

ARCHITECTURALE KARAKTERISTIEKE N

CASE

- Sillycon Symposia: bedrijf dat ludieke technologieconferenties organiseert
- Ter ondersteuning: Lafter, een sociaal medium voor sprekers en bezoekers

REQUIREMENTS DOCUMENT

(1)

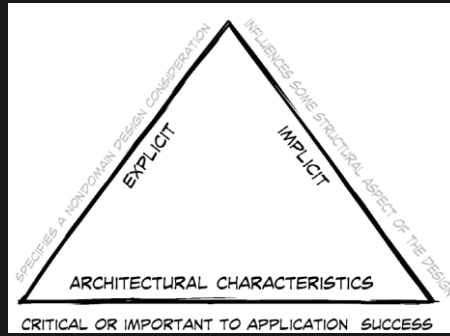
- Gebruikers: (honderden) sprekers,
(duizenden) bezoekers
- Requirements:
 - Users kunnen een account aanmaken
 - Users kunnen "jokes" (lange posts) en "puns" (korte posts) maken
 - Users kunnen berichten tot 281 karakters posten
 - Users kunnen links posten

REQUIREMENTS DOCUMENT

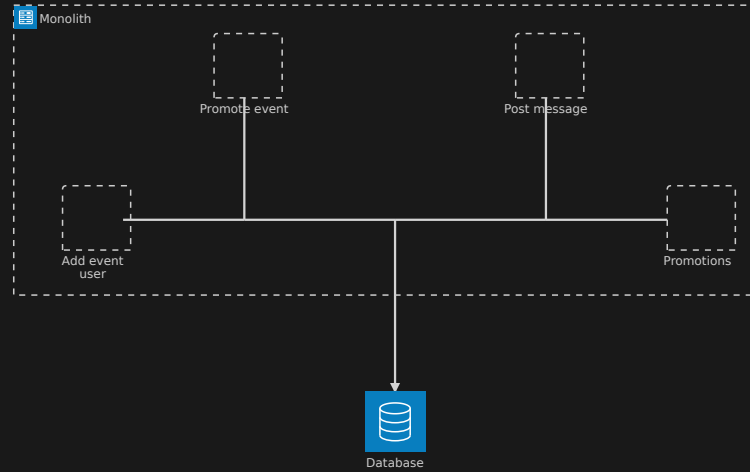
(2)

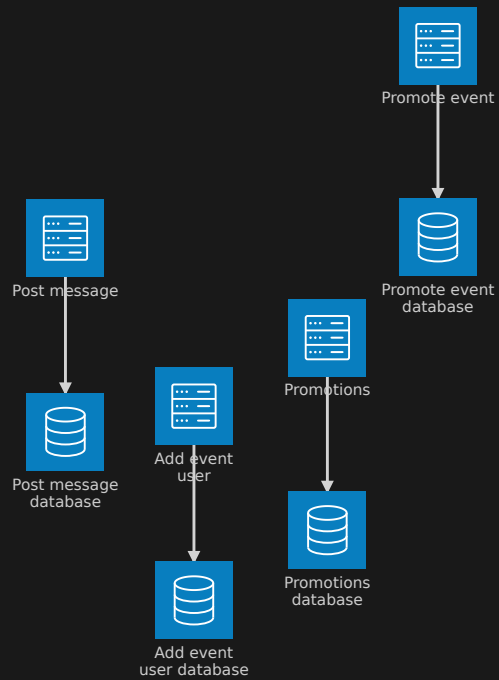
- Followers kunnen "Haha" of "Giggle" reageren
- Sprekers hebben een eigen icoontje
- Sprekers kunnen een forum opzetten over hun eigen topic
- Context:
 - Ondersteuning voor internationaal gebruik
 - Kleine support staff

- "Nondomain design considerations":
karakteristieken zijn niet het **wat**, maar het **hoe** van de oplossing
 - "Domain" = het gebied waarin we bezig zijn
- "De architecturale karakteristieken vormen criteria voor succes, geven aan hoe we de requirements moeten implementeren en waarom bepaalde keuzes gemaakt zijn."



IMPLICIETE INVLOED OP STRUCTUUR





BEPERKEN VAN DE KARAKTERISTIEKEN

- kunnen niet alles doen
- geen exhaustieve lijst of universele terminologie
- synergieën
- toename van criteria

BRONNEN VAN KARAKTERISTIEKEN

- het probleemdomein
- kennis van de omgeving (is klant een startup, een groot bedrijf,...?)
- ervaring met het domein

KARAKTERISTIEKEN SILLYCON SYMPOSIA

- account aanmaken
- "jokes" en "puns" maken
- berichten tot 281 karakters posten
- links posten
- "Haha" of "Giggle" reageren
- een eigen icoontje
- sprekers kunnen een forum opzetten over hun eigen topic
- ondersteuning voor internationaal gebruik
- kleine support staff
- pieken in verkeer (tijdens conferenties)
- scalability (veel users)
- elasticity (pieken)
- authentication (users)
- authorization (soorten users)
- internationalization (zie context)
- customizability (aanmaak forums, icoontjes)
- fault tolerance (kleine support staff)

OPLOSSINGEN VS. REQUIREMENTS

- klant: "nieuw vliegtuig moet mach 2.5 halen"
- redenering: duur, dus moet kunnen ontsnappen
- probleem: beperkt door technologie
- oplossing: zeer manoeuvreerbaar i.p.v. zeer snel


COMPOSITES

- brede karakteristieken zoals "performant" of "betrouwbaar"
- soms nodig op te breken in **meetbare** karakteristieken
 - bv. performance
 - eerste page load time
 - totale laadtijd
 - gemiddelde responstijd
 - bovengrens responstijd

KEUZES MAKEN

- kies de belangrijkste 7 ("driving characteristics")
- maak achteraf een top 3
- impliciete zijn altijd zinvol, herhaal ze eventueel onder "driving"

KEUZES MAKEN (DIAGRAM)

Top 3	Driving Characteristics	Implicit Characteristics
<input type="checkbox"/>	<hr/>	<hr/> <i>feasibility (cost/time)</i> <hr/>
<input type="checkbox"/>	<hr/>	<hr/> <i>security</i> <hr/>
<input type="checkbox"/>	<hr/>	<hr/> <i>maintainability</i> <hr/>
<input type="checkbox"/>	<hr/>	<hr/> <i>observability</i> <hr/>
<input type="checkbox"/>	<hr/>	 <p>We've found these four to be pretty common implicit characteristics. Feel free to replace these with whatever makes sense for you.</p>
<input type="checkbox"/>	<hr/>	
<input type="checkbox"/>	<hr/>	

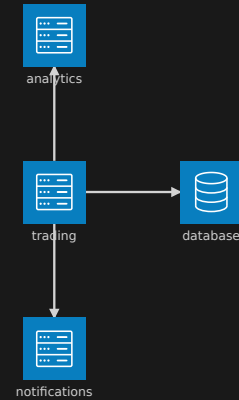
ARCHITECTURALE BESLISSINGEN

CASE: TWO MANY SNEAKERS

van



naar

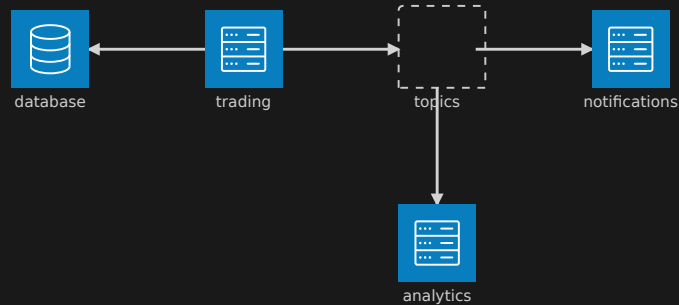
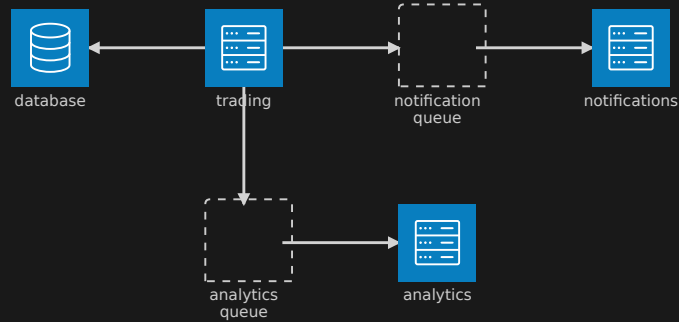


"Hoe gaan we deze bijkomende interacties implementeren?"

OPTIES

1. web services (REST/GraphQL/...)
2. message queues (kan met RabbitMQ)
3. topics (kan ook met RabbitMQ)

MESSAGE QUEUES VS. TOPICS



web services	queues	topics
goed gekend	matig gekend	matig gekend
heterogene berichten	heterogene berichten	homogene berichten
synchroon	asynchroon	asynchroon
sterke koppeling	matige koppeling	zwakke koppeling
service mag niet uitvallen	buffering + gedetailleerde scaling	buffering
extra werk security		extra werk security
...

EERSTE WET VAN SOFTWARE ARCHITECTUUR

Everything is a trade-off.

ARCHITECTURALE BESLISSINGEN

- kunnen structuur beïnvloeden
- kunnen genomen worden om karakteristiek te verkrijgen

TWEEDE WET VAN SOFTWARE ARCHITECTUUR

Why is more important than how.

- "How" is simpel: kijk naar de codebase
- "Why": **niet** op het zicht duidelijk

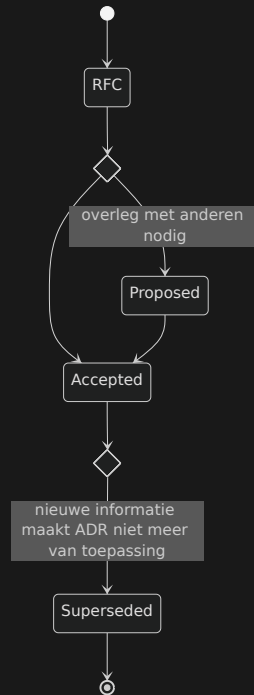
ARCHITECTURAL DECISION RECORDS

- document per architecturale beslissing
- vaste structuur (kan per bedrijf variëren)
- geen "user-facing documentation"
 - wel uitstekend voor nieuwe developers
- append-only
- plain text formaat is handigst en kan in versiebeheer
- typisch per project, kunnen op organisatieniveau

MOGELIJKE TEMPLATE

- title: nummer en samenvatting op één regel
- status: RFC, proposed, accepted of superseded
 - "superseded by XYZ" en "supersedes ABC" voor betere verwijzingen
- context: alle factoren om in rekening te brengen, zonder beslissing
- decision: de gemaakte keuze, kordaat uitgedrukt, samen met **objectieve** verantwoording
- consequences: wat wordt (on)mogelijk, welk werk ontstaat of valt weg,...
- governance: hoe zorgen we dat dit wordt opgevolgd (bewustmaking, audits,...)
- notes: algemene opmerkingen, metadata (kan deels via versiebeheersysteem)

TOESTANDSDIAGRAM



- Title: 012: Gebruik van queues voor asynchrone messaging vanaf tradingdienst
- Status: Accepted
- Context: De trading service moet andere diensten (voorlopig: analytics en notifications) op de hoogte brengen van nieuwe producten en van elke verkoop. Dit kan via web services (REST e.d.) of via asynchrone messaging (queues of topics).
- Decision: **We zullen queues gebruiken.** Queues maken het systeem veelzijdig, aangezien elke queue andere soorten berichten kan afleveren. Ze maken het systeem ook veiliger aangezien de trading service steeds weet wie de queues uitleest.
- Consequences: De koppeling tussen diensten zal hoger zijn, want elke nieuwe queue moet ondersteund worden door de trading service. We zullen ook infrastructuur voor de queues moeten voorzien.

