13 JUNI 2020

DOMAIN-DRIVEN DESIGN

ONDERZOEK

Fontys
University of Applied Sciences

FURKAN DEMIRCI
FAMBOOK
Rachelsmolen 1, Eindhoven

Informatie

Versie: 1.0
Versiedatum: 13 jun. 20
Status: Definitief
Auteur: Furkan Demirci

Document geschiedenis

Versie	**Wijziging**	**Auteur**	**Datum**
0.1	Opstellen document	Furkan Demirci	10 jun. 20
0.2	Invullen deelvragen	Furkan Demirci	11 jun. 20
0.3	Invullen conclusie	Furkan Demirci	13 jun. 20
1.0	Afronden document	Furkan Demirci	13 jun. 20

Inhoud

Hoofdvraag

"Wat voor mogelijkheden biedt het toepassen van Domain-Driven Design met betrekking tot het realiseren van webapplicaties?"

Sub-vragen

1. Wat is Domain-Driven Design?
   Strategie: ***Bieb***

2. Wat zijn de voor- en nadelen van Domain-Driven Design?
   Strategie: ***Bieb***

3. Hoe wordt Domain-Driven Design toegepast in een Microservice architectuur?
   Strategie: ***Bieb*** en ***lab***

Wat is Domain-Driven Design?

Om een domein gedreven ontwerp te definiëren moeten we eerst vaststellen wat we in deze context onder domein gedreven ontwerp verstaan. De definitie van een domein in een algemeen woordenboek is: "Een kennis- of activiteitengebied". Als we daar iets van afleiden, verwijst het domein in het domein van software engineering meestal naar het vakgebied waarop de applicatie van toepassing is. Met andere woorden, tijdens de ontwikkeling van een applicatie is het domein het "kennis- en activiteitengebied waar de applicatielogica om draait".

Een andere veelgebruikte term die tijdens de softwareontwikkeling wordt gebruikt is de domein laag of de domein logica, die bij veel ontwikkelaars misschien beter bekend is als de business logic. De business logic van een applicatie verwijst naar de regels op een hoger niveau voor de manier waarop business objecten met elkaar omgaan om gemodelleerde gegevens te creëren en te wijzigen.

Domein gedreven ontwerp is de uitbreiding en toepassing van het domeinconcept, zoals het van toepassing is op de ontwikkeling van software. Het streeft ernaar om de creatie van complexe toepassingen te vergemakkelijken door de bijbehorende onderdelen van de software te verbinden tot een steeds evoluerend model.

DDD richt zich op drie kernprincipes:

- Focus op het kerndomein en de domeinlogica.
- Baseer complexe ontwerpen op modellen van het domein.
- Voortdurend samenwerken met domeinexperts om het toepassingsmodel te verbeteren en opkomende domein gerelateerde problemen op te lossen.

Domein gedreven ontwerp legt ook sterk de nadruk op continuous integration, die het hele ontwikkelingsteam vraagt om één gedeelde code repository te gebruiken en er dagelijks commits naar te pushen. Een automatisch proces voert aan het einde van de werkdag de integriteit van de gehele codebasis uit, waarbij geautomatiseerde unit-tests en dergelijke worden uitgevoerd om eventuele problemen die in de laatste commits zijn geïntroduceerd, snel op te sporen.

Wat zijn de voor- en nadelen van Domain-Driven Design?

Er zijn velen voor- en nadelen van Domain-Driven Design. Het meest belangrijkste voordelen en nadelen worden hier beschreven.

Gemakkelijke communicatie

Voordeel

Met een vroege nadruk op het creëren van een algemene en universele taal met betrekking tot het domeinmodel van het project, zullen teams vaak veel gemakkelijker kunnen communiceren gedurende de gehele ontwikkelingscyclus. Typisch zal DDD minder technisch terminologie vereisen bij het bespreken van aspecten van de applicatie, aangezien de in het begin vastgestelde universele taal waarschijnlijk eenvoudigere termen zal definiëren om te verwijzen naar die meer technische aspecten.

Verbetert de flexibiliteit
Voordeel

Omdat DDD zo sterk gebaseerd is op de concepten van objectgeoriënteerde analyse en ontwerp, zal bijna alles binnen het domeinmodel gebaseerd zijn op een object en dus vrij modulair en ingesloten zijn. Dit maakt het mogelijk om verschillende componenten, of zelfs het hele systeem als geheel, op een regelmatige, doorlopende wijze aan te passen en te verbeteren.

Benadrukt domein over interface
Voordeel

Aangezien DDD de praktijk is van het bouwen rond de domeinconcepten en wat de domeinexperts binnen het project adviseren, zal DDD vaak applicaties produceren die nauwkeurig geschikt zijn voor en representeren van het betreffende domein, in tegenstelling tot die applicaties die de nadruk leggen op de gebruikersinterface in de eerste plaats. Hoewel een duidelijke balans nodig is, betekent de focus op het domein dat een DDD-aanpak een product kan opleveren dat goed aansluit bij het publiek dat met het domein geassocieerd wordt.

Vereist robuuste domeinkennis
Nadeel

Zelfs met de meest technisch onderlegde mensen die aan de ontwikkeling werken, is het allemaal voor niets als er niet minstens één domeinexpert in het team zit die de exacte ins en outs kent van het vakgebied waarop de applicatie moet worden toegepast.

Stimuleert iteratieve handelingen
Voordeel/Nadeel

Hoewel velen dit als een voordeel beschouwen, kan niet worden ontkend dat DDD-praktijken sterk afhankelijk zijn van constante iteratie en continue integratie om een plooibaar project op te bouwen dat zichzelf kan aanpassen wanneer dat nodig is. Sommige organisaties kunnen moeite hebben met deze praktijken, vooral als hun ervaring in het verleden grotendeels gekoppeld is aan minder flexibele ontwikkelingsmodellen, zoals het watervalmodel of iets dergelijks.

Slecht geschikt voor zeer technische projecten
Nadeel

Aangezien DDD zo sterk de nadruk legt op de noodzaak (en het belang) van domeinexperts om de juiste taal en vervolgens het juiste domeinmodel te genereren waarop het project is gebaseerd, kan een project dat ongelooflijk technisch complex is voor domeinexperts een uitdaging zijn om te begrijpen, waardoor er later problemen kunnen ontstaan

Hoe wordt Domain-Driven Design toegepast in een Microservice architectuur?

DDD gaat over het modelleren van het echte domein door het eerst volledig te begrijpen, en alle terminologie, regels en logica te plaatsen in een abstracte representatie binnen je code, meestal in de vorm van een domeinmodel. DDD zelf is geen framework, maar heeft wel een aantal bouwstenen of concepten die je in je oplossing kunt verwerken:

De alomtegenwoordige taal

(Universele taal)

Om het domein te beschrijven moeten ontwikkelaars, architecten, domeinexperts en iedereen die betrokken is bij een project dezelfde taal spreken, dit wordt in de DDD-methodologie Alomtegenwoordige taal genoemd.

Entiteiten

Entiteit in DDD bevat gegevens en gedrag in het domeinmodel. Elke logica die betrekking heeft op een entiteit moet erin worden opgenomen. Entiteiten zijn de dingen die een identiteit nodig hebben, die er gedurende het hele leven bij zal blijven. Voor een databaseperspectief kunnen we denken aan een entiteit als een databasetabel, maar dan met een bepaald gedrag.

Aggregates

Het domeinmodel heeft een methode nodig om deze associaties te beheren. Belangrijker nog, logische groepen van entiteiten en waarde objecten moeten een interface definiëren die andere entiteiten met hen laat werken. Zonder een dergelijke structuur kan de interactie tussen groepen van objecten verwarrend zijn en later tot problemen leiden.

Domeindiensten

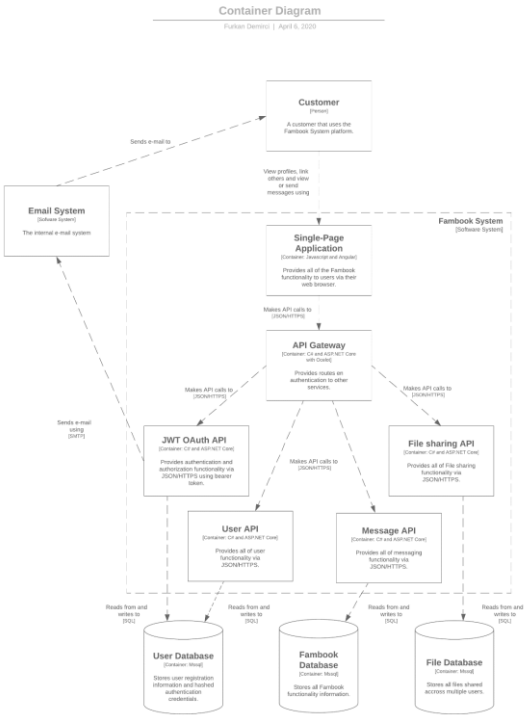
In DDD zijn methoden die niet echt op een enkele entiteit passen of die toegang tot de repository vereisen, opgenomen in domeinservices, De domeinservice laag kan ook eigen domeinlogica bevatten en maakt net zo goed deel uit van het domeinmodel als entiteiten en waarde objecten.

Repository

Het Repository-patroon, functioneert als een in-memory-verzameling voor bedrijfsentiteiten, waarbij de onderliggende data-infrastructuur volledig wordt weggenomen. Dit patroon stelt in staat om domeinmodel vrij te houden van alle zorgen over de infrastructuur.

Layered architecture

Als laatst is het uit elkaar halen van alle onderdelen een belangrijk concept omdat dit helpt met de separation of concerns. Een voorbeeld is hoe het nu in de proftaak en individueel is toegepast. Zie figuur 1.



Figuur 1 - Multi layered architecture

Conclusie

Als een software engineer die zich later meer wil focussen op het bedenken van complexe architecturen in web apps kan Domain-Driven Design een rol in spelen. Het bedenken en maken van applicatie voor klanten is het kennen van het domein hier noodzakelijk. Voor onze proftaak hebben we in het begin al met elkaar besloten om DDD hiertoe te passen. Gezamenlijk als team en onze PO voeren we gesprekken zodat we als team het domein (dus de in's and out's van het concept) beter begrijpen. Het toepassen van microservices met de benodigde bouwstenen voor een DDD-aanpak, de gemakkelijke communicatie tussen ons, flexibiliteit, domein over interface en iteratieve handelingen is wat we toegepast hebben in onze proftaak. Niet alleen in de proftaak maar ook individueel heb ik de bouwstenen voor een DDD-aanpak toegepast.

Een antwoord op de vraag:

"Wat voor mogelijkheden biedt het toepassen van Domain-Driven Design met betrekking tot het realiseren van webapplicaties?"

DDD biedt veel mogelijkheden genoemd in deelvraag 2 als voordelen. Het is voor mij belangrijk dat ik weet hoe het werkt als een team domein gedreven te werk gaat. Dit wordt vaak toegepast bij bedrijven waar webapplicaties wordt gemaakt voor andere klanten. Als klant wil je natuurlijk een werkend applicatie naar jouw wensen dat aansluit op jouw domein. Een klant is niet echt geïnteresseerd in welke technologieën of modellen je hebt toegepast. Wat DDD ook als voordeel komt voor een klant is dat de applicatie continu up en running zal moeten blijven. Continues integration speelt hier dus ook een rol bij. Als student die later bezig zal zijn met dit soort ontwerpen heb ik veel geleerd over hoe DDD werkt en hoe het wordt toegepast.

Verwijzingen

Fotso, J. P. (2019, November 20). *Domain Driven Design (DDD)in Microservice architecture in a nutshell*. Opgehaald van Medium: https://medium.com/@jpdeffo/domain-driven-design-ddd-in-microservice-architecture-for-nutshell-19c7c579009a

Kraak, A. (2020, Februari 18). *Domain Driven Design (DDD) in gewoon Nederlands*. Opgehaald van Bergler: https://www.bergler.nl/domain-driven-design-in-gewoon-nederlands-artikel-2-het-model

Meuter, E. (sd). *Een eigenzinnige kijk op Domain Driven Design*. Opgehaald van nljug: https://nljug.org/java-magazine/een-eigenzinnige-kijk-op-domain-driven-design/

Powell-Morse, A. (2017, April 21). *Domain-Driven Design – What is it and how do you use it?* Opgehaald van Airbrake: https://airbrake.io/blog/software-design/domain-driven-design

Shah, S. (2018, Maart 12). *Role of Enterprise Architecture in Domain-Driven Design*. Opgehaald van LinkedIn: https://www.linkedin.com/pulse/role-enterprise-architecture-domain-driven-design-shreyansh-shah/