Ontwerpdocument

Isotita

|  |  |
| --- | --- |
| Leereenheid: | Isotita |
| Versienummer: | 1.0 |
| Auteur(s): | Mohamad Alamoroush |
| Datum: | 15-06-2022 |

Hiernaast staan vier invulvelden. Drie daarvan worden ‘hergebruikt’ in de header en de footer van het document. Om de inhoud van de invulvelden naar de header en footer door te voeren, moet je een print preview genereren.

De tekst die je nu leest zal niet afgedrukt worden.

# Versiebeheer

|  |  |  |  |
| --- | --- | --- | --- |
| Datum | Versie | Wie | Wijzigingen |
| 10-4-2022 | 0.1 | Mohamad | Nieuwe document: Ontwerpdocument |
| 22-4-2022 | 0.2 | Mohamad | Naamgeving aangepast + Class diagram aangepast |
| 25-04-2022 | 0.3 | Mohamad | Nieuwe versie van Class diagram en databaseontwerp |
| 10-05-2022 | 0.4 | Mohamad | Aanpassingen op Class diagram en Databaseontwerp. Architectuur diagram toegevoegd |
| 21-05-2022 | 0.5 | Mohamad | Aanpassingen op alle diagrammen + toelichting |
| 31-05-2022 | 0.6 | Mohamad | Aanpassingen op class diagram + toelichting op class diagram en database model |
| 03-06-2022 | 0.7 | Mohamad | Architectuurdiagram aangepast + toelichting |
| 07-06-2022 | 0.8 | Mohamad | Architectuurdiagrammen aangepast door UnitTest-project toe te voegen |
| 13-06-2022 | 0.9 | Mohamad | Aanpassingen op Architectuurdiagrammen op basis van feedback sprintoplevering 5 |
| 15-06-2022 | 1.0 | Mohamad | Pijlen verandert in Architectuurdiagrammen |

Contents

[Versiebeheer 2](#_Toc106196318)

[Projectbeschrijving: 4](#_Toc106196319)

[Logica laag van Isotita-app 4](#_Toc106196320)

[Class diagram toelichting 4](#_Toc106196321)

[Databaseontwerp 6](#_Toc106196322)

[Database ontwerp toelichting: 6](#_Toc106196323)

[Architectuurdiagram 8](#_Toc106196324)

[Architectuurdiagram toelichting 10](#_Toc106196325)

# Projectbeschrijving:

Isotita is een Grieks woord en het betekent ‘gelijkheid’. Dat is ook het doel van hun gevraagde webapplicatie. Ze willen een systeem hebben waar bedrijven hun beschikbare vacatures kunnen plaatsen. Het systeem gebruikers kunnen anoniem solliciteren dat betekent dat een bedrijf ziet alleen gegevens zoals ervaring en opleidingen. Maar geen personeel informatie zoals naam, geboortedatum of afkomst. Zo willen ze focussen dat bedrijven eerste gaan kijken naar wat iemand kan en dan wie hij is.

De belangrijkste rollen van de app:

1- Bedrijven die vacatures plaatsen en kunnen aanpassen

2- Sollicitanten die op een vacature kunnen solliciteren en status van hun sollicitaties bekijken.

Wij solliciteren voor jou is een webapplicatie die ongeveer overheen komt met wat Isotita wil hebben. Toch zijn er verschillende punten. Bijvoorbeeld bij wij solliciteren voor jou gaan ze zelf aan de slag met de sollicitatie proces. Dat helpt maar voor meerdere sollicitaties is dat toch wat lastiger te volgen. Bij ons gaan sollicitanten dat zelfstandig doen en ze kunnen zelf de status van hun sollicitaties nakijken.

# Logica laag van Isotita-app

# Classdiagram:

Diagram

Description automatically generated

Figuur 1: Class diagram

## Class diagram: toelichting

Mijn class-diagram heeft alle klassen wat ik nodig zou hebben om de gevraagde functionaliteit te kunnen uitvoeren.

In Isotita-app wordt gebruikt gemaakt van Rich-Domain-Models.

Het rijke domeinmodel is het technische deel van het implementeren van DDD en omvat bouwstenen zoals entiteiten, waardeobjecten. Het doel is om een ​​gemeenschappelijke taal op te bouwen tussen ontwikkelaars en belanghebbenden met behulp van eisen die bedrijfsregels beschrijft.

De voordelen daarvan zijn:

1. Onderhouden: In het begin van elke applicatie is het lastig om voor te stellen hoe complex het zou zijn. Wat als straks grote vraagstukken erbij komen? Door Rich-Domain-Models wordt het makkelijker om nieuwe functionaliteit te brengen in de applicatie.
2. Logica-laag testen: Applicatie testen is belangrijke onderdeel om te voorkomen dat er iets aanpassingen worden gemaakt die negatieve invloed hebben op andere functionaliteiten. Met Rich-Domain-Models wordt het makkelijker om het logica laag te testen.
3. Veiliger: De eigenschappen van het domeinmodel moeten worden bewaakt met aparte setters, zodat alleen het model zijn eigen eigenschappen kan instellen, door zijn eigen gedrag. Anders is het "promiscue".
4. Komt beter overheen met OO: Een model heeft zijn eigen gedragen en proporties dus OO-principe wordt makkelijker toegepast. Het helpt ook om op juiste manier de SOLID-principe toe te passen.

Toelichting op class diagram zelfs:

Door gebruik te maken van Rich-Domain-Models mag een model zichzelf niet aanmaken, want het object bestaat nog niet. Het ophalen van een object mag ook niet de object ophalen want dit is de taak van een Container class. Daarom heb ik Company Container Model die de taak heeft om een nieuwe Company aan te maken, dat is bij de registratie het geval. Kijken of het Company Object bestaat en het company object ophalen, bijvoorbeeld als Company-account inlogt.

Een Company-Model is heeft zijn eigen proporties. Tegelijkertijd wordt hij gezien als een Container voor Job-Model. Want een Company kan een nieuw bedrijf toevoegen. Zij kan ook alle gemaakte Jobs door hun ophalen.

Een Job-Model kan zichzelf aanpassen en verwijderen. Door die model worden ook alle sollicitanten opgehaald die op een vacature hebben gesolliciteerd. Door de Job model worden ook nieuwe sollicitaties toegevoegd aan Applied Model.

In de Job Container worden methodes zoals GetJobs & GetById. Het reden waarom ze hier staan en niet bij de company is dat alle vacatures ophalen geen taak is van bedrijf, want een bedrijf kan alleen zijn vacatures ophalen. Op Applied Model worden de statussen van sollicitaties aangepast.

# Databaseontwerp

Graphical user interface, application

Description automatically generated

Figuur 2: Database ontwerp

## Database ontwerp toelichting:

In figuur 4 heb ik een database ontwerp gemaakt. Daar wil ik de data van Isotita-app opslaan.

Mijn database model bestaat uit 3 tabellen. Die 3 tabellen voldoen aan wat de applicatie moet doen.

Een gast kan registreren als bedrijf. In het tabel Company worden zijn gegevens opgeslagen, zodat hij straks kan inloggen.

Wanneer iemand met bedrijf account wil inloggen voert hij zijn email en wachtwoord. Vervolgens wordt gecheckt of de gegevens echt bestaan in de database zo ja, dan logt hij succesvol in, anders krijgt hij foutmelding. Zodra dat hij ingelogd is kan de gebruiker vanuit dit bedrijf gebruik maken van CRUD-functies die uitgevoerd worden op Job tabel. Door de CRUD-functies kan de gebruiker een nieuwe vacature toevoegen. Gegevens van een vacature aanpassen of vacature helemaal verwijderen. Ten tweede kan hij alle vacaturen ophalen wat hij gemaakt heeft. Daarom staat een company\_id in Job tabel, die veld verwijst naar de relatie, want een vacature kan alleen gemaakt worden door bedrijf account. Dat relatie helpt ook om de sollicitanten per vacature op te halen. Die sollicitanten worden opgeslagen in de database wanneer ze gebruik maken van Isotita-app. Ze kunnen door de Job tabel alle geplaatste vacatures ophalen en dan daarop solliciteren. Dat hele sollicitatie proces wordt opgeslagen in Applied-tabel. Als iemand solliciteert op een vacature kan het bedrijf het sollicitantgegevens bekijken en vervolgens behandelen. Dat behandeling proces wordt als Updatefunctie gezien want de status en treated velden bij de Applied worden aangepast.

Omdat een sollicitatie wordt op vacature gedaan, staat ook een relatie tussen de twee tabellen dat is te zien door een job\_id in Applied tabel.

De relatie tussen een bedrijf en vacaturen is een op veel want een bedrijf kan meerder vacatures plaatsen. Het relatie naam geldt ook tussen vacature en sollicitaties. Dat komt omdat op een vacature meerdere sollicitaties kunnen komen.

Er wordt gebruik gemaakt van MySQL. De redenen zijn:

1. Mijn ruim ervaring in MySQL die mij kan helpen.
2. Geen hoge kosten
3. Voldoende informatie te vinden op internet als er problemen zijn.

# Architectuurdiagram

Diagram

Description automatically generated

Figuur 3: Architectuurqdiagram

De vorige figuur was de basisidee. Uitwerking met echte classes is in figuur 3 te vinden:

Diagram, schematic

Description automatically generated

Figuur 4: Architectuurdiagram uitgebreid

# Architectuurdiagram toelichting

Mijn lagen verdeling helpt mij SOLID-principe toe te passen.

Single Responsibility principle: die principe pas ik toe door te zorgen dat een class alleen methodes kan gebruiken wat hij mag gebruiken en omdat een methode in een class maar een actie kan uitvoeren. Dat wordt ook makkelijker omdat ik Rich-Domain-Models gebruik. De eisen om dat methodiek te gebruiken is te zorgen dat een class heeft zowel zijn proporties als zijn methodes.

Open-closed principle: Door te zorgen dat de S uit SOLID op het juiste manier wordt toegepast. Kan ik wel de Open-close principle toepassen. Omdat ik geen gebruik maak van onnodige proporties of methodes. Dat is ook vaker toegepast in mijn-applicatie want ik moest vaker nieuwe methoden of classes toevoegen maar nooit oude code aanpassen of verwijderen.

Liskov substitution principle: Hiervoor werd de Factory-laag aangemaakt. Het is niet mogelijk om per direct een interface object aan te maken. Daarom maak ik in de Factory-laag een methodes die een interfaces aanmaken maar die gebruiken een Dal-laag class die een interface implementeert.

Dat is ook het idee van dit principe namelijk Het principe definieert dat objecten van een superklasse kunnen worden vervangen door objecten van zijn subklassen zonder de toepassing te verbreken.

Een andere reden waarom ik Factory-laag gebruik en dat is namelijk het belangrijkste: Ik wil voorkomen dat er dependance ontstaat tussen Presentatie-laag en Dal-laag. Anders wordt het mogelijk om per direct toegang te krijgen tot Dal-laag classes en functies wat niet de bedoeling is.

Interface-Segregation pricipe: Dat houdt in dat grote interfaces kleiner en specifieker kunnen worden. Voorbeeld daarvan staat ook in figuur 4. Company class mag een nieuwe Job aanmaken en ten tweede mag hij ook zijn Jobs ophalen. Dat staat in een interface namelijk ICompanyJobDal. Tegelijkertijd heeft Job methodes zoals aanpassen verwijderen of sollicitaties ophalen. Dat staat in een andere interface namelijk IJobDal. Uiteindelijk worden beide interfaces geïmplementeerd door hetzelfde class omdat het gaat om Job-tabel in de database. Zo zorgde ik dat een grote interface wordt verdeeld en dat een client alleen gedragen kan uitvoeren die interessant zijn voor hem of waar hij rechten op heeft.

Dependency Inversion principle: Het houdt in dat hoge-niveau modules mogen geen dependance hebben op een lage-niveau modules, maar beide zullen een dependance hebben op een abstracties. Het tweede punt van dependance inversie is dat abstracties of interfaces mogen niet afhankelijk zijn van details maar details moeten afhankelijk zijn van abstracties of interfaces.

Het uitvoeren van dit principe wordt in behulp van interfaces gedaan. Zoals het duidelijk is in figuur 4, de pijlen van de Dal-laag gaan richting de Logica-laag, maar nier andersom wat ook de bedoeling is. Een interface heeft gedragen, wanneer een class in de Dal-laag die interface implementeert kan hij ook dat gedragen uitvoeren.

Andere algemene punten die te vinden zijn in mijn architectuurdiagram:

1. ViewModel = Die zijn classes waar validatie worden vastgelegd bij de invoer van data. Met validaties bedoel ik welke velden zijn verplicht. Welke minimum of maximumlengte geldt voor een bepaald veld. Welke data accepteert een veld zoals email om nummers etc.
2. PresLogicDTO = Die zijn DTO’s tussen presentatie-laag en logica-laag. Het doel daarvan is om geen nieuwe objecten te vullen in de presentatie-laag maar een DTO-object wordt gestuurd naar de logica-laag en daar worden proporties van de class gegeven om verder te converteren naar een DTO-object.
3. DTO-object: Die zijn DTO’s tussen de logica-laag en de dal-laag met doel om alleen proporties te sturen naar de dal-laag en daar worden ze verwerkt op basis van de gemaakte actie. Door dat proces te doen voorkom ik dat in de dal-laag een object van de logica-laag wordt gemaakt en vervolgens toegang kunnen hebben tot die class methodes.