ONDERZOEKSVERSLAG HEXAGONAL ARCHITECTURE

Onderzoek naar Hexagonal Architecture in de context van een CRM Applicatie

Auteurs: OOSE-team Thompson, studenten HBO-ICT Opleiding HAN
Opdrachtgever: JDI

Inhoudsopgave

1. Abstract

Dit document bevat het onderzoek wat uit is gevoerd door team Thompson voor de klant JDI naar de technologie Hexagonal Architecture. Er is een demo applicatie opgesteld om te kijken wat de voor en nadelen zijn aan de technologie.

De deelvragen die wij hebben beantwoord met dit onderzoek zijn als volgt:

1. Wat is Hexagonal Architecture
2. Wat is een CRM
3. Wat zijn de voordelen van Hexagonal Architecture
4. Wat zijn de nadelen van Hexagonal Architecture
5. Hoe ziet een implementatie van Hexagonal Architecture er uit binnen een CRM context

Tijdens dit onderzoek hebben we praktijk- en literatuuronderzoek gedaan naar Hexagonal Architecture. Zoals eerder genoemd is er op basis van het literatuuronderzoek een demo applicatie opgesteld waarnaar we onze bevindingen hebben geformuleerd.

Tijdens het onderzoek zijn we er achter gekomen dat Hexagonal Architecture de complexiteit van een applicatie verhoogd maar dat de applicatie hierdoor uiteindelijk wel onderhoudbaarder is. Als een applicatie echter te kleinschalig is dan is het niet verstandig om Hexagonal Architecture toe te passen, dit omdat de voordelen van de techniek pas bij complexe applicaties met veel onderdelen zichtbaar worden.

2. Inleiding

Dit onderzoek is opgesteld op basis van een gesprek met JDI. In dit gesprek gaven zij aan meer te weten te willen komen over de voor en nadelen van hexagonal architecture en hoe ze deze techniek in hun bedrijf kunnen toepassen. De volgende hoofdvraag is voor dit onderzoek geformuleerd: "Wanneer is hexagonal architecture een effectieve methode van ontwikkelen."

Er is onderzocht hoe hexagonal architecture kan worden toegepast binnen de context van een CRM applicatie. Voor dit onderzoek is lab onderzoek verricht door een demo applicatie te bouwen die gebruik maakt van hexagonal architecture (zie hoofdstuk 2). De source code van deze applicatie is te vinden op BitBucket

3. Materialen en methoden

Dit hoofdstuk bevat de verschillende materialen en onderzoeksmethoden die zijn gebruikt om Hexagonal Architecture te onderzoeken.

3.1. Literatuuronderzoek

Voor dit onderzoek wordt er gebruik gemaakt van literatuuronderzoek. De literatuur bestaat veelal uit online bronnen. Alle bronnen worden streng gecontroleerd op de volgende punten:

- Actualiteit
- Auteur
- Meningen
- Onpartijdigheid van de auteur

Elke gebruikte bron is terug te vinden in de Literatuurlijst aan het einde van dit onderzoeksrapport. Alle bronnen worden gecontroleerd op bovenstaande punten en deze resultaten komen in het hoofdstuk Discussie aan bod. De genoemde punten worden gemeten op de volgende voorwaarde(n):

Actualiteit

Een bron mag niet ouder zijn dan 2 jaar op het moment van raadplegen *of* de informatie in de bron moet nog steeds relevant zijn, d.w.z. dat hetgeen wat in de bron staat in de loop van tijd niet veranderd kan zijn. Dit laatste moet verantwoord (bewezen) worden in het hoofdstuk Discussie.

Auteur

De auteur moet voldoende kennis hebben over hetgeen waar hij/zij over schrijft. Dat wil zeggen dat de auteur gediplomeerd is in het vakgebied waar de bron over gaat *of* de auteur is minimaal 2 jaar werkzaam in het vakgebied waar de bron over gaat. De kennis van de auteur moet worden verantwoord in het hoofdstuk Discussie.

Meningen

Alle bronnen moeten gecontroleerd worden op meningen. Bronnen moeten objectief zijn. Bronnen mogen alleen een vorm van mening bevatten wanneer er wordt gesproken over voor- en nadelen, omdat dit altijd een persoonlijke mening is. Echter moeten er bij dit soort bronnen minimaal 2 andere bronnen hebben, die ook voldoen aan alle andere beoordelingscriteria, die dezelfde mening ondersteunen. Gebruik van meningen moet duidelijk (d.w.z. concreet in woorden vermeld aan het begin van een alinea) worden vermeld.

Onpartijdigheid van de auteur

De auteur mag geen direct relatie hebben met een partij die belang heeft bij een bron. Een voorbeeld: een bron over de manier van werken bij Tesla mag niet geschreven zijn door de teamleider van Tesla, omdat hij/zij een directe relatie heeft met de inhoud van de bron, waardoor al snel een subjectieve en/of verheerlijkende bron ontstaat. De relatie van de auteur (of de organisatie waar de auteur voor werkt) met de inhoud van de bron moet worden onderzocht en verantwoord worden in het hoofdstuk Discussie.

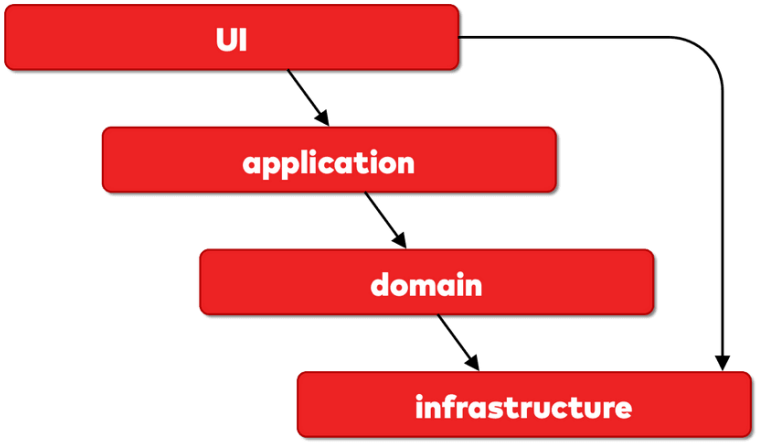
3.2. Praktijkonderzoek

Voor het onderzoek wordt ook gebruik gemaakt van praktijkonderzoek. Dit gebeurd door middel van het toepassen van de techniek in een voorbeeld CRM-applicatie. Deze onderzoekstechniek wordt toegepast om te kijken of Hexagonal Architecture daadwerkelijk een goede toevoeging is in de praktijk, om zo bij te dragen aan het doel van het overkoepelende onderzoek CRM-JDI (Plan van Aanpak - Hoofdstuk 3).

4. Wat is hexagonal architecture?

Hexagonal architecture, uitgevonden door Alistair Cockburn in 2005, is een manier van object georiënteerd software ontwikkelen die in theorie zorgt dat bekende valkuilen van object georiënteerd programmeren worden voorkomen. Zo worden bijvoorbeeld ongewilde dependencies op verschillende lagen voorkomen wanneer er wordt gewerkt met domain driven design. In principe is Hexagonal Architecture dus een vorm van Domain Driven Desgin, er wordt bijvoorbeeld ook gebruik gemaakt van het layer pattern.

(Wikipedia contributors, 2021)



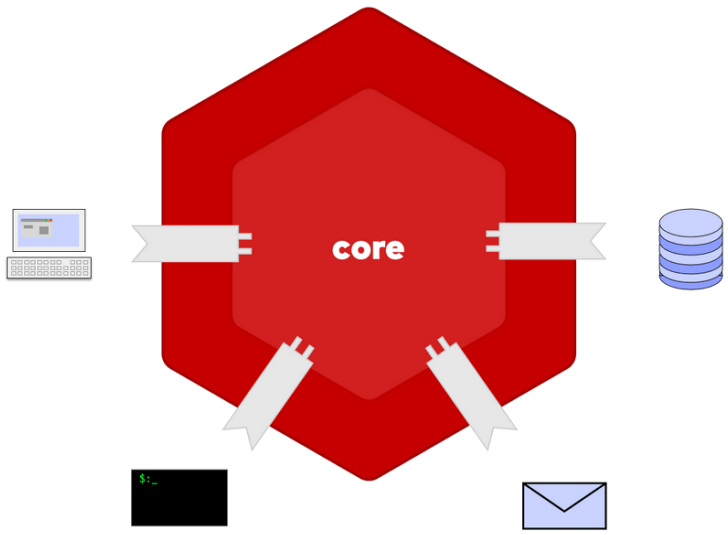
Figuur 1: Hexagonal architecture layers toepasbaar bij Hexagonal Architecture

4.1. Het principe

Hexagonal architecture werkt door een systeem op te delen in verschillende modulaire en laag gekoppelde onderdelen. Bij deze onderdelen kan er bijvoorbeeld worden gedacht aan een database, de user interface of een kleiner onderdeel zoals logica voor het sturen van een e-mail. Door deze onderdelen los te koppelen van de kern van de applicatie is het mogelijk om makkelijk te kunnen wisselen van gebruikte technieken, zo kan bijvoorbeeld gewisseld worden van het type database zonder dat daarvoor de complete applicatie hoeft te worden herschreven.

(Cockburn, 2008)

Het verschil tussen een "normaal" lagen patroon en hexagonale architectuur wordt zichtbaar in de volgende illustratie:



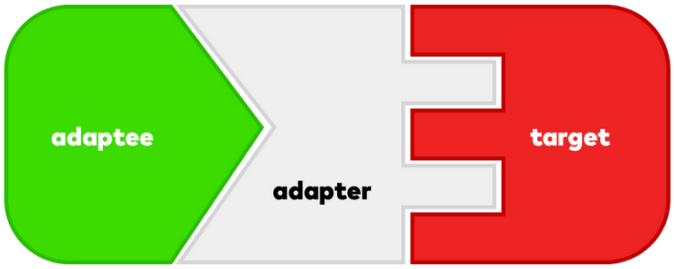
Figuur 2: Overview van Hexagonal Architecture

Op *figuur 1* is te zien hoe een voorbeeld applicatie is opgedeeld. Er is een hexagoon te zien in het midden van het figuur, dit illustreert de kern van de applicatie (ook wel het domein). Binnen deze kern valt de primaire business logica, zonder framework of boilerplate.

Buiten de hexagoon staan verschillende *adapters* afgebeeld. Een *adapter* kan een externe API zijn van de applicatie of een cliënt zijn van een compleet ander systeem. Ze vertalen de koppeling van externe systemen (bijvoorbeeld een zoekmachine of bestandsserver) naar de koppeling die het domein blootstelt. Deze koppelingen van het domein heten *poorten (ports)*.

Poorten maken het mogelijk om adapters *in the pluggen* in het hoofddomein. Een voorbeeld zou een *repository* kunnen zijn met een methode die de inhoud van een artikel teruggeeft als simpele string. Door een poort te declareren, bijvoorbeeld als een standaard Java interface, declareert het domein een soort contract wat het volgende aangeeft: "Ik geef je een *id* en ik verwacht tekst terug, hoe je eraan komt is jouw probleem". Het domein heeft alleen te maken met artikelen, welke een titel en content hebben. Het domein heeft verder totaal geen weet van JSON, binaire bestanden of hoe de data dan ook wordt opgehaald.

(*Hexagonal Architecture by Example - a Hands-on Introduction*, 2020)



Figuur 2: Een illustratie van een poort binnen Hexagonal Architecture. De adaptee kan hier een database zijn, de adapter beschrijft de API en de target is de kern van de applicatie

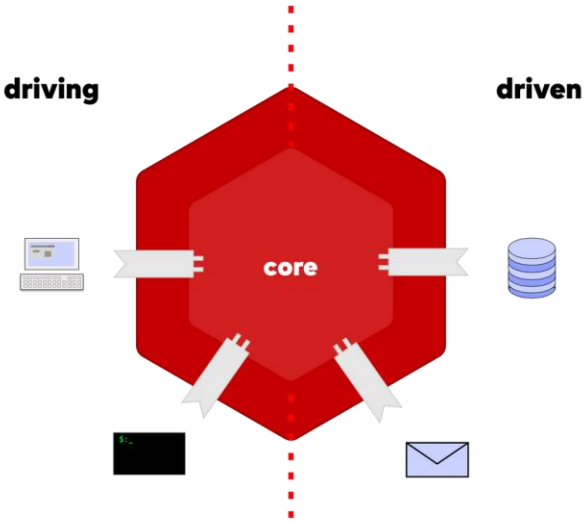
4.2. Dieper over poorten en adapters

In Hexagonal Architecture wordt gesproken over verschillende soorten poorten; input en output poorten. Het verschil tussen deze twee poorten zit hem eigenlijk al in de naam. Input poorten worden aangeroepen door input adapters om iets gedaan te krijgen. Ze zijn geen onderdeel van de kern maar hebben er wel interactie mee. Output poorten worden aangeroepen door output adapters die weer worden aangeroepen vanuit usecases binnen de kern. Poorten worden altijd aangeroepen door adapters.

(*Hexagonal architecture demystified,* 2019)

Een input adapter, ook wel *driving adapter* genoemd, zou bijvoorbeeld een web interface kunnen zijn. Wanneer een gebruiker op een knop klikt in de browser roept de web adapter een input poort aan die daarna een usecase aanroept in de kern.

Een output adapter, ook wel *driven adapter* genoemd, kan bijvoorbeeld een database zijn. Wanneer de usecase data uit de database nodig heeft roept de usecase een output adapter aan die een output port aanroept.

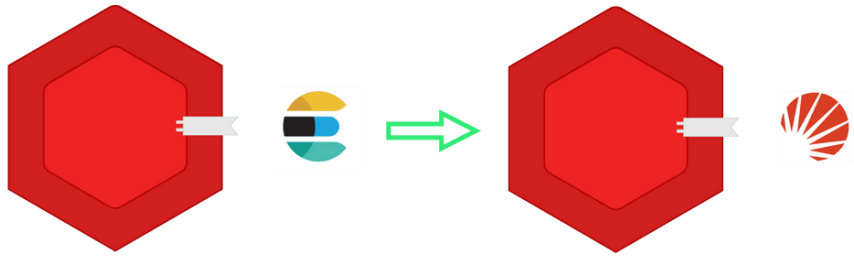


Figuur 3: Een illustratie van driving adapters (links) en driven adapters (rechts)

5. Voor- / Nadelen

5.1. Wisselen van technologieën

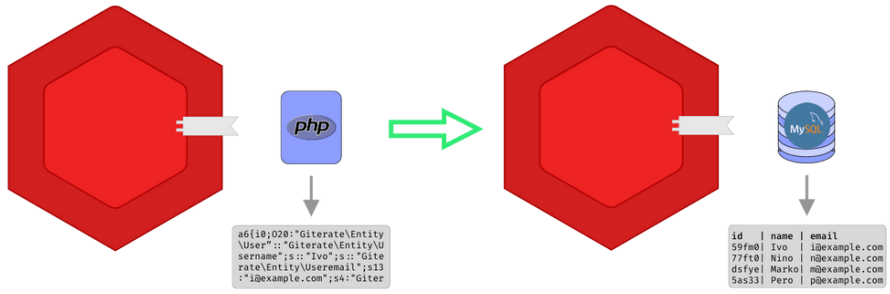
Adapters maken het zoals eerder genoemd makkelijk om de achterliggende technologie te veranderen zonder dat een heel stuk van de applicatie hoeft te worden herschreven. Plug een nieuwe adapter in voor een andere database en er hoeft in principe niks worden aangepast omdat de API van de adapter vooraf is gedefinieerd.



Figuur 4: Een illustratie van de mogelijkheid van het wisselen van technologieën

5.2. Uitstellen van technische keuzes

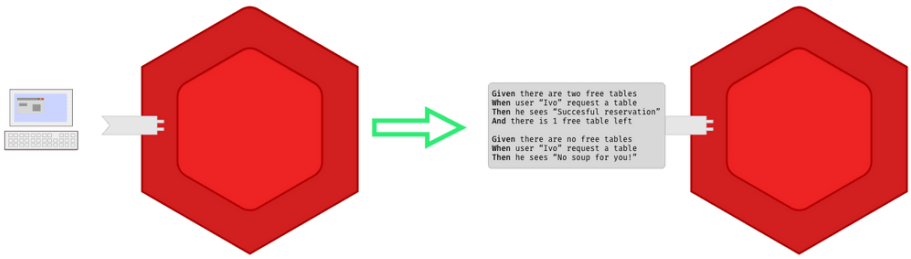
Doordat er een API wordt vastgelegd aan het begin van het project hoeft er ook niet direct gekozen te worden welke technologie er op de achtergrond wordt gebruikt. Meestal kies je bij het starten van een nieuw project direct welke database en front-end je gaat gebruiken alleen hoeft dit bij Hexagonal Architecture niet. Er kan bijvoorbeeld worden gekozen voor het gebruik van een in-memory database totdat de applicatie in productie gaat. Wanneer dit gebeurt kan er makkelijk worden gewisseld naar bijvoorbeeld een SQL database.



Figuur 5: Een PHP implementatie (links) en een MySQL implementatie (rechts)

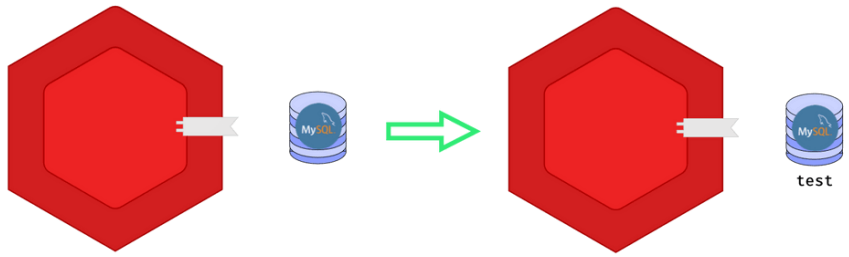
5.3. Testbaarheid code

Als het gaat om testen maakt Hexagonal Architecture het mogelijk de applicatie te testen los van externe afhankelijkheden. Aan de *driving* kant kunnen we onze echte adapters verwisselen met testadapters:



Figuur 6: Een illustratie van een testadapter aan de driving kant

Aan de *driven* kant kunnen we onze echte adapters verwisselen met gemockte adapter-implementaties:



Figuur 7: Een illustratie van een gemockte testadapter aan de driven kant.

Doordat de adapters zo makkelijk gemockt kunnen worden is het testen van deze applicatie een stuk overzichtelijker. Ook maakt het het schrijven van unit-tests makkelijker omdat je precies weet welke API je moet implementeren voor een mock. Door een framework als Mockito (https://site.mockito.org/) te gebruiken is het mogelijk om een gemockte adapter te maken in 3 regels code.

5.4. Complex

Door alle porten en adapters worden programma's zeer complex. Deze moeten aan het begin van het project worden uitgedacht en geïmplementeerd en ieder voegt een extra laag toe aan de architectuur. Als een port moet worden aangepast, moeten alle adapters die de port gebruiken ook worden aangepast. Ook worden er meer value objects en data mappers gebruikt, omdat de kern gescheiden moet blijven van de buitenwereld. Dit alles kost meer tijd en geld om te maken en onderhouden.

6. Demo-applicatie

Om een nog beter beeld te krijgen hoe Hexagonal Architecture in zijn werk gaat is er voor dit onderzoek een demo applicatie gemaakt. De basisfunctionaliteiten van een CRM zijn geïmplementeerd om te kijken hoe Hexagonal Architecture zich voor zo'n project schikt. Om de applicatie zo simpel mogelijk te houden en om de focus op het onderzoek te leggen is in overleg met JDI gekozen om alleen API endpoint(s) te exposen die kunnen worden aangeroepen met Swagger.

6.1. Wat is een CRM?

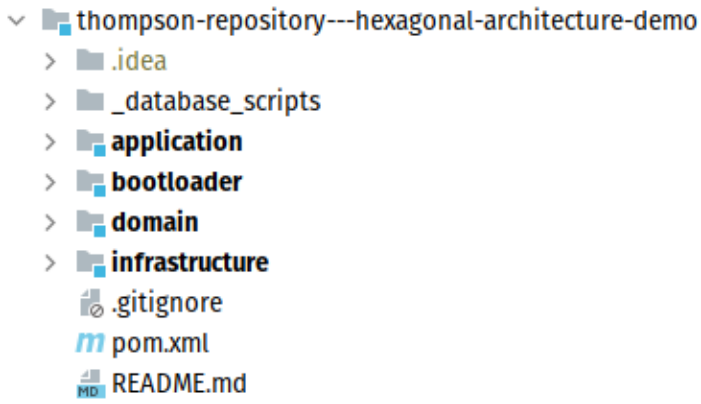
Customer relationship management *oftewel* ***CRM*** *is een Engelstalige benaming voor klantrelatiebeheer, soms ook relatiemarketing of verkoopbeheersysteem genoemd. Het is een werkwijze alsmede een technologie waarbij klantgegevens worden geanalyseerd om de zakelijke relatie met klanten te verbeteren, met als doel hen aan het bedrijf of de organisatie te binden en zo uiteindelijk de inkomsten te verhogen.*

(Wikipedia-bijdragers, 2021)

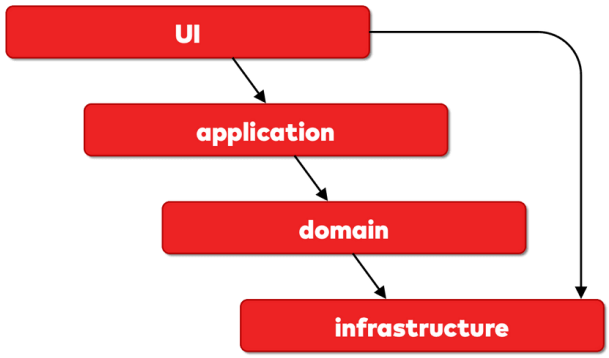
Omdat dit project de focus legt op Hexagonal Architecture en niet de CRM zelf hebben we in overleg met JDI gekozen om de implementatie zo simpel mogelijk te houden. De applicatie bevat een CRUD **(create, read, update, delete)** endpoint voor een klant **(customer).**

6.2. Applicatiestructuur

De structuur van de demo applicatie is te zien in *figuur 8.* Zoals te zien zijn de verschillende lagen (zoals eerder genoemd in H4) opgedeeld door gebruik te maken van folders. Dit zorgt er voor dat al in een oogopslag duidelijk wordt waar welk gedeelte van de code te vinden is. Elke folder bevat een Maven module die onafhankelijk van de rest van de modules kan worden gebouwd. Omdat deze applicatie geen UI heeft (endpoints worden aangeroepen met Swagger) ontbreekt deze folder uit de structuur. De **bootloader** folder bevat de code die nodig is om de applicatie op te starten (Spring Boot initialisatie).



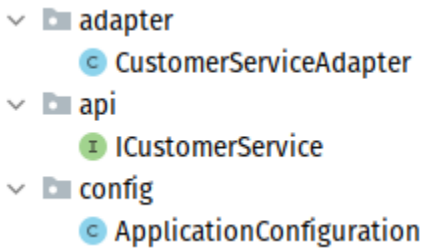
Figuur 8: Applicatiestructuur demo HA



Figuur 9: Lagen uit H3 ter verduidelijking

6.2.1. *De application laag*

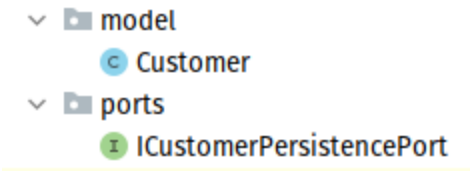
De application laag bevat de definitie van services en adapters die worden gebruikt binnen het project. De *api* package bevat een interface die de API definieert voor elke port die gebruikt wordt binnen de applicatie. In de *ApplicationConfiguration* wordt aangegeven welke implementatie van de adapter gebruikt moet worden, omdat er meerdere in een project kunnen zitten. Denk hierbij aan een in-memory- en MySQL database.



Figuur 10: Codestructuur application laag

6.2.2. De *domain* laag

De domain laag bevat alle business logica van de applicatie, de entiteiten, events en alle andere types die business logica bevatten. De structuur is in *figuur 11* afgebeeld. De *model* package bevat de entiteiten die worden gebruikt in de applicatie, meestal komen deze uit de database. De *ports* package bevat definities van de *poorten* de kunnen worden geimplementeerd binnen de applicatie. De interface *ICustomerPersistancePort* bevat een uitlijning van de API om een customer op te slaan als voorbeeld, deze code staat ter verduidelijking ook in *figuur 12* opgenomen.



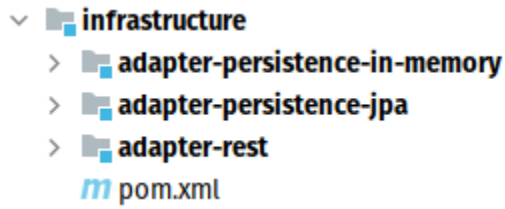
Figuur 11: Codestructuur domain laag

```
public interface ICustomerPersistencePort {
    void addCustomer(Customer customer);
    void removeCustomer(Integer customerId);
    List<Customer> getCustomers();
    Customer getCustomerById(Integer customerId);
}
```

Figuur 12: Code ICustomerPersistancePort

6.2.3. De *infrastructure* laag

De *infrastructuur* laag bevat alle implementaties van de adapters die in de *application* laag zijn gedefinieerd. Het demo project bevat een persistence adapter voor in-memory opslag en een persistence adapter voor JPA opslag. In *figuur 13* is te zien dat door een project op deze manier op te delen er gelijk duidelijk is welke functionaliteiten de applicatie bevat. Ook is het mogelijk om achteraf te wisselen van implementatie zonder een stuk van de applicatie te herschrijven. Er is bij de demo applicatie bijvoorbeeld gekozen om eerst een in-memory implementatie van de database te maken en daarna een JPA implementatie.



Figuur 13: Codestructuur infrastructuur laag

6.3. Uitbreiden demo-applicatie

De demo applicatie is een mooie weergave van hoe Hexagonal Architecture toegepast kan worden binnen een CRM. Natuurlijk is het alleen nog wel redelijk abstract, daarom wordt in dit hoofdstuk beschreven hoe de demo applicatie verder kan worden uitgebreidt. Dit om hands-on ervaring op te doen en een beter beeld te krijgen van hoe een implementatie eruit kan zien. De stappen om een nieuwe driving adapter toe te voegen staan hieronder uitgewerkt. Ze beschrijven het proces om een *command line* adapter toe te voegen aan de applicatie waarmee deze kan worden aangesproken.

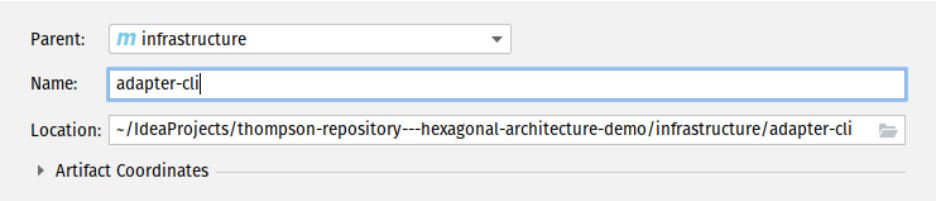
Er wordt in deze guide gebruik gemaakt van de IntelliJ IDE. In principe kunnen de instructies worden toegepast op elke IDE alleen zijn sommige handelingen dan net anders.

6.3.1. Stap 1 - Toevoegen adapter in infrastructure laag

De eerste stap in het toevoegen van een adapter (driving of driven) is het aanmaken van een nieuwe Java module in de *infrastructure* laag. Dit kan simpelweg worden gedaan door een rechterklik op de infrastructure module en de optie *New→Module* te kiezen binnen IntelliJ.

Kies voor de SDK die de module gebruikt de zelfde SDK als het project, in dit geval 17. Voor de naam van de module kan in principe alles worden gekozen. Om de standaarden van de demo applicatie aan te houden wordt de naam geprefixt met *adapter-*.

Let op! Selecteer als parent de infrastructure module! Deze staat standaard op de hoofdmodule (hexagonal-architecture-demo).

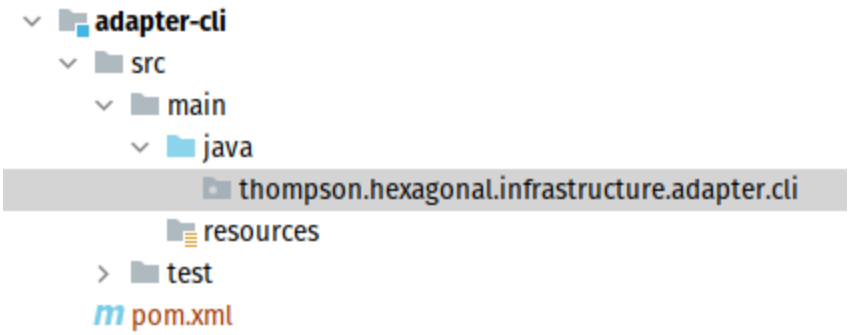


Figuur 14: Toevoegen nieuwe adapter-cli module.

Wanneer de naam akkoord is kan op *Finish* worden gedrukt.

6.3.2. Stap 2 - Toevoegen package binnen adapter-cli module

Nu dat de nieuwe module is aangemaakt kan er binnen de main java folder een nieuwe package worden aangemaakt. Het belangrijkste is om de package te prefixen met *thompson.hexagonal.infrastructure.* Er is hier gekozen voor de naam *thompson.hexagonal.infrastructure.adapter.cli.*



Figuur 15: Overzicht folderstructuur na stap 2

6.3.3. Stap 3 - Aanmaken controller voor Customer entiteit

Nu dat alle folders op de juiste manier binnen het project staan kan de controller worden aangemaakt voor de Customer entiteit. //TODO: Afmaken

6.4. Springboot in een HA applicatie

Omdat de CRM die uiteindelijk gebouwd wordt gebruik maakt van Springboot hebben we er voor gekozen om bij de demo applicaties ook van dit framework gebruik te maken, zodat het samenvoegen van de demo's uiteindelijk makkelijker is. Natuurlijk is er geen verplichting om bij Hexagonal Architecture gebruik te maken van Springboot, er kan binnen elke taal die object georiënteerd is een versie van Hexagonal Architecture worden geïmplementeerd.

De Inversion of Control (IoC) container binnen Springboot maakt het echter wel makkelijker om tot hetzelfde resultaat te komen. De dependencies binnen de applicatie hoeven namelijk niet door de programmeur zelf beheerd te worden. (Crusoveanu, 2021) Een repository klasse kan bijvoorbeeld geinjecteerd worden op alle plekken binnen de applicatie waar deze nodig is, zonder dat er elke keer een nieuwe instantie van aan hoeft worden gemaakt.

De concrete logica van de use-cases binnen de applicatie maken verder geen gebruik van Springboot specifieke technieken. De klasses voor het ophalen van data uit de database bijvoorbeeld zouden zo gekopieerd kunnen worden naar een project waar geen gebruik wordt gemaakt van Springboot.

7. Conclusie

Uit dit onderzoek is gebleken dat Hexagonal Architecture het erg makkelijk maakt om modulaire software te schrijven die goed testbaar is. Het is naar ons idee een erg schaalbare manier voor het indelen van een software applicatie. Door technische keuzes niet aan het begin van een project te hoeven maken is de opstart van projecten makkelijker. Een project kan simpeler worden getest omdat de domein laag volledig staat losgekoppeld van de rest van de applicatie. De use-cases kunnen dus los worden getest.

Door de adapter structuur toe te passen is het ook makkelijk om verschillende technieken te wisselen. Wanneer er bijvoorbeeld gewisseld moet worden van database is het makkelijk om hier een adapter voor te schrijven en deze toe te voegen. De rest van de applicatie hoeft verder niet aangepast te worden. Ook kan er makkelijk een andere front-end worden toegevoegd, bijvoorbeeld een mobiele app die de zelfde use-cases gebruikt als de desktop applicatie.

Het enige nadeel wat wij hebben ervaren tijdens het schrijven van de demo applicatie van Hexagonal Architecture is dat er veel tijd gaat zitten in de boilerplate code om de adapter structuur op te zetten. Er moeten aan het begin van het project al redelijk veel API's worden vastgelegd (adapters) zodat de applicatie daaromheen kan worden geschreven. Dit uit zich in de praktijk door veel verschillende interfaces waar vroeg in de ontwikkeling van het project al over nagedacht moet worden. Dit probleem kan worden opgelost door bijvoorbeeld gebruik te maken van een al bestaande boilerplate (H7.1). Deze templates geven al een goed ingerichte applicatiestructuur waarin verder ontwikkelt kan worden. Wanneer er gebruik wordt gemaakt van zo'n boilerplate zijn de negatieve kanten van Hexagonal Architecture eigenlijk volledig weggewerkt.

Zelfs met gebruik van een boilerplate schikt Hexagonal Architecture zich toch beter voor grotere projecten. Wanneer er snel iets gebouwd moet worden waarbij er weinig complexe relaties zitten tussen entiteiten is het makkelijker om voor een traditionele manier van softwareontwikkeling te gaan (zoals een gelaagde architectuur). De opzet van het project duurt een stuk korter en de voordelen van Hexagonal Architecture komen niet goed tot hun recht (gebeurt pas bij complexe grote applicaties).

TLDR:

Hexagonal Architecture werkt het best voor grote software applicaties. De techniek is zeer modulair en makkelijk te onderhouden / testen alleen vereist veel boilerplate code. Er bestaan gelukkig al veel starter template projecten, sommige van deze staan in H7.1 benoemd.

7.1. Boilerplates / Templates

Hieronder een aantal boilerplates die gebruikt kunnen worden tijdens het werken met Hexagonal Architecture:

- https://github.com/dolap-tech/spring-ddd-boilerplate (Java)
- https://github.com/CodelyTV/typescript-ddd-example (JavaScript / TypeScript)
- https://github.com/ivanpaulovich/hexagonal-architecture-acerola (C#)

8. Advies

In dit hoofdstuk willen we als team Thompson concreet advies geven binnen welke projecten Hexagonal Architecture toepasbaar zou kunnen zijn binnen JDI. Er is gekeken naar verschillende cases die JDI in het verleden heeft opgepakt en voor deze cases zijn de hypothetische voor en nadelen opgesteld.

8.1. Case 1 - Circulus Berkel: meegroeiende webapplicatie

https://www.jdi.nl/cases/case-circulus-berkel-meegroeiende-webapplicatie

Beschrijving van de case op de site van JDI: *Door digitalisering op basis van talloze geautomatiseerde koppelingen met haar interne ICT-infrastructuur wil Circulus Berkel haar ruim 610.000 klanten maximaal ondersteunen. Deze case beschrijft de reis naar hun stip aan de horizon: een uniek meegroeiende webapplicatie met koppelingen naar diverse systemen.*

Een project als deze lijkt ons een goede kandidaat voor de toepassing van Hexagonal Architecture. De case wordt op de site verder beschreven als *een unieke webapplicatie met koppelingen naar diverse systemen.* Hexagonal Architecture maakt het mogelijk om een applicatie modulair te laten groeien door nieuwe adapters met functionaliteit toe te voegen. Een koppeling maken naar veel diverse systemen is een van de beste voorbeelden waarbij Hexagonal Architecture toegepast kan worden. Door voor elk soort systeem een adapter aan te maken kan de applicatie blijven groeien terwijl hij onderhoudbaar blijft omdat de koppeling tussen de verschillende systemen erg laag is.

Het makkelijk laten groeien van de applicatie past dan weer goed bij een ander stuk van de beschrijving op de pagina van de case:

Optimaliseren: maximale ondersteuning door meegroeiende webapplicatie

Gezamenlijk blijven wij continu onderzoeken welke mogelijkheden en kansen er te benutten zijn. Dit met als doel dat de webapplicatie meegroeit met de organisatie en de steeds veranderende behoefte van de klant. Zo is er onlangs een proef gedraaid met luiercontainers. Waarmee gehoor werd gegeven aan de oproep van de inwoners om luiers en incontinentiemateriaal van het restafval te kunnen scheiden. Deze proef was een succes en heeft geresulteerd in een vaste luiercontainer. Mede door dit soort acties en te blijven onderzoeken welke kansen er te benutten zijn, blijft Circulus Berkel haar klanten alsmede haar interne organisatie maximaal online ondersteunen bij de vragen die zij hebben.

Ook weer na het lezen van deze beschrijving lijkt Hexagonal Architecture een passende oplossing. Een applicatie die blijft groeien moet makkelijk en onderhoudbaar uit te breiden zijn, Hexagonal Architecture maakt dit mogelijk.

Team Thompson adviseert voor cases die vergelijkbaar zijn met deze Hexagonal Architecture te gebruiken.

8.2. Case 2 - LED-Scores app met verdienmodel

https://www.jdi.nl/cases/led-scores-app-met-verdienmodel

Beschrijving van de case op de site van JDI: *LED Visuals is gespecialiseerd in het ontwikkelen, leveren en monteren van indoor en outdoor LED schermen. Deze schermen zijn vaak terug te vinden bij onder andere evenementen, langs de weg of langs sportvelden als scorebord. Juist voor dit laatste zag LED Visuals een kans liggen. Het invoeren van informatie op een scorebord gebeurd namelijk bij het scorebord zelf en is niet op afstand mogelijk. Reden voor LED Visuals om bij ons aan te kloppen om het op afstand besturen van LED schermen mogelijk te maken!*

Dit project lijkt ons een minder goede kandidaat voor de toepassing van Hexagonal Architecture. Bij de case staat de volgende verdere beschrijving:

De start van het ontwikkelen van de webapplicatie begon met het inrichten van een demo-omgeving in de bestaande backoffice van JDI. In deze omgeving was het mogelijk om de app na te bootsen en de aansturing van het scorebord te testen. Daarnaast werd er een separaat scherm ingericht met daarin alle functionaliteiten gevangen die ook in de app gaan komen. Dit scherm werd gekoppeld aan de demo-omgeving, zodat er een interactie ontstond tussen beiden. Aangezien het daadwerkelijke scorebord niet de hele tijd beschikbaar was, is het scorebord vervangen door een webpagina.

Voor het ontwikkelen van een demo applicatie vereist Hexagonal Architecture naar ons idee teveel boilerplate code per use-case om er een realistisch voordeel uit te halen. Tijdens het schrijven van een dergelijke testapplicatie wil je snel functionaliteit kunnen toevoegen en wijzigen, en omdat deze functionaliteit uiteindelijk opnieuw geïmplementeerd moet worden in

de productie applicatie, maakt de onderhoudebaarheid van deze code minder uit. Een "normaal" lagen patroon zou bij een demo applicatie als deze beter passen.

Ook bij het maken van de productie app in het geval van deze case adviseert team Thompson tegen het gebruik van Hexagonal Architecture. Op de pagina van de case staat dat de volgende technieken worden gebruikt:

JDI Backoffice, Flutter, WebSockets, Vue.js, SpringBoot, Java.js, S3 Objectstore

Flutter heeft meerdere verschillende technieken voor het opdelen van code die als industry standard worden gezien. Neem bijvoorbeeld het bloc pattern (https://bloclibrary.dev/). Verschillende paradigma's uit Hexagonal Architecture en Domain Driven Development worden overgenomen door het bloc pattern. Het voordeel van dit patroon is alleen dat er op de achtergrond al veel boilerplate code is geschreven. De library bevat verschillende "helper" widgets die ervoor zorgen dat het schrijven van onderhoudebare Flutter code simpeler is. Mede hierdoor adviseren wij dat een rouwe implementatie van Hexagonal Architecture hieraan inferieur is.

Vue.js heeft ook zijn eigen industry standards voor het opdelen van code. Meestal wordt er gebruik gemaakt van een standaard gelaagde architectuur. Hexagonal Architecture kan worden overwogen als het inzicht bestaat dat de applicatie in de toekomst veel gaat groeien. Door gebruik te maken van het adapter patroon binnen Hexagonal Architecture wordt de code hierdoor goed onderhoudbaar.

Naar ons idee is Hexagonal Architecture voor Springboot wel een goede kandidaat. Tijdens het schrijven van onze demo applicaties zijn we erachter gekomen dat wanneer er een solide fundatie ligt met Hexagonal Architecture het uitbreiden van de app erg vlot verloopt.

Het is voor deze case lastig om concreet advies te geven omdat er veel verschillende technieken samenwerken om de applicatie te bouwen. Het advies wat wordt uitgebracht geld hier voor de app (front-end) van de applicatie.

Team Thompson adviseert voor cases die vergelijkbaar zijn met deze geen Hexagonal Architecture te gebruiken.

9. Discussie

De bronnen die gebruikt zijn tijdens dit onderzoek zouden nog verder gecontroleerd kunnen worden op authenticiteit en juistheid. Sommige bronnen zoals het Wikipedia artikel over Hexagonal Architecture zijn goed om een globaal overzicht te krijgen van de techniek maar zijn wat minder goed voor details en verdieping.

Het advies is redelijk oppervlakkig omdat team Thompson geen verdere inzage heeft in de opbouw van de code van de verschillende cases. Feedback is gegeven op basis van de beschrijvingen op de site van JDI.

Bij de demo applicatie was het achteraf beter geweest om een complexere applicatie te bouwen dan alleen wat simpele CRUD functionaliteit als CRM. Als er complexere use-cases waren opgenomen in de applicatie zou het doel van Hexagonal Architecture beter naar voren komen. Dit zou voor het rapport beter weergeven wanneer Hexagonal Architecture toepasbaar is. Toch is dit ergens ook positief omdat het de conclusie die we al eerder hadden getrokken (dat het nu van HA wegvalt bij kleinere applicaties) wordt bewezen. (Deze argumenten gelden alleen voor de tussentijdse HA demo. De volledige demo applicatie bevat wel complexe use-cases).

10. Bronnen

Blog, N. T. (2020, 12 maart). *Ready for changes with hexagonal architecture - Netflix TechBlog*. Medium. Geraadpleegd op 29 november 2021, van https://netflixtechblog.com/ready-for-changes-with-hexagonal-architecture-b315ec967749

Cockburn, A. (2008). *Hexagonal architecture*. Alistair Cockburn. Geraadpleegd op 29 november 2021, van https://alistair.cockburn.us/hexagonal-architecture/

Hexagonal Architecture by example - a hands-on introduction. (2020, 21 mei). Allegro.Tech. Geraadpleegd op 29 november 2021, van https://blog.allegro.tech/2020/05/hexagonal-architecture-by-example.html

Hexagonal architecture with Java and Spring. (2019, 3 november). Reflectoring.Io. Geraadpleegd op 29 november 2021, van https://reflectoring.io/spring-hexagonal/

Wikipedia contributors. (2021, 26 juli). *Hexagonal architecture (software)*. Wikipedia. Geraadpleegd op 29 november 2021, van https://en.wikipedia.org/wiki/Hexagonal_architecture_(software)

Wikipedia-bijdragers. (2021, 19 februari). *CRM*. Wikipedia. Geraadpleegd op 7 december 2021, van https://nl.wikipedia.org/wiki/CRM

Crusoveanu, L. (2021, 8 december). *Intro to Inversion of Control and Dependency Injection with Spring*. Baeldung. Geraadpleegd op 15 december 2021, van https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring

Spajic, Z. (2019, 4 augustus). *Hexagonal architecture demystified*. Madewithlove. Geraadpleegd op 2 december 2021, van https://madewithlove.com/blog/software-engineering/hexagonal-architecture-demystified/