Technisch ontwerp showcase

Sil Gosker

**Versiebeheer**

|  |  |  |
| --- | --- | --- |
| **Wie** | **Wat** | **Wanneer** |
| Sil Gosker | Opzet van het document | 2025-02-05 |
| Sil Gosker | Initialiseren van UC1 | 2025-02-05 |
| Sil Gosker | Initialiseren van UC2 | 2025-02-07 |
| Sil Gosker | Initialiseren van UC3 & UC4 | 2025-02-25 |
|  |  |  |

Contents

[2 Inleiding 4](#_Toc191366423)

[2.1 Probleemstelling 4](#_Toc191366424)

[2.2 Oplossing 4](#_Toc191366425)

[3 Systeem context 5](#_Toc191366426)

[3.1 Actoren 5](#_Toc191366427)

[3.2 Website 5](#_Toc191366428)

[3.3 Mailserver 6](#_Toc191366429)

[4 Container context – website 7](#_Toc191366430)

[4.1 Component context – Frontend 9](#_Toc191366431)

[4.1.1 UC1: Code context – Homepagina 10](#_Toc191366432)

[4.1.2 UC2: Code context – Contactpagina 11](#_Toc191366433)

[4.2 Component context – Backend API 12](#_Toc191366434)

[4.2.1 UC2: Code context – Mailing module 13](#_Toc191366435)

# Inleiding

Dit document bevat de technische documentatie. Dit omvat visueel en tekstueel vormgegeven ontwerpen op basis van requirements en ontwerpen uit het functioneel ontwerp. Er zal van een globale scope naar detail genavigeerd worden om overzicht te behouden. In de inleiding zal naast deze introductie ook de probleemstelling- en oplossing geïntroduceerd worden. Het verkleinen en detailleringsniveau van de scope zal middels het C4-model vormgegeven worden. De vierde laag zal echter hybride vormgegeven zijn om ruimte te geven de use cases hierin te verwerken.

## Probleemstelling

Stage aanbieders, werkgevers en recruiters zijn allemaal geïnteresseerd in nieuwe ontwikkelaars en willen oppervlakkig in contact met hen komen. Dit wordt in hoge mate vermoeilijkt door het feit dat er online geen verbinding gelegd kan worden met deze ontwikkelaars. Hierdoor wordt teruggevallen op sociale media platformen zoals LinkedIn, wat niet wenselijk is.

## Oplossing

De ontwikkelaar in kwestie zal een website ontwikkelen waarin duidelijk is wat hij kan en waar hij geïnteresseerd in is. Daarnaast moet een contactformulier geïntegreerd worden in de website waardoor oppervlakkig contact opgenomen kan worden.

# Systeem context

Afbeelding met tekst, visitekaartje, schermopname, Lettertype

Door AI gegenereerde inhoud is mogelijk onjuist.In dit hoofdstuk zal de globale systeemcontext uitgelegd worden. Deze kan gevonden worden in Figuur 1: Systeemcontext.

Figuur 1: Systeemcontext

## Actoren

In het onderstaande overzicht kan gevonden worden welke actoren relevant zijn voor het huidige systeem:

|  |  |
| --- | --- |
| **Actor** | **Beschrijving** |
| Geïnteresseerde | Een bezoeker op de website die geïnteresseerd is in de ontwikkelaar van de website zelf. |
| Plantenhouder | Iemand die een plant heeft en die wilt laten onderhouden door K.A.S. |

## Website

De website is waar de geïnteresseerde naar gaat als hij meer wilt weten over de ontwikkelaar. De website bevat verschillend componenten die toekomstige uitbreidingen en ontwikkelingen mogelijk maken.

## Mailserver

Om berichten te versturen en contact te leggen moeten e-mails verstuurd worden. Dit zal gedaan worden middels een externe mailserver. Omdat mailserving en het onderhouden van zo’n server buiten de scope van het project ligt zal hier niet dieper op ingegaan worden.

# Container context – website

In dit hoofdstuk zal de context van de website zelf beschreven worden. Dit omvat de frontend, backend en databases. Ieder container zal globaal beschreven worden, waarna in detail gekeken zal worden naar de mogelijkheden van het container middels de laag drie en vier van het C4 model. Alle elementen buiten het website-systeem om zullen niet besproken worden. Denk hierbij aan actoren of systemen die buiten de scope liggen.

Afbeelding met tekst, schermopname, Lettertype, diagram

Door AI gegenereerde inhoud is mogelijk onjuist. **Frontend**

Figuur 2: Container context website

De frontend omvat alles wat de gebruiker ziet; dit is waar de wireframes in het functionele ontwerp gerealiseerd worden. Echter, hier zitten geen functionaliteiten aan vast. Er is minimale interactie mogelijk. Voor uitgebreide interactie en volledige realisatie van de requirements is een backend nodig. Om consistentie te verhogen is gebruik gemaakt van TypeScript. Daarnaast maken het ontwikkelingsproces webcomponents makkelijker, consistenter en sneller. Hiervoor is svelte gekozen. Om de UI consistent te houden, is tailwind gekozen als CSS framework.

**Backend API**

De backend API verwerkt alle data. Denk aan het versturen van emails middels het contactformulier. Het verwerkt dan ook alleen data. Om consistentie te verhogen is gekozen voor een gecompileerde, hard gecodeerde programmeertaal zoals C# en ASP.NET. Voor het beveiligen van data is gekozen om gebruik te maken van Entity Framework Core in combinatie met het NpgSQL pakket. Toegangspunten (endpoints) worden onderhouden door FastEndPoints.

**SQL Database**

De database is verantwoordelijk voor het correct opslaan van data binnen de applicatie. Denk hierbij aan accounts of planten. Een van de meest gebruikte databases is PostgreSQL, en om goede redenen. Verschillende functionaliteiten zoals de coalitie-functionaliteiten zijn vrijwel uniek tot Postgres en zorgen voor een hogere consistentie voor het behouden van de integriteit van de data.

## Component context – Frontend

Dit hoofdstuk zal dieper ingaan op de verschillende componenten die gevonden kunnen worden in de Frontend-container. Een visuele weergave hiervan kan gevonden worden in Figuur 2: Container context website. Ieder component zal kort besproken worden, waarna dieper ingegaan zal worden op de individuele componenten middels de vierde laag van het C4 model. Deze laag is vormgegeven d.m.v. UML klassendiagrammen. Ondanks dat er geen klassen worden gerealiseerd, kan een klassendiagram wel de relatie tussen verschillende componenten en hun eigenschappen weergeven.

**Homepagina**

Afbeelding met tekst, schermopname

Door AI gegenereerde inhoud is mogelijk onjuist.De homepagina omvat UC1: bekijken profiel. Denk hierbij aan het inzien van foto’s, de introductie tot de ontwikkelaar of vaardigheden/eigenschappen.

**Contactpagina**

De contactpagina is de pagina waar, wanneer relevant, de geïnteresseerde contact kan leggen met de ontwikkelaar. Dit gebeurt middels een contactformulier waar een aantal verplichte data ingevoerd moet worden.

Figuur 3: Component context Frontend

### UC1: Code context – Homepagina

De homepagina bestaat uit een aantal componenten die teruggevonden kunnen worden in de code. Denk hierbij aan de slideshow of de beschrijving. Hou er rekening mee dat dit geen klassen representeren, maar (svelte)componenten. Een visuele weergave van het diagram kan gevonden worden in Figuur 4: code context Home pagina. Per component zal een beschrijving gegeven worden aan de functie van dat component.

A blue screen with black text

Description automatically generated

**HomePage**

De HomePage is het “root”- component. Dit betekent dat dit component initieel wordt ingeladen als naar “/” genavigeerd wordt. Dit component laadt vervolgens 3 andere componenten in: AboutMeDescription, SlideShow en SkillGraph

Figuur 4: code context Home pagina

**AboutMeDescription**

Dit component bevat een nette, stijlvolle beschrijving voor de geïnteresseerde.

**SlideShow**

Dit is een container-component waar een slideshow gestart kan worden. Alle elementen die hier in terecht komen worden als individuele slides gezien die horizontaal voorbij “scrollen”.

**Slide**

Dit representeert een individuele slide. Dit omvat een standaardgrootte- en breedte en andere stijlen.

**SkillGraph**

Dit is het component dat verantwoordelijk is voor het tonen van eigenschappen van de ontwikkelaar. Dit gaat door middel van een naam (bijv. C#) en een percentage (0-100), waarbij 0 compleet onbekend is, en 100 expert/senior-niveau aan kennis.

### UC2: Code context – Contactpagina

De contactpagina bestaat uit een aantal componenten die teruggevonden kunnen worden in de code. Denk hierbij aan input-velden met automatische validatie. Een visuele weergave van het diagram kan gevonden worden in Figuur 5: Code context Contact pagina.

**A blue rectangular object with text

AI-generated content may be incorrect.**

**ContactPage**

De ContactPage is het “root”-component. Dit betekent dat dit component initieel wordt ingeladen als naar “/contact” genavigeerd wordt. Dit component laadt vervolgens het ContactForm component in.

**ContactForm**

Het contactformulier bevat een aantal invoervelden en is verantwoordelijk voor het onderhouden van de gecombineerde data. Als het formulier verstuurd wordt, wordt een aanvraag naar de Backend API gedaan, waarnaar een melding getoond wordt dat de e-mail verstuurd is.

Figuur 5: Code context Contact pagina

**FormField**

Een herbruikbaar component dat gebruikt wordt voor het automatisch valideren van input. In de huidige situatie maakt ContactForm gebruik van 3 van deze velden: één voor de email; één voor het onderwerp en één voor het bericht zelf.

### UC3: Code context – Login pagina

De loginpagina bestaat uit een aantal componenten die in hun totaliteit teruggevonden kunnen worden in de code. Denk hierbij aan de relaties tot de input-velden, naamgeving, etc. Een visuele weergave van het diagram kan gevonden worden in Figuur 6: Code context login pagina.

Afbeelding met tekst, diagram, schermopname, Rechthoek

Door AI gegenereerde inhoud is mogelijk onjuist.

Figuur 6: Code context login pagina

**LoginPage**

De LoginPage is het “root”- component. Dit betekent dat dit initeel wordt ingeladen als naar “/login” genavigeerd wordt. Dit component laadt vervolgens de standaard-layout in, gecombineerd met het LoginForm component.

**LogiBack1nForm**

Het loginForm bevat naast een aantal invoervelden ook minimale validatie. Grotendeels van de validatie wordt gedaan door het FormField component. Het LoginForm is verantwoordelijk voor het behouden van de totaliteit van de data. Als het formulier verstuurd wordt, wordt een aanvraag naar de Backend API gedaan, waarnaar een melding getoond wordt of het inloggen gelukt is of niet.

**FormField**

Zoals eerder besproken is het FormField een representatie van een input-veld in het formulier. Het is verantwoordelijk voor het valideren van één onderdeel van de algehele data dat verwerkt wordt in het LoginForm.

### UC4: Code context – registratiepagina

De registratiepagina bestaat uit een aantal componenten die in hun totaliteit teruggevonden kunnen worden in de code. Denk hierbij aan de relaties tot de input-velden, naamgeving, etc. Een visuele weergave van het diagram kan gevonden worden in Figuur 7: Code context registratiepagina.

Afbeelding met tekst, schermopname, diagram, Rechthoek

Door AI gegenereerde inhoud is mogelijk onjuist.

Figuur 7: Code context registratiepagina

**RegistrationPage**

De RegistrationPage is het “root”- component. Dit betekent dat dit initeel wordt ingeladen als naar “/register” genavigeerd wordt. Dit component laadt vervolgens de standaard-layout in, gecombineerd met het RegistrationForm component.

**RegistrationForm**

Het RegistrationForm bevat naast een aantal invoervelden ook validatie voor het controleren van de data in hun geheel. Grotendeels van de validatie wordt gedaan door het FormField component. Het RegistrationForm is verantwoordelijk voor het behouden van de totaliteit van de data. Als het formulier verstuurd wordt, wordt een aanvraag naar de Backend API gedaan, waarnaar een melding getoond wordt of het registreren gelukt is of niet.

**FormField**

Zoals eerder besproken is het FormField een representatie van een input-veld in het formulier. Het is verantwoordelijk voor het valideren van één onderdeel van de algehele data dat verwerkt wordt in het LoginForm.

### UC5: Code context – Inzien planten pagina

Het visualiseren van de overzichtspagina’s voor de planten is vrijwel overbodig, gezien dit zal bestaan uit de eerder genoemde “Root-componenten”, waarbij voor de index- en detailpagina een component beschikbaar wordt gesteld.

### UC6: Code context – Create planten pagina

**Afbeelding met tekst, schermopname, diagram, Rechthoek

Door AI gegenereerde inhoud is mogelijk onjuist.**De pagina voor het aanmaken van een plant is in een component-diagram vrij simplistisch. Zoals standaard zijn er 3 componenten aanwezig, waarvan 2 slecht herbruikbaar. Ieder component zal besproken worden.

**CreatePlantPage**

Figuur 8: Code context create plant pagina

Het CreatePlantPage- component is het “root”- component, betekene dat dit initieel wordt ingeladen als naar de corresponderende url genavigeerd wordt. Dit laadt vervolgens het CreatePlantForm- component in.

**CreatePlantForm**

Het CreatePlantForm- component is verantwoordelijk voor het managen en versturen van de data van een plant. Deze laadt 2 velden in: één voor de naam en één voor de duratie.

## Component context – Backend API

Dit hoofdstuk zal dieper ingaan op de verschillende componenten die gevonden kunnen worden in de Backend API-container. Een visuele weergave hiervan kan gevonden worden in Figuur 4: Component context Backend API. Ieder component zal kort besproken worden, waarna dieper ingegaan zal worden op de individuele componenten middels de vierde laag van het C4 model. Deze laag is vormgegeven d.m.v. UML klassendiagrammen.

**Mailing module**

Afbeelding met tekst, schermopname, Lettertype, ontwerp

Door AI gegenereerde inhoud is mogelijk onjuist.De emailmodule is verantwoordelijk voor het versturen van e-mails op basis van de informatie die is meegegeven in het formulier op de contactpagina.

**Account module**

De account module is verantwoordelijk voor het opslaan en managen van gebruikersgegeven. Deze maakt ook verbinding met de database middels Entity Framework.

**View Plant Component**

Omdat de Plant- module vrij uitgebreid is, is deze opgesplitst op basis van use cases. Voor iedere use case zal een klassendiagram gemaakt worden. Het View Plant component is verantwoordelijk voor het inzien van de planten, en meesturen van de correcte data.

Figuur 9: Component context Backend API

**Create Plant Component**

De Create Plant Component is verantwoordelijk voor het opslaan van planten. Dit omvat validatie, en ervoor zorgen dat de gebruiker niet meer dan 5 planten tegelijk kan laten verbinden met de server.

### UC2: Code context – Mailing module

A close-up of several folders

AI-generated content may be incorrect.Dit hoofdstuk zal dieper ingaan op de inhoud van de mailing module. Dit is vormgegeven d.m.v. een klassendiagram. Alle relevante klassen zullen individueel besproken worden om hun rol en relevantie te bespreken. Het klassendiagram is gevisualiseerd in Figuur 7: Code context Mailing module.

Figuur 10: Code context Mailing module

**SendMailRequest**

Dit representeert het dataformaat dat verwerkt wordt. Daarom zitten hier alle aspecten in die relevant zijn voor het versturen van een e-mail, zoals de verstuurder, het bericht en het onderwerp.

**SendMailEndpoint**

Het aanvraagpunt (endpoint) wordt gedefinieerd door de SendMailEndPoint klasse. Deze bevat de IEmailService om de email te versturen. Dit wordt d.m.v. Dependency Injection resolved.

**IEmailService**

De IEmailService is de interface die definieert wat er met emails gedaan kan worden. Door een expliciete interface te gebruiken, wordt de mogelijkheid opengesteld om de SendMailEndpoint te unit testen.

**SmtpEmailService**

De EmailService-klasse is de implementatie van de IEmailService, en gebruikt de SmtpClient om emails te versturen.

**EmailOptions**

De EmailOptions-klasse representeert alle configuratie die mogelijk is bij het versturen van emails. Het is mogelijk om hier later extra properties aan toe te voegen wanneer meer configuratie nodig is.

### UC3, UC4: Code context – Account module

Dit hoofdstuk zal dieper ingaan op de inhoud van de account module. Dit is vormgegeven d.m.v. een klassendiagram. Alle relevante klassen zullen individueel besproken worden om hun rol en relevantie te bespreken. Het klassendiagram is gevisualiseerd in Figuur 7: Code context registratiepagina.

Authenticatie en autorisatie wordt middels Bearer Authentication gedaan. Dit betekent dat de server een ASP.NET Identity (ClaimsIdentity) serialized tot een encrypted stuk tekst, die in de browser opgeslagen wordt. Bij iedere aanvraag die de browser naar de API doet, wordt dit token meegestuurd en deseralized op de server.

Afbeelding met tekst, schermopname, diagram, Plan

Door AI gegenereerde inhoud is mogelijk onjuist.**RegisterEndpoint**

Figuur 11: code context account module

Dit representeert het aanvraagpunt om een account te registreren. Het ontvangt een RegisterRequest klasse, waar alle relevante informatie in staat om een account te maken.

**LoginEndpoint**

Dit representeert het aanvraagpunt om in te loggen. Dit omvat het verkrijgen van een bearer token en informatie over je account. Het ontvangt een LoginRequest en stuurt een LoginResponse terug.

**RegisterRequest**

Dit representeert alle benodigde informatie om een account te maken. In de toekomst kan hier eventueel extra informatie aan toegevoegd worden om extra data aan een account toe te voegen.

**LoginRequest**

Dit representeert alle benodigde data om in te loggen. Voor nu is een email of wachtwoord voldoende.

**LoginResponse**

Dit representeert een blauwdruk voor het verwachte antwoord van de server. Dit omvat een token om mee in te loggen en de rol die de gebruiker heeft. Het e-mail adres wordt bewust niet meegestuurd omdat de gebruiker die inlogt deze al heeft.

**IAccountService**

Deze interface is verantwoordelijk voor het definiëren van de functionaliteit van een account. Dit omvat het aanmaken en inloggen van een account.

**DbEntity**

Dit representeert een gegeneraliseerde database entity. Deze heeft functionaliteit voor soft-delete en een standaard-primary key.

**AccountService**

Dit is een implementatie van de IAccountService die gebruik maakt van de Postgres database.

**KasDbContext**

Dit representeert de huidige database in een klasse. Iedere property met het type DbSet<> kan gezien worden als een tabel.

**Account**

Dit representeert een account in de database. Voor veiligheidsredenen worden wachtwoorden niet in plaintext opgeslagen, maar middels salting én hashing opgeslagen. Dit zal middels Bcrypt gedaan worden.

**Role**

Deze enum representeert welke rol een gebruiker heeft. Omdat er relatief weinig functionaliteiten in de showcase zitten, is het niet nodig een permissie-gebaseerd systeem in te bouwen.

### UC5: Plant View Module

Afbeelding met tekst, schermopname, diagram, Plan

Door AI gegenereerde inhoud is mogelijk onjuist.Dit hoofdstuk zal dieper ingaan op de inhoud van de “plant view module”. De module is verantwoordelijk voor het openstellen van API endpoints zodat plantenhouders hun eigen planten kunnen inzien. Dit is vormgegeven d.m.v. een klassendiagram. Alle relevante klassen zullen individueel besproken worden om hun rol en relevantie te bespreken. Het klassendiagram is gevisualiseerd in Figuur 11: Code context Plant module.

Figuur 12: Code context Plant module

**GetPlantsEndpoint**

Omdat de Plant zo’n kleine entiteit is, is maar 1 endpoint nodig om te voorzien in alle benodigdheden. Dit endpoint is verantwoordelijk voor het terugsturen van alle planten op basis van de Id van de gebruiker.

**PlantResponse**

De PlantResponse is verantwoordelijk voor het representeren van een plant voor de gebruiker. Deze bevat een identificatienummer dat overeenkomt met dat in de database, en de naam van de plant. Op de client kan bekeken worden of deze ook verbonden is met de server.

**IPlantService**

De IPlantService interface representeert de functionaliteiten die mogelijk zijn bij het verwerken van data gerelateerd aan een plant. Denk hierbij aan het ophalen van planten, en in de toekomst updaten en/of verwijderen.

**PlantService**

De implementatie van de IPlantService die middels de database de functionaliteiten biedt.

**KasDbContext**

De KasDbContext is aangevuld met de Plants- property.

**Plant**

De plant representeert een plant in de database. Voor meer informatie kan naar het ERD gerefereerd worden, gezien het een 1 op 1 representatie is.

### UC6: Plant Create Module

Afbeelding met tekst, diagram, schermopname, lijn

Door AI gegenereerde inhoud is mogelijk onjuist.Dit hoofdstuk zal dieper ingaan op de inhoud van de “plant create module”. De module is verantwoordelijk voor het openstellen van API endpoints zodat plantenhouders hun eigen planten kunnen aanmaken. Dit is vormgegeven d.m.v. een klassendiagram. Alle relevante klassen zullen individueel besproken worden om hun rol en relevantie te bespreken. Het klassendiagram is gevisualiseerd in Figuur 12: Code context Plant Create Module.

Figuur 13: Code context Plant Create Module

**Validator**

Dit is onderdeel van de FluentValidation- package. De **CreatePlantValidator** kan middels deze package regels stellen voor specifieke properties voor de **CreatePlantRequest**.

**IPlantService**

De IPlantService heeft een extra methode gekregen om een plant aan te maken. Deze methode geeft een string terug, wat het wachtwoord in plaintext moet representeren. In de database wordt deze plaintext gehashed middels bcrypt.

**Plant**

De plant heeft een extra property gekregen, de Hash. Deze hash wordt gebruikt voor het geven van water (UCX). Daarnaast heeft hij de duratie waarin water gegeven moet worden.

**CreatePlantRequest**

Deze klasse representeert de aanvraag die gedaan wordt naar de server. De naam en duratie moeten verplicht meegegeven worden, waarbij de limieten toegepast worden die in het functioneel ontwerp genoemd worden.

**CreatePlantResponse**

Deze klasse representeert het antwoord van de server. Dit omvat de naam, de duratie en het wachtwoord in tekst. Deze kan vervolgens aan de gebruiker getoond worden, zodat hij het in zijn script kan gebruiken.

**CreatePlantEndpoint**

Deze klasse is verantwoordelijk voor het verwerken van de aanvraag en het terugsturen van het antwoord.

## Component context – SQL Database

Afbeelding met tekst, schermopname, Lettertype, Rechthoek

Door AI gegenereerde inhoud is mogelijk onjuist.Dit hoofdstuk zal dieper ingaan op de database server. Onder andere worden de databases in de server besproken. Een visualisatie hiervan kan gevonden worden in Figuur 11: Component Context SQL Database.

Figuur 14: Component Context SQL Database

Zoals te zien is er in de huidige situatie maar één database. In vervolghoofdstukken zal de ERD van deze database besproken worden.

### Code context KasDb

In dit hoofdstuk zal de database van de KasDb uitgewerkt worden middels een ERD. Hou er rekening mee dat het lastig is om exact te bepalen hoe de ERD eruit ziet. Dit heeft te maken met de code-first praktijk binnen de applicatie. De database wordt compleet beheerd door EntityFramework. Alleen de kolommen, datatypen en relaties kunnen weergegeven worden. Een visualisatie van de database kan gevonden worden in Figuur 12: ERD KasDb. Iedere tabel zal individueel besproken worden.

Afbeelding met tekst, schermopname, Lettertype, nummer

Door AI gegenereerde inhoud is mogelijk onjuist.

Figuur 15: ERD KasDb

**Account**

Een account representeert een gebruiker in de database. Hij heeft zoals standaard een integere waarde als primary key (ID) en een kolom om soft-delete te bepalen (Deleted). Daarnaast heeft hij de verplichte kolommen om een gebruiker te kunnen representeren. Denk aan de Email/Wachtwoord combinatie (Wachtwoord d.mv. Hash en Salt), en de Role. In de huidige situatie kan deze 1 v/d de onderstaande waarden bevatten:

* 0: Een plantenhouder
* 1: Een administrator

**Plant**

Een plant representeert een fysieke plant op kantoor of thuis. Hij heeft de standaard- elementen die ook in een Account voorkomen die een database- entiteit representeren (Id, Deleted). Daarnaast heeft hij ook een naam om de planthouder een indicatie te geven over welke plant het gaat. Naast een naam moet ook bijgehouden worden hoeveel water de plant per dag moet ontvangen. Dit gebeurt middels de WaterDuration kolom, die weergegeven wordt in seconden. De Hash zorgt ervoor dat de plant in kan loggen op de server middels websockets. Degene die de plant maakt wordt ook automatisch gekoppeld, zodat andere gebruikers er niet meer bij kunnen.