EXTERNE API – API GATEWAYS

Veerle Ongenae

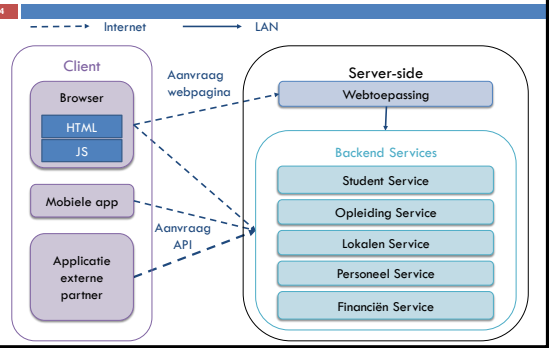
Bron

- Gebaseerd op hoofdstuk 8 uit het boek "Microservices Pattern" van Chris Richardson

Inhoud hoofdstuk

- Verschillende aspecten/overwegingen bij externe API-ontwerp
  - API's die verschillende clients ondersteunen – een uitdaging
- API gateway pattern en backends voor front end pattern
- Ontwerpen en implementeren van een API gateway
  - Kant en klaar producten
  - Frameworks voor eigen ontwikkeling (Spring Cloud Gateway, GraphQL)
- API samenstellen vereenvoudigen met reactive programming

Voorbeeld entreprise applicatie



Externe API voor services

- Verschillende clients
  - Mobiele apps
  - JS in een browser
  - Applicaties ontwikkeld door partners
- REST, SOAP, … API
  - Verschillende microservices met elke eigen API
    - Welke api beschikbaar voor clients?
    - Rechtstreeks de services aanspreken?

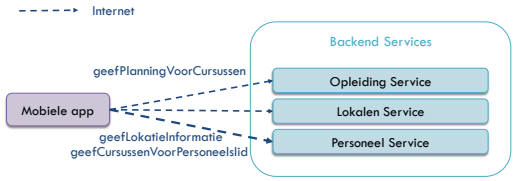
Uitdagingen ontwikkeling externe API

- Diversiteit clients
  - Verschillende data
    - Webapplicaties voor desktop meer data nodig dan mobiele app (toont meer)
    - Toegang tot API via verschillende types netwerken
      - LAN met firewall
      - Internet
      - Mobiel netwerk
- Een API voor alle type clients vaak niet zinvol

API-ontwerp

7

- Optie 1
  - Clients spreken services rechtstreeks aan
- Voorbeeld uurrooster voor personeelslid



API Composer

8

- Client heeft de rol van API Composer
  - Verschillende services aanspreken
  - Resultaten combineren
- Problemen/nadelen
  - Slechte gebruikerservaring
    - Verschillende aanvragen naar verschillende services om de nodige data op te halen
    - Inefficiënt
  - Gebrek aan afscherming backend
    - Clients kennen interne structuur
    - Moeilijk om architectuur te veranderen
  - Services kunnen IPC-mechanismen gebruiken
    - Niet handig voor externe clients

Probleem 1: slechte gebruikerservaring

9

- Slechte gebruikerservaring omdat er verschillende aanvragen gedaan worden → niet responsive (traag netwerk)
  - Parallel of sequentieel?
- Client moet complexere code bevatten om de resultaten van de API's te combineren
  - Primaire taak is echter goeie gebruikerservaring
- Verschillende aanvragen zijn slecht voor de batterij

Probleem 2: afscherming backend

10

- Backend niet afgeschermd
- Verandering backend → aanpassing code frontend
  - Opdeling in services kan ook aangepast worden
    - Nieuwe
    - Opsplitsing
    - Samenvoegen
- Uitrol nieuwe versie app
  - Kan verschillende dagen duren
  - Niet alle clients downloaden de nieuwe versie
- Ontwikkeling API's wordt belemmerd
- Externe partners hebben nood aan een stabiele API
  - Oude versies moet lang ondersteund worden
  - Aparte publieke API nodig

Probleem 3

11

- Services kunnen IPC-mechanismen gebruiken → client onvriendelijk
- Protocollen moeilijk/niet ondersteund door de client
- Buiten firewall
  - HTTP
  - WebSockets
- Services
  - gRPC
  - AMQP (messaging)
  - Lukt niet altijd door firewall
  - Niet eenvoudig aan te roepen in clients

Inhoud hoofdstuk

12

- Verschillende aspecten/overwegingen bij externe API-ontwerp
  - API's die verschillende clients ondersteunen – een uitdaging
- API gateway pattern en backends voor front end pattern
- Ontwerpen en implementeren van een API gateway
  - Kant en klaar producten
  - Frameworks voor eigen ontwikkeling (Spring Cloud Gateway, GraphQL)
- API samenstellen vereenvoudigen met reactive programming

API Gateway - waarom

13

- Directe toegang tot services is nadelig
  - Niet praktische om API-compositie uit te voeren over het internet
  - Gebrek aan afscherming maakt het moeilijk voor ontwikkelaars om services te veranderen
  - Niet alle services gebruiken communicatieprotocollen die geschikt zijn om te gebruiken buiten een firewall
- Beter
  - API Gateway

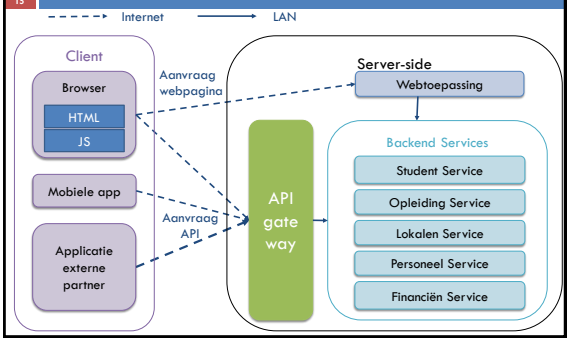
API Gateway

14

- Toegangspunt tot de service voor de buitenwereld
- Verantwoordelijk voor
  - Request routing
  - API-compositie, aggregatie/samenvoegen resultaten
  - Vertaling naar protocollen geschikt voor de client
  - Andere functionaliteit
    - Authenticatie, …

Voorbeeld entreprise applicatie

15



API gateway pattern

16

- Client één request naar API gateway
- API gateway is service
  - Enige toegangspunt voor API-aanvragen van buitenaf
  - ~ Facade
  - Afschermen interne structuur
  - (Vereenvoudigde) API voor clients
  - Eventueel andere verantwoordelijkheden: autenticatie, monitoring, “rate limiting”

Request Routing

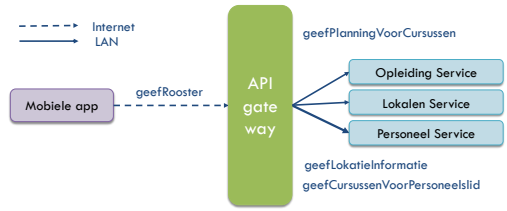
17

- Doorsturen naar de juiste service
  - Routing map
    - Bv. Http-methode + pad HTTP URL service

API Compositie

18

- Eén vraag van de client → meerdere requests naar services → resultaten samenvoegen



Vertaling protocol

19

- RESTful API voor clients
- Intern andere API's mogelijk bv. gRPC

Elk type client heeft een API

20

- Verschillende types clients – andere noden
- Bv.
  - Applicatie partner
    - Volledig rooster nodig
  - Mobiele app
    - Enkel rooster voor deze week
- API gateway voorziet voor elke client een API
  - Bv. een API voor mobiele clients, eventueel verschillende voor Android en iPhone, één voor externe partners, …

Implementeren randfunctionaliteit

21

- Belangrijkste verantwoordelijkheden API gateway
  - Routing
  - Compositie
- Daarnaast kan de API gateway ook randfunctionaliteit voorzien
  - Authenticatie
  - Autorisatie
  - Beperken aantal aanvragen
  - Caching
  - Logging
  - Metingen bv. voor facturatie

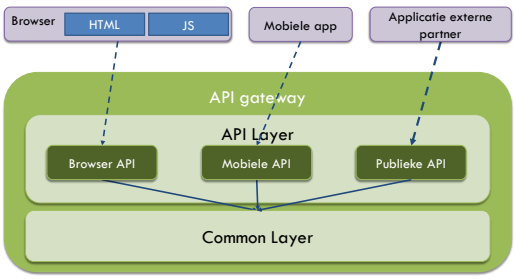
Randfunctionaliteit – waar?

22

- Backend services
  - Zinvol voor caching en metingen
  - Authenticatie beter "aan de rand"
- Aparte service voor de API gateway: eerst authenticatie, dan pas toegang tot gateway
  - Scheiding van de belangen
  - Gemeenschappelijk voor verschillende publieke API's
  - Nadeel: meer netwerkvertraging, complexere applicatie
- In de API gateway zelf

API gateway architectuur

23



API-module

24

- Implementeert API-opdrachten
- Twee opties
  - De opdracht correspondeert met één operatie van een API
    - Routing naar service
    - Eventueel vastgelegd in configuratiebestand voor een routing module
  - Meer complexe opdrachten
    - API-compositie
    - Code implementeren
      - Verschillende services aanspreken
      - Resultaten combineren

API gateway ownership model

25

- Wie is verantwoordelijk voor de ontwikkeling en onderhoud van de API gateway?
- Opties
  - Apart team
  - Client teams en API gateway team

Apart API gateway team

26

- Toegang tot een bepaalde backend service → wachten op API gateway team
- Tegen filosofie van autonome onafhankelijke teams
- API gateway- verantwoordelijkheid vaag/gedeeld
  - Verschillende teams werken aan dezelfde code
  - In strijd met 'if you build it you own it'

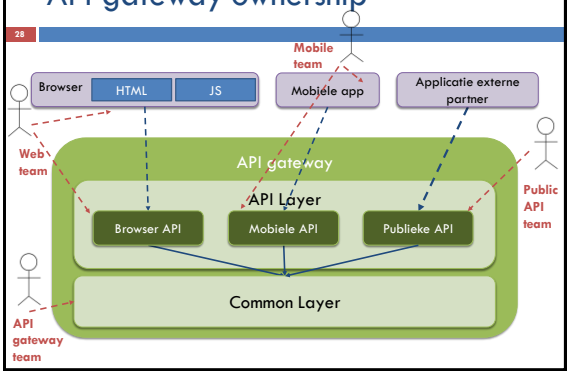
Backends for front ends pattern (BFF)

27

- Oplossing
  - Client teams en API gateway team
    - Elk client team beheert eigen API-module
    - API gateway voor elke client
    - API gateway team is verantwoordelijk voor de gemeenschappelijk module en beheersaspecten (operational)
- Backends for front ends pattern (BFF)

API gateway ownership

28



Voordelen BFF

29

- Duidelijke verantwoordelijkheden
- Elke API-module is geïsoleerd → betere betrouwbaarheid
  - Fout in één module heeft geen impact op de andere modules
- Elke API-module is een apart process → beter te observeren
- Elke API-module is apart schaalbaar
- Startoptijd wordt gereduceerd
  - API gateways zijn kleiner en eenvoudiger

Voordelen API gateway

30

- Afschermen interne structuur applicatie
  - Clients spreken niet met de specifieke services, maar met de gateway
- API is specifiek voor de client
  - Minder berichten heen en weer
  - Eenvoudigere client code

Nadelen API gateway

31

- Extra component om te ontwikkelen, te publiceren en te beheren
  - Altijd beschikbaar (highly available)
- Risico op bottleneck
  - Ontwikkelaars moeten API gateway aanpassen om hun services beschikbaar te stellen
    - Update proces moet licht (lightweight) zijn
    - Anders risico op wachtrij van ontwikkelaars

Voorbeeld

32

- Netflix
  - Aparte API voor elk type toestel

Inhoud hoofdstuk

33

- Verschillende aspecten/overwegingen bij externe API-ontwerp
  - API's die verschillende clients ondersteunen – een uitdaging
- API gateway pattern en backends voor front end pattern
- Ontwerpen en implementeren van een API gateway
  - Kant en klaar producten
  - Frameworks voor eigen ontwikkeling (Spring Cloud Gateway, GraphQL)
- API samenstellen vereenvoudigen met reactive programming

Ontwikkeling API gateway

34

- Rekening houden met de volgende factoren
  - Performantie en schaalbaarheid
  - Onderhoudbare code
  - Afhandelen gedeeltelijk falen
  - De architectuur van de applicatie volgen

Performantie en schaalbaarheid

35

- API gateway = voordeur applicatie
  - Alle aanvragen passeren hier
  - Performantie en schaalbaarheid is dus belangrijk
- Synchrone of asynchrone IO?

Performantie en schaalbaarheid

36

- Synchrone IO
  - Elke connectie in een thread (~servlets)
    - Beperkt aantal threads → beperkt aantal gelijktijdige verbindingen

Performantie en schaalbaarheid

37

- Asynchrone IO (non-blocking)
  - Één eventloop verdeelt IO-requests over eventhandlers (~nodeJS)
  - Geen overhead van verschillende threads
  - Callback-based programming → complexer
  - Eventhandlers moeten snel terugkeren → voorkomen blokkeren eventloop
- Kan maar hoeft geen verbetering te zijn

Onderhoudbare code

38

- API-compositie roept verschillende backend services aan
- Sommige services hebben resultaten van andere services nodig
  - Verschillende services sequentieel oproepen → antwoordtijd = som antwoordtijd verschillende services
  - Gelijktijdig oproepen vermindert antwoordtijd
- Uitdaging onderhoudbare "concurrente" code schrijven

Onderhoudbare code

39

- Asynchrone en event-driven IO maakt gebruik van callbacks
- Code voor API-compositie met traditionele aanpak via asynchrone callback is moeilijk onderhoudbaar (callback hell)
  - Tangled
  - Moeilijk te begrijpen
  - Foutgevoelig, zeker bij een mix van parallele en sequentiële aanvragen
- Beter: reactive programming

Onderhoudbare code

40

- Reactive programming
  - Java
    - Java 8 CompletableFutures
    - Project Reactor Monos
    - RxJava (Reactive Extensions for Java) Observables, which was created byNetflix specifically to solve this problem in their API gateway
    - Scala Futures
  - Javascript
    - Promises
    - RxJS

Afhandelen gedeeltelijk falen

41

- API gateway moet betrouwbaar zijn
  - Load balancer
    - Instantie faalt → doorgestuurd naar andere instantie
  - Gefaalde aanvragen of te trage aanvragen degelijk afhandelen

Volgen applicatiearchitectuur

42

- Als de architectuur van de applicatie een aantal patterns gebruikt voor zijn service bv. voor het ontdekken of observeren van services, dan moet de API gateway die ook volgen

Implementatie API gateway

43

- Routering van aanvragen
- API-compositie
- Randfunctionaliteit
- Vertaling van het protocol
- Applicatiearchitectuur volgen

Implementatie API gateway

44

- Opties
  - Een klant en klaar product gebruiken (OFS off-the-shelf API gateway)
    - Ondersteunt geen API-compositie
  - Eigen API gateway implementeren
    - API gateway framework
    - Web framework

Klant en klaar

45

- AWS API gateway (Amazone Web Services)
- AWS application load balancer
- Kong (gebaseerd op NGINX HTTP server)
- Traefik (in GoLang)

Zelf ontwikkelen

46

- Webapplicatie die proxy is voor andere services
  - Mechanisme voor routering voorzien → verminderen code
  - HTTP proxy-ing gedrag juist implementeren (bv. afhandelen headers)
- Gebruik een framework → minder code zelf schrijven
  - Netflix Zuul
  - Spring Cloud Gateway