol of succesvol aanmelden krijgt de tester een gepaste alert te zien.

#### Beoordeling resultaat

[In te vullen nadat de test is uitgevoerd]

De foutmeldingen werden niet correct getoond. Dit werd in de code aangepast.

#### Remediëring

[In te vullen nadat de re-test is uitgevoerd]

De resultaten zijn voor het React-gedeelte zoals verwacht.

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* Authenticatie is voorzien.
* De databank werd gevuld met mock-data van bestaande gebruiker.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **Risico** | **Beschrijving** | **Impact** | **Mitigatieplan** |
| API-data (consistentie) | De mock-data bevindt zich in een API voor React, welke aanvragen in bepaalde formaten moet blijven verwerken. Als een functionaliteit aangepast wordt kan de mock-data | Groot | De testers voeren exploratieve en contract testen voor de API calls uit. |
| Mock-data (performantie) | De mock-data weerspiegelt geen realistische niveaus inzake laadtijden en performantie. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank uitgevoerd. |

### Test matrix

Testen op domeinregels voor use-case 6:

|  |  |  |
| --- | --- | --- |
|  | DOMEINREGEL TEST |  |
| FEATURES | Gebruikersnaam is uniek en moet bestaan uit minstens 2 en hoogstens 20 karakters | Wachtwoord moet bestaan uit minstens 5 en hoogstens 20 karakters |
| Aanmelden pop-up pagina | Getest | Getest |

## Test Use Case 7: Aanpassen gebruikersrol

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (2u), Java, React |
| Henri De Bel | 33% (2u), React |
| Christof Remeysen | 33% (2u), React en Java |

### Features

De ontwikkelde features voor deze use cases staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Aanpassen gebruikersrol pagina | Een admin kan een bestaande gebruiker kiezen uit een lijst met alle gebruikers en zijn rol aanpassen. |
| Routing | De routing zorgt ervoor dat je van de dashboard naar de aanpassen gebruikersrol pagina kan navigeren en omgekeerd. |
| REST API (React)  JPA (Java) | De REST API/JPA ontvangt en behandelt verzoeken van de front-end. De REST API/JPA bevat seeds met mock-data; |
| MySQL databank | Deze bevat de data van een gebruiker (Java) (React). |
| CSS stijlen | De CSS stijlen zorgen voor de opmaak van een scherm:   * React: Chakra, CSS. * Java: JavaFX, CSS. |
| Alerts | Er worden alerts voorzien om de gebruiker te informeren over het slagen/falen van de uitgevoerde taak. |
| Foutmeldingen | Er worden foutmeldingen voorzien bij de inputvelden volgens de domeinregels. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Aanpassen gebruikersrol pagina | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de aanpassen gebruikersrol pagina  en selecteert een gebruiker. Vervolgens kan de tester de rol van de gekozen gebruiker wijzigen. Voor React gebeurt dit semiautomatisch met Cypress. |
| Routing | Functioneel  Performance | De functionele testen verlopen exploratief, zowel voor Java als React. De tester controleert of de juiste pagina’s bij navigatie openen.    Voor Java, verifieert de tester de compilatieduur en snelheid van laden op exploratieve wijze. Voor React, zet de tester hiervoor Google Lighthouse en React Profiler in. |
| REST API (React)  JPA (Java) | Integratie  Performance | De integratietest verloopt exploratief. De REST API (Nest.js) stuurt een verzoek naar de databank voor het ophalen van een lijst met bestaande gebruikers. (React) (JPA voor Java).  De tester verifieert op het zicht of de aanpassen gebruikersrol verzoeken goed en snel genoeg door de Nest.js API worden uitgevoerd. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. De tester verifieert op het zicht of de data goed en snel vanuit de databank wordt opgehaald. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen, responsive design. |
| Alerts | Unit en component  Functioneel | De tester verifieert exploratief of de juiste alerts getriggerd worden bij het succesvol of niet succesvol wijzigen van een gebruikersrol. |

#### Verwacht resultaat

Bij het onsuccesvol of succesvol wijziging van een rol zal de tester een gepaste alert zien.

#### Beoordeling resultaat

[In te vullen nadat de test is uitgevoerd]

/

#### Remediëring

[In te vullen nadat de re-test is uitgevoerd]

/

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* Authenticatie is voorzien.
* De databank werd gevuld met mock-data van bestaande gebruikers.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **Risico** | **Beschrijving** | **Impact** | **Mitigatieplan** |
| API-data (consistentie) | De mock-data bevindt zich in een API voor React, welke aanvragen in bepaalde formaten moet blijven verwerken. Als een functionaliteit aangepast wordt kan de mock-data | Groot | De testers voeren exploratieve en contract testen voor de API calls uit. |
| Mock-data (performantie) | De mock-data weerspiegelt geen realistische niveaus inzake laadtijden en performantie. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank uitgevoerd. |

### Test matrix

Testen op domeinregels voor use-case 7:

|  |  |  |  |
| --- | --- | --- | --- |
|  | DOMEINREGEL TEST |  |  |
| FEATURES | Tester heeft een Admin rol | Minimaal 1 andere niet admin gebruiker | Een admin kan niet de rol van een adere admin wijzigen |
| Aanpassen gebruikersrol pagina | Wordt getest | Wordt getest | Wordt getest |

## Test Use Case 8: Raadplegen datasource

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (30 minuten), React |
| Henri De Bel | 33% (30 minuten), Java |
| Christof Remeysen | 33% (30 minuten), Java en React |

### Features

De ontwikkelde features voor deze use case staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Detailpagina | Deze geeft de details van de datasource weer: naam. |
| Accordeon | Deze bevat de informatie van elke datasource die tot een doelstelling behoort. |
| Data viewer | Deze toont de gegevens van de dataset: naam, type, data. |
| REST API (Nest.js) | Deze bevat de data van een datasource (React). |
| MySQL databank | Deze bevat de data van een datasource (Java). |
| CSS stijlen | Deze zorgen voor de opmaak van een scherm:   * React: Chakra, CSS. * Java: JavaFX, CSS. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Detailpagina | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de detailpagina en controleert deze. Voor React gebeurt dit semiautomatisch met Cypress. |
| Accordeon | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.  De functionele testen verlopen exploratief, zowel voor Java als React. De tester klikt op een accordeon-item en controleert de getoonde info. Voor React gebeurt dit semiautomatisch met Cypress. |
| Data viewer | Functioneel | De functionele testen verlopen exploratief, zowel voor Java als React. De tester selecteert een datasource op de detailpagina en controleert de gegevens op het zicht. |
| REST API (Nest.js) | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht en via Postman of de data goed en snel genoeg vanuit de Nest.js API inlaadt. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor Java. De tester verifieert op het zicht of de data goed en snel genoeg vanuit de MySQL databank inlaadt. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen en leesbaarheid van tekst. |

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* De “mock” data is aanwezig in de persistentie laag om de exit criteria te kunnen bereiken.
* Elke rol kan testen. Authenticatie wordt pas in volgende use cases (UC5 en UC6) voorzien.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Risico** | **Impact** | **Mitigatieplan** |
| Afwezigheid van authenticatie (vertrouwelijkheid) | De data is zichtbaar voor alle gebruikers, ook al hebben ze de juiste rol niet. | Groot | De testers zorgen voor fictieve data. Zodra de applicatie live kan, testen we authenticatie en de zichtbaarheid van features voor rollen in detail. |
| Capaciteit en mock-data (performantie) | Het aantal gelijktijdige verzoeken voor de REST API en databank en de hoeveelheid mock-data weerspiegelen geen realistische niveaus inzake laadtijden en performantie. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank en -API uitgevoerd. |
| Dataformaat (consistentie) | De data bevindt zich in een API voor React en in een databank voor Java, welke aanvragen in bepaalde formaten moeten blijven verwerken. | Groot | De testers voeren exploratieve en contract testen voor de API en DAO verzoeken uit. |

### Test matrix

Testen op domeinregels voor use case 8 (DR\_DATASOURCE):

|  |  |  |  |
| --- | --- | --- | --- |
|  | **FEATURE** | | |
| **DOMEINREGEL** | Detailpagina met accordeon | REST API (Nest.js) | MySQL databank |
| Unieke naam | Getest | Getest | Getest |

Legende matrix:

* Getest: de tester dient de domeinregel tijdens het testen voor de feature te verifiëren.
* Niet getest: de tester dient de domeinregel niet tijdens het testen voor de feature te verifiëren.

De test is geslaagd.

## Test Use Case 9: Toevoegen datasource

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (30 minuten), Java en React |
| Henri De Bel | 33% (30 minuten), React |
| Christof Remeysen | 33% (30 minuten), Java |

### Features

De ontwikkelde features voor deze use case staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Detailpagina | Deze geeft de details van de datasource weer: naam. |
| Call-to-action | Deze nodigt de gebruiker uit om een datasource aan een doelstelling toe te voegen. |
| Formulier | Dit biedt de functionaliteit aan om een datasource aan te maken. Deze velden moeten worden ingevuld om het formulier geldig in te dienen: naam, type, data. Het formulier bevat een annuleer-functie om terug naar de detailpagina te keren. |
| Validatieboodschap | Deze geeft feedback inzake aanmaak van de datasource. |
| REST API (Nest.js) | Deze bevat een HTTP POST methode om een datasource toe te voegen (React). |
| MySQL databank | Deze luistert naar een DAO methode om een datasource toe te voegen (Java), welke via de datasource repository vanuit de Domeincontroller in de GUI wordt aangesproken. |
| CSS stijlen | Deze zorgen voor de opmaak van een scherm:   * React: Chakra. * Java: JavaFX CSS. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Detailpagina | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de detailpagina en controleert deze. Voor React gebeurt dit semiautomatisch met Cypress. |
| Call-to-action | Functioneel | De functionele testen verlopen exploratief, zowel voor Java als React. De tester klikt op de call-to-action en controleert of het formulier goed verschijnt. Voor React gebeurt dit semiautomatisch met Cypress. |
| Formulier | Functioneel | De functionele testen verlopen exploratief, zowel voor Java als React. Eerst tracht de tester het formulier ongeldig in te dienen en keert hij terug. Hierna doet hij het op een geldige manier. Voor React gebeurt dit semiautomatisch met Cypress. |
| Validatieboodschap | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.  De functionele testen verlopen exploratief, zowel voor Java als React. De tester checkt of de boodschappen correct verschijnen wanneer het formulier ongeldig en geldig wordt ingediend. Voor React gebeurt dit semiautomatisch met Cypress. |
| REST API (Nest.js) | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht of de data enkel na geldige indiening van het formulier wordt opgeslagen. Daarnaast checkt hij of de Nest.js API snel genoeg op het POST verzoek reageert. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor Java. De tester verifieert op het zicht of de data enkel na geldige indiening van het formulier wordt opgeslagen. Daarnaast checkt hij of de MySQL databank snel genoeg op het DAO verzoek reageert. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen en leesbaarheid van de formuliervelden. |

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* De “mock” data is aanwezig in de persistentie laag om de exit criteria te kunnen bereiken.
* Elke rol kan testen. Authenticatie wordt pas in volgende use cases (UC5 en UC6) voorzien.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Risico** | **Impact** | **Mitigatieplan** |
| Afwezigheid van authenticatie (vertrouwelijkheid) | Elke gebruiker kan een datasource toevoegen, ook al heeft hij de juiste rol niet. | Groot | We aanvaarden het risico voor staging. Zodra de applicatie live kan, testen we authenticatie en de zichtbaarheid van features voor rollen in detail. |
| Capaciteit (performantie) | Tijdens het testen op staging, zijn het aantal gelijktijdige verzoeken voor de REST API en databank niet zoals in de realiteit. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank en -API uitgevoerd. |
| Dataformaat (consistentie) | De data bevindt zich in een API voor React en in een databank voor Java, welke aanvragen in bepaalde formaten moeten blijven verwerken. | Groot | De testers voeren exploratieve en contract testen voor de API en DAO verzoeken uit. |

### Test matrix

Testen op domeinregels voor use case 9 (DR\_DATASOURCE):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **FEATURE** | | | |
| **DOMEINREGEL** | Detailpagina met accordeon | Formulier met validatie | REST API (Nest.js) | MySQL databank |
| Unieke naam | Getest | Getest | Getest | Getest |
| Naam <= 50 tekens | Niet getest | Getest | Getest | Getest |
| Type IN [Bar, Doughnut, Line] | Niet getest | Getest | Getest | Getest |
| Alfanumerieke tekens | Getest | Getest | Getest | Getest |
| Maximum 1 doelstelling | Niet getest | Getest | Getest | Getest |
| Alle velden verplicht | Niet getest | Getest | Getest | Getest |

Legende matrix:

* Getest: de tester dient de domeinregel tijdens het testen voor de feature te verifiëren.
* Niet getest: de tester dient de domeinregel niet tijdens het testen voor de feature te verifiëren.

De test is geslaagd.

## Test Use Case 10: Wijzigen datasource

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (30 minuten), Java |
| Henri De Bel | 33% (30 minuten), Java en React |
| Christof Remeysen | 33% (30 minuten), React |

### Features

De ontwikkelde features voor deze use case staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Detailpagina | Deze geeft de details van de datasource weer: naam. |
| Call-to-action | Deze nodigt de gebruiker uit om een datasource via het formulier te wijzigen. |
| Formulier | Dit biedt de functionaliteit aan om een datasource te wijzigen. Deze velden moeten worden ingevuld om het formulier geldig in te dienen: naam, type, data. Het formulier bevat een annuleer-functie om terug naar de detailpagina te keren. |
| Validatieboodschap | Deze geeft feedback inzake wijziging van de datasource. |
| REST API (Nest.js) | Deze bevat een HTTP PUT methode om een datasource te wijzigen (React). |
| MySQL databank | Deze luistert naar een DAO methode om een datasource te wijzigen (Java), welke via de datasource repository vanuit de Domeincontroller in de GUI wordt aangesproken. |
| CSS stijlen | Deze zorgen voor de opmaak van een scherm:   * React: Chakra. * Java: JavaFX CSS. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Detailpagina | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de detailpagina en controleert deze. Voor React gebeurt dit semiautomatisch met Cypress. |
| Call-to-action | Functioneel | De functionele testen verlopen exploratief, zowel voor Java als React. De tester klikt op de call-to-action en controleert of het formulier goed verschijnt. Voor React gebeurt dit semiautomatisch met Cypress. |
| Formulier | Functioneel | De functionele testen verlopen exploratief, zowel voor Java als React. Eerst tracht de tester het formulier ongeldig in te dienen en keert hij terug. Hierna wijzigt hij elk veld op een geldige manier. Voor React gebeurt dit semiautomatisch met Cypress. |
| Validatieboodschap | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.  De functionele testen verlopen exploratief, zowel voor Java als React. De tester checkt of de boodschappen correct verschijnen wanneer het formulier ongeldig en geldig wordt ingediend. Voor React gebeurt dit semiautomatisch met Cypress. |
| REST API (Nest.js) | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht of de data enkel na geldige indiening van het formulier wordt opgeslagen. Daarnaast checkt hij of de Nest.js API snel genoeg op het PUT verzoek reageert. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor Java. De tester verifieert op het zicht of de data enkel na geldige indiening van het formulier wordt opgeslagen. Daarnaast checkt hij of de MySQL databank snel genoeg op het DAO verzoek reageert. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen en leesbaarheid van de formuliervelden. |

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* De “mock” data is aanwezig in de persistentie laag om de exit criteria te kunnen bereiken.
* Elke rol kan testen. Authenticatie wordt pas in volgende use cases (UC5 en UC6) voorzien.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Risico** | **Impact** | **Mitigatieplan** |
| Afwezigheid van authenticatie (vertrouwelijkheid) | Elke gebruiker kan een datasource wijzigen, ook al heeft hij de juiste rol niet. | Groot | We aanvaarden het risico voor staging. Zodra de applicatie live kan, testen we authenticatie en de zichtbaarheid van features voor rollen in detail. |
| Capaciteit en mock-data (performantie) | Het aantal gelijktijdige verzoeken voor de REST API en databank en de hoeveelheid mock-data weerspiegelen geen realistische niveaus inzake laadtijden en performantie. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank en -API uitgevoerd. |
| Dataformaat (consistentie) | De data bevindt zich in een API voor React en in een databank voor Java, welke aanvragen in bepaalde formaten moeten blijven verwerken. | Groot | De testers voeren exploratieve en contract testen voor de API en DAO verzoeken uit. |

### Test matrix

Testen op domeinregels voor use case 10 (DR\_DATASOURCE):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **FEATURE** | | | |
| **DOMEINREGEL** | Detailpagina met accordeon | Formulier met validatie | REST API (Nest.js) | MySQL databank |
| Unieke naam | Getest | Getest | Getest | Getest |
| Naam <= 50 tekens | Niet getest | Getest | Getest | Getest |
| Type IN [Bar, Doughnut, Line] | Niet getest | Getest | Getest | Getest |
| Alfanumerieke tekens | Getest | Getest | Getest | Getest |
| Maximum 1 doelstelling | Niet getest | Getest | Getest | Getest |
| Alle velden verplicht | Niet getest | Getest | Getest | Getest |

Legende matrix:

* Getest: de tester dient de domeinregel tijdens het testen voor de feature te verifiëren.
* Niet getest: de tester dient de domeinregel niet tijdens het testen voor de feature te verifiëren.

De test is geslaagd.

## Test Use Case 11: Verwijderen datasource

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (30 minuten), Java |
| Henri De Bel | 33% (30 minuten), React |
| Christof Remeysen | 33% (30 minuten), Java en React |

### Features

De ontwikkelde features voor deze use case staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Detailpagina | Deze geeft de details van de datasource weer: naam. |
| Call-to-action | Deze nodigt de gebruiker uit om een datasource te verwijderen. |
| Confirmatie pop-up | Deze vraagt aan de gebruiker of hij zeker is dat de datasource mag worden verwijderd. |
| REST API (Nest.js) | Deze bevat de data van een datasource (React). |
| MySQL databank | Deze bevat de data van een datasource (Java). |
| CSS stijlen | Deze zorgen voor de opmaak van een scherm:   * React: Chakra. * Java: JavaFX CSS. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Detailpagina | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de detailpagina en controleert deze. Voor React gebeurt dit semiautomatisch met Cypress. |
| Call-to-action | Functioneel | De functionele testen verlopen exploratief, zowel voor Java als React. De tester klikt op de call-to-action en controleert of de confirmatie pop-up goed verschijnt. Voor React gebeurt dit semiautomatisch met Cypress. |
| Confirmatie pop-up | Functioneel | De functionele testen verlopen exploratief, zowel voor Java als React. De tester klikt afwisselend op de annuleer- en bevestig-functie. Voor React gebeurt dit semiautomatisch met Cypress. |
| REST API (Nest.js) | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht en via Postman of de data goed en snel genoeg in de Nest.js API wordt verwijderd. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor Java. De tester verifieert op het zicht of de data goed en snel genoeg in de MySQL databank wordt verwijderd. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen en leesbaarheid van tekst. |

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* De “mock” data is aanwezig in de persistentie laag om de exit criteria te kunnen bereiken.
* Elke rol kan testen. Authenticatie wordt pas in volgende use cases (UC5 en UC6) voorzien.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Risico** | **Impact** | **Mitigatieplan** |
| Afwezigheid van authenticatie (vertrouwelijkheid) | Elke gebruiker kan een datasource verwijderen, ook al heeft hij de juiste rol niet. | Groot | We aanvaarden het risico voor staging. Zodra de applicatie live kan, testen we authenticatie en de zichtbaarheid van features voor rollen in detail. |
| Capaciteit en mock-data (performantie) | Het aantal gelijktijdige verzoeken voor de REST API en databank en de hoeveelheid mock-data weerspiegelen geen realistische niveaus inzake laadtijden en performantie. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank en -API uitgevoerd. |
| Dataformaat (consistentie) | De data bevindt zich in een API voor React en in een databank voor Java, welke aanvragen in bepaalde formaten moeten blijven verwerken. | Groot | De testers voeren exploratieve en contract testen voor de API en DAO verzoeken uit. |

### Test matrix

Testen op domeinregels voor use case 11 (DR\_DATASOURCE):

|  |  |  |  |
| --- | --- | --- | --- |
|  | **FEATURE** | | |
| **DOMEINREGEL** | Detailpagina met accordeon | REST API (Nest.js) | MySQL databank |
| Unieke naam | Getest | Getest | Getest |

Legende matrix:

* Getest: de tester dient de domeinregel tijdens het testen voor de feature te verifiëren.
* Niet getest: de tester dient de domeinregel niet tijdens het testen voor de feature te verifiëren.

De test is geslaagd.

## Test Use Case 12: Raadplegen MVO-doelstelling

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (30 minuten), React |
| Henri De Bel | 33% (30 minuten), Java |
| Christof Remeysen | 33% (30 minuten), Java en React |

### Features

De ontwikkelde features voor deze use case staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Detailpagina | Deze geeft de details van de doelstelling weer: naam, beschrijving, status, SDGs[[1]](#footnote-2) entiteiten, datasource. |
| Accordeon | Deze bevat de informatie van elke doelstelling die tot een categorie behoort. |
| REST API (Nest.js) | Deze bevat de data van een doelstelling en een datasource (React). |
| MySQL databank | Deze bevat de data van een doelstelling en een datasource (Java). |
| CSS stijlen | Deze zorgen voor de opmaak van een scherm:   * React: Chakra. * Java: JavaFX CSS. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Detailpagina | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de detailpagina en controleert deze. Voor React gebeurt dit semiautomatisch met Cypress. |
| Accordeon | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.  De functionele testen verlopen exploratief, zowel voor Java als React. De tester klikt op een accordeon-item en controleert de getoonde info. Voor React gebeurt dit semiautomatisch met Cypress. |
| REST API (Nest.js) | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht en via Postman of de data goed en snel genoeg vanuit de Nest.js API inlaadt. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor Java. De tester verifieert op het zicht of de data goed en snel genoeg vanuit de MySQL databank inlaadt. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen en leesbaarheid van tekst. |

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* De “mock” data is aanwezig in de persistentie laag om de exit criteria te kunnen bereiken.
* Elke rol kan testen. Authenticatie wordt pas in volgende use cases (UC5 en UC6) voorzien.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Risico** | **Impact** | **Mitigatieplan** |
| Afwezigheid van authenticatie (vertrouwelijkheid) | De data is zichtbaar voor alle gebruikers, ook al hebben ze de juiste rol niet. | Groot | De testers zorgen voor fictieve data. Zodra de applicatie live kan, testen we authenticatie en de zichtbaarheid van features voor rollen in detail. |
| Capaciteit en mock-data (performantie) | Het aantal gelijktijdige verzoeken voor de REST API en databank en de hoeveelheid mock-data weerspiegelen geen realistische niveaus inzake laadtijden en performantie. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank en -API uitgevoerd. |
| Dataformaat (consistentie) | De data bevindt zich in een API voor React en in een databank voor Java, welke aanvragen in bepaalde formaten moeten blijven verwerken. | Groot | De testers voeren exploratieve en contract testen voor de API en DAO verzoeken uit. |

### Test matrix

Testen op domeinregels voor use case 12 (DR\_DOELSTELLING):

|  |  |  |  |
| --- | --- | --- | --- |
|  | **FEATURE** | | |
| **DOMEINREGEL** | Detailpagina met accordeon | REST API (Nest.js) | MySQL databank |
| Unieke naam | Getest | Getest | Getest |

Legende matrix:

* Getest: de tester dient de domeinregel tijdens het testen voor de feature te verifiëren.
* Niet getest: de tester dient de domeinregel niet tijdens het testen voor de feature te verifiëren.

De test is geslaagd.

## Test Use Case 13: Toevoegen MVO-doelstelling

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (30 minuten), Java en React |
| Henri De Bel | 33% (30 minuten), React |
| Christof Remeysen | 33% (30 minuten), Java |

### Features

De ontwikkelde features voor deze use case staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Detailpagina | Deze geeft de details van de doelstelling weer: naam, beschrijving, status, SDGs entiteiten, datasource. |
| Call-to-action | Deze nodigt de gebruiker uit om een doelstelling via het formulier toe te voegen. |
| Formulier | Dit biedt de functionaliteit aan om een doel aan te maken. Deze velden moeten worden ingevuld om het formulier geldig in te dienen: naam, beschrijving, status, SDGs entiteiten. Het formulier bevat een annuleer-functie om terug naar de detailpagina te keren. |
| Validatieboodschap | Deze geeft feedback inzake aanmaak van de doelstelling. |
| REST API (Nest.js) | Deze bevat een HTTP POST methode om een doelstelling toe te voegen (React). |
| MySQL databank | Deze luistert naar een DAO methode om een doelstelling toe te voegen (Java), welke via de doelstelling repository vanuit de Domeincontroller in de GUI wordt aangesproken. |
| CSS stijlen | Deze zorgen voor de opmaak van een scherm:   * React: Chakra. * Java: JavaFX CSS. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Detailpagina | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de detailpagina en controleert deze. Voor React gebeurt dit semiautomatisch met Cypress. |
| Call-to-action | Functioneel | De functionele testen verlopen exploratief, zowel voor Java als React. De tester klikt op de call-to-action en controleert of het formulier goed verschijnt. Voor React gebeurt dit semiautomatisch met Cypress. |
| Formulier | Functioneel | De functionele testen verlopen exploratief, zowel voor Java als React. Eerst tracht de tester het formulier ongeldig in te dienen en keert hij terug. Hierna doet hij het op een geldige manier. Voor React gebeurt dit semiautomatisch met Cypress. |
| Validatieboodschap | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.  De functionele testen verlopen exploratief, zowel voor Java als React. De tester checkt of de boodschappen correct verschijnen wanneer het formulier ongeldig en geldig wordt ingediend. Voor React gebeurt dit semiautomatisch met Cypress. |
| REST API (Nest.js) | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht of de data enkel na geldige indiening van het formulier wordt opgeslagen. Daarnaast checkt hij of de Nest.js API snel genoeg op het POST verzoek reageert. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor Java. De tester verifieert op het zicht of de data enkel na geldige indiening van het formulier wordt opgeslagen. Daarnaast checkt hij of de MySQL databank snel genoeg op het DAO verzoek reageert. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen en leesbaarheid van de formuliervelden. |

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* De “mock” data is aanwezig in de persistentie laag om de exit criteria te kunnen bereiken.
* Elke rol kan testen. Authenticatie wordt pas in volgende use cases (UC5 en UC6) voorzien.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Risico** | **Impact** | **Mitigatieplan** |
| Afwezigheid van authenticatie (vertrouwelijkheid) | Elke gebruiker kan een doelstelling toevoegen, ook al heeft hij de juiste rol niet. | Groot | We aanvaarden het risico voor staging. Zodra de applicatie live kan, testen we authenticatie en de zichtbaarheid van features voor rollen in detail. |
| Capaciteit (performantie) | Tijdens het testen op staging, zijn het aantal gelijktijdige verzoeken voor de REST API en databank niet zoals in de realiteit. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank en -API uitgevoerd. |
| Dataformaat (consistentie) | De data bevindt zich in een API voor React en in een databank voor Java, welke aanvragen in bepaalde formaten moeten blijven verwerken. | Groot | De testers voeren exploratieve en contract testen voor de API en DAO verzoeken uit. |

### Test matrix

Testen op domeinregels voor use case 13 (DR\_DOELSTELLING):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **FEATURE** | | | |
| **DOMEINREGEL** | Detailpagina met accordeon | Formulier met validatie | REST API (Nest.js) | MySQL databank |
| Unieke naam | Getest | Getest | Getest | Getest |
| Naam <= 50 tekens | Niet getest | Getest | Getest | Getest |
| Beschrijving <= 250 tekens | Niet getest | Getest | Getest | Getest |
| Alfanumerieke tekens | Getest | Getest | Getest | Getest |
| 1 categorie | Niet getest | Getest | Getest | Getest |
| Maximum 1 datasource | Niet getest | Niet getest | Niet getest | Niet getest |
| Alle velden verplicht | Niet getest | Getest | Getest | Getest |

Legende matrix:

* Getest: de tester dient de domeinregel tijdens het testen voor de feature te verifiëren.
* Niet getest: de tester dient de domeinregel niet tijdens het testen voor de feature te verifiëren.

De test is geslaagd. De domeinregel “<= 1 datasource” wordt in een volgend testplan opgenomen.

## Test Use Case 14: Wijzigen MVO-doelstelling

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (30 minuten), Java |
| Henri De Bel | 33% (30 minuten), Java en React |
| Christof Remeysen | 33% (30 minuten), React |

### Features

De ontwikkelde features voor deze use case staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Detailpagina | Deze geeft de details van de doelstelling weer: naam, beschrijving, status, SDGs entiteiten, datasource. |
| Call-to-action | Deze nodigt de gebruiker uit om een doelstelling via het formulier te wijzigen. |
| Formulier | Dit biedt de functionaliteit aan om een doel te wijzigen. Deze velden moeten worden ingevuld om het formulier geldig in te dienen: naam, beschrijving, status, SDGs entiteiten. Het formulier bevat een annuleer-functie om terug naar de detailpagina te keren. |
| Validatieboodschap | Deze geeft feedback inzake wijziging van de doelstelling. |
| REST API (Nest.js) | Deze bevat een HTTP PUT methode om een doelstelling te wijzigen (React). |
| MySQL databank | Deze luistert naar een DAO methode om een doelstelling te wijzigen (Java), welke via de doelstelling repository vanuit de Domeincontroller in de GUI wordt aangesproken. |
| CSS stijlen | Deze zorgen voor de opmaak van een scherm:   * React: Chakra. * Java: JavaFX CSS. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Detailpagina | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de detailpagina en controleert deze. Voor React gebeurt dit semiautomatisch met Cypress. |
| Call-to-action | Functioneel | De functionele testen verlopen exploratief, zowel voor Java als React. De tester klikt op de call-to-action en controleert of het formulier goed verschijnt. Voor React gebeurt dit semiautomatisch met Cypress. |
| Formulier | Functioneel | De functionele testen verlopen exploratief, zowel voor Java als React. Eerst tracht de tester het formulier ongeldig in te dienen en keert hij terug. Hierna wijzigt hij elk veld op een geldige manier. Voor React gebeurt dit semiautomatisch met Cypress. |
| Validatieboodschap | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.  De functionele testen verlopen exploratief, zowel voor Java als React. De tester checkt of de boodschappen correct verschijnen wanneer het formulier ongeldig en geldig wordt ingediend. Voor React gebeurt dit semiautomatisch met Cypress. |
| REST API (Nest.js) | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht of de data enkel na geldige indiening van het formulier wordt opgeslagen. Daarnaast checkt hij of de Nest.js API snel genoeg op het PUT verzoek reageert. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor Java. De tester verifieert op het zicht of de data enkel na geldige indiening van het formulier wordt opgeslagen. Daarnaast checkt hij of de MySQL databank snel genoeg op het DAO verzoek reageert. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen en leesbaarheid van de formuliervelden. |

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* De “mock” data is aanwezig in de persistentie laag om de exit criteria te kunnen bereiken.
* Elke rol kan testen. Authenticatie wordt pas in volgende use cases (UC5 en UC6) voorzien.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Risico** | **Impact** | **Mitigatieplan** |
| Afwezigheid van authenticatie (vertrouwelijkheid) | Elke gebruiker kan een doelstelling wijzigen, ook al heeft hij de juiste rol niet. | Groot | We aanvaarden het risico voor staging. Zodra de applicatie live kan, testen we authenticatie en de zichtbaarheid van features voor rollen in detail. |
| Capaciteit en mock-data (performantie) | Het aantal gelijktijdige verzoeken voor de REST API en databank en de hoeveelheid mock-data weerspiegelen geen realistische niveaus inzake laadtijden en performantie. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank en -API uitgevoerd. |
| Dataformaat (consistentie) | De data bevindt zich in een API voor React en in een databank voor Java, welke aanvragen in bepaalde formaten moeten blijven verwerken. | Groot | De testers voeren exploratieve en contract testen voor de API en DAO verzoeken uit. |

### Test matrix

Testen op domeinregels voor use case 14 (DR\_DOELSTELLING):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **FEATURE** | | | |
| **DOMEINREGEL** | Detailpagina met accordeon | Formulier met validatie | REST API (Nest.js) | MySQL databank |
| Unieke naam | Getest | Getest | Getest | Getest |
| Naam <= 50 tekens | Niet getest | Getest | Getest | Getest |
| Beschrijving <= 250 tekens | Niet getest | Getest | Getest | Getest |
| Alfanumerieke tekens | Getest | Getest | Getest | Getest |
| 1 categorie | Niet getest | Getest | Getest | Getest |
| Maximum 1 datasource | Niet getest | Niet getest | Niet getest | Niet getest |
| Alle velden verplicht | Niet getest | Getest | Getest | Getest |

Legende matrix:

* Getest: de tester dient de domeinregel tijdens het testen voor de feature te verifiëren.
* Niet getest: de tester dient de domeinregel niet tijdens het testen voor de feature te verifiëren.

De test is geslaagd. De domeinregel “<= 1 datasource” wordt in een volgend testplan opgenomen.

## Test Use Case 15: Verwijderen MVO-doelstelling

### Resources

De onderstaande tabel geeft aan wie de testen voor deze use case zal uitvoeren.

|  |  |
| --- | --- |
| **Tester** | **% Werkuren** |
| Rayme Emin | 33% (30 minuten), Java |
| Henri De Bel | 33% (30 minuten), React |
| Christof Remeysen | 33% (30 minuten), Java en React |

### Features

De ontwikkelde features voor deze use case staan hieronder beschreven.

|  |  |
| --- | --- |
| **Feature** | **Beschrijving** |
| Detailpagina | Deze geeft de details van de doelstelling weer: naam, beschrijving, status, SDGs entiteiten, datasource. |
| Call-to-action | Deze nodigt de gebruiker uit om een doelstelling te verwijderen. |
| Confirmatie pop-up | Deze vraagt aan de gebruiker of hij zeker is dat de doelstelling mag worden verwijderd. |
| REST API (Nest.js) | Deze bevat de data van een doelstelling en een datasource (React). |
| MySQL databank | Deze bevat de data van een doelstelling en een datasource (Java). |
| CSS stijlen | Deze zorgen voor de opmaak van een scherm:   * React: Chakra. * Java: JavaFX CSS. |

### Methoden

Deze sectie geeft aan hoe de hierboven vernoemde features worden getest.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Type test** | **Beschrijving** |
| Detailpagina | Unit en component  Functioneel | De unit en component testen verlopen automatisch. Ze worden enkel voor Java (JUnit) voorzien.    De functionele testen verlopen exploratief, zowel voor Java als React. De tester opent de detailpagina en controleert deze. Voor React gebeurt dit semiautomatisch met Cypress. |
| Call-to-action | Functioneel | De functionele testen verlopen exploratief, zowel voor Java als React. De tester klikt op de call-to-action en controleert of de confirmatie pop-up goed verschijnt. Voor React gebeurt dit semiautomatisch met Cypress. |
| Confirmatie pop-up | Functioneel | De functionele testen verlopen exploratief, zowel voor Java als React. De tester klikt afwisselend op de annuleer- en bevestig-functie. Voor React gebeurt dit semiautomatisch met Cypress. |
| REST API (Nest.js) | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor React. De tester verifieert op het zicht en via Postman of de data goed en snel genoeg in de Nest.js API wordt verwijderd. |
| MySQL databank | Integratie  Performance | De integratietest verloopt exploratief. Ze gebeurt enkel voor Java. De tester verifieert op het zicht of de data goed en snel genoeg in de MySQL databank wordt verwijderd. |
| CSS stijlen | Functioneel  Usability | De tester verifieert exploratief of de CSS stijlen goed inladen, zowel voor Java als React. Daarbij let hij op gebruiksvriendelijke aspecten, zoals goed aanklikbare knoppen en leesbaarheid van tekst. |

### Assumpties

De tester maakt de volgende assumpties bij aanvang:

* De applicatie is in één taal gebouwd: het Nederlands.
* De “mock” data is aanwezig in de persistentie laag om de exit criteria te kunnen bereiken.
* Elke rol kan testen. Authenticatie wordt pas in volgende use cases (UC5 en UC6) voorzien.

### Risico’s

De onderstaande tabel geeft risico’s mee voor het verdere verloop van het project.

Legende:

* Klein: de gevolgen zijn klein wanneer fouten op de productie-omgeving optreden. De gebruiker voert taken met weinig hinder in de applicatie uit. De aanbieder ondervindt geen schade: financieel, reputatie, etc.
* Gemiddeld: de gevolgen zijn gemiddeld wanneer fouten op de productie-omgeving optreden. De gebruiker ervaart enige moeite om taken in de applicatie uit te voeren. De aanbieder ondervindt enige schade: financieel, reputatie, etc.
* Groot: de gevolgen zijn groot wanneer fouten op de productie-omgeving optreden. De gebruiker kan de taken in de applicatie niet uitvoeren. De aanbieder ondervindt substantiële schade: financieel, reputatie, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Risico** | **Impact** | **Mitigatieplan** |
| Afwezigheid van authenticatie (vertrouwelijkheid) | Elke gebruiker kan een doelstelling met bijbehorende datasource verwijderen, ook al heeft hij de juiste rol niet. | Groot | We aanvaarden het risico voor staging. Zodra de applicatie live kan, testen we authenticatie en de zichtbaarheid van features voor rollen in detail. |
| Capaciteit en mock-data (performantie) | Het aantal gelijktijdige verzoeken voor de REST API en databank en de hoeveelheid mock-data weerspiegelen geen realistische niveaus inzake laadtijden en performantie. | Gemiddeld | De testers voeren manuele laadtesten uit. Vlak voor go live, wordt een integratietest op de productie-databank en -API uitgevoerd. |
| Dataformaat (consistentie) | De data bevindt zich in een API voor React en in een databank voor Java, welke aanvragen in bepaalde formaten moeten blijven verwerken. | Groot | De testers voeren exploratieve en contract testen voor de API en DAO verzoeken uit. |

### Test matrix

Testen op domeinregels voor use case 15 (DR\_DOELSTELLING):

|  |  |  |  |
| --- | --- | --- | --- |
|  | **FEATURE** | | |
| **DOMEINREGEL** | Detailpagina met accordeon | REST API (Nest.js) | MySQL databank |
| Unieke naam | Getest | Getest | Getest |

Legende matrix:

* Getest: de tester dient de domeinregel tijdens het testen voor de feature te verifiëren.
* Niet getest: de tester dient de domeinregel niet tijdens het testen voor de feature te verifiëren.

De test is geslaagd.

1. Sustainable Development Goals [↑](#footnote-ref-2)